CSE474/574 Introduction to Machine Learning
Programming Assignment 2
Report

Group number:
CSE574 Programming Assignment 14
Group Members:
Vinita Venktesh Chappar
Prajit Krisshna Kumar
Harsh Hemendra Shah

**All the time values mentioned in the report are in "Seconds"*

Part I - Sentiment Analysis

Approach 1 - Word Count Vectorization

Report 1

In this approach, we implemented a simple hand-crafted classifier that can label a movie review as positive or negative.
Although simple, this algorithm takes a lot of time to execute (50 mins atleast) depending on the size of the data which is not at all ideal.
Moreover, the accuracy that we got was also not upto the mark:

In [102]:
```
predictions_test = []
for r in reviews_test:
    l = nonml_classifier(r,pos_neg_ratios)
    predictions_test.append(l)

# calculate accuracy
correct = 0
for l,p in zip(sentiments_test,predictions_test):
    if l == p:
        correct = correct + 1
print('Accuracy of the model = {}'.format(correct/len(sentiments_test)))
```

```
Accuracy of the model = 0.762
```

As you can see, the accuracy of the model is 0.76 i.e 76% , which is not so good.

Approach 2 - Neural Network Based Sentiment Classification

Here, we are going to solve the sentiment problem using a machine learning model instead of a simple hand-crafted classifier. Then we'll compare our results to determine which approach is better

Report 2

Input vairables:
num_epochs = 50
n_hidden_1 = 10
Layers = 1 (Singleton preceptron)
Avg Time for execution: 71.09334 seconds.

	Training Accuracy	Test Accuracy
10	0.903125	0.84625
20	0.929583	0.84
30	0.9255	0.85125
40	0.939417	0.8425
50	0.932	0.85625

Avg Training data accuracy: 0.92
Avg Test data accuracy:0.85

Q. Is it better or worse than your rule-based algorithm implemented in Approach 1?
As we can see, the avg accuracy we got using the neural networks for test data was 0.85. This accuracy is far greater than what we got with the approach 1. That is why neural network models are preferred.
Another thing to note is that, the model reduced the execution time significantly.

Report 3

Here, we'll test the neural network accuracies for different set of input values.

1. **Increasing width**
   Input vairables:
   num_epochs = 50
   n_hidden_1 = 10
   Layers = 1 (Singleton preceptron)
   Avg Time for execution: 21.33644580 seconds

No. of Epochs	Training Accuracy	Test Accuracy
10	0.50025	0.5
20	0.500583	0.50125
30	0.7575	0.76125
40	0.768667	0.7475
50	0.78725	0.7575

   Conclusions: Increasing the width brought time down but also brought down the accuracy to 0.75.

2. **Increasing number of Epochs** Input vairables:
   num_epochs = 100
   n_hidden_1 = 10
   Layers = 1 (Singleton preceptron)
   Avg Time for execution: 142.98443031 seconds

No. of Epochs	Training Accuracy	Test Accuracy
10	0.877833	0.8425
20	0.912417	0.85
30	0.928583	0.86
40	0.926208	0.85625
50	0.917625	0.86625
60	0.94325	0.86375
70	0.941292	0.86625
80	0.935292	0.8625
90	0.951667	0.86
100	0.951917	0.85375

   Conclusions: Increasing the number of Epochs, increased the execution time considerably as the number if iterations increased. But the accuracy of the system increased significantly from 0.75 to 0.85

3. **Increasing number of Epochs and width of hidden layers**
   Input vairables:
   num_epochs = 100
   n_hidden_1 = 50
   Layers = 1 (Singleton preceptron)
   Avg Time for execution: 164.494992 seconds

No. of Epochs	Training Accuracy	Test Accuracy
20	0.888667	0.845
40	0.908542	0.85125
60	0.919542	0.8475
80	0.930458	0.85
100	0.932833	0.86375

Conclusions: By increasing both the number of epochs and the width, we successfully increased the accuracy but the execution time was increased drastically.

4. **Increasing number of layers**
   Input vairables:
   num_epochs = 50
   n_hidden_1 to n_hidden_5 = 10
   Layers = 5 (Multilayered preceptron)
   Avg Time for execution: 79.319952011 seconds

No. of Epochs	Training Accuracy	Test Accuracy
10	0.876958	0.84125
20	0.893	0.8375
30	0.910125	0.86
40	0.918958	0.855
50	0.920917	0.84875

   Conclusions: Increasing the complexity of the model by adding more layers, increased the accuracy as well as the execution time.

5. **Increasing number of layers and width**
   Input vairables:
   num_epochs = 50
   n_hidden_1 to n_hidden_5 = 50
   Layers = 5 (Multilayered preceptron)
   Avg Time for execution: 73.580284595489

No. of Epochs	Training Accuracy	Test Accuracy
10	0.895417	0.83
20	0.922542	0.845
30	0.929708	0.85
40	0.933583	0.85625
50	0.934625	0.8575

   Conclusions: Here, the layers and the width were increased. This made the accuracy go high and time to go down.

6. **Increasing number of layers and Epoch**
   Input vairables:
   num_epochs = 100
   n_hidden_1 to n_hidden_5 = 10
   Layers = 5 (Multilayered preceptron)
   Avg Time for execution: 145.27267622947693 seconds

No. of Epochs	Training Accuracy	Test Accuracy
10	0.885583	0.8375
20	0.913625	0.84875
30	0.925625	0.84625
40	0.925542	0.8375
50	0.932625	0.83375
60	0.931125	0.84625
70	0.936042	0.845
80	0.935583	0.84625
90	0.937042	0.84375
100	0.940333	0.83375

Conclusions: When we increase the number of epochs, naturally the execution time increases due to increase in the number of iterations. The accuracy decreased, but not by much.

7. **Increasing number of layers, epochs and width of the hidden layers.**
   Input vairables:
   num_epochs = 100
   n_hidden_1 to n_hidden_5 = 50
   Layers = 5 (Multilayered preceptron)
   Avg Time for execution: 167.945810079574 seconds

No. of Epochs	Training Accuracy	Test Accuracy
10	0.876833	0.835
20	0.9025	0.835
30	0.917542	0.85625
40	0.921792	0.85
50	0.923833	0.86375
60	0.927833	0.855
70	0.923708	0.8525
80	0.918458	0.83875
90	0.914708	0.8425
100	0.918583	0.825

Conclusions : Here, as we increased the width, the accuracy went down from 0.83 to 0.82.

8. **Removing neutral words to compare the performance**
   Input vairables:
   num_epochs = 100
   n_hidden_1 to n_hidden_5 = 50
   Layers = 5 (Multilayered preceptron)
   Avg Time for execution: 177.75455904006958 seconds

No. of Epochs	Training Accuracy	Test Accuracy
10	0.864708	0.8225
20	0.878042	0.83375
30	0.901875	0.84375
40	0.906292	0.87
50	0.913417	0.8375
60	0.917125	0.8675
70	0.917083	0.85125
80	0.919917	0.8625
90	0.917583	0.8675
100	0.925417	0.85125

Conclusion: When the neutral words are ignored, the accuracy went up from 0.82 to 0.85. While, the execution time increased because of the extra implementation of removing the neutral words.

Part II - Image Classification on the AI Quick Draw Dataset

Here, we are provided with a large dataset from the AI Quick draw tool. Firstly, we studied the performance of a neural network (Multilayered Perceptron) having a single layer. Next, we tried to increase the hidden layers to compare their accuracies. We also studied the performance of the network when the resolution of the image changes.
The Code attached consists of the multilayered perceptron approach with 5 layers and ignore words functionality implemented.

Report 4

---

1. **Run the evaluation of the 2 hidden layer neural network in the notebook - PA2-Part2.ipynb and report the test accuracy and the run time.**

```
#with two layers(28*28)
predict_test = model.predict_classes(data_test)
acc_test = np.where(label_test1==predict_test)[0].shape[0]/data_test.shape[0]
print('Testing accuracy{}'.format(acc_test))
print("time taken : "+ str(time_taken))
```

```
Testing accuracy0.24928
time taken : 7160.738799571991
```

Output:
Test data accuracy: 0.24928
Time taken : 7160.738799571991

2. **Compare the performance when the number of hidden layers are increased to 3 and 5.**

```
#with three layers(28*28)
#time taken 8926.286591 seconds
predict_test = model.predict_classes(data_test)
acc_test = np.where(label_test1==predict_test)[0].shape[0]/data_test.shape[0]
print('Testing accuracy{}'.format(acc_test))
```

```
Testing accuracy0.23084
```

Output:
Test data accuracy: 0.23084
Time taken : 8926.286591

Conclusions: As you can see here, the accuracy went from 0.24928 to 0.23084 after adding one more layer to the neural network. Not a significant change, but we can see that accuracy went down a bit.

```
#with five layers(28*28)
predict_test = model.predict_classes(data_test)
acc_test = np.where(label_test1==predict_test)[0].shape[0]/data_test.shape[0]
print('Testing accuracy{}'.format(acc_test))
print("time taken : "+ str(time_taken))
```

```
Testing accuracy0.1
time taken : 13418.315411806107
```

Output:
Test data accuracy: 0.1
Time taken : 13418.315411806107

Conclusions: Here, the accuracy of the network dropped furthermore when 5 layers were added. Hence, we can say that as the complexity of the model increases, the accuracy decreases.
As for the time, the performance is directly proportional to the complexity. Time for execution increases as the number of layers decreases.

3. **Use the resize images() function to reduce the resolution of the images to (20 x 20), (15 x 15), (10 x 10) and (5 x 5). Using the 2 hidden layer architecture, compare the performance at different resolutions, including the original (28 x 28) resolution, both in terms of test accuracy and time.**

   For a 2 layered Perceptron with the original image size (28x28), we had the folloing observations:
   Test data accuracy: 0.24928
   Time taken : 7160.738799571991
   Let's resize the image and compare the performance:

```
#20*20  image resize ( 2 layer)
data_test = resize_images(data_test,(20,20))
predict_test = model.predict_classes(data_test)
acc_test = np.where(label_test1==predict_test)[0].shape[0]/data_test.shape[0]
print('Testing accuracy{}'.format(acc_test))
print("time taken : "+ str(time_taken))
```

```
Testing accuracy0.25028
time taken : 5388.183333158493
```

Output:
Test data accuracy: 0.25028
Time taken : 5388.183333158493

```
#15*15  image resize (2 layer)
data_test = resize_images(data_test,(15,15))
predict_test = model.predict_classes(data_test)
acc_test = np.where(label_test1==predict_test)[0].shape[0]/data_test.shape[0]
print('Testing accuracy{}'.format(acc_test))
print("time taken : "+ str(time_taken))
```

```
Testing accuracy0.233
time taken : 4508.592120409012
```

Output:
Test data accuracy: 0.233
Time taken : 4508.592120409012

```
#10*10  image resize(2 layer)
data_test = resize_images(data_test,(10,10))
predict_test = model.predict_classes(data_test)
acc_test = np.where(label_test1==predict_test)[0].shape[0]/data_test.shape[0]
print('Testing accuracy{}'.format(acc_test))
print("time taken : "+ str(time_taken))
```

```
Testing accuracy0.17048
time taken : 3595.2470846176147
```

Output:
Test data accuracy: 0.17048
Time taken : 3595.2470846176147

```
#5*5  image resize ( 2 layer)
data_test = resize_images(data_test,(5,5))
predict_test = model.predict_classes(data_test)
acc_test = np.where(label_test1==predict_test)[0].shape[0]/data_test.shape[0]
print('Testing accuracy{}'.format(acc_test))
print("time taken : "+ str(time_taken))
```

```
Testing accuracy0.50996
time taken : 3682.2164652347565
```

Output:
Test data accuracy: 0.50996
Time taken : 3682.2164652347565

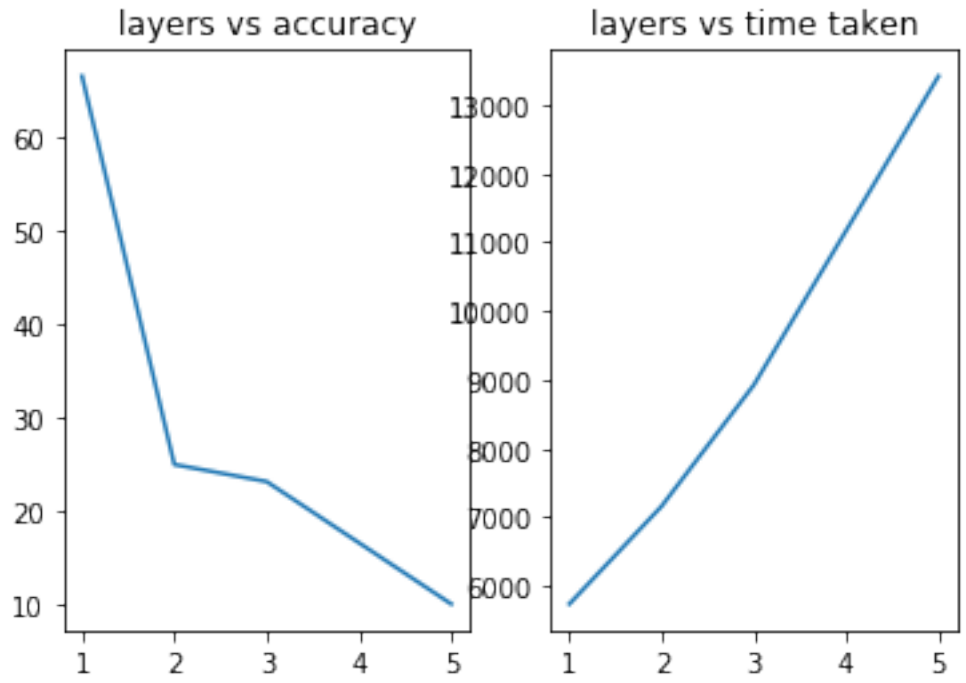
Conclusions:

a. Comparison with accuracy:
As we can see, as we decreased the resolution of the images, the accuracy began to decline. Before for an image with resolution 28x28 we had an accuracy of 0.24928. As the resolution decreased to 20x20, we got the accuracy of 0.25028.
Next, for 15x15 we got 0.233 then for 10x10 we got 0.17048. Then for resolution 5x5, the accuracy suddenly increased to 0.50996.
When we reduce the resolution of the image, we are losing some of the information from it. Hence, the accuracy goes on decreasing.

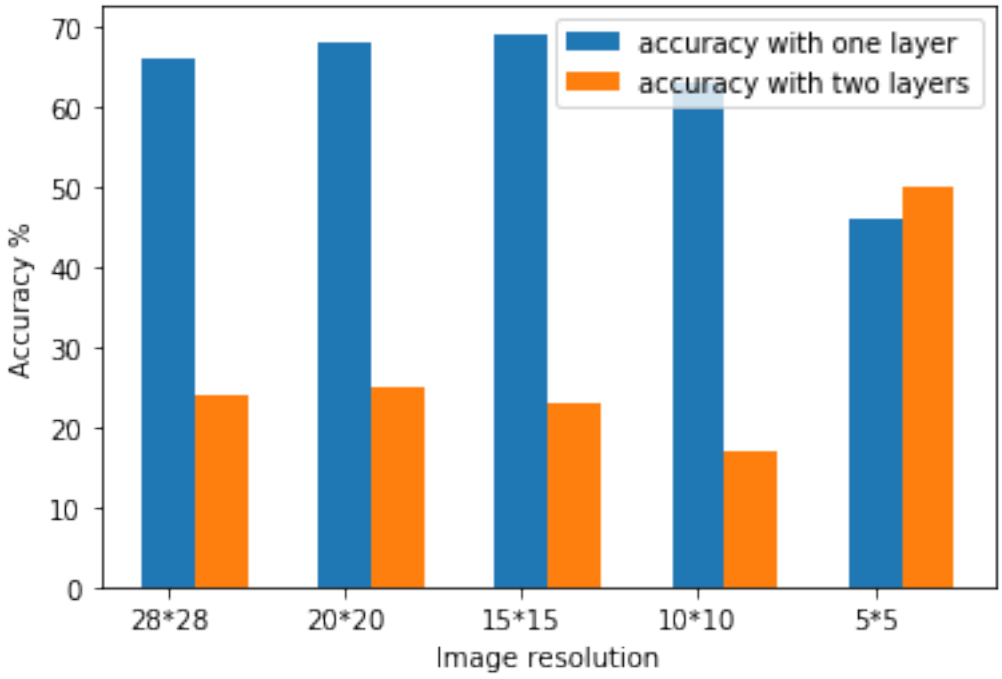
b. Comparison with execution time:
As for the execution time, as the resolution decreased, the execution time also decreased.

Final conclusions

The following graphs show the relationship between the layers in a perceptron model with accuracy and execution time.
These graphs plotted by rounding off the values to nearest integer.



The below bar graph is a plot of image resolution vs accuracy.
As you can see, one layered perceptron gives us better results with changes in image resolution.



The next graph gives us the plot for image resolutions vs execution time.
The one layered perceptron gives smaller execution time.