Deep Learning

Assignment 1

Task 2 Report

Abstract

To develop a connected neural network to process the spam data from the dataset spambase, split the train and test sets and split the train sets again into validation and test sets, train the developed model and finally to measure the accuracy by binary class classification.

Problem Statement

Spambase is a dataset collection with 4600 instances of mail and 57 attributes determining if the mail is spam or not. The data is loaded into the model and is split into train and test sets. Then the train set is again split into train and validation set. Finally the accuracy of model across the validation and the test sets is obtained.

Proposed Solution

The purpose of the classification is to determine whether the given mail is spam or not. Since there are only two classes, spam or not spam binary classification is selected. In case of model design, the activation functions are relu for all the layers and sigmoid function for the last layer as this is binary classification. Finally the optimizer used is rmsprop.

Loading the data:

The data was found in the .DATA format with all the instances and attributes. The final column determined if the given mail was spam (1) or not (0). Loading the data included the following steps:

1. The data was split into test and train randomly.
2. The labels of the split data were popped out.
3. Finally the input to the model was the data without the label column.

Binary classification:

The data fed into the model finally fits into either of the class – spam or not spam. Thus binary classification is implemented which determines the result using labels that are binary.

Model design:

Activation = relu

RELU function is used for activation. The purpose of this is to obtain the maximum value when the given input is greater than zero and to substitute zero when the given input is negative. RELU activation gives better performance than other activation functions such as sigmoid and tanh in the hidden layers.

Activation = sigmoid

Sigmoid function is extensively used for binary classification. It creates probabilities for the output obtained from the hidden layers. With this output created the mail is determined to be either a spam or not a spam. Since there is only one output possible the last layer has only one class.

Optimizer = rmsprop

Rmsprop is used in this model. It converts data to mini batches and iterates through them to learn from it. It is similar to the gradient descent algorithm. By this way it is easy to tune the model more accurately. This optimizer is more suitable for huge datasets.

Implementation Details

Implemented using anaconda and Jupyter.

Implementation issues and processes includes:

1. Since the datasets were huge, it was necessary to increase the number of hidden layers to give better results. It was also necessary to increase the number of iterations through the training set to make the model better.
2. The task was implemented in Jupyter notebook. It was found that Jupyter stored the values of the models and at each hyper tuning the kernel was restarted.
3. The datasets were found to be huge and thus processing the data with so many epochs and hidden layers were difficult.
4. The process of running such an entire data on the local CPU was difficult.

Evaluation metrics

Initially the validation data obtained from the train data was ran across the model and finally the test data was ran across the model which was tuned with the test label.

Out of 3221 train data 2221 were considered as train set and 1000 were taken as validation data.

The accuracy obtained were as follows:

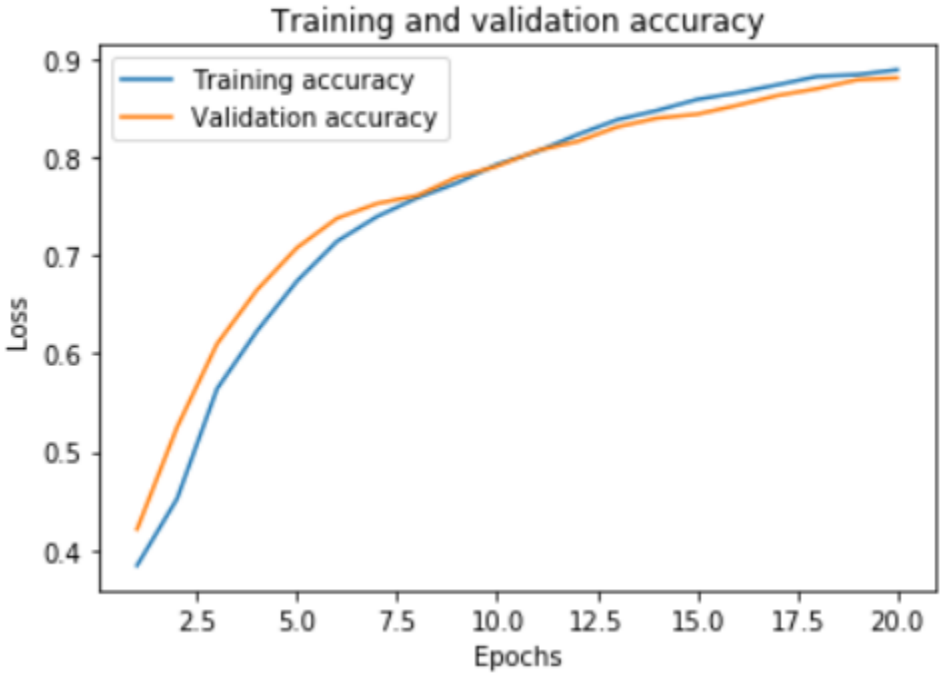
Validation set accuracy at each epoch:

```
Train on 2221 samples, validate on 1000 samples
Epoch 1/20
2221/2221 [==============================] - 1s 280us/step - loss: 0.7642
- accuracy: 0.3841 - val_loss: 0.7289 - val_accuracy: 0.4210
Epoch 2/20
2221/2221 [==============================] - 0s 17us/step - loss: 0.7139 -
accuracy: 0.4520 - val_loss: 0.6965 - val_accuracy: 0.5250
Epoch 3/20
2221/2221 [==============================] - 0s 28us/step - loss: 0.6841 -
accuracy: 0.5642 - val_loss: 0.6719 - val_accuracy: 0.6100
Epoch 4/20
2221/2221 [==============================] - 0s 18us/step - loss: 0.6604 -
accuracy: 0.6231 - val_loss: 0.6512 - val_accuracy: 0.6650
```

```
Epoch 5/20
2221/2221 [==============================] - 0s 16us/step - loss: 0.6394 -
accuracy: 0.6740 - val_loss: 0.6314 - val_accuracy: 0.7080
Epoch 6/20
2221/2221 [==============================] - 0s 15us/step - loss: 0.6192 -
accuracy: 0.7145 - val_loss: 0.6124 - val_accuracy: 0.7380
Epoch 7/20
2221/2221 [==============================] - 0s 13us/step - loss: 0.5993 -
accuracy: 0.7398 - val_loss: 0.5929 - val_accuracy: 0.7530
Epoch 8/20
2221/2221 [==============================] - 0s 16us/step - loss: 0.5787 -
accuracy: 0.7587 - val_loss: 0.5734 - val_accuracy: 0.7610
12500/12500 [==============================] - 1s 113us/step - loss: 0.7838 -
accuracy: 0.6732 - val_loss: 0.8042 - val_accuracy: 0.6456
Epoch 5/50
12500/12500 [==============================] - 1s 112us/step - loss: 0.7552 -
accuracy: 0.6874 - val_loss: 0.9881 - val_accuracy: 0.5276
Epoch 6/50
12500/12500 [==============================] - 1s 114us/step - loss: 0.7498 -
accuracy: 0.6899 - val_loss: 0.7370 - val_accuracy: 0.6968
```

.....

The training loss and the validation loss across the epochs were plotted as follows:



Observations from the graph:

The loss of both the classes training and validation increases from the initial iteration.

Since the two lines go at the same pace, the process of learning takes at a better pace.

However at certain places the training loss outcurves the validation loss meaning that the process pf overfitting takes place. This could be due to repeated learning of same data.

Test set accuracy at the epoch:

1380 test samples of the three classes were taken and the result obtained from the model were as follows:

```
1380/1380 [==============================] - 0s 32us/step
[0.31000121581381646, 0.904347836971283]
```

Finally the accuracy of the test sample is obtained to be 90%. This accuracy is obtained after man y hyper tunings by

1. Changing the number of layers of the model
2. Changing the epochs
3. Changing the batch size
4. Choosing the correct optimizer.

References:

1. Keras code given by the professor.