Lisa Meng
Shagun Gupta
INF 522
Homework 5

REPORT

Part 1: Implementation

The algorithm was implemented using the following libraries: pandas, numpy, matplotlib, matplotlib, math.
The images were read using the matplotlib.image function and converted to pandas dataframe where each row had the flattened, normalised value of the image pixel array.
The algorithm was implemented with the following parameters: 1000 epochs with the model going over the entire dataset over and over again and 0.1 learning rate. The logistic sigmoid function was used.
A function was created to implement the following parts of the algorithm:

1. Get_weights: to initialise random weights to an array of size n between range -0.1 and 0.1
2. Feedforward: implements step 1 of the algorithm to calculate x_{jL} for each perceptron in the hidden layer except the bias and the output layer with no bias
3. Get_delta: implements step 2 of the algorithm to calculate δ_{jl-1} for the hidden layer except the bias and the output layer
4. Step3_weights: implements step 3 of the algorithm to adjust the weights based on the values obtained above
5. Testing: to test the testing data and return the final predictions and accuracy

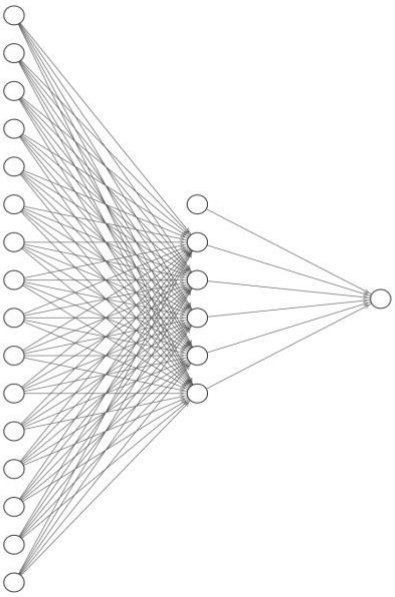
The accuracy obtained is : 77.10843373493976%
The final predictions are attached in the file **prediction.csv** with the following format:

Prediction	Label	Final	Compare
The prediction	The actual label	0 if prediction < 0.5 else 1	If final == label

The finals weight are attached in the file **weights.csv** with the following format:

Row 1: weights between input and hidden layer	[Array of 100]: weights from perceptron 1 in input layer to all perceptrons in hidden layer	[Array of 100]: weights from perceptron 2 in input layer to all perceptrons in hidden layer
Row 2: weights between hidden layer	[Array of 1]: weight from perceptron 1 in hidden layer to perceptron in output layer	[Array of 1]: weight from perceptron 2 in hidden layer to perceptron in output layer

The architecture implemented by the neural network is explained below:



Input Layer Hidden Layer Output Layer
960 perceptrons 100 perceptrons 1 perceptron
1 bias perceptron 1 bias perceptron no bias perceptron

The challenges faced were mainly in handling the overflowing exponential values generated in the implementation of the sigmoid function. This was resolved by switching to the numpy library instead of using the math library. Another solution is to put a lower limit on the value of theta and automatically return 0 as the value would be as low as $1/10^7$.

Part 2: Software Familiarization

For this part of the assignment, we researched many software/libraries that have good implementation of the Back Propagation algorithm for Feed Forward Neural Networks. Due to the recent popularity with AI and Neural Networks, there were numerous software, such as Keras and Google's Tensorflow. But ultimately, we decided to use the tried-and-true SKLearn because of its reliability and ease of use.

The image below shows how we imported the library and the parameters we used. Note that the activation parameter, "logistic", is for the logistic sigmoid function, which returns f(x) = 1 / (1 + exp(-x)), and the optimizer/"solver" we used was "sdg" (Stochastic Gradient Descent), as the professor taught in lecture. We used the same values for the size of our hidden layer, epochs, and learning rate as was requested in Part 1, which are 100, 1000 and 0.1, respectively. We then

trained our model using the fit() function on the training data set.

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(100), activation='logistic',solver='sgd',
                    learning_rate='constant',learning_rate_init=0.1,max_iter=1000)
mlp.fit(train_X, train_y.values.ravel())
```

```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
              beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=100, learning_rate='constant',
              learning_rate_init=0.1, max_iter=1000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='sgd', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

We then used SKLearn's predict() function on the test set. The array belows shows the model's predictions, 1 indicating "thumbs down" and all else as 0.

```
predictions = mlp.predict(test_X)
predictions
```

```
array([1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0])
```

To calculate the model's accuracy, we used the SKLearn's classification_report and confusion_matrix function. With the confusion matrix, you can see that the model misclassified 9 images, but classified the rest correctly, resulting in an 89% accuracy, shown in detail in the classification table.

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(test_y,predictions))
print(classification_report(test_y,predictions))
```

```
[[60  4]
 [ 5 14]]
              precision    recall  f1-score   support

           0       0.92      0.94      0.93        64
           1       0.78      0.74      0.76        19

    accuracy                           0.89        83
   macro avg       0.85      0.84      0.84        83
weighted avg       0.89      0.89      0.89        83
```

Comparing the results from our code from scratch versus the results of implementing a library, our model from scratch had an accuracy of 77% (misclassifying 19 images), whereas the model from the library had an accuracy of 89% (misclassifying 9 images). Although we feel our code did sufficiently well, ways we can improve our code to get a higher accuracy are to add

more hidden layers, decrease the learning rate, and increase the number of epochs in order to increase our model's learning.

Part 3: Applications

Neural Networks consists of many different techniques but 3 types commonly used are:

1. Classification
    a. Similar to this assignment, Neural Networks can be used for classification and categorization for text, image, and numerical/integer data.
        i. For images, NN can be used for image and facial recognition. Such examples are classifying whether an image is of a cat or not, categorizing dog breeds, and identifying a person based on an image of their face.
        ii. For text, NN can be used for Natural Language Processing. Example applications are sentiment analysis, information filtering, web searching, and language identification.
        iii. In terms of numerical/integer data, NN can be used to classify patients with likelihood of having diabetes but using numerical data from their age, weight, fitness level, blood test, genome, and other health issues.
2. Prediction
    a. A Neural Network model can be trained to return outputs that are forecasting from a given input/event.
        i. An example is stock market prediction.
3. Clustering
    a. As an unsupervised classification technique, NN can be used to identify inherent structures or unique features and classify/categorize them without any prior knowledge.
        i. This method is useful for pattern recognition, image segmentation, and data mining.

Contributions:

- ➢ For Part 1, Lisa and Shagun collectively discussed the concepts of Neural Networks and the steps to implementing the Back Propagation algorithm for Feed Forward Neural Networks. Through collaborative work, both Lisa and Shagun coded Part 1 by taking turns coding each function and explaining the process as they went along.
- ➢ For Part 2, Lisa researched and implemented the SKLearn library for the Back Propagation algorithm for Feed Forward Neural Networks.
- ➢ Part 3 was researched and completed by Lisa.
- ➢ The report was compiled by Lisa and Shagun.