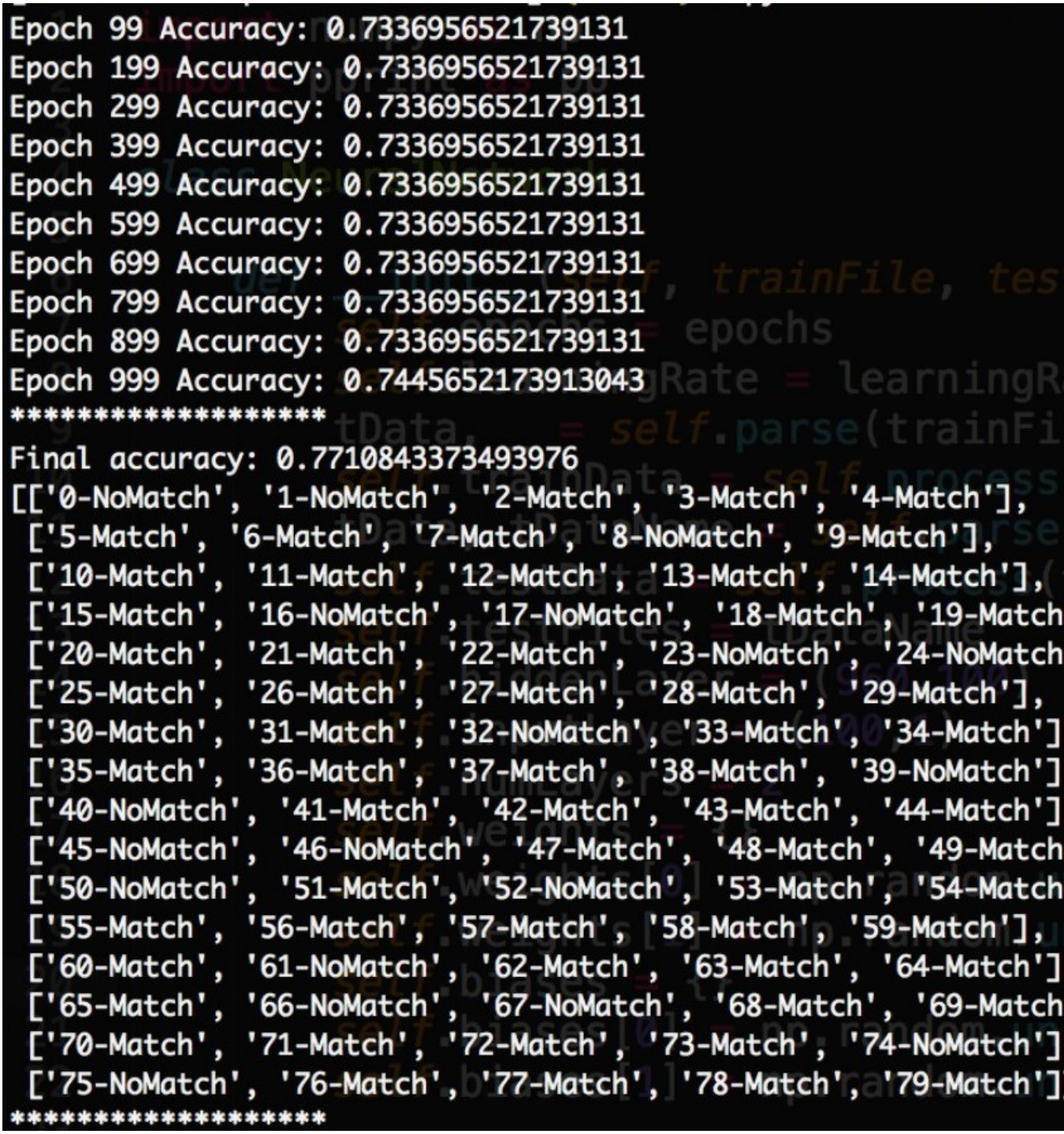
Assignment 5

Gaurav Baney, Wei-Han Chang gbaney@usc.edu, weihanc@usc.edu
04/09/2020

1. Implementation

Our implementation of the back propogation algorithm using neural networks is in the file 'NeuralNetwork.py'
The training and test files have been hardcoded to run upon executing the file.

We tested our implementation with the initial random weights of 0.01 and got these results:



We also tied a tighter fit of values between 0.75 and -0.75 and got much better results:

```
Epoch 99 Accuracy: 0.782608695652174
Epoch 199 Accuracy: 0.8260869565217391
Epoch 299 Accuracy: 0.8586956521739131
Epoch 399 Accuracy: 0.875
Epoch 499 Accuracy: 0.9130434782608695
Epoch 599 Accuracy: 0.9184782608695652
Epoch 699 Accuracy: 0.9239130434782609
Epoch 799 Accuracy: 0.9402173913043478
Epoch 899 Accuracy: 0.9510869565217391
Epoch 999 Accuracy: 0.9565217391304348
**********
Final accuracy: 0.927710843373494
[['0-Match', '1-Match', '2-Match', '3-Match', '4-Match'],
 ['5-Match', '6-Match', '7-Match', '8-Match', '9-NoMatch'],
 ['10-Match', '11-Match', '12-Match', '13-Match', '14-Match'],
 ['15-Match', '16-Match', '17-Match', '18-Match', '19-Match'],
 ['20-Match', '21-Match', '22-Match', '23-NoMatch', '24-NoMatch
 ['25-Match', '26-Match', '27-Match', '28-Match', '29-Match'],
 ['30-Match', '31-Match', '32-Match', '33-Match', '34-Match'],
 ['35-Match', '36-Match', '37-Match', '38-NoMatch', '39-NoMatch
 ['40-Match', '41-Match', '42-Match', '43-Match', '44-Match'],
 ['45-Match', '46-Match', '47-Match', '48-Match', '49-Match'],
 ['50-Match', '51-Match', '52-Match', '53-Match', '54-Match'],
 ['55-Match', '56-Match', '57-Match', '58-Match', '59-Match'],
 ['60-Match', '61-Match', '62-Match', '63-Match', '64-Match'],
 ['65-Match', '66-Match', '67-NoMatch', '68-Match', '69-Match']
 ['70-Match', '71-Match', '72-Match', '73-Match', '74-Match'],
 ['75-Match', '76-Match', '77-Match', '78-Match', '79-Match']]
**********
```

1.0.1 Implementation

In our implementation we use numpy arrays to improve the speed of the algorithm (SIMD operations) and also for certain math features that can be directly applied to the numpy arrays(Transpose, dot, summations, etc)

Using NumPy I can directly subtract out the means from the parsed data, calculate the covariance and also the eigenvalues and eigenvectors

1.0.2 Data Structures

We used numpy arrays to vectorize all the pmg input and they labels they carry
We use lists to keep track of per epoch data for each of the input
We use a dictionary to store the results of single forward and single backward movement
We save the weights and biases in a separate dictionaries.

1.0.3 Code Level Optimzations

We hardcoded our neural nets architecture according to the input size and given number of layers to make it easier to set up the initials weights and biases

1.1 Software Familiarization

1.1.1 sklearn.neural$_n etwork.MLPClassifier$

The SkLearn Results were as follows

```
('              precision    recall  f1-score   support\n'
 '\n'
 '           0       0.77      1.00      0.87        64\n'
 '           1       0.00      0.00      0.00        19\n'
 '\n'
 '    accuracy                           0.77        83\n'
 '   macro avg       0.39      0.50      0.44        83\n'
 'weighted avg       0.59      0.77      0.67        83\n')
C:\Users\gaurd\AppData\Local\Programs\Python\Python38\lib\si
 to 0.0 in labels with no predicted samples. Use `zero_divis
  _warn_prf(average, modifier, msg_start, len(result))
('              precision    recall  f1-score   support\n'
 '\n'
 '           0       0.73      1.00      0.85       135\n'
 '           1       0.00      0.00      0.00        49\n'
 '\n'
 '    accuracy                           0.73       184\n'
 '   macro avg       0.37      0.50      0.42       184\n'
 'weighted avg       0.54      0.73      0.62       184\n')
```

Our answers matched those of SkLearn at init weights of 0.01 a

Sklearns package ran much faster than our implementation and we think its because of they way they store their initial weights and data, and pass it on in a more efficient way.

Sklearns NeuralNet package also offered a lot more control giving more precision on the kind of fit you want to create with the data.

2. Applications

2.1 Recurrent Neural Networks

RNNs are a derived form of Neural Networks but the graph formed by the nodes is directed graph along a straight sequence of nodes. They also have access to the state of other internal nodes and because of this are really good at sequence generation. Their primary use cases are handwriting and speech recognition.