CS577 – Deep Learning
Homework 1 – Task 2

Niti Wattanasirichaigoon
A20406934

Problem Statement

Load the spam email data from the UCI repository "spambase". Prepare the data you loaded as a tensor suitable for neural network. Normalize features as needed. Explain the steps you perform in preparing the data and justify them. Write a function "*load_spam_data*" to load the data and split it into training and testing subsets. Repeat the steps you performed in task 1.

Proposed Solution

Using the same approach as the first task, we should first study our dataset, address the type of problem, design the appropriate model, and tune hyper parameters.

1. **Data Preparation:**
   The spambase data [1] consists of 57 attributes of emails and a binary label (1 for spam, 0 for not spam). The dataset contains a total of 4601 tuples. First we must normalize the features because some of they have been measured in different units (not normalizing the features will cause the neural network to weigh higher values during fitting). We then split roughly 20% of the data for testing and 80% for training purposes. The validation split will be done during the *model.fit* method of model compilation. The data is already in a vectorized tensor, ready for input.

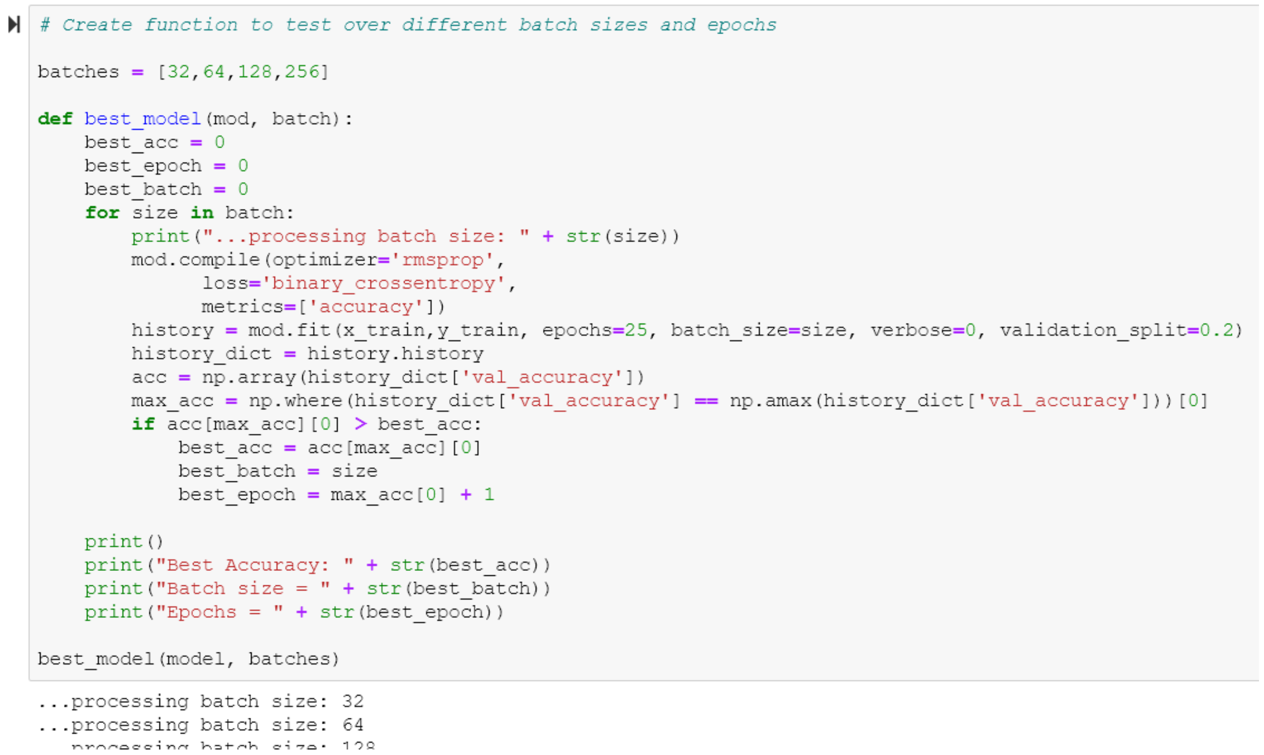
2. **Model Initialization:**
   We start with a model with 32 nodes for the input and hidden layers and one single node in the output layer. Here I chose 32 nodes because this dataset has much less features than our data in task 1 and one node for the final output layer because we will use a sigmoid activation function since this is a binary classification problem. The initial model will use RMSprop optimizer, binary cross-entropy for the loss function, and measured by accuracy.

3. **Training and Validation:**
   The models will be trained with 20 epochs each. Hyper parameters will then be turned over a range of number of hidden layers, nodes per hidden layer, batch sizes, epochs and loss function. We will be comparing the models by measuring loss and accuracy.

4. **Final Model Evaluation:**
   We finally select the best combination of hyper parameters to train the final model and evaluate it by using the test data.

Implementation Details

We revert back to the same problem as in task 1, where it is resource demanding in finding the best hyper parameters for the model. Here I used a different approach by writing a function that will find us the number of epochs and batch size that will yield us the best accuracy for each model.



```python
# Create function to test over different batch sizes and epochs

batches = [32,64,128,256]

def best_model(mod, batch):
    best_acc = 0
    best_epoch = 0
    best_batch = 0
    for size in batch:
        print("...processing batch size: " + str(size))
        mod.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
        history = mod.fit(x_train,y_train, epochs=25, batch_size=size, verbose=0, validation_split=0.2)
        history_dict = history.history
        acc = np.array(history_dict['val_accuracy'])
        max_acc = np.where(history_dict['val_accuracy'] == np.amax(history_dict['val_accuracy']))[0]
        if acc[max_acc][0] > best_acc:
            best_acc = acc[max_acc][0]
            best_batch = size
            best_epoch = max_acc[0] + 1

    print()
    print("Best Accuracy: " + str(best_acc))
    print("Batch size = " + str(best_batch))
    print("Epochs = " + str(best_epoch))

best_model(model, batches)
```

```
...processing batch size: 32
...processing batch size: 64
...processing batch size: 128
```

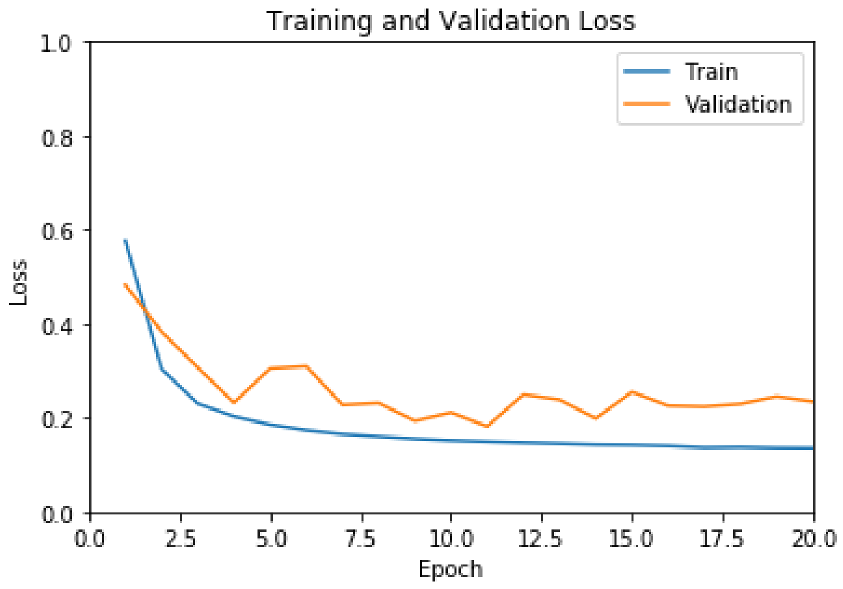
I ran the function over different models and chose the one with the best accuracy as the best model. After observing our training sample, approximately half (1813/3681) are labelled as spam. Since this is a binary classification, the sample is considered balanced enough to use accuracy as our main metric of evaluation. The loss in this case is calculated by binary cross-entropy which gets lower easily because there are only two classes but can swing by a large value if a single bad prediction is made [2].

Results and Discussion

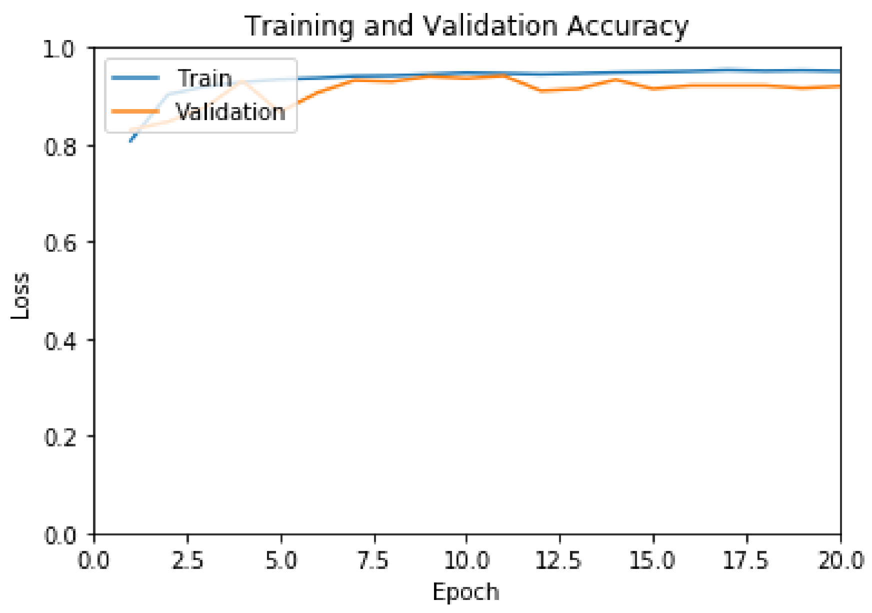
The *best_model* function was ran over a variety of models according to the table below. The output layer will always contain the single sigmoid node.

Model	Best Validation Accuracy	Epochs	Batch Size
1. Input(32), hidden(32)	0.9416553378105164	15	64
2. Input(16), hidden(16)	0.9430122375488281	16	32
3. Input(32), hidden(32), hidden(16)	0.9538670182228088	7	32
4. Input(32), hidden(16), hidden(32)	0.9470827579498291	10	64
5. Input(32), hidden(16), hidden(8), hidden(4)	0.9375848174095154	8	64

From the experiment, we can see that the best accuracy lies around 0.94 for all the models. Although slightly different, all models performed better with smaller batch sizes of 32 or 64. The final model will be the one with the best validation accuracy achieved (model 3). We can confirm the best number of epochs by graphing the losses and accuracies one more time.

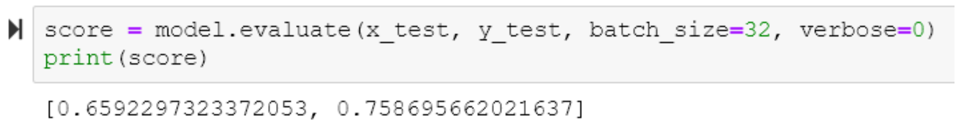


We can see from the figure that the validation loss stops changing significantly after 7 epochs



Here we can also see that the model accuracy stabilizes after around 7 epochs as well.

Compiling the final model at 7 epochs and evaluating it on the test set we get an **accuracy of 0.7587**. This happened to be lower than out validation accuracy, although still much better than the estimated accuracy for a binary random classifier (0.5). After some investigation, I found out that the test data was imbalanced, and contained only samples labelled as non-spam only. The model might perform better if we had a more balanced test set.



```
score = model.evaluate(x_test, y_test, batch_size=32, verbose=0)
print(score)
```

```
[0.6592297323372053, 0.758695662021637]
```

References

1. UCI Repository - Spambase dataset. https://archive.ics.uci.edu/ml/datasets/Spambase
2. Why Loss and Accuracy Metrics Conflict? http://www.jussihuotari.com/2018/01/17/why-loss-and-accuracy-metrics-conflict/