**Assignment 3**

**Problem 1:**

1) Try out the tutorial on TensorFlow. You can find details from https://www.tensorflow.org/versions/ r1.1/get\_started/mnist/beginners. The tutorial explains multi-class logistic regression with softmax on MNIST dataset. Use the same setting that you used in your Homework 2 and compare its result to what you had from Homework 2. Do they match?

*Output for the Homework3: Problem 1: Softmax with TensorFlow:*

Extracting MNIST\_data/train-images-idx3-ubyte.gz

Extracting MNIST\_data/train-labels-idx1-ubyte.gz

Extracting MNIST\_data/t10k-images-idx3-ubyte.gz

Extracting MNIST\_data/t10k-labels-idx1-ubyte.gz

Accuracy: 0.9205

*Output for the homework2: Problem 2:*

*OUTPUT:*

Epoch 0. Loss: 1.31376479625 Train\_acc 0.858383 Test\_acc 0.8625

Epoch 1. Loss: 0.595704523579 Train\_acc 0.882967 Test\_acc 0.8857

Epoch 2. Loss: 0.502324586852 Train\_acc 0.894133 Test\_acc 0.8967

Epoch 3. Loss: 0.452291325553 Train\_acc 0.899667 Test\_acc 0.8997

Epoch 4. Loss: 0.419943816221 Train\_acc 0.903817 Test\_acc 0.9013

Epoch 5. Loss: 0.396343981425 Train\_acc 0.906917 Test\_acc 0.9059

Epoch 6. Loss: 0.377986954149 Train\_acc 0.907383 Test\_acc 0.9039

Epoch 7. Loss: 0.364052859215 Train\_acc 0.912767 Test\_acc 0.9098

Epoch 8. Loss: 0.352651608233 Train\_acc 0.91275 Test\_acc 0.9071

Epoch 9. Loss: 0.343372354233 Train\_acc 0.91695 Test\_acc 0.9113

Epoch 10. Loss: 0.334975646194 Train\_acc 0.918167 Test\_acc 0.9139

Epoch 11. Loss: 0.327387751714 Train\_acc 0.917267 Test\_acc 0.9104

Epoch 12. Loss: 0.321502842379 Train\_acc 0.921183 Test\_acc 0.9151

Epoch 13. Loss: 0.31585445176 Train\_acc 0.921517 Test\_acc 0.9151

Epoch 14. Loss: 0.311217189145 Train\_acc 0.918417 Test\_acc 0.9124

Epoch 15. Loss: 0.307078578277 Train\_acc 0.9207 Test\_acc 0.9149

Epoch 16. Loss: 0.302349377406 Train\_acc 0.922583 Test\_acc 0.9159

Epoch 17. Loss: 0.299740602593 Train\_acc 0.922167 Test\_acc 0.9155

Epoch 18. Loss: 0.295952921053 Train\_acc 0.924017 Test\_acc 0.9163

Epoch 19. Loss: 0.292886624074 Train\_acc 0.92325 Test\_acc 0.9161

testing\_confusion\_matrix:

[[ 948 0 2 4 0 7 14 3 2 0]

[ 0 1108 5 2 0 4 4 3 9 0]

[ 5 7 920 20 4 4 17 9 37 9]

[ 1 0 14 939 0 19 3 14 17 3]

[ 2 2 9 5 880 1 17 7 8 51]

[ 10 1 3 53 7 749 20 8 32 9]

[ 6 3 9 1 8 11 915 3 2 0]

[ 0 8 22 10 4 0 0 954 3 27]

[ 8 9 11 42 7 32 12 14 826 13]

[ 8 10 1 14 13 7 2 25 7 922]]

sanity check: 10000

True\_Positive: [ 948 1108 920 939 880 749 915 954 826 922]

False\_Positive: [ 40 40 76 151 43 85 89 86 117 112]

False\_Negative: [ 34 29 114 73 104 145 45 76 150 89]

[ 0.96537678 0.97449428 0.88974855 0.92786561 0.89430894 0.83780761

0.953125 0.92621359 0.84631148 0.91196835]

Final Recall:

0.912722019242

[ 0.95951417 0.96515679 0.92369478 0.86146789 0.95341278 0.89808153

0.91135458 0.91730769 0.87592789 0.89168279]

Final Precision:

0.915760090166

accuracy: 0.9161

**Conclusion:**

1. **The program designed from the scratch takes approximately 30 minutes to run the program that contains 10000 epochs and produce the output with accuracy of** 0.9161.
2. **The tensorflow tutorial runs in micro seconds and produces the accuracy of** 0.9205.
3. **Thus, using the tensorflow library to run the program increases the precision, accuracy and recall also reducing the execution time.**

2) Try out Convolution Neural Network (CNN) tutorial on MNIST using Tensorflow. You can find details from https://www.tensorflow.org/versions/r1.1/get\_started/mnist/pros. Use the images of 0 to 4 (i.e., 4 classes) and see if the classification accuracy improves over the previous result.

*Output for the Homework3: Problem 1: CNN with TensorFlow:*

*OUTPUT:*

Extracting MNIST\_data/train-images-idx3-ubyte.gz

Extracting MNIST\_data/train-labels-idx1-ubyte.gz

Extracting MNIST\_data/t10k-images-idx3-ubyte.gz

Extracting MNIST\_data/t10k-labels-idx1-ubyte.gz

WARNING:tensorflow:From <ipython-input-1-9b3e4ab67cef>:12: softmax\_cross\_entropy\_with\_logits (from tensorflow.python.ops.nn\_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow

into the labels input on backprop by default.

See tf.nn.softmax\_cross\_entropy\_with\_logits\_v2.

0.9207

step 0, training accuracy 0.1

step 500, training accuracy 0.98

step 1000, training accuracy 1

step 1500, training accuracy 0.98

step 2000, training accuracy 0.98

step 2500, training accuracy 0.98

step 3000, training accuracy 0.96

step 3500, training accuracy 0.94

step 4000, training accuracy 1

step 4500, training accuracy 1

step 5000, training accuracy 0.98

step 5500, training accuracy 1

step 6000, training accuracy 1

step 6500, training accuracy 1

step 7000, training accuracy 1

step 7500, training accuracy 0.98

step 8000, training accuracy 1

step 8500, training accuracy 0.98

step 9000, training accuracy 1

step 9500, training accuracy 1

test accuracy 0.9902

*Output for the homework2: Problem 2:*

*OUTPUT:*

Reading data

data read and cleaned

Epoch: 0

Epoch 0

Training Accuracy: 93.2583333333

Epoch 0 : Testing Accuracy: 95.06 %

Epoch: 1

Epoch 1

Training Accuracy: 96.5283333333

Epoch 1 : Testing Accuracy: 96.11 %

Epoch: 2

Epoch 2

Training Accuracy: 97.3216666667

Epoch 2 : Testing Accuracy: 96.61 %

Epoch: 3

Epoch 3

Training Accuracy: 97.7216666667

Epoch 3 : Testing Accuracy: 96.72 %

Epoch: 4

Epoch 4

Training Accuracy: 98.14

Epoch 4 : Testing Accuracy: 96.83 %

Testing confuion matrix

[[ 967 1 1 0 0 1 3 3 3 1]

[ 0 1125 0 3 0 1 1 1 4 0]

[ 3 2 999 2 4 1 6 5 7 3]

[ 0 0 3 977 0 15 0 4 5 6]

[ 1 0 2 0 944 0 7 0 1 27]

[ 4 3 0 4 3 853 10 2 7 6]

[ 6 4 0 0 1 13 929 0 5 0]

[ 0 7 12 2 4 0 0 983 3 17]

[ 4 1 3 2 8 4 6 3 936 7]

[ 5 6 0 5 7 2 3 4 7 970]]

true positive:

[ 967 1125 999 977 944 853 929 983 936 970]

False Positive:

[23 24 21 18 27 37 36 22 42 67]

False Negative:

[ 980 1135 1032 1010 982 892 958 1028 974 1009]

True Negative:

[8997, 8841, 8947, 8972, 8991, 9071, 9006, 8950, 8984, 8924]

label precision recall

0.000000 0.977 0.987

1.000000 0.979 0.991

2.000000 0.979 0.968

3.000000 0.982 0.967

4.000000 0.972 0.961

5.000000 0.958 0.956

6.000000 0.963 0.970

7.000000 0.978 0.956

8.000000 0.957 0.961

9.000000 0.935 0.961

precision total: 0.968107135952

precision total: 0.967914337655

accuracy: 0.9683

**Conclusion:**

1. **The program designed from the scratch takes approximately 30 minutes to execute the program with 10000 epochs while the program that uses the Tensorflow uses hardly 15 minutes to execute the programs for 10000 epochs.**
2. **Also, the Tensorflow program provides the better accuracy than that of the program that was developed from scratch. The accuracy the Tensorflow program provides is more than 99% which is almost equal to 1. That means the algorithm classifies almost all images correctly.**
3. **The program developed from scratch has accuracy 96% so it still makes mistake to identify the classes for the images.**

**Problem 2:**

1. Implement hard-margin linear SVM. You can use a library (e.g. MATLAB ”quadprog” function) to solve the quadratic programming problem. For the rest of the implementation, write your own code.

*Formula for Hard Margin SVM:*

*Where* w= weights

b=bias.

*Answer:*

*OUTPUT:*

Running Fold: 1 for the Hard Margin SVM:

The Hard-Margin SVM. Please wait...

*Accuracy is*  97.9

svm: [97.9]

Running Fold: 2 for the Hard Margin SVM:

The Hard-Margin SVM. Please wait...

Accuracy is 97.95

svm: [97.9, 97.95]

Running Fold: 3 for the Hard Margin SVM:

The Hard-Margin SVM. Please wait...

Accuracy is 97.875

svm: [97.9, 97.95, 97.875]

Running Fold: 4 for the Hard Margin SVM:

The Hard-Margin SVM. Please wait...

Accuracy is 97.825

svm: [97.9, 97.95, 97.875, 97.825]

Running Fold: 5 for the Hard Margin SVM:

The Hard-Margin SVM. Please wait...

Accuracy is 97.925

svm: [97.9, 97.95, 97.875, 97.825, 97.925]

Average accuracy for SVM 97.89500000000001

*Explanation:*

*The program uses the* CVXOPT *library in python to solve the quadratic equation. The program uses the 2 fold cross validation (5x2) for 5 times as it* is generally better for determining approximate average error *for the algorithm.*

*The hard margin Support vector has no slack thus the slack value=0*

*Output Explanation:*

*The SVM runs the two fold cross validation 5 times on the faces database. The accuracy is calculated for each fold. This accuracy is saved in an array named ‘svm’. Each time the accuracy is calculated the array named ‘svm’ is appended to add one more element in the array. When the two fold cross validation is run 5 times, it calculates the average of all the accuracies saved in the ‘svm’. That is said to be the final accuracy of the SVM.*

2) Implement soft-margin linear SVM. You can use a library to solve the quadratic programming problem. For the rest of the implementation, write your own code. Try different parameters for the slack variable, C = {0.001, 0.01, 1, 10, 100}. How does the result change over different C? Also, compare the result with the hard-margin case.

*Answer:*

Where i=1,2,3…..,*m*

W= weights

C= Slack Variable

Two-Fold Cross Validation:

We randomly divide the data into 2 blocks or, randomly divide each category into two blocks if doing stratified cross-validation. Then, train on block A and evaluate on B. Next, we reserve it and train on B and evaluate on A. Then, repeat the process. We divide the data randomly into two blocks using a different seed value. Perform the two evaluations. Then, repeat again (actually, until you have done this 5 times).

OUTPUT:

Enter any number between 1-5(or 0 to quit):

1: SVM with c=0.001

2: SVM with c=0.01

3: SVM with c=1

4: SVM with c=10

5: SVM with 100

0: To quit

Please enter your choice: 5

Running SVM with c=100

Running Fold: 1 for SVM

Running SVM. Please wait...

Accuracy is 96.55

svm: [96.55]

Running Fold: 2 for SVM

Running SVM. Please wait...

Accuracy is 96.05

svm: [96.55, 96.05]

Running Fold: 3 for SVM

Running SVM. Please wait...

Accuracy is 96.975

svm: [96.55, 96.05, 96.975]

Running Fold: 4 for SVM

Running SVM. Please wait...

Accuracy is 96.8

svm: [96.55, 96.05, 96.975, 96.8]

Running Fold: 5 for SVM

Running SVM. Please wait...

Accuracy is 96.35

svm: [96.55, 96.05, 96.975, 96.8, 96.35]

Average accuracy for SVM 96.545

Enter any number between 1-5(or 0 to quit):

1: SVM with c=0.001

2: SVM with c=0.01

3: SVM with c=1

4: SVM with c=10

5: SVM with 100

0: To quit

Please enter your choice: 0

*Output Explanation:*

*The program gives the user an option to pick a slack variable from the {0.001, 0.01, 1, 10, 100}. The user MUST pick the values between 1-5 or 0 to quit. The program then takes this value and calls the cross\_validate function which in turn calls the soft\_margin\_svm with the same value to set the slack variable value. Then the program divides the dataset into 2 different parts for training and testing. And then runs the 2 fold cross validation 5 times to get better average error for the SVM algorithm. And calculate the accuracy for that fold and store it in an array. Then it will calculate the average for the accuracy to give the average accuracy for the 5 times that it runs the 2 fold cross validation.*

*Output Explantion:*

*Program will ask for the user input in the form of a switch case where user must enter any number between 1-5 or 0 to quit (entering a float value will throw an exception) and then the program runs the 2 fold cross validation for 5 times and shows the accuracy for each time. This accuracy will then be stored in an array and then after all the five times the 2 fold cross validation is run it will calculate the average accuracy.*