**Sayan Banerjee**

UWM, EE-890/490

Mini-project 3 report

**Mini Project 3: Hand Written Digit Recognition Using a Single Layer Perceptron**

In this project, I have used single layer 2-D Perceptron to predict Hand Written Digits.

The Learning Rate I have used in this system is time/iteration variant.

Learning Rate(n)= Initial Learning Rate/(1+(n/TAU)), Where Initial Learning Rate and TAU are User Defined Constant values, and n is the current number of iteration. As the iteration goes by the Learning Rate will have smaller and smaller value.

Accuracy, Precision and Recall of the system strongly depend on the Particular digit the system is working currently and their structure overlap with the other digits(Negative Class).

The training Labels have been converted to [1,-1] depending on the particular digit the System is looking at.

When Training with “1” the system gives the best performance on the Test Dataset. The accuracy is 99%. But the same system when training with “8” gives the worst performance where the accuracy is 94.3%. As this a single layer Perceptron, the measure of accuracy can’t really tell us the how good the system is when predicting a particular digit. To get the accurate picture of how the system is performing for a particular digit we need turn towards Precision and Recall.

The system gives the best Precision for digit: 1 and the wort for dight “8” and the values are 0.9531 abd 0.6509 respectively.

The system gives the best Reacll for digit: 1 and the wort for dight “8” and the values are 0.9683 abd 0.7753 respectively.

The reason behind is “8” has most overlapping structure in the dataset than any other digit and “1” has least overlapping structure with the rest of digits. So, the learned weight from the training set can almost perfectly distinguish beween “1” and other digits where as it fails to provide a good performace when it comes to predicting “8”.

I have made some tweaks in the implementation to make the system perform better.

1. I have normalised the all the feature values by deviding the “train” matrix, 255 (max value). This makes all the feature values in the range of [0,1].
2. I have taken TAU=2500.00 and Initial Learning Rate = 10.0
3. \*\*When training I have made System to go over every training example by no. of “iterations” (user defined constant). With every iteration the Initial Learning Rate got decreased. The formula for that is:

Initial Learning Rate(iter)= Initial Learning Rate /(3^(iter-1))

**Pseudo Code:**

For iter = 1: iterations

Learning Rate(iter)= Initial Learning Rate /(3^(iter-1))

For n= 1: No. of Examples

\*\*\*\*Do the Training\*\*\*\*

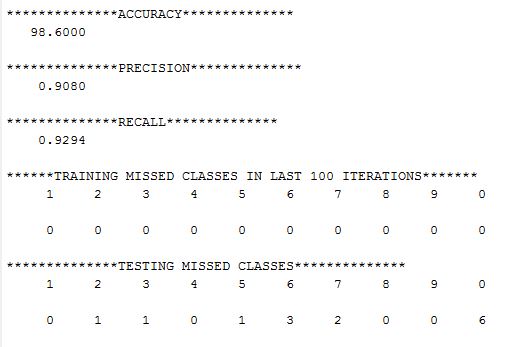
Learning Rate(iter,n)= Initial Learning Rate(iter)/(1+(n/TAU))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

End

End

**System Performance: Digit: 0**

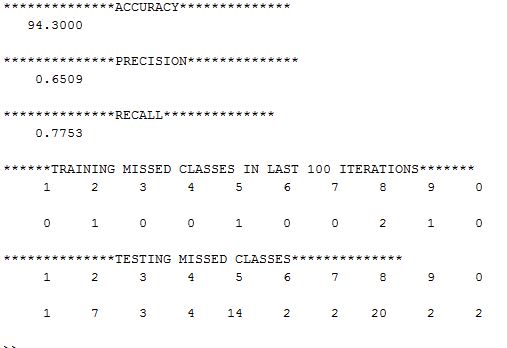


**Learned Weight for Digit: 0**



The picture of the Learned Weight for “0” shows that what are the general features got extracted from train examples of “0” picture. The black color indicates negative or 0 weight whereas the white color indicates the positive weights. From the learned weight picture “0” we can say a hand-written digit picture to be recognized as “0”, the test input need to have positive pixel values the same places as the white colors appear in this picture.

**System Performance: Digit: 8**



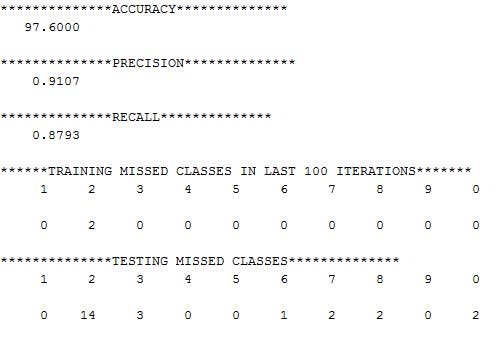
**Learned Weight for Digit: 8**



The picture of the Learned Weight for “8” shows that what are the general features got extracted from train examples of “8” picture. The black color indicates negative or 0 weight whereas the white color indicates the positive weights. From the learned weight picture “8” we can say a hand-written digit picture to be recognized as “8”, the test input need to have positive pixel values the same places as the white colors appear in this picture.

For this system, it appears that the system gives its worst performance for digit “8”. The reason behind this performance failure is clear from the learned weight picture for digit “8”. We can fit any digits on the feature map learned for digit “8”. Because of this most overlapping structure of digit “8”, the system performs worst for digit “8”.

**System Performance: Digit: 2**



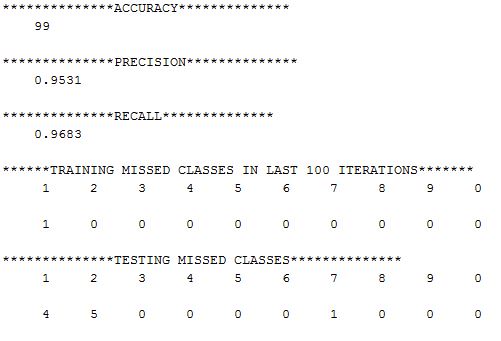
**Learned Weight for Digit: 2**



Performance for digit “2” is far better than the system’s performance for digit “8”. The accuracy got raised to 97.60%, Precision 91.07% and Recall with 87.93%. The picture of the Learned Weight for “2” shows that what are the general features got extracted from train examples of “2” picture. The black color indicates negative or 0 weight whereas the white color indicates the positive weights. From the learned weight picture “2” we can say a hand-written digit picture to be recognized as “2”, the test input need to have positive pixel values the same places as the white colors appear in this picture.

From the high precision and relatively low recall we can say that the system gives positive output with high confidence only. “2” has most over-lapping structure with “3”, “7”, “8” and “0”.

**System Performance: Digit: 1**



**Learned Weight for Digit: 1**



The system’s performance for digit “1” is the highest. It gives 99.00% accuracy on the test set. The Precision and Recall for predicting digit “1” are 0.9531 and .9653 respectively. 1 has least overlapping structure with other digits. It only has overlapping structure with “7” and “2” and this is the main reason behind the better performance of the system than other digits.

The picture of the Learned Weight for “1” shows that what are the general features got extracted from train examples of “1” picture. The black color indicates negative or 0 weight whereas the white color indicates the positive weights. From the learned weight picture “1” we can say a hand-written digit picture to be recognized as “1”, the test input need to have positive pixel values the same places as the white colors appear in this picture.

**MATLAB Code:**

clear;

clc;

d=2;

ILR=10.0;

TAU=2500.0;

error=0;

Train\_ErrorClass=[1 2 3 4 5 6 7 8 9 0];

Test\_ErrorClass=[1 2 3 4 5 6 7 8 9 0];

Train\_ErrorClassCount=zeros(1,10);

Test\_ErrorClassCount=zeros(1,10);

TruePositive=0;

TrueNegative=0;

FalsePositive=0;

FalseNegative=0;

iteration=10;

load digits.mat

for k = 1:5000

dummy = train(:,k) ;

for i = 1:28

for j = 1:28

%x(i,j,k) = double(dummy((i-1)\*28 + j)) ;

end

end

end

train\_T=double(train');

test\_T=double(test');

[q,e]=size(train\_T);

Train\_Normalized\_Features=train\_T/255;

Train\_Normalized\_Features=[ones(q,1) Train\_Normalized\_Features];

train\_label=oneHotEncoding(trainlabels,d);

Test\_Normalized\_Features=test\_T/255 ;

[q,e]=size(test\_T);

Test\_Normalized\_Features=[ones(q,1) Test\_Normalized\_Features];

test\_label=oneHotEncoding(testlabels,d);

%=========================================================================================================

%======================TRAINING=======================

[q,e]=size(Train\_Normalized\_Features);

W=zeros(1,e);

for iter= 1:iteration

ILR=ILR/(3^(iter-1));

for i=1:q

value=Train\_Normalized\_Features(i,:)\*W';

if value > 0

v=1;

else

v=-1;

end

if train\_label(i,1)~=v && i>4900 && iter==iteration

error=error+1;

disp ('==============\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*===============');

disp('ERROR');

disp(error);

disp('ITERATION');

disp(i);

disp('ORIGINAL LABEL');

disp (trainlabels(i,1));

disp ('==============\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*===============');

msdcls=trainlabels(i,1);

if msdcls==0

Train\_ErrorClassCount(1,10)=Train\_ErrorClassCount(1,10)+1;

else

Train\_ErrorClassCount(1,msdcls)= Train\_ErrorClassCount(1,msdcls)+1;

end

end

if train\_label(i,1)~=v && v==1

LR= ILR/(1+(i/TAU));

W=W-LR\*(Train\_Normalized\_Features(i,:));

%LR = 0.0;

end

if train\_label(i,1)~=v && v==-1

LR= double(ILR/(1.0+double((i/TAU))));

disp(LR);

W=W+LR\*(Train\_Normalized\_Features(i,:));

%LR = 0.0;

end

end

end

disp(LR);

%=========================================================================================================

%======================TESTING=======================

disp('\*\*\*\*\*\*\*\*\*\*\*\*\*\*TESTING\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

total\_error=0;

[q,e]=size(Test\_Normalized\_Features);

for i=1:q

value=0.0;

v=0;

value=Test\_Normalized\_Features(i,:)\*W';

if value >= 0

v=1;

else

v=-1;

end

if test\_label(i,1)~=v

total\_error=total\_error+1;

if v==1

FalsePositive=FalsePositive+1;

end

if v==-1

FalseNegative=FalseNegative+1;

end

disp ('==============\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*===============');

disp('ERROR');

disp(total\_error);

disp('ITERATION');

disp(i);

disp('ORIGINAL LABEL');

disp (testlabels(i,1));

disp ('==============\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*===============');

msdcls=testlabels(i,1);

if msdcls==0

Test\_ErrorClassCount(1,10)=Test\_ErrorClassCount(1,10)+1;

else

Test\_ErrorClassCount(1,msdcls)= Test\_ErrorClassCount(1,msdcls)+1;

end

else

if v==1

TruePositive=TruePositive+1;

end

if v==-1

TrueNegative=TrueNegative+1;

end

end

end

disp('\*\*\*\*\*\*\*\*\*\*\*\*\*\*ACCURACY\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

accuracy=((q-total\_error)/q)\*100;

disp (accuracy);

disp('\*\*\*\*\*\*\*\*\*\*\*\*\*\*PRECISION\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

precision=TruePositive/(TruePositive+FalsePositive);

disp(precision);

disp('\*\*\*\*\*\*\*\*\*\*\*\*\*\*RECALL\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

recall=TruePositive/(TruePositive+FalseNegative);

disp(recall);

Weights=W(1,2:785);

%Weights=(Train\_std.\*(Weights))+Train\_mean;

W\_D=reshape(Weights,28,28);

imshow(W\_D);

%imagesc(W\_D)

colormap('gray')

colorbar

pbaspect([1 1 1])

disp('\*\*\*\*\*\*TRAINING MISSED CLASSES IN LAST 100 ITERATIONS\*\*\*\*\*\*\*');

disp(Train\_ErrorClass);

disp(Train\_ErrorClassCount);

disp('\*\*\*\*\*\*\*\*\*\*\*\*\*\*TESTING MISSED CLASSES\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

disp(Test\_ErrorClass);

disp(Test\_ErrorClassCount);

% oneHotEncoding

function [x]=oneHotEncoding(labels,d)

x= zeros(size(labels));

[q,e]=size(labels);

for i=1:q

if (labels(i,1)==d)

x(i,1)=1;

else

x(i,1)=-1;

end

end

end