Handwriting Recognition Using Machine Learning

Parth Shah
SEAS, University at Buffalo
Buffalo, NY, 41214
Person Number: 50291125
parthnay@buffalo.edu

Abstract

The project aims to create a Machine Learning Model that can accurately recognize two same Handwritings based on a number of features of the two images.

1 Introduction

The given problem is a binary regression problem.We use Machine Learning to recognize two same Handwritings based on a number of features as provided by the two datasets that are Human Observed dataset and GSC(Gradient Structural Concavity) dataset. As we can see that this is a regression problem. We use three methods to perform regression on the datasets provided.

1. Linear Regression using Gradient Descent
2. Logistic Regression
3. Neural Networks

2 DataSet and Data Splitting

We are given two datasets. Each instance in the CEDAR \AND" training data consists of set of input features for each hand-written “AND" sample. The features are obtained from two different sources:
1. Human Observed features: Features entered by human document examiners manually
2. GSC features: Features extracted using Gradient Structural Concavity (GSC) algorithm.
The target values are scalars that can take two values(binary) They are 1:same writer or 0:different writers.
Based on feature extraction process, we have provided two dataset settings:
Setting 1: Feature Concatenation: Here, the set of features for two pairs are concatenated using the pandas function pd.concat on two image rows/attributes. Thus we get

a. 18 features for Human Observed Dataset
b. 1024 features for GSC Dataset

Setting 2: Feature subtraction: Here we calculate the absolute difference between the two same features for both the images. Thus we get

a. 9 features for Human Observed
b. 512 features for GSC

The Models are made to run on each of the databases settings created.

The 4 DataSets thus created were partitioned as follows. 80% for the data was used for training. 10% of the data was used for validating our model and the remaining 10% was used to test our model for the performance.

3 Multivariate Linear Regression [Using Gradient Descent]

The given problem was solved using Multivariate Linear Regression. Linear Regression is solved by the equation of the form y(x,w) = w^T(x). where w is the weight vector to be learnt from regression and x is the input vector. Multiple linear regression attempts to model the relationship between two or more feature variables and a target variable by fitting a linear equation to observed data. Every value of the independent variable x is associated with a value of the dependent variable y. The regression equation for n variables $x_1, x_2, \ldots, x_n$ is defined to be

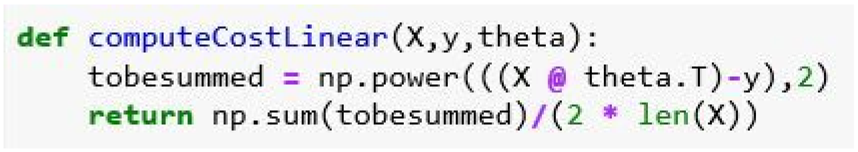
$$y = b_0 + b_1x_1 + b_2x_2 + \ldots + b_nx_n$$

where, $b_{0,}$ $b_{1,}$ $b_{2, \ldots,}$ b_n are the coefficients of regression. They can also be called as the weights that are generated after performing linear regression on the training dataset and adjusted after performing linear regression on the validation dataset to improve accuracy.

The regression is divided into two functions:

1. ComputeCost
2. Gradient Descent

ComputeCost - The ComputeCost function computes the value of the features with the current weights. The ComputeCost function takes X,y and theta as parameters and computes the cost for the iteration. This cost is the cost of the current state of the dataset.The compute cost function looks like



```python
def computeCostLinear(X,y,theta):
    tobesummed = np.power(((X @ theta.T)-y),2)
    return np.sum(tobesummed)/(2 * len(X))
```

Gradient Descent - Gradient Descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point. If instead one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.The stochastic gradient descent algorithm first takes a random initial value w(0). Then it updates the value of w_0 using

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)}$$

Where τ denotes the number of iterations and Δw denotes the weight update. Where,

$$\Delta\mathbf{w}^{(\tau)} = -\eta^{(\tau)}\nabla E$$

is the learning rate and the update is given by

$$\nabla E_D = -(t_n - \mathbf{w}^{(\tau)\top}\boldsymbol{\phi}(\mathbf{x}_n))\boldsymbol{\phi}(\mathbf{x}_n)$$

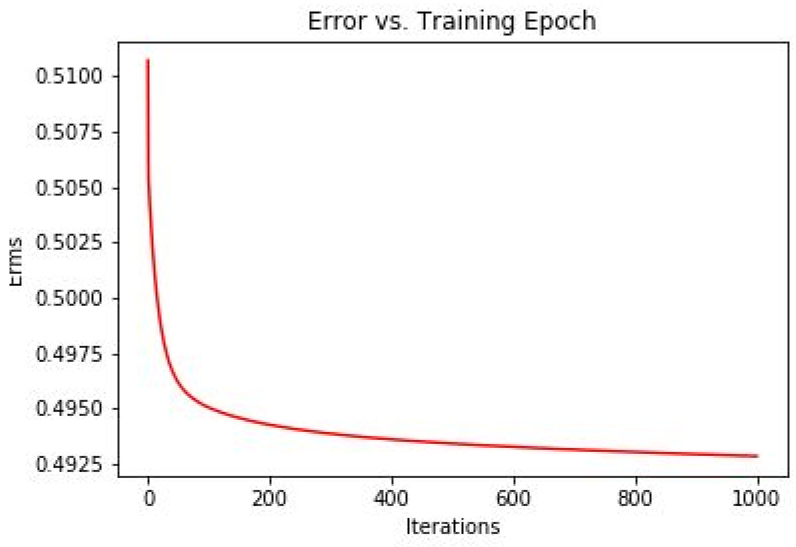
$$\nabla E_W = \mathbf{w}^{(\tau)}$$

They python code for the same can be written as

```
def gradientDescentLinear(X,y,theta,iters,alpha):
    Erms = [0 for x in range(iters)]
    accuracy = [0 for x in range(iters)]
    cost = np.zeros(iters)
    for i in range(iters):
        theta = theta - (alpha/len(X)) * np.sum(X * (X @ theta.T - y), axis=0)
        cost[i] = computeCostLinear(X, y, theta)
        y_new = GetValTest(X,theta)
        Erms[i],accuracy[i] = GetErms(y_new,y)
        if i%10 == 0:
            print("cost = ",cost[i])
    print("final weights are:",theta)
    print("Erms = ",Erms)
    return theta,cost,Erms,accuracy
```

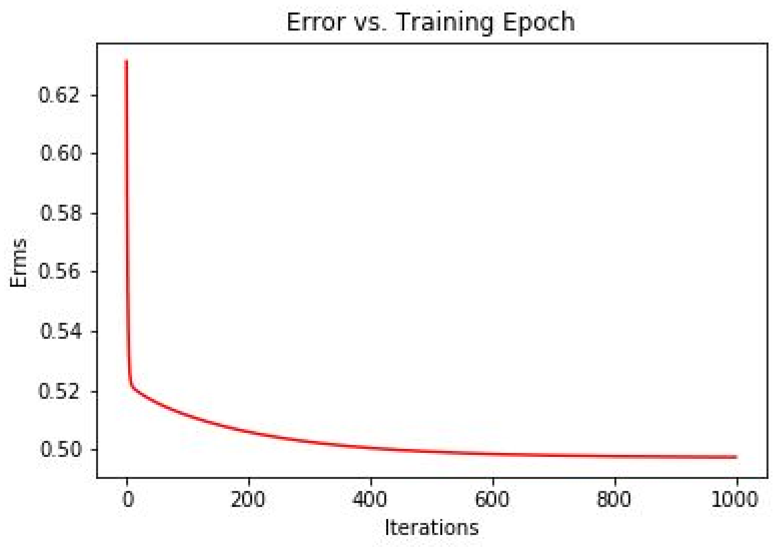
The following observations were made after performing linear regression on the Human observed dataset:

Cost Before Training = 0.24407582938388625
Cost After Training = 0.12145344971746858
Cost After Training = 0.12174575823383825
The Erms can be seen as:



For Human Observed Subtracted dataset

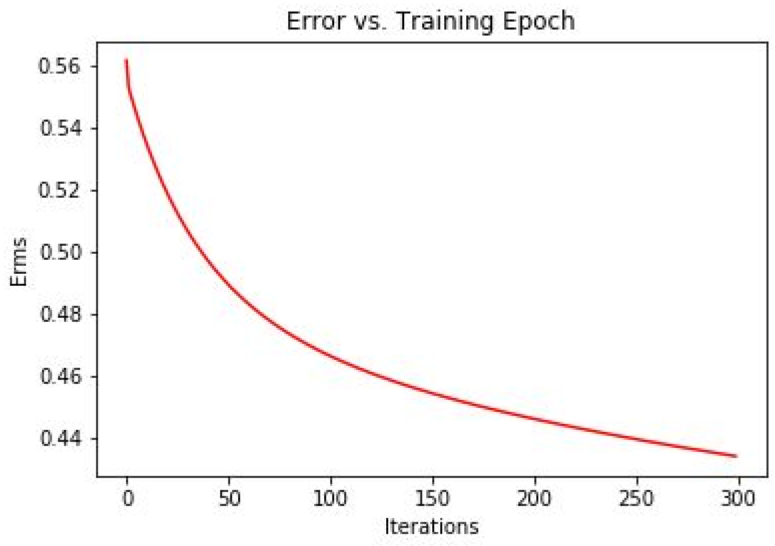
Cost Before Training = 0.2539494470774092
Cost After Training = 0.1236411917943889
Cost After Training = 0.12174575823383825
The Erms can be seen as:



The following observations were made after performing Logistic regression on the GSC dataset:(Since the Dataset is too huge, It was trained on ten sets of 8000 shuffled rows each and then the last set was partitioned into 80-10-10 as described in data partition for validation and testing)

For GSC Concatenated dataset

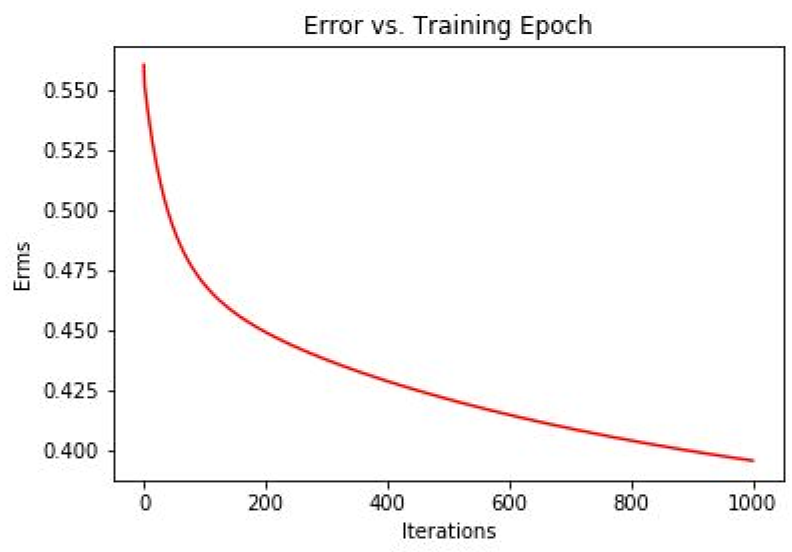
Cost Before Training = 0.2124940625
Cost After Training = 0.07730604325029898
Cost After Training = 0.07931231215029898
The Erms can be seen as:



For GSC Subtracted dataset

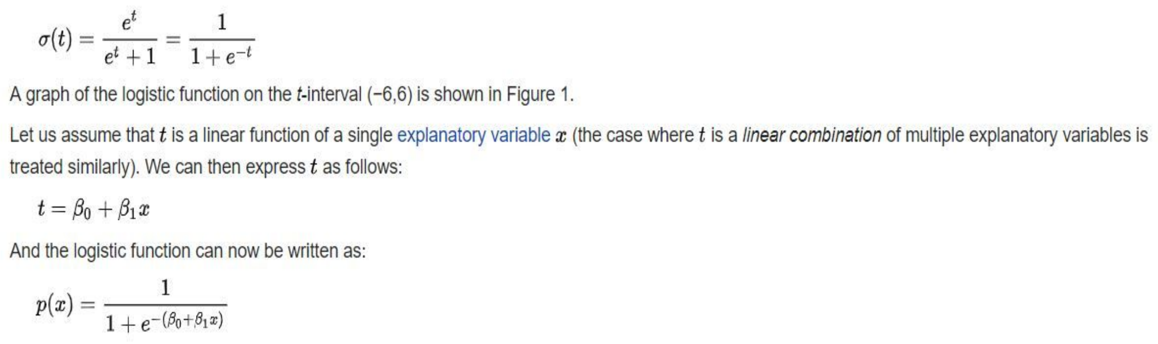
Cost Before Training = 0.24940625
Cost After Training = 0.07830604325029898
Cost After Training = 0.081313132502989

The Erms can be seen as:



4 Logistic Regression

Logistic Regression is similar to the linear regression as above, Here the cost is computed and modified by an activation function. We choose the sigmoid activation in out implementation. A sigmoid function, takes any real input t (t ∈ {R}) and outputs a value between zero and one.For the logit, this is interpreted as taking input log-odds and having output probability. The logistic function sigmoid(t) is defined as follows:



$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

A graph of the logistic function on the t-interval (−6,6) is shown in Figure 1.

Let us assume that t is a linear function of a single explanatory variable x (the case where t is a *linear combination* of multiple explanatory variables is treated similarly). We can then express t as follows:

$$t = \beta_0 + \beta_1 x$$

And the logistic function can now be written as:

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

We calculate the Log-Likelihood and maximize it with the help of updated weights using gradient descent

.

The following observations were made after performing Logistic regression on the Human observed dataset:

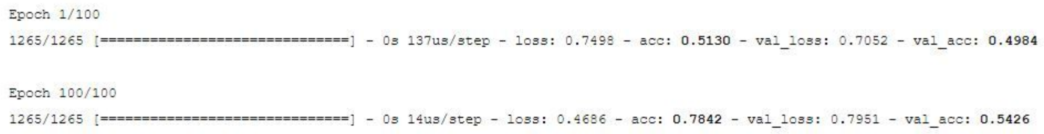
For Human Observed Concatenated dataset

5 Neural Network

Artificial neural networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. The neural network itself isn't an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs.[1]
A two-layer neural network was created. The hidden layer had 512 nodes and the output layer has 2 nodes for the two values of the target (1=writer match, 0=writer mismatch). The input layer contains 18 nodes for the concatenated datasets and 9 nodes for the subtracted datasets.
The hyperparameters available to be tweaked were batch size, optimizer, loss, number of layers, number of nodes.
By rough estimation and a few runs, The hyperparameters that were considered are batch size = 32, optimizer = rmsprop, loss= categorical_crossentropy.

The following observations were made after training the Neural Network model on the Human observed dataset:

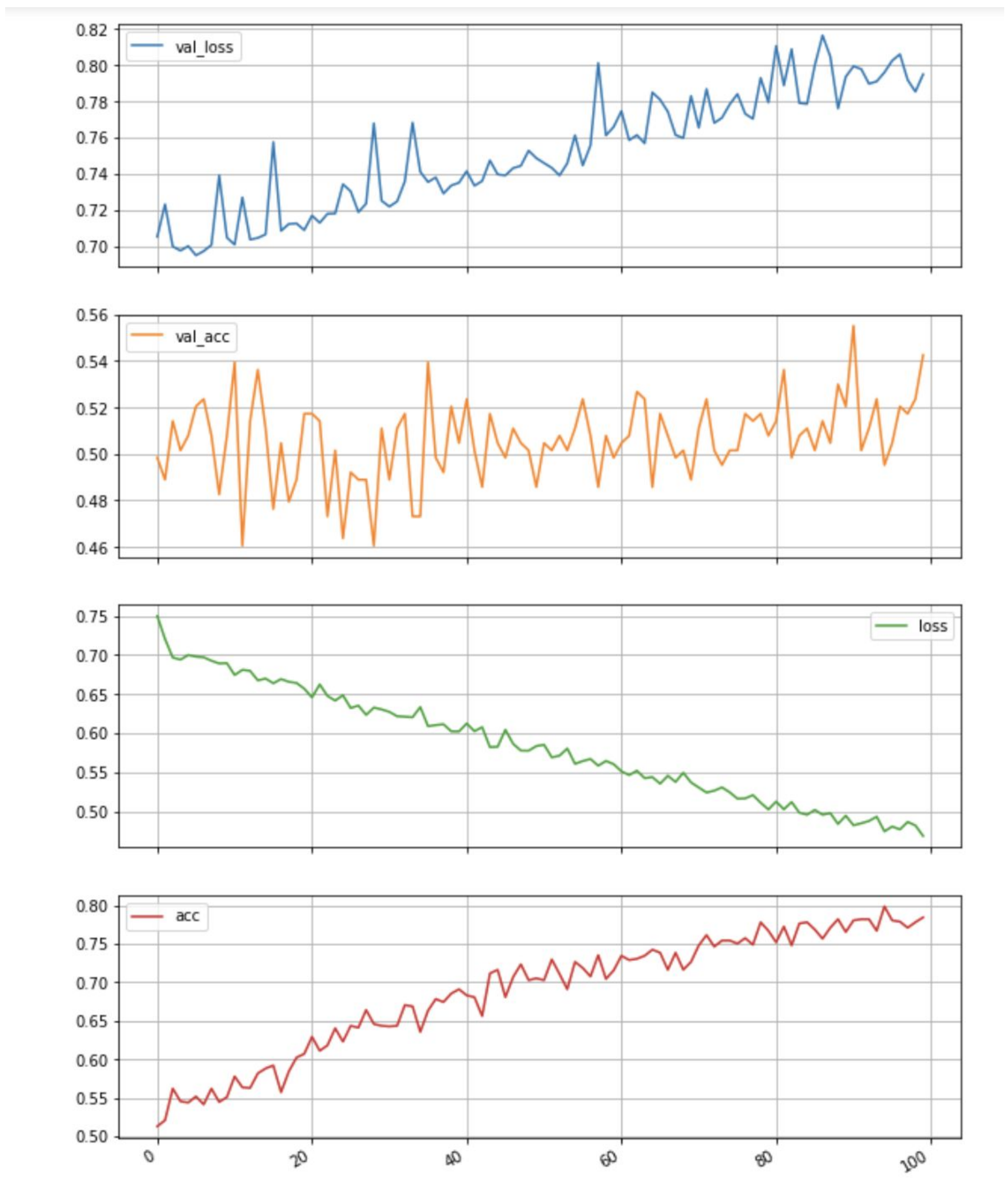
For Human Observed Concatenated dataset



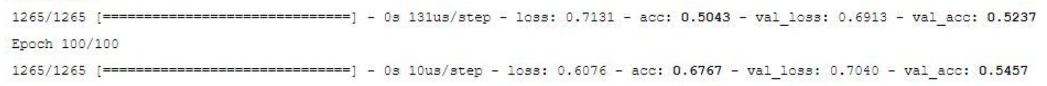
```
Epoch 1/100
1265/1265 [==============================] - 0s 137us/step - loss: 0.7498 - acc: 0.5130 - val_loss: 0.7052 - val_acc: 0.4984

Epoch 100/100
1265/1265 [==============================] - 0s 14us/step - loss: 0.4686 - acc: 0.7842 - val_loss: 0.7951 - val_acc: 0.5426
```

The graphs can be seen as :

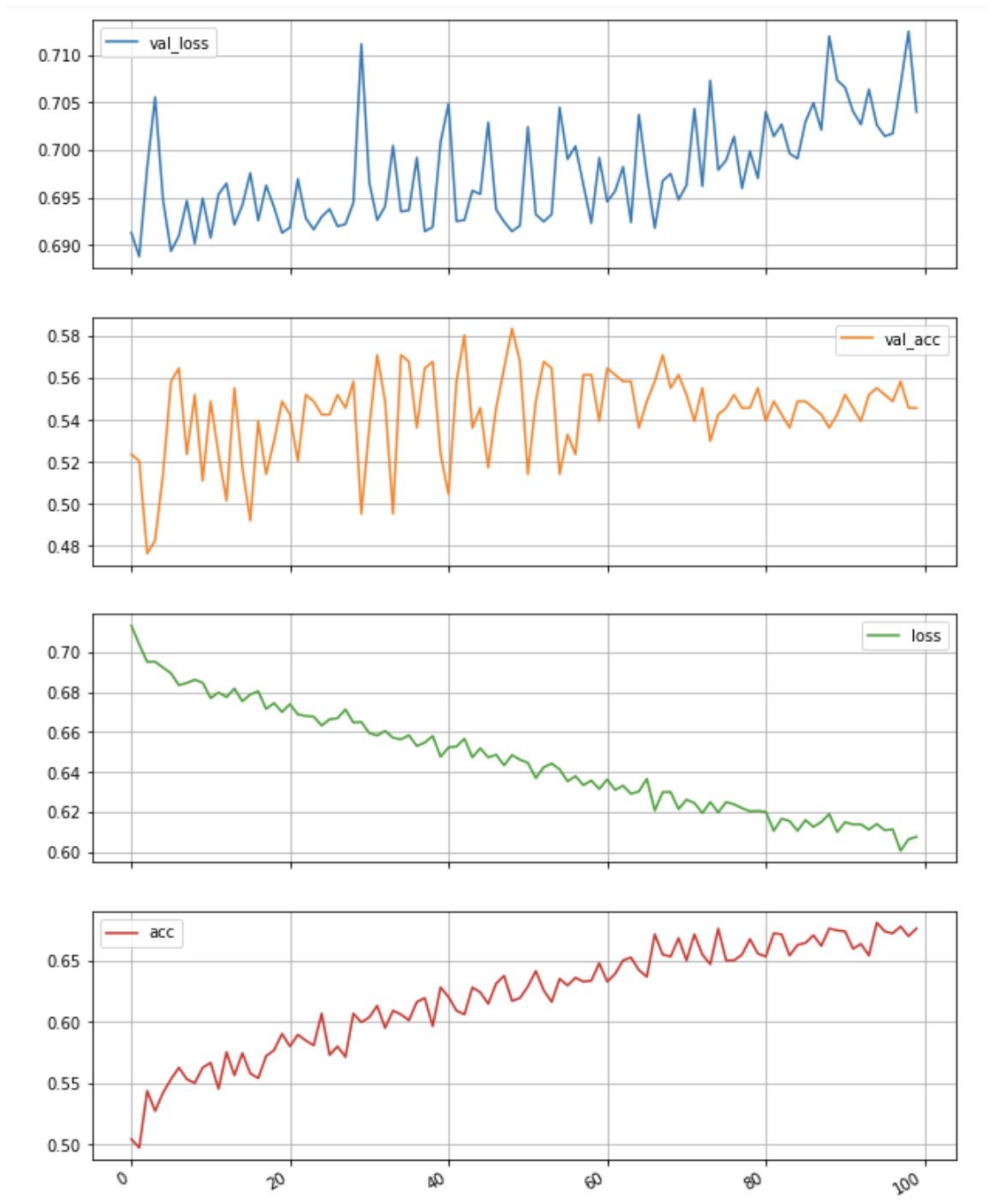


For Human Observed Subtracted dataset



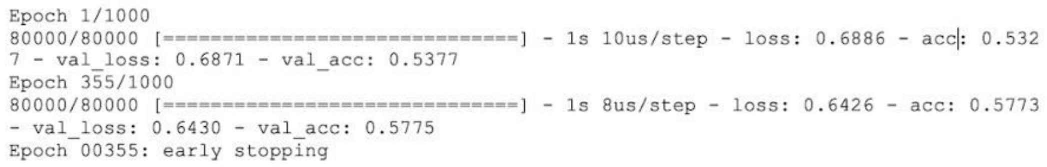
```
Epoch 1/100
1265/1265 [==============================] - 0s 131us/step - loss: 0.7131 - acc: 0.5043 - val_loss: 0.6913 - val_acc: 0.5237
Epoch 100/100
1265/1265 [==============================] - 0s 10us/step - loss: 0.6076 - acc: 0.6767 - val_loss: 0.7040 - val_acc: 0.5457
```

The graphs can be seen as :



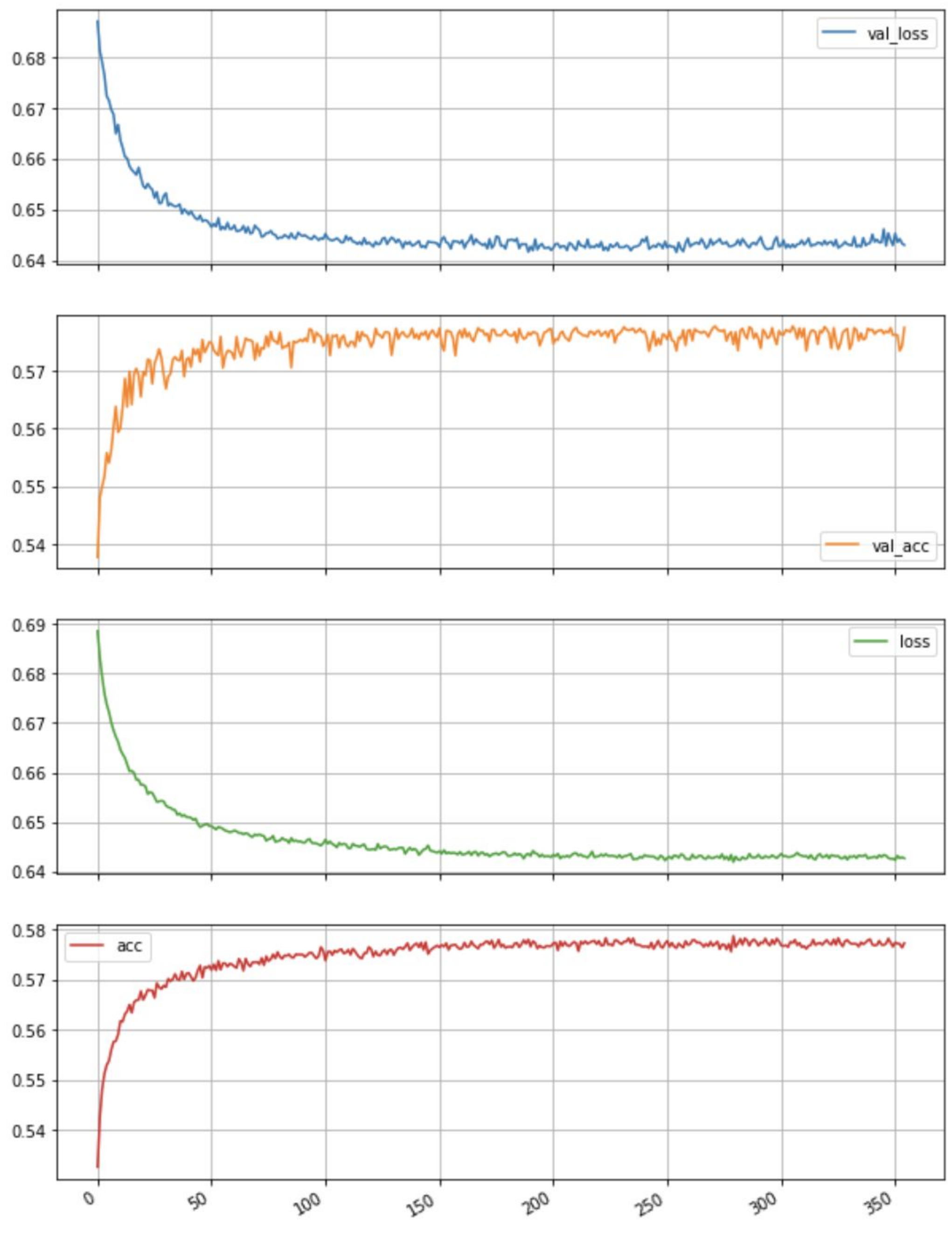
The following observations were made after training the Neural Network model on the GSC dataset:(Since the Dataset is too huge, It was trained on ten sets of 8000 shuffled rows each and then the last set was partitioned into 80-10-10 as described in data partition for validation and testing)

For GSC Concatenated dataset



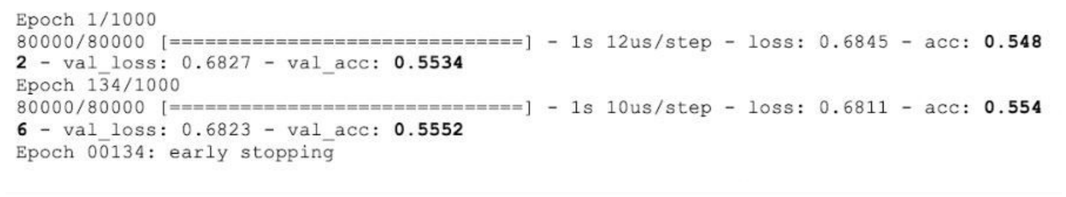
```
Epoch 1/1000
80000/80000 [==============================] - 1s 10us/step - loss: 0.6886 - acc: 0.532
7 - val_loss: 0.6871 - val_acc: 0.5377
Epoch 355/1000
80000/80000 [==============================] - 1s 8us/step - loss: 0.6426 - acc: 0.5773
- val_loss: 0.6430 - val_acc: 0.5775
Epoch 00355: early stopping
```

The graphs can be seen as :

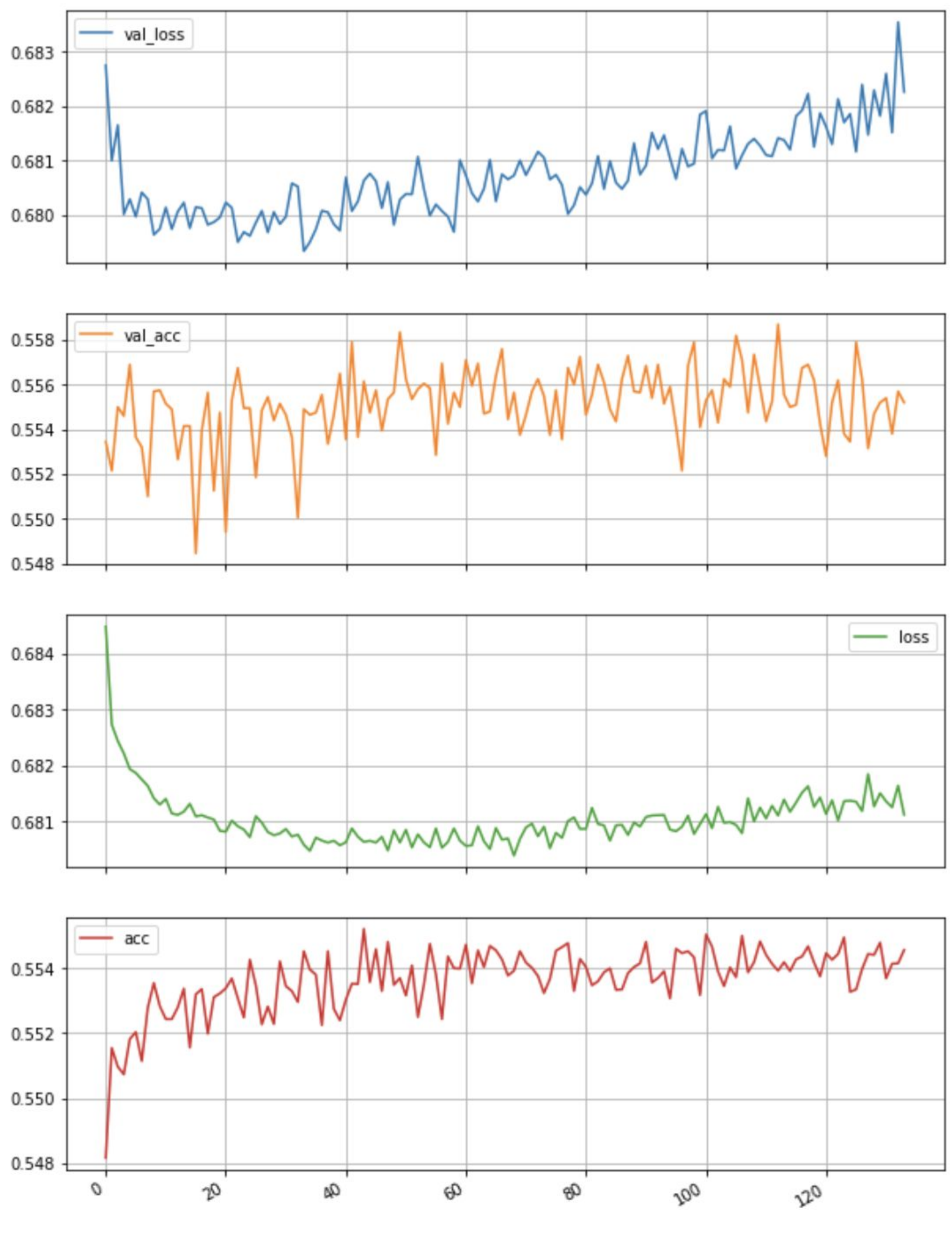


For GSC Subtracted dataset

The graphs can be seen as :



```
Epoch 1/1000
80000/80000 [==============================] - 1s 12us/step - loss: 0.6845 - acc: 0.548
2 - val_loss: 0.6827 - val_acc: 0.5534
Epoch 134/1000
80000/80000 [==============================] - 1s 10us/step - loss: 0.6811 - acc: 0.554
6 - val_loss: 0.6823 - val_acc: 0.5552
Epoch 00134: early stopping
```



6 Conclusion

Regression was performed on the four given datasets. Linear regression is not a good machine learning approach for this dataset as the targets are discrete binary classes. Logistic regression performs the best under such conditions. The GSC dataset contains more features and thus, gives more accurate machine learning models. The deep learning approach is an altogether better approach for any machine learning applications.

Acknowledgments

This report was prepared in accordance to the Project2.pdf provided, Multiple Linear Regression, Logistic Regression and Artificial Neural Network documentation available on Wikipedia and Learn data science,com

References

[1] Pattern Recognition and Machine Learning: Christopher Bishop

[2] Gradient descent: https://en.wikipedia.org/wiki/Gradient_descent

[3]http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex3/ex3.html

[4]Logistic Regression: https://en.wikipedia.org/wiki/Logistic_regression

[5]Artificial Neural Network: https://en.wikipedia.org/wiki/Artificial_neural_network