Predicting Stock Prices Using An RNN Model

Shlomo Amor
ID. 000803254

Yarden Kuperberg
ID. 311149082

Submitted as final project report for the Deep Learning course, IDC, 2021

1 Introduction

Being able to predict market trends is of great importance. There are many complicated financial indicators which can help predict stock market trends. However, as technology is advancing, the opportunity to gain a steady fortune from the stock market has increased. This has allowed experts to uncover the most informative indicators, henceforth allowing them to make better predictions, while managing to keep their risks low.

Recurrent neural networks (RNN) have proved one of the most powerful models for processing sequential data. Long Short-Term memory (LSTM) is one of the most successful RNNs architectures. LSTM introduces the memory cell, a unit of computation that replaces traditional artificial neurons in the hidden layer of the network.

The FTSE 100 is an index composed of the 100 largest companies, by market cap, listed on the London Stock Exchange (LSE). These are often referred to as 'blue chip' companies, and the index is seen as a good indication of the performance of major companies listed in the UK.
This paper explores the power of using LSTMs, and their ability to keep and forget information. This will hopefully allow us to predict successfully and accurately the market trends and directional movement of certain stocks (48 in this case) within the FTSE 100.
We aim to achieve this by:

- Creating 48 different RNN's, one for each of the selected companies in the FTSE 100. Each RNN will be trained on a specific company's historical market data. The input to each RNN will be the company's opening price, with the aim to predict that stock's next day's directional movement.
- Creating a single generic RNN. This RNN will be trained on a combination of each of the selected companies in the FTSE 100's historical market data. The input to this generic RNN will be a company's opening price, as well as the industry the company belongs to.

We will then compare the accuracy between the company specific method vs the generic model method. The approach about how we will measure the model's accuracy will be discussed further in the 'Experimental Results' section.

1.1 Related Works

There are various different resources that can be found online that describe different methods and approaches in using LSTMs to predict daily stock prices. We used the following links:

- https://towardsdatascience.com/stock-prediction-using-rnn
- https://towardsdatascience.com/stock-market-predictions-with-rnn

The articles above gave a clear overview of how to model and design the RNNs. These articles sparked an interest in the use of RNNs for stock predictions and inspired the latter half of this research, specifically the use of encoding sector names into vector format and utilising this as an additional input for the generic RNN. The above articles also were, we felt, expandable because of the fact that only one stock was monitored. In our research, we have chosen 48 stocks to monitor and use to help increase our predictive power.

2 Solution

2.1 General approach

We wanted to define an RNN in such a way that required no difference in architectural structure between both methods (Stock Specific and Generic). We split the research into 4 sections:

- Section 1: Data extraction and Pre-processing: We began by installing and importing a special library called yfinance. We were able to extract historical daily market data for a selection of the FTSE 100 companies. The data extracted for each stock was presented in the following format:

Date	Open	High	Low	Close	Volume
2005-03-29	6.584420	6.587054	6.584420	6.584420	807900
2005-03-30	6.584420	6.587054	6.584420	6.584420	18300
2005-03-31	6.584421	6.589689	6.584421	6.587055	21400
2005-04-01	6.589688	6.589688	6.584420	6.584420	10300
2005-04-04	6.587054	6.587054	6.584420	6.584420	30900

Figure 1: Data

– Section 2: Building a specific RNN for each stock: The specific architecture of the RNN will be discussed in more detail later, in the 'Design' section. Briefly, the 'time-step' is a given time frame used within the historical data set to group data into a uniform format that can be used throughout the research. In our case, the 'time-step' size we chose was 5 days, hence the use of 5 days worth of data to predict the 6th day's opening price. The RNN's input is a vector of the time-step containing opening prices and the output is the predicted opening price on the time-step+1 day. An important hyper-parameter here is time-step, should we simply take the previous opening price or should we take the previous 100 daily opening prices?

– Section 3: Building a Generic RNN for all stocks: The architecture of this RNN is identical to the Specific-RNN. Part of the RNN's input is a vector of the time-step containing opening prices. A One-hot encoding of the sector that the company belongs to, is concatenated with that opening price vector and is used as the input.

– Section 4: Analysis Comparison of both approaches: We introduce here the idea of 'monotonicity measures'. We define this as the model's accuracy in predicting movements in the direction that the stock actually moves in. We measured our model's accuracy by quantifying it's ability to look at the real data, where the price falls/rises on day 'time-step+1', and predict a movement in the same direction.

2.2 Design

We will begin by explaining the RNN architecture. The following block of code is the RNN's definition:

```
def build_rnn(X_train,y_train, epochs, batch_size, dropout, rnn, first_stage):
    if first_stage:
        # Initialising the RNN
        rnn = Sequential()

        # Adding the first LSTM layer and some Dropout regularisation
        rnn.add(LSTM(units = 50, return_sequences = True,
                     input_shape = (X_train.shape[1], 1)))
        rnn.add(Dropout(dropout))

        # Adding a second LSTM layer and some Dropout regularisation
        rnn.add(LSTM(units = 50, return_sequences = True))
        rnn.add(Dropout(dropout))

        # Adding a third LSTM layer and some Dropout regularisation
        rnn.add(LSTM(units = 50, return_sequences = True))
        rnn.add(Dropout(dropout))

        # Adding a fourth LSTM layer and some Dropout regularisation
        rnn.add(LSTM(units = 50))
        rnn.add(Dropout(dropout))

        # Adding the output layer
        rnn.add(Dense(units = 1))

        rnn.compile(optimizer = 'adam', loss = 'mean_squared_error')

    # Fitting the RNN to the Training set
    rnn.fit(X_train, y_train, epochs = epochs, batch_size = batch_size)

    return rnn
```

Figure 2: RNN architecture

As one can see from the code snippet above, our RNN included 3 hidden layers with a dropout of 20%. We also used a mean sqaure loss function, as well as an Adam optimizer since, Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

When it came to training the model, we tested various different epoch values and decided to choose an epochs value of 10. We felt it was the right balance between 'time to train' and 'over-fitting'. Below is an example of the time it took to train a specific RNN on a specific stock:

```
Training RNN for: BME
Epoch 1/10
100/100 [==============================] - 8s 14ms/step - loss: 0.0301
Epoch 2/10
100/100 [==============================] - 1s 14ms/step - loss: 0.0013
Epoch 3/10
100/100 [==============================] - 1s 15ms/step - loss: 8.5167e-04
Epoch 4/10
100/100 [==============================] - 1s 14ms/step - loss: 9.7475e-04
Epoch 5/10
100/100 [==============================] - 1s 14ms/step - loss: 7.1948e-04
Epoch 6/10
100/100 [==============================] - 1s 14ms/step - loss: 6.5874e-04
Epoch 7/10
100/100 [==============================] - 1s 15ms/step - loss: 6.4419e-04
Epoch 8/10
100/100 [==============================] - 1s 14ms/step - loss: 6.7286e-04
Epoch 9/10
100/100 [==============================] - 1s 14ms/step - loss: 5.6146e-04
Epoch 10/10
100/100 [==============================] - 1s 14ms/step - loss: 6.3073e-04
```

Figure 3: Training losses for BME company

For the Generic approach, we kept the RNN architecture the same (See Figure 2) and only had to change the input vector to contain the One-hot encoding of the sector the company belonged to. The following block of code explains how we created a One-hot encoding of the sectors:

```
# Create Onehot vector
onehot_lists = pd.get_dummies(FTSE100_df.Sector, prefix='Sector')
onehot_lists = onehot_lists.values
onehot_lists = onehot_lists.astype(np.float64)
print(onehot_lists[0])
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Figure 4: One-hot Encoding

Finally, we had very different running times between the Specifc models and the Generic models. One contributing factor could have been due to the model already being tuned with previous stocks or the fact that the input contains the sector. Below is an example of the time it took to train the Generic model for the same stock as shown in Figure 3:

```
Training RNN for: BME
Epoch 1/10
100/100 [==============================] - 10s 99ms/step - loss: 0.0332
Epoch 2/10
100/100 [==============================] - 10s 97ms/step - loss: 0.0216
Epoch 3/10
100/100 [==============================] - 10s 98ms/step - loss: 7.7510e-04
Epoch 4/10
100/100 [==============================] - 10s 99ms/step - loss: 5.8902e-04
Epoch 5/10
100/100 [==============================] - 10s 97ms/step - loss: 5.1458e-04
Epoch 6/10
100/100 [==============================] - 10s 97ms/step - loss: 4.5269e-04
Epoch 7/10
100/100 [==============================] - 10s 98ms/step - loss: 5.5472e-04
Epoch 8/10
100/100 [==============================] - 10s 96ms/step - loss: 4.5429e-04
Epoch 9/10
100/100 [==============================] - 10s 98ms/step - loss: 3.8282e-04
Epoch 10/10
100/100 [==============================] - 10s 97ms/step - loss: 4.0318e-04
```

Figure 5: Training losses for BME company

3 Experimental results

When considering trading, or attempting to make money within the financial markets, the direction of the market, Bearish or Bullish, is probably the most important factor one could consider. The specifics of prices help when trying to gauge whether to buy or sell, but on a large scale and with certain frequencies of trades, the direction holds more importance because success can come in a wide variety of ways as long as a direction is known. Knowing whether to hold, sell, go short, or long all are based on the 'hunch' of which direction a market will move in. To be able to use a predictive model to guide this 'hunch' can allow for major profits to be made.

After training and testing the specific models for each stock we were able to produce the following graph, the one shown below for example is the output graph for the stock with ticker BME:

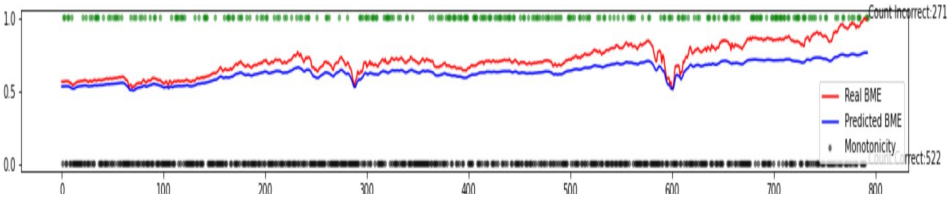

Figure 6: Predictive results for BME using specific model
The graph above contains a few important features:

· Red curve: This is the actual stock opening price for BME.

- Blue curve: This is the predicted stock opening price for BME.
- Black dots at y=0: These dots indicate if our prediction and the real market movement were the same. The fewer gaps in the black line along y=0, the higher the accuracy for this model is. As we can see we predicted 522/793 correct market movements.
- Green dots at y = 1: These dots indicate if our prediction and the real market movement were in the opposite direction hence, we predicted a fall whilst the true movement was a rise or vice versa. The fewer the green dots we have along the line y = 1 the higher the accuracy for this model. As we can see we predicted 271/793 incorrect market movements.

Similarly, we achieved the following graph for the Generic model:

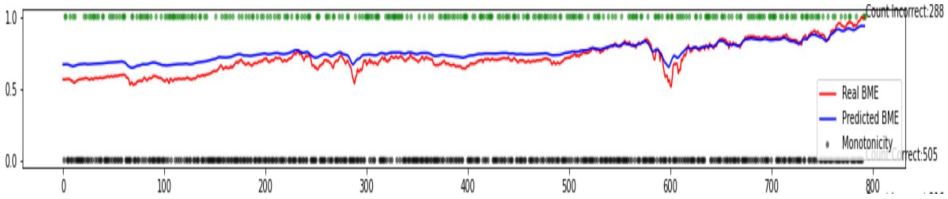

Figure 7: Predictive results for BME using Generic model

As it can be seen the accuracy is not the same and actually lower than the specific model with a predictive power of 501/793 correct market movements. Showing both results on the same graph we get the following:

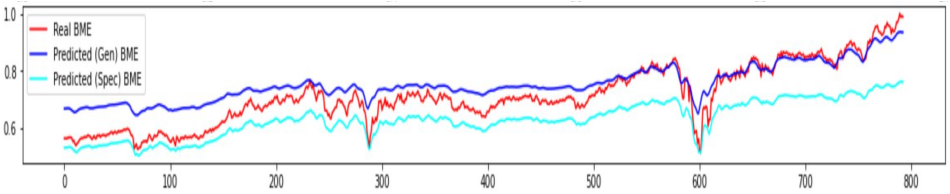

Figure 8: Predictive results for BME using both model

- Red curve: This is the actual stock opening price for BME.
- Blue curve: This is the predicted opening price using the Generic Model.
- Cyan curve: This is the predicted opening price using the Specific Model.

Before beginning the project we were almost certain that the specif models would out-preform the generic model in almost all cases. However, to our surprise we achieved the following results:

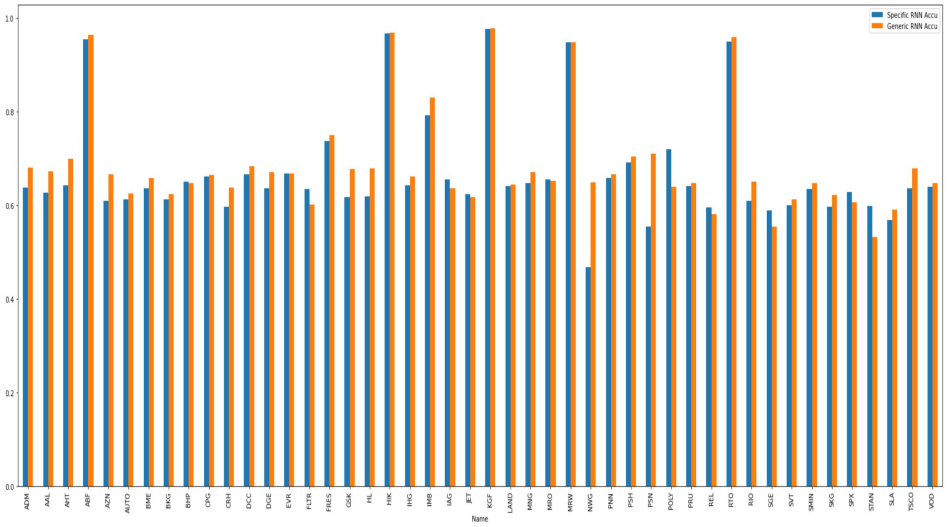

Figure 9: Accuracy comparison of both models

Overall, the generic model out- preformed the specific stock models, which was truly a surprise. We believe that this is due to how we chose to measure the model's accuracy, and the fact that the generic model concentrated more on the trends and less on actual price prediction, since it wasn't tuned on a specific stock, but rather on a sector.

4 Discussion

This research paper opened our eyes to the incredibly powerful uses and insights one can gain when deep learning is applied to real life problems. Although attempting to predict stock market movements can sometimes feel extremely daunting, the confidence that came from using RNNs meant that the bulk of our work was out of our hands. The main challenge for us, was the building and structural architecture of the RNNs. This involved choosing the number of hidden layers, batch sizes and what percentage of dropout to choose for each layer. The guidance from the articles explained above was paramount to our success. The use of RNNs worked in the way we felt would give us the most accurate results, and looking back, the RNN and LSTM models aligned perfectly with our hypothesis question and research direction and we would not change our technique if we repeated this type of research again. As mentioned before, attempting to predict stock market movements in a mighty challenge, however the confidence gained from this paper and doing this research has only encouraged us more to use our knowledge in the future to tackle daily challenges through the use of artificial neural networks. This research project proved to only scratch the surface of the potential that RNNs have on the world of financial markets and quantitative finance. To be in a position to tackle the challenges

head on is something this research question has given us the desire to continue to do. Specifically, we would like to further pursue the questions regarding finding the optimal time-step, as well as RNN architecture related topics including drop out values and batch sizes. A question this has lead to, is can we use the power of RNNs to process unconventional data, such as news articles and current events, and use them as inputs to train models similar to this.

5 Code

Please find in the following link to the code and all the resources used in this research paper. Click Here.