MONASH University

ECE4179 – NEURAL NETWORKS AND DEEP LEARNING PROJECT S2 2020

Buy or Sell Stock Classifier

Ridhwan Mohamed 28892674, Jeremy Vasic 27771458

Emails: rmoh0004@student.monash.edu, jvas9@student.monash.edu

TABLE OF CONTENTS

1. SUMMARY

1.1. Brief Background

The aim of this project is to train a neural network to classify a trading/investment stock as 'BUY' or 'SELL' for short-term profit gains. The network will consider the stock's recent closing prices, market index, public interest (through google trends) and most recent recommendation from valuation firms. As both of us are Electrical Engineering and Commerce (Finance) double degree students, this topic was particularly interesting for us.

Accurate prediction of stock prices can be challenging given the volatile and non-linear nature of the financial market. There is the huge problem of overfitting with stock price NNs, with the network converging adequately but performing poorly with unseen test data [1]. The prices of stocks are largely dependent on external factors such as news and community sentiment rather than historical performance. We plan to combat this using regularization techniques and utilizing more than just simple historical data to train the network.

In this report we detail our process of implementing a deep binary classification neural network to achieve this goal. We trial the network using different numerical and categorical data sources, and as test its performance for different company's stocks - our main case study being TSLA: Tesla Motors.

1.2. Overview of Proposed Method

We chose a binary classification (1 = BUY, 0 = SELL) system due to its ease of implementation and minimizing confusion for the network outputs. The deep model utilizes four linear layers, ReLU activation, batch normalization and drop-out regularization.

The network structure includes a combination of standard numerical input, but also categorical input in the form of the most recent trading recommendations from valuation firms. The numerical and categorical input fields of the network are shown in the tables below on the right:

A key innovation in our approach is the inclusion of daily external factors that affect the stock price – many other models naively use historical stock prices and company statistics exclusively. It is common for these models to experience overfitting, or memorization of the training dataset and perform poorly with unseen data. The neural network output when only given historical prices results in a network that approximates a 'best fit' function that minimizes the loss of the training set.

To avoid the downfalls of using exclusively time series data and approximating a function, we felt that a binary classification model that advises if we should take a long (buying) or short (selling) position on a stock today. We included external factors such as the daily google trends score and firm recommendations. It is relevant to note that these inputs would only useful in cases where the chosen company is high profile – such as Tesla – where internet traffic, media coverage, and frequent valuations impact the stock price heavily.

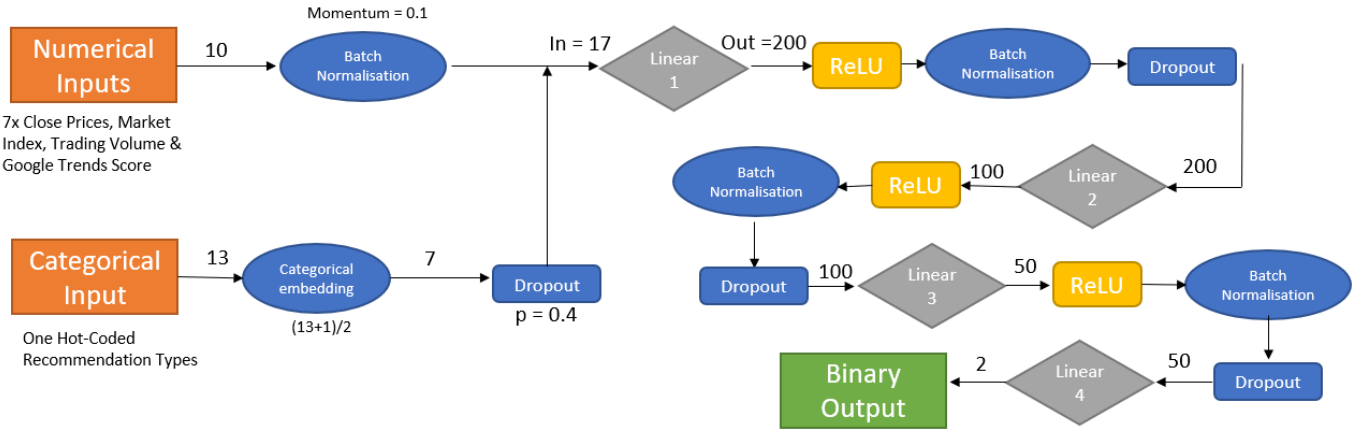
Numerical (10)
6x Historical Close Prices
Current Close Price
Daily Trading Volume
Index Close Price
Google Trends Score

These 13 categorical entries are one-hot coded for the input.

1x Categorical 'Recent Recommendations' Field
Buy
Equal-Weight
Hold
Market Outperform
Neutral
Outperform
Overweight
Peer Perform
Sector Perform
Sell
Underperform
Underweight

Recommendations are made by firms such as JP Morgan, Citigroup, etc.

A flow diagram of the model structure is shown below:



The final output produces a buy (1) or sell (0) recommendation, given the initial 23 inputs. The output labelling for training the network was determined by the next day's (t+1) price– for example; if the price tomorrow is higher than the price today, we label today's output as '1' or buy. Once we have trained the network on data where we know the next day's price, we will test it on an unseen input set where it is the model's job to determine whether we should take a long or short position on the stock today to make a profit tomorrow.

2. IMPLEMENTION

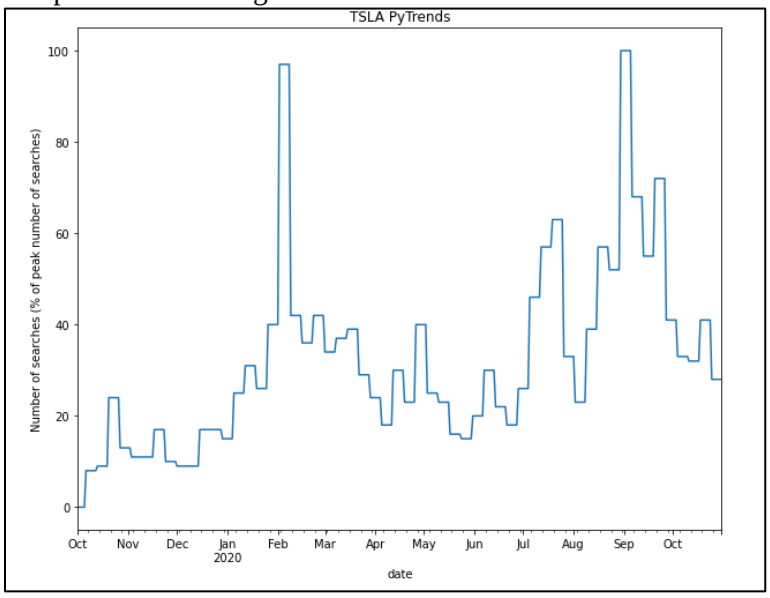
2.1. Data sources

The dataset for this model was sourced from a combination of two sources. Yahoo Finance and Google Trends. Both have python libraries that can be smoothly imported to our Jupyter Notebook. The scope of our dataset was a year of daily trading data from 1st of October 2019 to 31st October 2020.

2.1.1. pyTrends (Google Trends)

Google Trends is website which provides analytics on Google searches, including the number of daily searches for given keywords. We attempted to use certain keywords as proxies for a stock's popularity and sentiment. For example, the company "Tesla, Inc" is publicly traded using the ticker symbol "TSLA". The number of Google searches for TSLA should have some relation to the popularity of the Tesla stock. As stock price is highly driven by consumer sentiment.

Graph of TSLA Google Trends from 1 Oct 2019 to 31 Oct 2020



Yahoo Finance TSLA stock price from 1 Oct 2019 to 31 Oct 2020

In order to source this search result data, we used a Python Google Trends API known as pyTrends [2]. Below is an example of request for the daily search data for the keyword “TSLA”.

Code requesting TSLA search data via pyTrends

```
#Using the .get_daily_data as, the above PyTrends methods only get WEEKLY data for pull requests over 3 months
pyTrendsdata_full = dailydata.get_daily_data('TSLA', 2019, 10, 2020, 10, geo = 'US')
clear_output(True)
#print(pyTrendsdata_full,"\n")

pyTrendsdata = pyTrendsdata_full['TSLA_monthly'] #Selecting only the search requests data column
pyTrendsdata = pyTrendsdata.fillna(0) #Removing all NaN values
#print(pyTrendsdata,"\n")
#print('pyTrendsdata Shape:', pyTrendsdata.shape,"\n")

image = pyTrendsdata.plot(title = 'TSLA PyTrends')
image.set_ylabel("Number of searches (% of peak number of searches)")
fig = image.get_figure()
fig.savefig('figure.png')
pyTrendsdata.to_csv('TSLA.csv', encoding='utf_8_sig')
```

2.1.2. yfinance (Yahoo Finance)

Yahoo Finance is a comprehensive source of stock market data such as the close and open price for any Australian listed stocks (ASX). Yfinance is a Python API that allows access to the historical stock prices data from Yahoo Finance, including trade volumes, prices, and dividends for ASX and internationally listed stocks [3].

Snapshot of the combined Yahoo Finance and Google Trends dataset
(total rows from 1/10/2019 to 31/10/2020)

	Close_tMinus4	Close_tMinus3	Close_tMinus2	Close_tMinus1	Close	Volume	Index_Close	TSLA_monthly	Recent_Recommendation	Prediction
1/10/2019	46.285999	47.543999	48.009998	48.905998	48.948002	31416500	287.676880	8.0	Market Perform	1
2/10/2019	47.543999	48.009998	48.905998	48.948002	49.577999	42377000	290.659180	8.0	Market Perform	1
3/10/2019	48.009998	48.905998	48.948002	49.577999	51.391998	51025000	290.335480	9.0	Market Perform	1
4/10/2019	48.905998	48.948002	49.577999	51.391998	51.577999	32164000	293.209900	9.0	Market Perform	1
5/10/2019	48.948002	49.577999	51.391998	51.577999	51.950001	33420500	292.738983	9.0	Market Perform	1

1 = BUY

Code requesting TSLA stock price data via yfinance library

```
import yfinance as yf
import numpy as np

stock = yf.Ticker('TSLA')
# print(stock.recommendations)

history = stock.history(start="2019-01-01", end="2019-12-31", interval = '1d')
#print(history)
close = history['Close']
volume = history['Volume']
grade = stock.recommendations['To Grade']
#print(grade)

index = yf.Ticker('SPY')
history2 = index.history(start="2019-01-01", end="2019-12-31", interval = '1d')

history2.rename(columns={'Close': 'Index_Close'}, inplace=True)
index_close= history2['Index_Close']


#Getting previous weeks price history
close_tMinus1 = close.shift(1)
close_tMinus1.rename('Close_tMinus1', inplace=True)

close_tMinus2 = close.shift(2)
close_tMinus2.rename('Close_tMinus2', inplace=True)

close_tMinus3 = close.shift(3)
close_tMinus3.rename('Close_tMinus3', inplace=True)

close_tMinus4 = close.shift(4)
close_tMinus4.rename('Close_tMinus4', inplace=True)

close_tMinus5 = close.shift(5)
close_tMinus5.rename('Close_tMinus5', inplace=True)
```

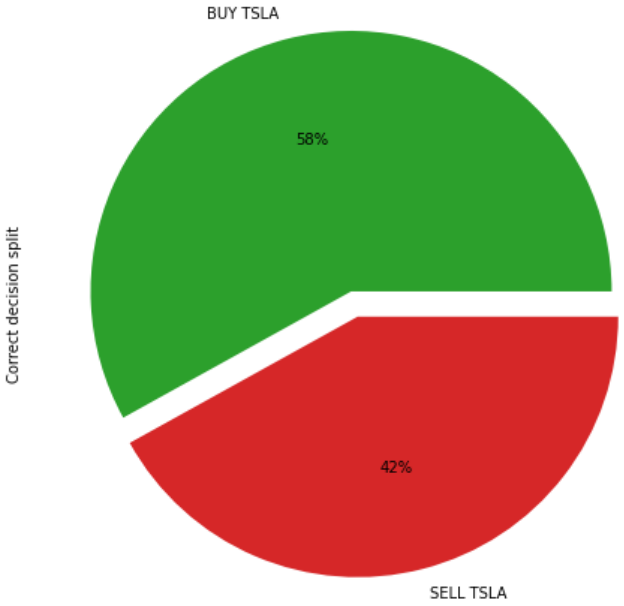
```
close_tMinus6 = close.shift(6)
close_tMinus6.rename('Close_tMinus6', inplace=True)


#Combining pyTrends and yfinance data
combined = pd.merge(close_tMinus6, close_tMinus5, left_index=True, right_index=True)
combined = pd.merge(combined, close_tMinus4, left_index=True, right_index=True)
combined = pd.merge(combined, close_tMinus3, left_index=True, right_index=True)
combined = pd.merge(combined, close_tMinus2, left_index=True, right_index=True)
combined = pd.merge(combined, close_tMinus1, left_index=True, right_index=True)
combined = pd.merge(combined, close, left_index=True, right_index=True)

#REMOVE combined = pd.merge(close, volume, left_index=True, right_index=True)
combined = pd.merge(combined, volume, left_index=True, right_index=True)
combined = pd.merge(combined, index_close, left_index=True, right_index=True)
combined = pd.merge(combined, pyTrendsdata, left_index=True, right_index=True)
combined = pd.merge_asof(combined, grade, left_index=True,right_index=True)
#print(combined)

#Removing first week of data, as no previous close price data for it
combined = combined.drop(pd.date_range('2018-12-31', '2019-01-08'), errors='ignore')

#renaming col names
combined.rename(columns={'To Grade': 'Recent_Recommendation'}, inplace=True)


#Creating target labels -1 and 1, using change in Close price from previous day
combined['Delta'] = (combined['Close'] - combined['Close'].shift(1)).fillna(0)
combined['Delta'] = combined['Delta'].shift(-1)
combined['Prediction'] = np.where(combined['Delta'] >0, 1, 0) #Prediction 1 = BUY, 0 = SELL
combined = combined.drop(labels=['Delta'],axis='columns')
print(combined)
combined.to_csv('OurData.csv', encoding='utf_8_sig', index = True)
```

Merging all the feature columns

2.2. Cleaning data

2.2.1. Designing labels (output)

True output labels split (before training)



Our output labels consisted of 1="BUY" or 0="SELL". These output labels were created from Yahoo Finance closing price data, based upon whether the given company's stock price rose (investor/trader should BUY) or fell (investor/trader should SELL) for the close of the next day (t+1).

2.2.2. Designing features (inputs)

Experimenting with different Google Trends keywords (e.g. "TSLA", "Model 3", "tesla stock price") and Yahoo Finance data (e.g. TSLA close price, recommendations by firms to Buy/Hold/Sell, trade volume) was the key to finding the most accurate model.

Some of the features where in the form of categorical data (strings). To overcome this issue, we implement a method [4] of one hot encoding the categorical data into 1's and 0's.

Code showing the datatypes of the features

```
categorical_columns = ['Recent_Recommendation']
numerical_columns = ['Close_tMinus6','Close_tMinus5','Close_tMinus4','Close_tMinus3','Close_tMinus2','Close_tMinus1','Close'
outputs = ['Predicted']
```

```
dataset.dtypes
```

```
Unnamed: 0               object
Close_tMinus6           float64
Close_tMinus5           float64
Close_tMinus4           float64
Close_tMinus3           float64
Close_tMinus2           float64
Close_tMinus1           float64
Close                   float64
Volume                    int64
Index_Close             float64
TSLA_monthly            float64
Recent_Recommendation    object
Prediction                int64
dtype: object
```

Both labels and features were final converted to PyTorch tensors (Long/64-bit integer (signed))

```
outputs = ['Prediction']
outputs = torch.tensor(dataset[outputs].values).flatten()
outputs[:5]
```

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

We split our total dataset, with 80% used for training and the remaining 20% used for testing. Additional data processing is required to properly combined the numerical and categorical data.

```
total_records = categorical_data.shape[0];
test_records = int(total_records * .2)

categorical_train_data = categorical_data[:total_records-test_records]
categorical_test_data = categorical_data[total_records-test_records:total_records]
numerical_train_data = numerical_data[:total_records-test_records]
numerical_test_data = numerical_data[total_records-test_records:total_records]
train_outputs = outputs[:total_records-test_records]
test_outputs = outputs[total_records-test_records:total_records]
```

2.3. Defining Model and Training

We define the model as described in section 1.2 – Overview of Proposed Method:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):

    def __init__(self, embedding_size, num_numerical_cols, output_size, layers, p=0.4):
        super().__init__()
        self.all_embeddings = nn.ModuleList([nn.Embedding(ni, nf) for ni, nf in embedding_size])
        self.embedding_dropout = nn.Dropout(p)
        self.batch_norm_num = nn.BatchNorm1d(num_numerical_cols)

        all_layers = []
        num_categorical_cols = sum((nf for ni, nf in embedding_size))
        input_size = num_categorical_cols + num_numerical_cols

        for i in layers:
            all_layers.append(nn.Linear(input_size, i))
            all_layers.append(nn.ReLU(inplace=True))
            all_layers.append(nn.BatchNorm1d(i))
            all_layers.append(nn.Dropout(p))
            input_size = i

        all_layers.append(nn.Linear(layers[-1], output_size))

        self.layers = nn.Sequential(*all_layers)

    def forward(self, x_categorical, x_numerical):
        embeddings = []
        for i,e in enumerate(self.all_embeddings):
            embeddings.append(e(x_categorical[:,i]))
        x = torch.cat(embeddings, 1)
        x = self.embedding_dropout(x)

        x_numerical = self.batch_norm_num(x_numerical)
        x = torch.cat([x, x_numerical], 1)
        x = self.layers(x)
        return x
```

Dropout rate

```
model = Model(categorical_embedding_sizes, numerical_data.shape[1], 2, [200,100,50], p=0.4)
```

Categorical Input

Numerical Input

# of outputs layers

Linear laver sizes

Output of Model Structure:

```
Model(
  (all_embeddings): ModuleList(
    (0): Embedding(11, 6)
  )
  (embedding_dropout): Dropout(p=0.4, inplace=False)
  (batch_norm_num): BatchNorm1d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layers): Sequential(
    (0): Linear(in_features=16, out_features=200, bias=True)
    (1): ReLU(inplace=True)
    (2): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Dropout(p=0.4, inplace=False)
    (4): Linear(in_features=200, out_features=100, bias=True)
    (5): ReLU(inplace=True)
    (6): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): Dropout(p=0.4, inplace=False)
    (8): Linear(in_features=100, out_features=50, bias=True)
    (9): ReLU(inplace=True)
    (10): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): Dropout(p=0.4, inplace=False)
    (12): Linear(in_features=50, out_features=2, bias=True)
  )
)
```

Defining Hyperparameters:

```
#Hyperparameters
epochs = 500
lr = 0.001 #learning rate
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr)
```

Recalling our experience during labs and assignments, we observed that ADAM optimizer outperformed Stochastic Gradient Descent (SGD) due to the dynamically changing learning rate. This combination of hyperparameters; a low LR and high max_epoch, led to the best results for our network.

Code used to train network and log losses, accuracy and best accuracy:

```
aggregated_losses = []

best_accuracy = 0
best_epoch = 0

testAccuracyLogger = []

for i in range(epochs):
    i += 1
    y_pred = model(categorical_train_data, numerical_train_data)
    single_loss = loss_function(y_pred, train_outputs)
    aggregated_losses.append(single_loss)

    if i%25 == 1:
        clear_output(True)
        print(f'epoch: {i:3} loss: {single_loss.item():10.8f}')
        print(best_accuracy)
        print(best_epoch)
    optimizer.zero_grad()
    single_loss.backward()
    optimizer.step()


    #####
    with torch.no_grad():
        y_val = model(categorical_test_data, numerical_test_data)
        loss = loss_function(y_val, test_outputs)
    y_val2 = np.argmax(y_val, axis=1)
    accuracy = accuracy_score(test_outputs, y_val2)
    testAccuracyLogger.append(accuracy)
    if (accuracy>best_accuracy):
        best_accuracy = accuracy
        best_epoch = i


clear_output(True)
print(f'epoch: {i:3} loss: {single_loss.item():10.10f}')
print('Best accuracy: ',"%.3f" % (best_accuracy*100),'% at epoch ',best_epoch,sep='')
print('Final accuracy: ',"%.3f" % (accuracy_score(test_outputs, y_val2)*100),'%',sep='')
```

```
epoch: 500 loss: 0.5310603976
Best accuracy: 71.698% at epoch 188
Final accuracy: 58.491%
```

3.1. Standard Model

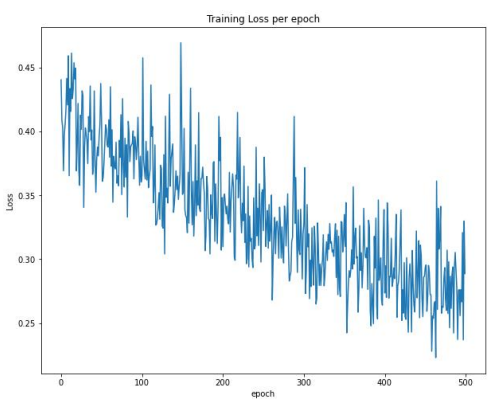
The dataset ranged from the 1st of October 2019 to the 31st of October 2020 – or 267 trading days. We felt it to be interesting to use the most recent years data, considering the highly volatile markets since COVID-19.

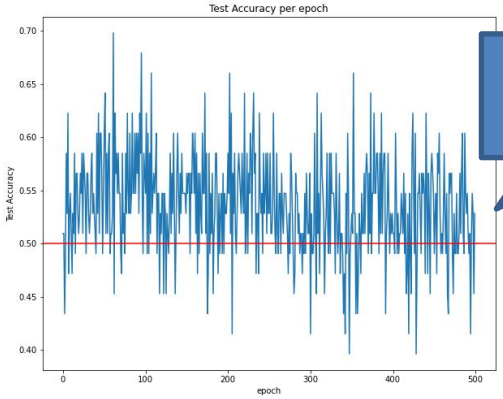
Our initial model did not use any google trends or categorical recommendation data. This was a combination of only the closing price data, historic data for the previous week, and the trade volume.

```
epoch: 500 loss: 0.2887781858
Best accuracy: 69.811% at epoch 62
Final accuracy: 52.830%
```

Low training loss implies overfitting with this model.

The model reached a final accuracy of 52.8% which was the lowest out of all our models. Having the lowest accuracy was expected as it had the least number of input features to form its prediction. Throughout training the test accuracy varied significantly between epochs, there did not appear to be a consistent improvement in accuracy as the number of epochs increased.

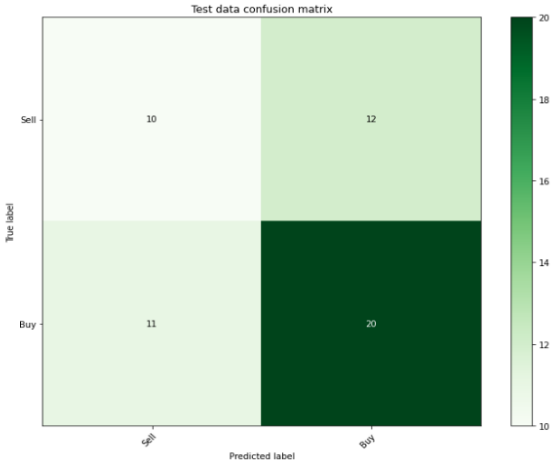




It is promising that the test accuracy was consistently above 50%

It reached a peak accuracy of 69.9%, this is relatively insignificant. As the model probably reached this level of accuracy through relatively random chance, after only a few iterations. In the accuracy plot on the right, it can be observed that the accuracy did not consistently stay above 60% for an extended period.

The model seems to overclassify days as 'BUY', which is expected considering the 58-42 buy/sell labelling.



ADAM GD
Cross Entropy Loss
LR = 0.001
Max_Epoch = 500

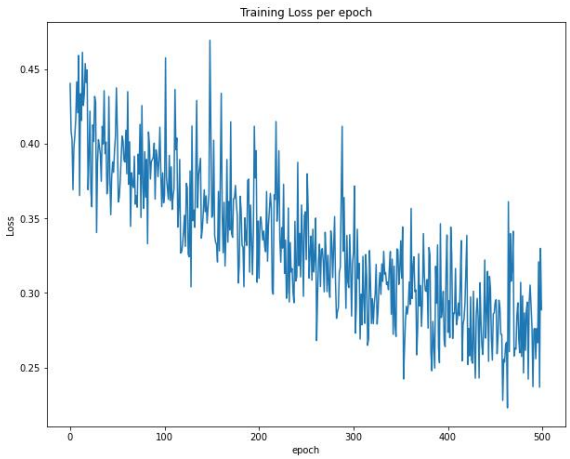
3.2. Model with Google Trends

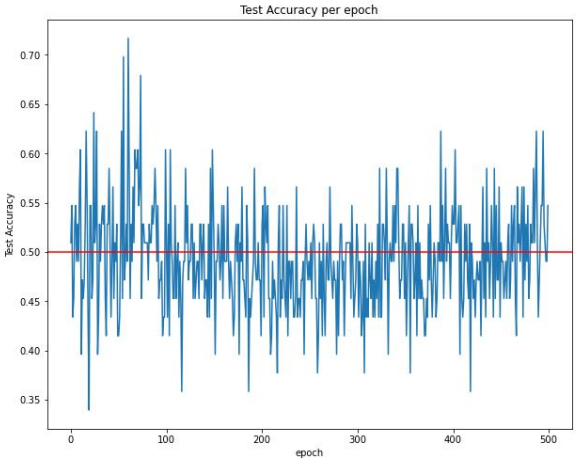
The results of the model after incorporating google trends data are as follows:

```
epoch: 500 loss: 0.4229362011
Best accuracy: 71.698% at epoch 61
Final accuracy: 54.717%
```

Higher training loss implies less overfitting with this model.

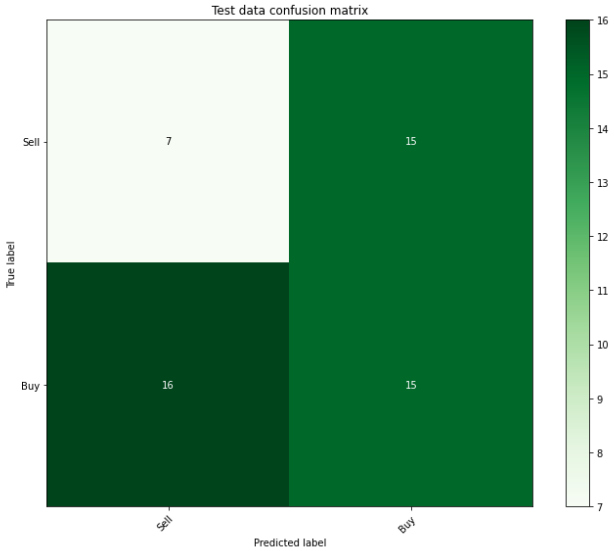
The model reached a final accuracy of 54.7% this was slightly higher than the model that did not contain any Google Trends data. The testing accuracy did not perform significantly better than the standard model. A possible reason for this could be that the number of searches did not capture whether the sentiment towards the stock is positive or negative. Individual traders could be searching TSLA stock prices and then shorting (selling) the stock, which would mean that this Google Trends data is comparable random noise. This could be improved by testing models with a variety of key terms such as "buy TSLA" or "sell TSLA" that more effectively capture the sentiment towards the TSLA stock price.





This version reached a peak accuracy of 71.7%. Throughout training the test accuracy varied significantly between epochs. The model appeared to perform at its best in the first 60 epochs, this may be due to overfitting for above 60 epochs. As the testing accuracy is stochastic and has a large variance, we cannot safely say overfitting as the cause for this decline.

The model continues to overclassify days as 'BUY'.



ADAM GD
Cross Entropy Loss
LR = 0.001
Max_Epoch = 500

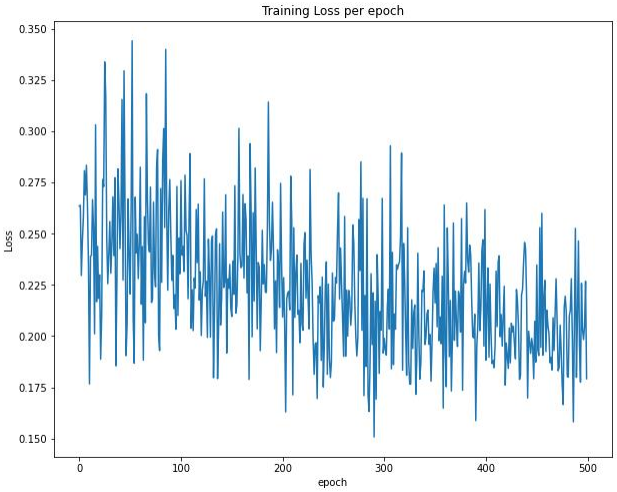
3.3. Model with Google Trends and Recent Recommendations

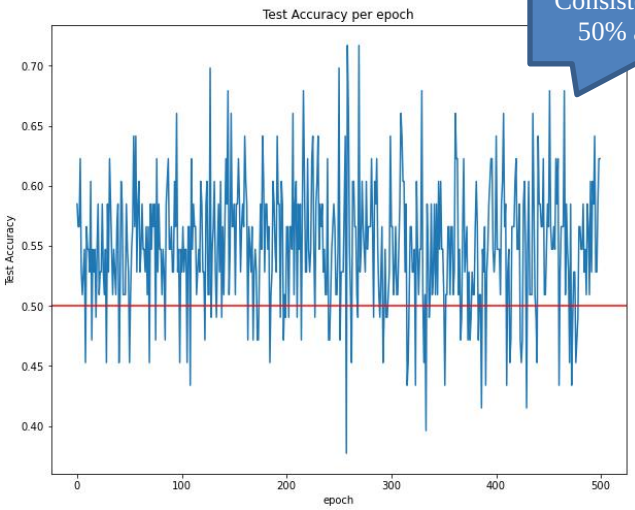
This final model utilized both google trends data as well as the most recent daily categorical recommendations from established firms such as JP Morgan and Citigroup. The results are as follows:

```
epoch: 500 loss: 0.1792159677
Best accuracy: 71.698% at epoch 259
Final accuracy: 62.264%
```

Lower training loss implies more overfitting with this model. Nevertheless, we still have superior performance with the test set.

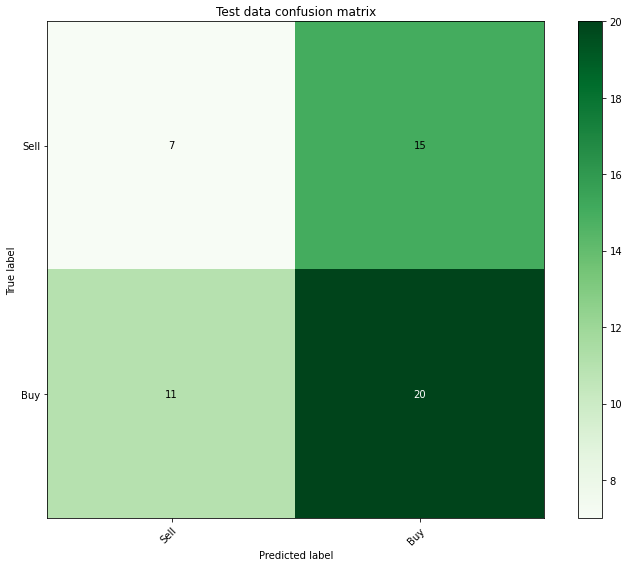
This model produced the best final test accuracy from all the previous models at 62%. We see that the final test accuracy consistently increases as we add more relevant data. Incorporating the most recent recommendations from valuation firms seemed to be very beneficial for the network, the test accuracy per epoch is volatile but is consistently above 50%.





Consistently above 50% accuracy.

The confusion matrix still shows over classification of 'BUY' but given the nature of the input dataset of Tesla this is to be expected. We trial this network for other stocks in the next section.

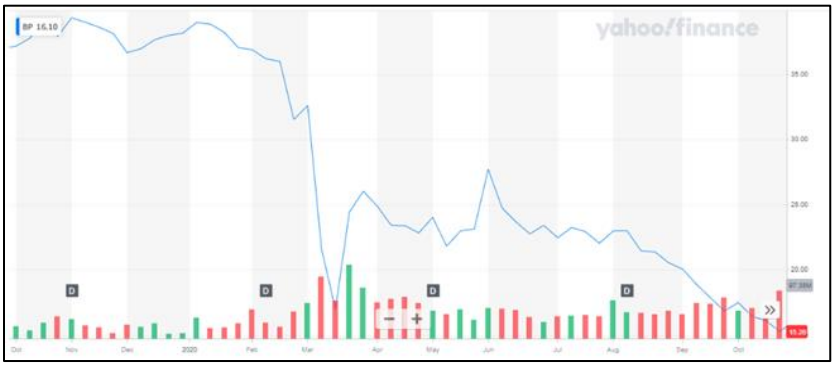


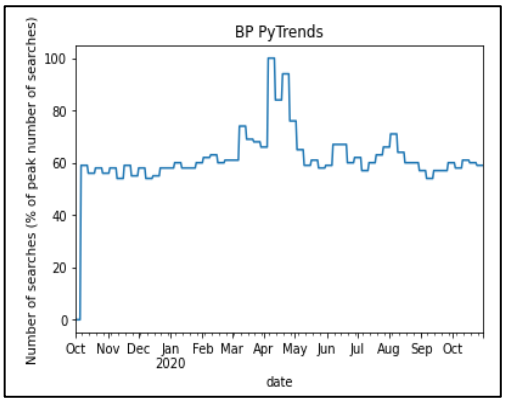
ADAM GD
Cross Entropy Loss
LR = 0.001
Max_Epoch = 500

3.4. Performance with Other Stocks – BP and MCD

Tesla was a stock which primarily appreciated over the past year, we felt it would be interesting if we trained and tested the model for stocks that were decreasing or volatile over the 12 months.

A stock that decreased from 1 Oct 2019 to 31 Oct 2020 was BP – Oil and Gas: The stock history and google trends chart are below:

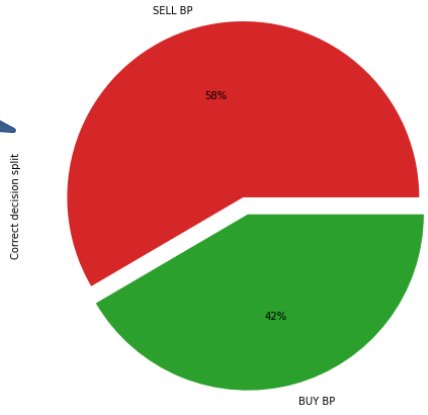


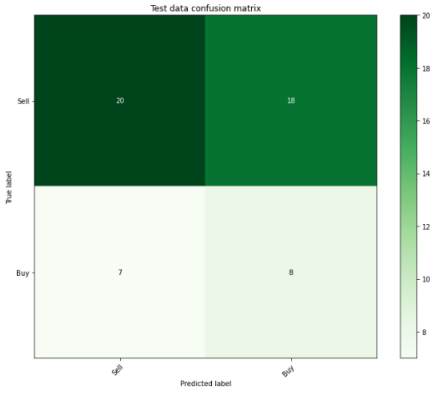


The results of the network after training with BP Data are as follows:

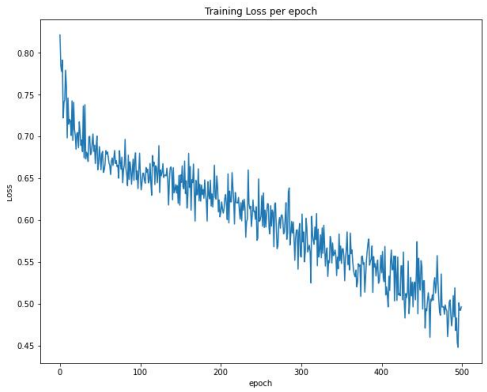
```
epoch: 500 loss: 0.4964604974
Best accuracy: 75.472% at epoch 52
Final accuracy: 66.038%
```

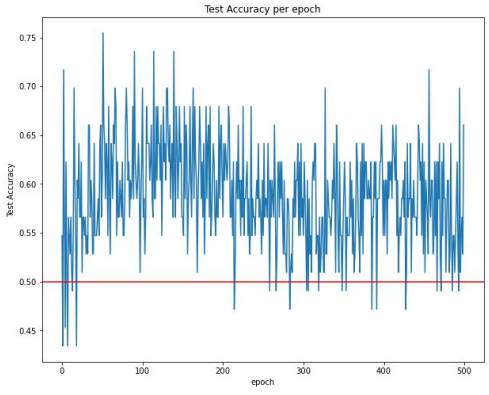
BP is more heavily biased towards 'SELL'





ADAM GD
Cross Entropy Loss
LR = 0.001
Max_Epoch = 500

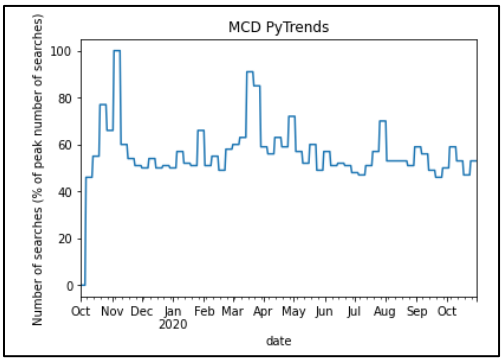




Consistently above 50%

From the confusion matrix we see the over classification of 'Sell' for BP stock (which is the opposite for TSLA). This is expected at BP primarily decreased over the dataset period. Test accuracy was finalised to be 66%, with the peak accuracy reaching 75% - which is the best so far. We see that the model performs very well with stocks that primarily move in a single direction.

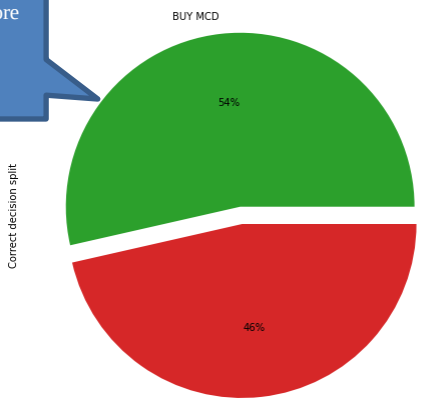
A stock that was more volatile from 1 Oct 2019 to 31 Oct 2020 was MCD – McDonalds:

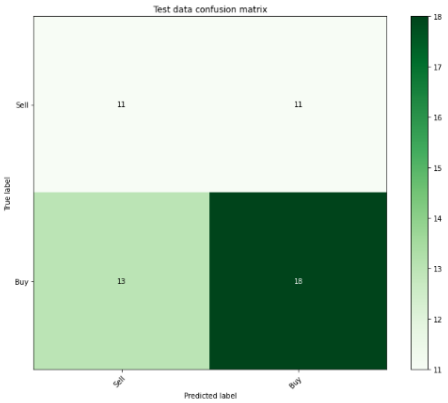


The results of the network after training with MCD Data are as follows:

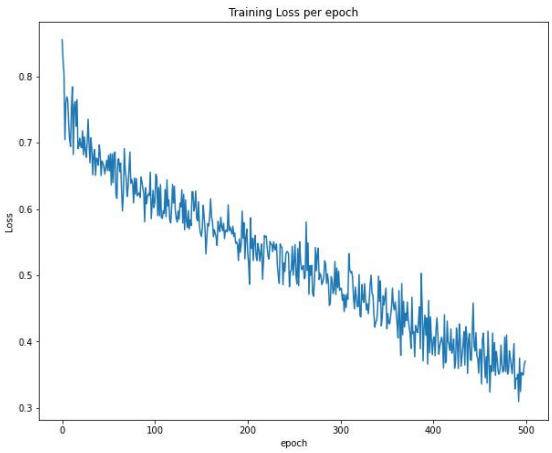
```
epoch: 500 loss: 0.3707025349
Best accuracy: 69.811% at epoch 2
Final accuracy: 60.377%
```

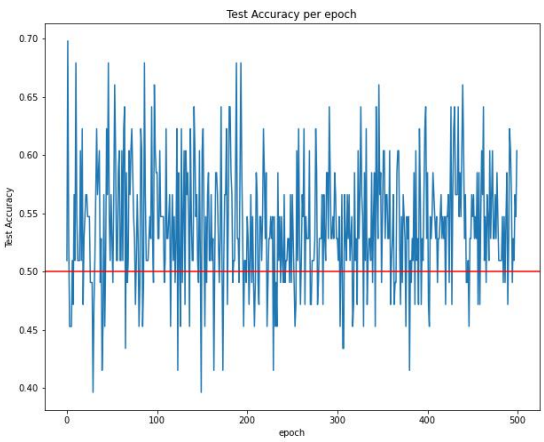
MCD has a more balanced BUY/SELL proportion





ADAM GD
Cross Entropy Loss
LR = 0.001
Max_Epoch = 500





The model with McDonalds data has a general test accuracy consistently above 50%, but it is quite volatile in its performance. Both Tesla and McDonalds had a relatively strong recovery since the COVID-19 crash in March, which results in relatively similar accuracy plots and confusion matrices. The final accuracy of above 60% shows that the model performs relatively well for volatile stocks, but overall performance is better with more stable movements up or down.

4. RECOMMENDATIONS AND CONCLUSION

In conclusion, our model performed adequately for predicting ‘Buy’ or ‘Sell’ when given only historical stock data. A small increase in accuracy was observed when we included google trends or public interest data. A much larger increase in accuracy was observed when we included recent professional recommendations. It was difficult to incorporate this one-hot encoded categorical data, but the added performance demonstrated its effectiveness. With this full model, we achieved consistent (albeit volatile) test accuracies of 60%+ which is significantly better than random guessing or an overfitted model. As we observed with Tesla and BP the model is best suited to stocks that primarily move in one direction, but it still performed well with volatile stocks such as McDonald’s.

A limitation of our model is that is relies on the stock being sufficiently ‘high profile’ in that there must be frequent recommendations from valuation firms and google trends interest. A company that is too small or lacks media coverage would lack the data needed for the model to make accurate predictions. Another key limitation is that this is a binary classification model, in that that there is no ‘strong buy’, ‘strong sell’ or ‘hold’ recommendations, in the case the price change only varies by a small amount.

A potential improvement would be to include trialing this model for a multitude of different stocks at the same time. This approach would have a different stock in every input row, resulting in a more generalized model to make a prediction for any specified stock without retraining. However, this may be too confusing and general for the network to work accurately. Another improvement would be to add additional input columns, this could be in the form of more historical and index data as well as additional google trends query’s for things like “Tesla Model 3” or “Elon Musk” which could add accuracy to the model’s performance for Tesla.

5. REFERENCES AND APPENDICES

[1] Ivan. “Is It Possible to Predict Stock Prices with a Neural Network?” Towards Data Science, 8 Feb. 2020, towardsdatascience.com/is-it-possible-to-predict-stock-prices-with-a-neural-network-d750af3de50b. Accessed: 15 Oct. 2020.

[2] J. Hogue and B. DeWilde, "pytrends", PyPI, 2020. Available: https://pypi.org/project/pytrends/#historical-hourly-interest. Accessed: 02 Nov. 2020

[3] R. Aroussi, "yfinance", PyPI, 2020. Available: https://pypi.org/project/yfinance/. Accessed: 02 Nov. 2020

[4] U. Malik, "Introduction to PyTorch for Classification", Stack Abuse, 2020 Accessed: 02 Nov. 2020

[5] G. Thushan. “Stock Market Predictions with LSTM in Python.”, DataCamp, 2 Jan. 2020, www.datacamp.com/community/tutorials/lstm-python-stock-market. Accessed: 15 Oct. 2020.

[6] “Types of Neural Networks and Definition of Neural Network”, Great Learning Team, 25 Apr. 2020, https://www.mygreatlearning.com/blog/types-of-neural-networks/. Accessed 15 Oct. 2020.

[7] “Long short-term memory 2020”, https://en.wikipedia.org/wiki/Long_short-term_memory Accessed: 15 Oct. 2020.

[8] Saldanha, Rodolfo. “Stock Price Prediction with PyTorch” Medium, 3 Jun. 2020, https://medium.com/swlh/stock-price-prediction-with-pytorch-37f52ae84632. Accessed: 15 Oct. 2020.