Exploring LSTM Neural Network Architectures for Advanced Forecasting of Stock Prices

Jordan Rowley

August 16th, 2022

University of
St Andrews

Word Count: 8,447

Abstract

Today, algorithmic trading dominates many markets such as foreign exchange [1] and cryptocurrency. This layer of trade exists between the institutional and retail investors where most of the exchanges are between two machines. As machine learning research has developed, neural networks, and especially recurrent neural networks, have become increasingly capable of learning signals within data. Being able to accurately predict time series has immense application across thousands of fields and it seems natural to apply this research to time series predictions which, in our case, is stock price predictions.

While RNNs have achieved state-of-the-art results in many benchmark problems, they suffer from the exploding and vanishing gradient problem in which the 'loops' within the recurrent architecture propagate weights to either exponential growth or decay. This is solved by long-short term memory neural networks that use gates inside memory cells to manage and control the flow of information through the model.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is 8,196 words long, including project specification and plan.

Acknowledgements

Firstly I would like to thank my supervisor, Professor Tom Kelsey, for supporting my research throughout while providing key insights and advice on how to push my project to the next level. The pride I have earned throughout this work is owed in-part to him. I am extremely grateful to have had to opportunity to work on such an exciting project.

I would also like to thank everyone else who has supported me throughout my career at the University of St Andrews including family, friends, professors, and tutors. Specifically I would like to note appreciation for Jon Lewis who set me on this path to graduation after supporting me through an administrative error that caused me to miss an entire semester of lectures.

Contents

1 Introduction

The research undertaken during this project shows how different LSTM neural network architectures perform at time series forecasting. Extending past the architecture, I also analyse how the data exposed to the models impact the results. By using wavelets in different ways, I also expose the amount of additional information that can be learned through such a transform. This work is underpinned by a research paper titled 'Forecasting stock prices with long-short term memory neural network based on attention mechanism' by Jianyu Qiu, Bin Wang, and Changjun Zhou [2]. While they were claiming to have very impressive results, the paper did little to explain their implementation beyond a high-level architecture overview. The initial challenge of this project was to decipher their publication and produce a report of my own sharing my findings.

Competitive price prediction performance is not valuable unless profits can be realised. To achieve this, the price predictions must be incorporated into a trading strategy which can be done any number of ways. The secondary focus of this project is to produce a trading strategy based on the LSTM outputs and compare it to other trading scenarios. I also intend on extending the research by building a sentiment analysis model such that market sentiment can be included as context for the trading strategy. Once this has been accomplished, I will explore how such technology could be deployed to users.

2 Requirements Specification

2.1 Primary Requirements

- Build an LSTM neural network with an attention mechanism as outlined in the underpinning paper [2].
- Implement the coiflet wavelet transform as used in the paper that underpins this project.
- Evaluate and compare the results between models with and without the attention layer and wavelet transforms.
- Build a sentiment analysis model trained to classify financial news headlines as positive, negative, or neutral.
- Combine the sentiment analysis model with the prediction model in an attempt to improve the performance of the trading strategy.
- Implement a trading strategy based on the predictions.

2.2 Secondary Requirements

- Compare how different models perform in the trading strategy.
- Produce an application that allows the user to build and train LSTM neural networks with attention on any stock price.

3 Context Survey

3.1 Neural Networks for Sequence Prediction Problems

Sequence prediction problems are different to traditional problems solved by neural networks. It refers to a problem whose data holds some temporal dimension such as financial time series or a set of historical weather data. Models that solve this kind of problem can have immense impact on modern society, consider DNA sequence classification for example.

There are four classes of sequence prediction problems which are sequence prediction, sequence classification, sequence generation, and sequence-to-sequence prediction. Sequence generation refers to generating new sequences that have the same features as the seen in the training dataset while sequence-sequence prediction, or seq2seq, is predicting the continuation of a sequence. In this project, I mainly focus on many-to-one sequence prediction for regression and sequence classification for sentiment analysis. Whilst not part of my core research, I also built and tested sequence-to-sequence models but was underwhelmed by it's performance.

In order to measure the performance of each of my neural networks, I have used the following four error metrics:

- Mean Absolute Error (MAE) = $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- Mean Squared Error (MSE) = $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- Root Mean Squared Error (RMSE) = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
- Coefficient of Determination (R2) = $1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$

The following two metrics were used to measure the performance of the sentiment analysis models:

- Categorical Cross-Entropy = $-log(\frac{exp(\hat{y})}{\sum_i^C exp(y_i)})$
- Accuracy = Correct predictions / Total predictions

3.2 LSTM Neural Networks

The LSTM, or Long-Short Term Memory neural network is a kind of Recurrent Neural Network that has can solve time series problems unsolvable by feed-forward networks [3]. This is achieved through a sliding window approach that will be discussed in the design section below. RNNs are models which incorporate 'loops' into the architecture allowing signals to be passed through the network both within and between the layers. In principle, this architecture allows the model to encode temporal context by dynamically affecting weights based on what has already been seen. This design change saw many state-of-the-art results be published for benchmark sequence prediction problems such as text summarisation.

LSTMs were designed to address the issue with training RNNs in which the recurrent nature can lead to weights shrinking or growing exponentially to the point of vanishing and having no effect or exploding and dominating the layer output. LSTM networks solve this through 'memory cells' which each consist of three weights and three gates. The first weight is used to weight input for the current time step as general weights do in multilayer perceptrons while the second weight is applied to the output for the last time step. The final weight represents the internal state of the memory cell and is used in the calculation for the current time step output. The gates are the forget gate, input gate, and output gate which are weighted functions that determine how information gets transmitted through the cell.

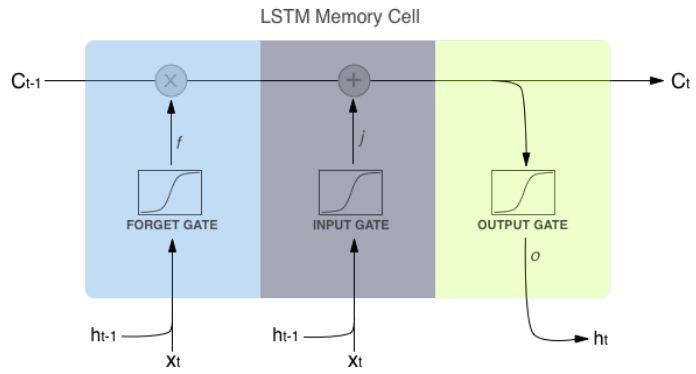

Figure 1: Information flow through an LSTM memory cell [4]

The forget gate calculates the proportion of the cell state from the previous time step to preserve in the cell state of the current time step; a value of 1 represents total preservation and 0 represents total discardment. This is achieved by combining the hidden state of the previous time step, h_{t-1}, with the external input from the current time step, x_t, into the tensor $[h_{t-1}, x_t]$ before applying the function: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) = \sigma(W_f \cdot h_{t-1} + W_f \cdot x_t + b_f)$ where σ is the sigmoid function and W_f, b_f are the gate's weight matrix and bias vector respectively.

The input gate is used to determine how much of the current time step input x_t is encoded into the current cell state, C_t. It applies the same function as the forget gate but with the input gate's weights and biases: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ where W_i, b_i are the weight matrix and bias vector of the input gate's sigmoid operator. Here, i_t represents how important it is for each of the values in the cell state to be updated on a scale between 0 and 1.

Alongside this process, a new candidate vector, $\hat{C}_t$, is calculated to representing how much each value is updated by. This part of the LSTM memory cell has its own weight matrix and bias vector which it applies a tanh activation function to resulting in outputs ranging from -1 to 1: $\hat{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$ where W_c, b_c are the weight matrix and bias vector of the input gate's tanh operator.

Once we have f_t, i_t, and $\hat{C}_t$ from the previous three equations, the cell state for the current time step can be calculated. This is achieved by multiplying the output from the forget gate, f_t, by the cell state from the previous time step, C_{t-1}, to get the information that should persist and then add it to the output from the input layer, i_t, to the candidate vector, $\hat{C}_t$: $C_t = f_t * C_{t-1} + i_t * \hat{C}_t$.

Lastly, we have the output gate which is responsible for producing the value of the next hidden state, h_t. This is done in two steps, the first of which uses a sigmoid activation to obtain weightings for how important it is for the values to persist: $O_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$ where W_o, b_o are the weight matrix and bias vector of the output gate. The final step is to apply the tanh activation function to the current cell state and multiply the result by the newly obtained O_t: $h_t = O_t * tanh(C_t)$.

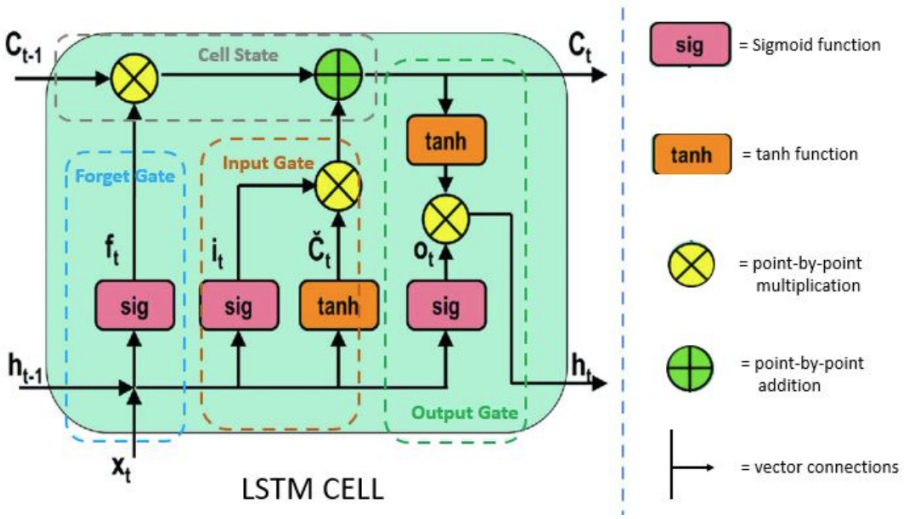

Figure 2: Internal structure of an LSTM memory cell [5]

There are many possible model architectures using LSTM layers of which I will be exploring three. Firstly, the standard LSTM model which only has one LSTM layer and one dense output layer will be used as a benchmark to compare future models to. The second is a stacked model which has two LSTM layers before the output layer. The first of which outputs the hidden state of every time step while the second only outputs it's final hidden state. Lastly, I explore the use of bidirectional LSTMs for sentiment analysis. This architecture involves two parallel recurrent layers that process the data forwards and backwards respectively. Theoretically, we can expect to see better results with this model when attacking natural language problems as sentences reveal context non-linearly.

Unfortunately, LSTMs are not perfect for every scenario. One instance where LSTM models fall short is in sequence prediction models where only the most recent few time steps hold the majority of the correlation with the prediction. This is because LSTMs are designed to learn contextual dependencies which can lead to underperformance when the weightings of the context are incorrectly balanced.

3.3 Backpropagation Through Time

Traditional backpropagation is an algorithm for training neural networks that involves calculating the derivatives of a model's loss function with respect to the network weights and adjusting the weights in order to minimise this loss.

Backpropagation through time (BPTT) is the variant of this algorithm that accounts for temporal structure and as such is used to train recurrent neural networks like those built in this project. BPTT starts by "unrolling" [6] the network such that each input time step has their own copy of the network and a single output. The loss is then calculated individually for each time step and accumulated representing the total model loss. The network is then rolled back up and the weights are updated by calculating the derivatives of the total loss with respect to the weights as in regular backpropagation.

3.4 Wavelet Transform

While Fourier transforms are popular for identifying signal characteristics and discarding noise, they are designed to find features on a global scope such as repeated frequencies. It follows that such a transform wouldn't reveal valuable information about financial time series which is noisy and irregular. A successful alternative is to use wavelet transforms which finds how close different time-localised wave functions match up to parts of the signal. This is achieved by sliding the defined wavelet across the signal and multiplying the frequencies to get a value representative of similarity. Since the frequency, wavelength, and shape can all be varied for these wavelets, we can essentially classify subsets of the signal as specific wavelets. Using these classifications that are represented as coefficients, we can reconstruct a new signal using the identified wavelets that represents the key information in our original signal.

The examples below show the effect of a transform with one layer of decomposition and a transform with three layers of decomposition on volume data respectively.

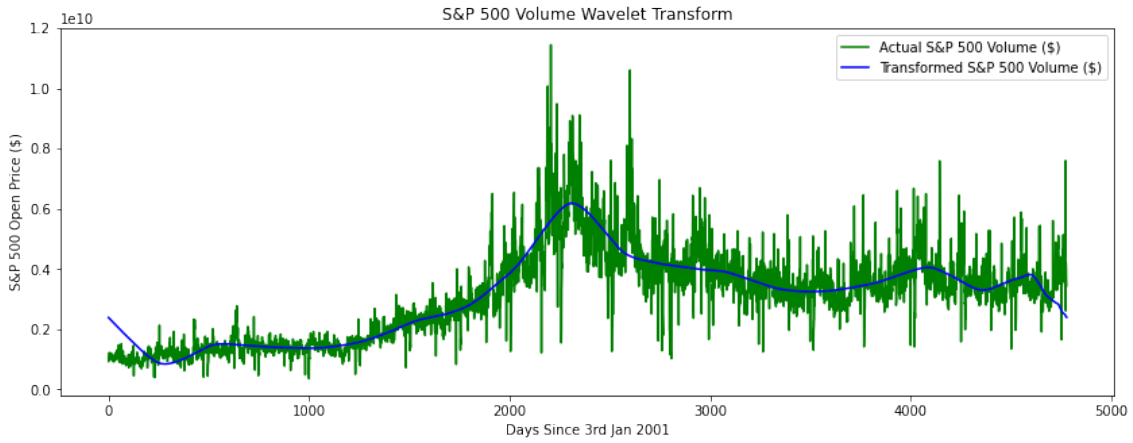

Figure 3: Single layer of decomposition wavelet transform

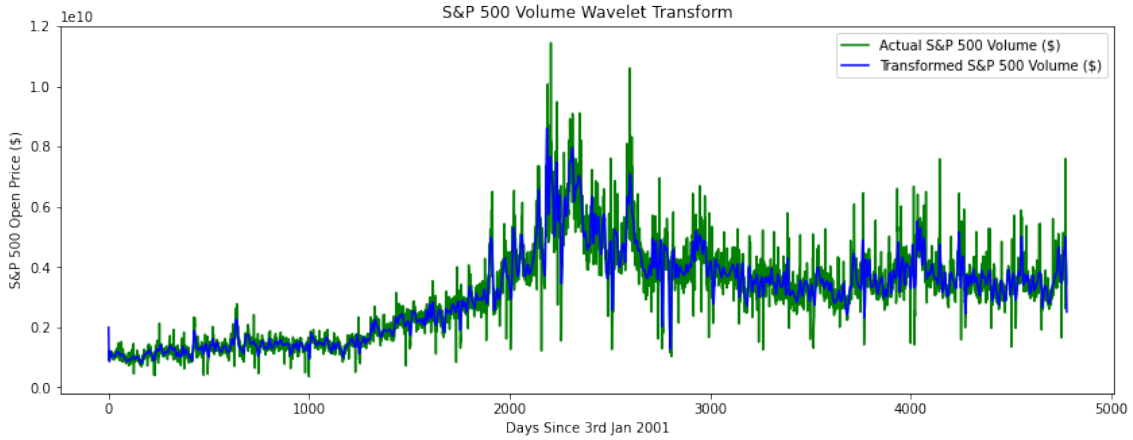

Figure 4: Three layers of decomposition wavelet transform

Another clear advantage that wavelet transforms have over Fourier transforms is that there are different families of wavelets to choose from each with different shapes and features. In this project I use the Coiflet 3 wavelet as used in the paper underpinning this research as they found that it has the highest Signal-to-Noise ratio and the lowest Root Mean Squared Error [2]. Since the shape of this coiflet, as shown below, is sharp, we can expect the transform to learn short-term spikes in the time series.

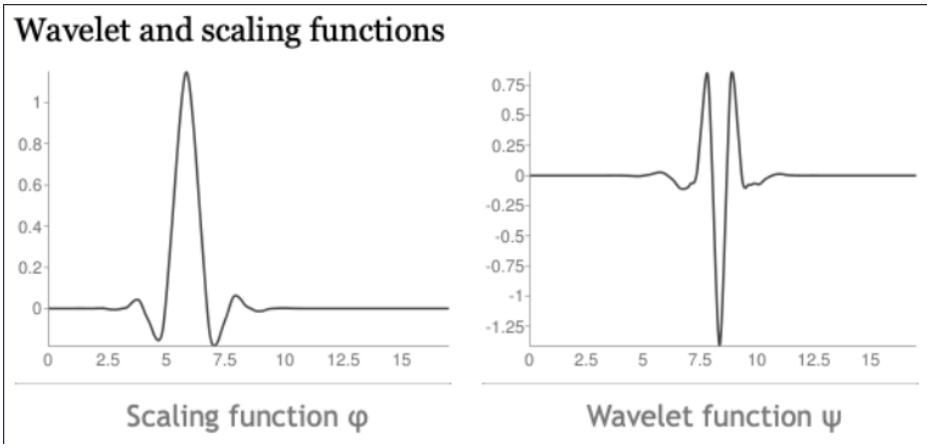

Figure 5: Coiflet 3 Wavelet [7]

3.5 Attention Mechanism

Attention in neural networks are biologically inspired to find and weight the more 'important' parts of the input information. This is achieved by first scoring the correspondence between each input element's hidden layer, $\bar{h}_s$, and the final LSTM hidden layer, h_t, and applying the softmax activation function to this score. Attention was first used by Bahdanau for natural language translation. He used an additive style of scoring whereas we will be using Luong's multiplicative style of scoring, defined as $score(h_t, \bar{h}_s) = h_t^T W_a \bar{h}_s$ where W_a is the learned weight matrix of the scoring function. Applying softmax leaves the attention weights:

- $\alpha_{ts} = softmax(score(h_t, \bar{h}_s)) = \frac{exp(score(h_t, \bar{h}_s))}{\sum_{s'=1}^{S} exp(score(h_t, \bar{h}_s))}$

By taking the sum of each internal hidden layer, $\bar{h}_s$, multiplied by its corresponding attention weight, α_{ts}, we get the context vector c_t:

- $c_t = \sum_s \alpha_{ts} \bar{h}_s$

The values in this vector represent which context from the inputs to pay attention to; essentially their 'importance'. This vector is concatenated with the final hidden state and put through a dense layer with a tanh activation function to receive the final output of the attention layer:

- $a_t = f(c_t, h_t) = tanh(W_c[c_t; h_t])$

where W_c is the learned weight matrix of the final dense layer. This final 'attention vector' is the output of the previous RNN layer augmented to account for information considered 'important'.

I, along with Qiu et al., use a many-to-one attention mechanism as the final layer after the LSTM but before the dense output layer [8].

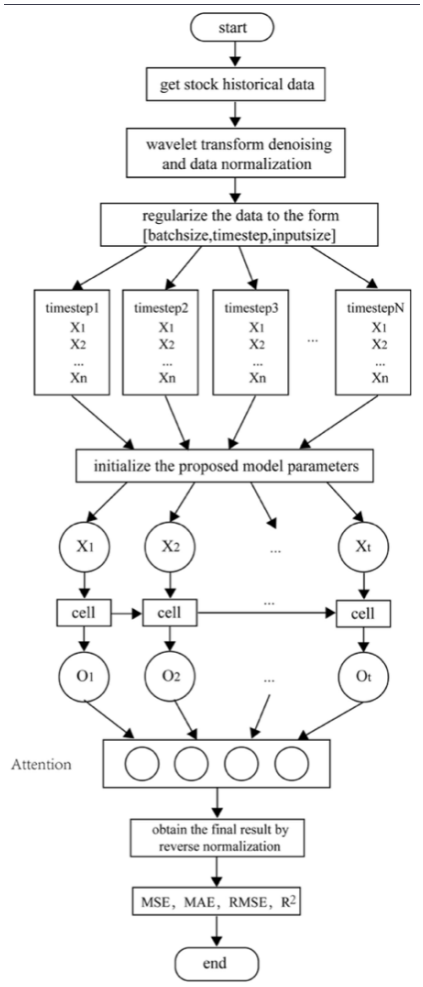

Figure 6: Model Architecture by Qiu et al. [9]

4 Resources, Tools, and Libraries

4.1 Data Sources

All models using SP 500 [10], DJIA [11], and HSI [12] datasets use the exact same data as was used by Qiu et al. [2]. For SP 500 and DJIA, this was from the 3rd January 2000 to 16th May 2019 for training and from the 17th May 2019 to the 1st July 2019 for testing. As for the HSI dataset, training was from the 2nd January 2002 until the 16th May 2019, and testing was on data from 17th May 2019 to the 1st July 2019.

The dataset used to train the sentiment analysis models on financial news headlines was taken from Kaggle [13].

4.2 Development Environment

All models were implemented in Jupyter notebooks which allows you to run Python code in custom orders just by running cells; an extremely useful tool during development. Additionally, Jupyter notebooks save the previous cell outputs as part of the file, as such, each model has its own notebook so that plots and performance metrics are preserved.

4.3 Libraries

Numpy - Adds support for large multi-dimensional tensors along with a number of high-level maths functions. [14]

Pandas - Helps manipulate large, organised, labelled data through their DataFrame object. [15]

Matplotlib Pyplot - Enables figures and plots to be built and displayed in python. [16]

Seaborn - Builds and displays figures but focussed on visualising key information in data. [17]

Keras - An open-source library that implements neural network layers as objects that can easily be compiled together as well as offering optimised implementations training algorithms. [18]

Scikit-Learn - A machine learning library that has useful preprocessing tools such as the StandardScaler and MinMaxScaler. [19]

PyWavelets - Open-source library for implementing various wavelet transforms. [20]

Attention - A third-party implementation of the attention mechanism compatible with Keras. [21]

Gensim - Open-source library of natural language processing tools used for word to vector translations in the sentiment analysis models. [22]

Tkinter - A Python library for producing simple graphical user interfaces. [23]

Yfinance - A third-party library that uses the Yahoo Finance APIs to collect various types of financial data. [24]

5 Design

5.1 Hyperparameters

My project focus is to compare the performance of different LSTM architecture styles and features used by Qiu et al. For this reason, I found a suitable set of hyperparameters for my models and used the same set for each to allow for fair comparisons. The values I decided on were found through trial and error and are by no means an optimal solution.

The hyperparameters include the activation function, size of the sliding window, number of LSTM memory cells, number of attention units, batch size, learning rates, number of training epochs.

5.2 Trading Strategies

In total I produced five trading strategies which were used to compare different model architectures. The first two act as control groups: randomly choosing to either buy or sell each day or buying on the first day and holding. These two results act as benchmarks to see how much the price predictions and sentiment classifications impact the trading performance which is measured in percentage value change over the test set period.

The trading strategy that every regression model uses is based on the price predictions. It will loop through each day of the predictions, y_hat, and calculate the percentage change from the current actual price to the next predicted price. If this percentage change is within a threshold value then it will not try to buy or sell. If the percentage change exceed the threshold then a buy or sell order will be artificially placed and tracked in the direction of the prediction; negative percentage change results in a sell order and vice versa. The threshold value here can be considered a hyperparameter which I improved through trial and error. Another trading algorithm hyperparameter was the bet proportion which represented how much of the current cash balance would be used in buy or sell orders. However, every model performed better when all orders were fulfilled using the maximum available balance on every trade.

The final two strategies follow the same logic as the previous one except use sentiment to improve performance. The first uses the sentiment analysis model to predict the sentiment of each of my staged sentences which represent the market sentiment for 25 days. If the sentiment result aligns with the direction of predicted price percentage change then the action threshold is reduced. By reducing the requirement for the model to make a trade in one direction, its balance is tipped towards either buy or sell depending on the current market sentiment.

5.3 Initial Models

My first set of regression models each are trained to predict the next open price of the SP 500, the Dow Jones Industrial Average, and the Hang Seng Index. The inputs to each model is a tensor in the shape (examples, window size, features) with the features: open, high, low, close, and volume. The idea is that for each prediction, the model will see the following 'window size' number of days and has to predict the following day's open price.

The first models acting as a benchmark for all future results do not use wavelet transforms or an attention mechanism. The data is standardised and then normalised before being used to train a model with a single LSTM layer with 50 units and a single Dense layer with one output.

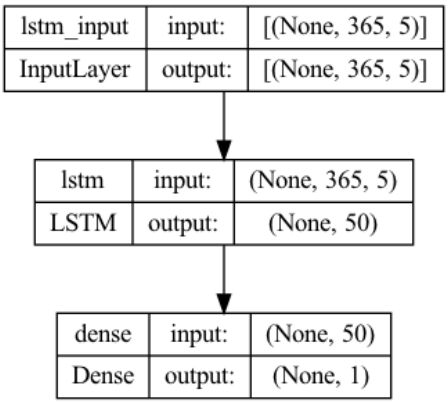

Figure 7: Initial Model Architecture

The next set of models use an identical architecture but make use of wavelet transforms to find signals within the data. Using a Coiflet 3 wavelet transform, a significant amount of noise in the data is removed and the network weights are trained to learn the signal.

I also produced a set of models that implemented the attention mechanism without the wavelet transforms. The attention layer has 16 outputs in all my models as shown in the figure below:

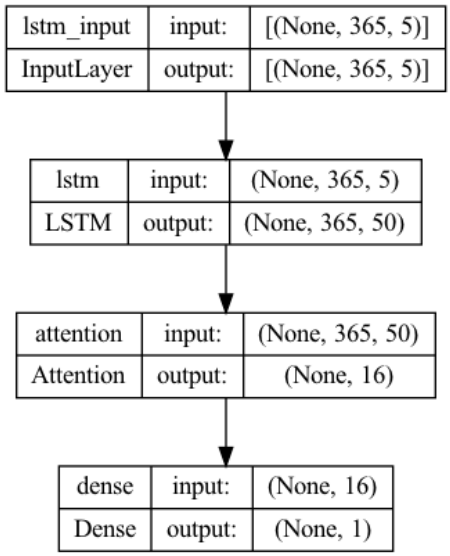

Figure 8: Initial Model Architecture

Notice that the LSTM layer in this architecture passes on data in the shape (365, 50) as opposed to (50) in the previous models. This is because the attention layer requires exposure to the entire sliding window in order to calculate its weights.

I then developed a set of single LSTM layer models that made use of both wavelet transforms and the attention mechanism. The architecture of these models is identical to the previous attention model architecture, however, now the training data is transformed using wavelets after standardisation and normalisation.

After testing each of these implementations I noticed that applying wavelet transforms to both the model inputs and outputs would increase the model performance on paper but was no closer to predicting real unscaled prices. To solve this issue, I rebuilt each of my models that use wavelet transforms to take both the pre- and post- wavelet transformed data as input and the pre-wavelet transformed open prices as output (file titles begin with 'New'). I thought that if the transform alone held less key information than the untransformed data then perhaps it can be used as a supplement rather than a replacement. Of course doubling the number of features increases training time, but also we see improved prediction performance.

5.4 Stacked LSTMs

There are many variants of LSTM network architectures such as bidirectional or generative LSTMs. I decided to build models using the 'stacked LSTM' architecture in which there are multiple hidden LSTM layers; specifically, in my stacked models there are two LSTM layers. I built models with this architecture twice for each dataset; one using the initial wavelet technique and one using the final technique - both using attention.

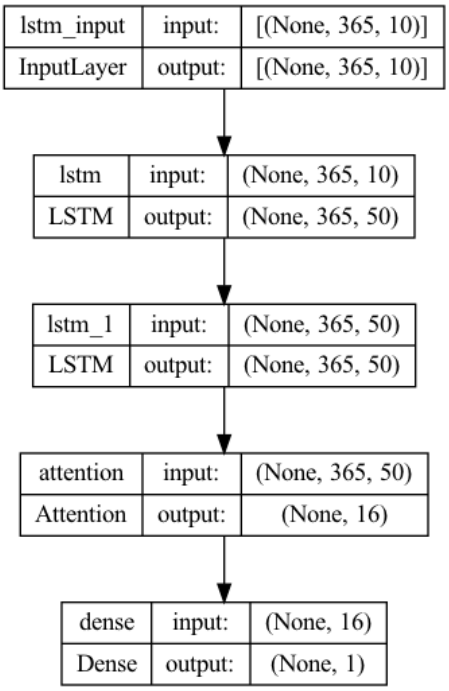

Figure 9: Stacked LSTM Architecture

By introducing more depth into the model, theoretically higher levels of abstraction can be learned as the first LSTM layer produces a representation of the data which is then interpreted by the second layer.

During implementation I also ran tests using Dropout layers to see if by purposefully discarding small amounts of data, the model will not overfit the data and will be able to generalise more accurately. Unfortunately, none of the results showed an increase in validation performance and as such they were not included in the final models.

5.5 Sentiment Analysis

My sentiment analysis model takes in a sentence given in natural language and outputs 'positive', 'neutral', or 'negative'; it is a ternary classifier. This is achieved through a more complex pipeline than in the regression model. We begin by checking the dataset is clean and then encoding each output label as an integer between 0 and 2. The Gensim natural language processing library is then used to build a word to vector (word2vec) model [25]. This model learns vector representations of English words which is critical for the LSTM to be able to learn patterns. In each variant of my sentiment analysis model, the word2vec model uses the same hyperparameters.

Once this is trained, the LSTM input and output series need to be built. To achieve this, the input sentences are split into words and replaced by their vector representations. On top of this, the vector sentences are padded so that sentences of different length can all be processed by the same shape neural network. When the inputs have been built, the outputs are one hot encoded and the training and testing sets are built.

There are four versions of this model to compare: a 'vanilla' LSTM, a 'vanilla' LSTM with attention, a bidirectional LSTM, and a bidirectional LSTM with attention. Each of these are trained for only 10 epochs to avoid overfitting the training set and use a sigmoid activation function in the dense layer to improve multiclass classification accuracy. The idea behind using a bidirectional LSTM is that context within natural language is revealed non-linearly and as such the model needs to be able to put past and future context together to be able to understand sentiment.

5.6 Combination of Models

To combine my regression and classification models into one trading strategy, I began by staging some news headlines. As it would've taken an extreme amount of time to produce a dated set of relevant financial news headlines, I wrote 5 fake headlines with various sentiment that I thought represented the market trend of each 25 day period. Of course this must be taken into account when measuring the validity of the results but it still proves the impact that a successful sentiment analysis model can have on a neural network driven trading strategy. Note that the fake headlines remained in the test set to avoid data leakage.

```
1  The market looks to be going green.,positive
2  Experts debate whether the market price will increase or decrease.,neutral
3  Report states that investors are profiting.,positive
4  Michael James Burry predicts a crash!,negative
5  Prices begin to go up as the market recovers.,positive
```

Figure 10: Staged Sentences

I would expect the combined model to perform better with 5+ dated news headlines for each day as my trading algorithm calculates an average daily sentiment. I designed it this way so that in modern predictions I can use the Yfinance library's *get_news*() method which returns all relevant news headlines posted on Yahoo Finance that day.

5.7 Seq2Seq Model

Peripherally to my core research, I produced a sequence to sequence prediction model that outputs a time distributed vector showing the predicted prices n days into the future. I tested the model at $n = 7$ as anything more seemed highly unreliable. The model design is slightly different to what we have already seen. The model follows an Encoder-Decoder architecture in which a first LSTM layer produces a representation of the inputs which is repeated using a RepeatVector layer. This repeated representation is then interpreted and decoded into a prediction by the second LSTM layer which precedes the final dense output layer which is inside a time distributed wrapper so that the model retains the temporal dimension in the output sequence.

5.8 Magic Window

After building and testing all aforementioned models, I decided to utilise the Yfinance library for its ability to get historical price data for any stock up until the current day. For this I built an applet that lets the user enter a ticker symbol, such as AAPL for Apple Inc, and a price prediction and trading recommendation will be shown.

Below are screenshots of my initial app and my re-design mock-up.

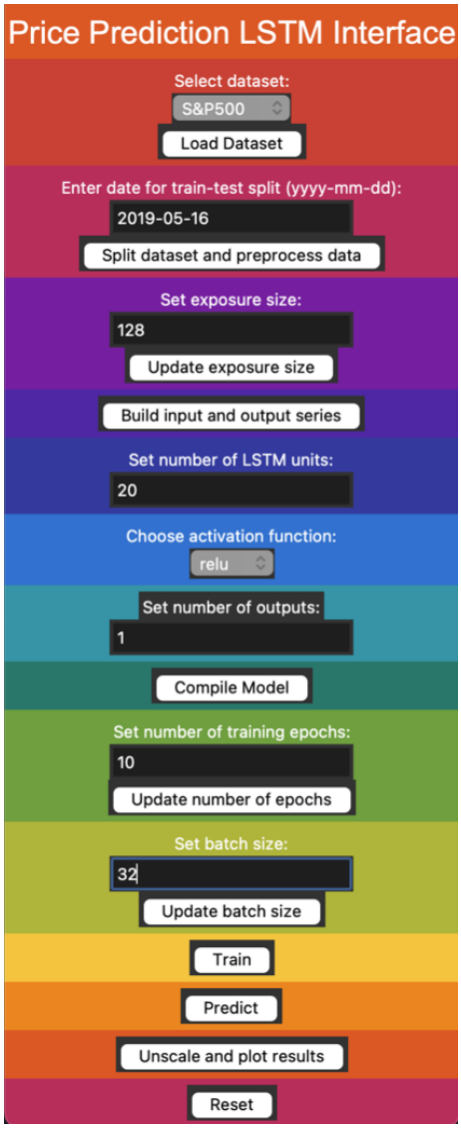

Figure 11: Initial App

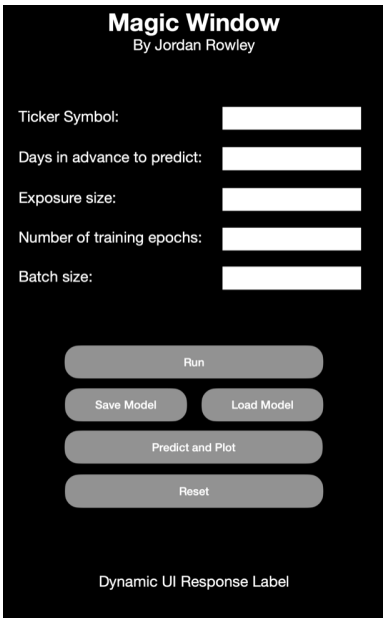

Figure 12: Mock-up of New App

After multiple iterations, the UI design ended up as:

Figure 13: Final App

Other than the ticker symbol I thought it would be important to give the user control over the training parameters. Additionally, the user is able to input the 'days to prediction', a value of 1 means the model will predict the following opening price which a value of 7 means the model will predict next week's opening price. It follows that the larger this value, the less accurate the predictions will be.

The ‘run’ and ‘predict and plot’ buttons were merged as the save/load model functionality was removed and as such they would always be clicked after one another. The save/load model function was removed because the custom attention layer conflicted with existing elements of the Keras library causing issues upon loading.The text label at the bottom updates upon each state update of the app. It will display when the app is reset and will show any errors with how the parameters were entered.

Once the app is run with valid parameters, the label will show the predicted price and a trading recommendation:

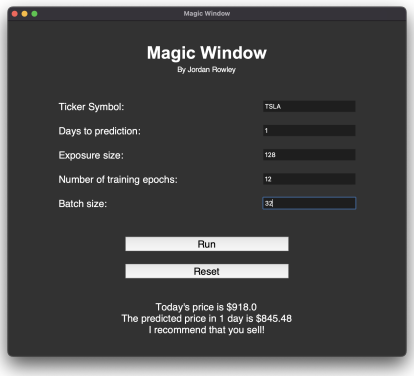

Figure 14: In-app Price Prediction

Four other windows will also now appear, the first of which is a plot of the price predictions over the test set compared to the actual prices. On top of this, two window display plots of the loss (MAE) and validation loss against epochs. Lastly, a window will show the ‘Training Report’ which shows the MAE, MSE, RMSE, and R2 values for both the training set and the testing set.

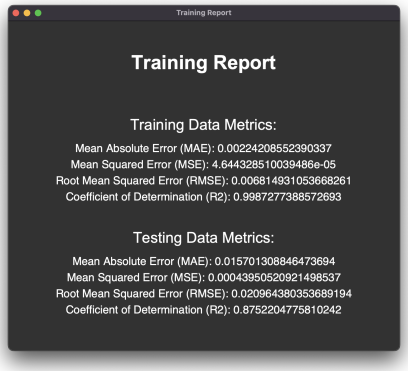

Figure 15: Training Report

6 Implementation

6.1 Trading Strategies

To implement the trading strategies as described above, I begin by defining my algorithm parameters: the initial balance, the action threshold, the sentiment influence, and the bet proportion. I then initialised global variables to track the performance of each competing strategy: balance, number of shares owned, and bet. The balance variable keeps track of how much cash is currently being held while the bet keeps track of the current trade size as it is dependent on the current balance.

To facilitate simulated trades, I implemented a Buy function and a Sell function which will effectively execute a trade at market price and update the bet, balance, and owned shares variables. The special cases are when the shares cannot be afforded in a Buy trade or when not enough shares are owned for a sell trade. In these situations, the maximum valid trade will be executed, i.e. the balance will all be spent on as many shares as possible or all owned shares will be sold.

I also needed a way to score the results of the sentiment analysis model in order for it to interact with the trading algorithm. To realise this, I defined a function that takes in an array of all sentiment labels from the current day. The method iterates through the labels accumulating a tally for positive, neutral, and negative sentiment. Once finished, the score that is output is defined as the number of positive results minus the number of negative results all divided by the total number of results. It follows that the result of this equation represents the polarity of the sentiment as well as a confidence ratio.

The first trading strategy that is implemented is random. Each day there is a 50% chance of making a Buy trade and a 50% chance of making a Sell trade. This acts as one of two control groups. The second control spends its entire balance on shares on day one and sells it all on the final day. These two algorithms act as a benchmark for the results achieved using the price prediction model.

To use the regression model in a trading strategy, I iterated through each price prediction in *y_hat* and calculated the percentage change from the current actual price to the next day's predicted price. If the absolute value of the predicted percentage change is less then than action threshold then the algorithm will not make a buy or sell order. If it does exceed the threshold, then an upwards prediction will trigger a buy trade and a downwards prediction will trigger a sell trade. The action threshold is set to 1% which I found to be an optimal value after some trial and error. The bet proportion is also a value I had to find through trial and error. After evaluating larger, smaller, and dynamic bet sizes, I found that going all in on every trade always produced the best results. This works because the markets I am trading are highly stable and the risk of a large loss is low.

With a 125 day test set period, I used each staged input sentence 25 days in a row to get five periods with distinct market sentiment. I organised the staged news headlines as such so that the labels align with the general trend of that time period. In a modern situation, we could use Yfinance's *get_news* method which returns an array of relevant news headlines from the current day. This way our daily sentiment scores would be much more accurate and would likely have a stronger positive effect on the trading algorithm's performance.

The new trading strategy that uses sentiment multiplies the sentiment score by a sentiment influence value for which I found an optimal value to be 0.5%. This value represents by how much the action threshold is reduced given a perfectly positive or negative sentiment score. The logic behind this is that there should be a smaller threshold to trade a price prediction if it is backed up by the current market sentiment shared in the news.

For comparison, I duplicated this new trading strategy such that one uses the predicted labels of the staged news headlines and one uses the actual labels of the staged news headlines. I did this because the sentiment analysis model can only correctly classify around 70% of input sentences.

6.2 Initial Models

The implementation for my initial models begin by loading the relevant dataset into a Pandas DataFrame and checking for null or NaN values. If the dataset is deemed 'clean' then it is split into training and testing sets at the 16th May 2019 to match Qiu et al.

To preprocess the price time series I used the StandardScaler and MinMaxScaler objects from scikit-learn to standardise and then normalise each column such that they have a mean of zero and standard deviation of 1 before being shifted and rescaled to range between 0 and 1. This is done so that the regression model is focused on learning the signal pattern.

After preprocessing, the exposure size is set which represents the number of days that will be shown to the model before each prediction. Now we can build the input series and corresponding outputs. This is done so that each input is the normalised open, high, low, and close price along with the volume for the previous number of days determined by the exposure size and the output is the next unseen day's open price.

Now that the training and testing sets have both been split into inputs and outputs, we can define our model. Using the Keras library, defining a neural network is fairly straight forwards. By creating a Sequential object, we can then individually add Keras Layer objects to it defining the shape of the neural network. Each of my initial models begin with an LSTM layer with 50 memory cells and an input shape of (batch size, exposure size, number of features). Since these initial models are supposed to act as a benchmark for future models, this single LSTM layer feeds into a single output dense layer to conclude the model architecture. The model is then compiled with the Adam optimiser with an initial learning rate of 0.01, using MAE as the loss to minimise. I also supplied the model with the additional metrics of MSE, RMSE, and R2 [26].

Once the model has been compiled we can begin training. By varying the learning rate during the training process, the gradient descent algorithm is able to get closer and closer to the discovered loss function minima. As such, I trained all my models for 16 epochs with a 0.01 learning rate, then 12 epochs with a 0.001 learning rate, and finally 8 epochs with a 0.0001 learning rate.

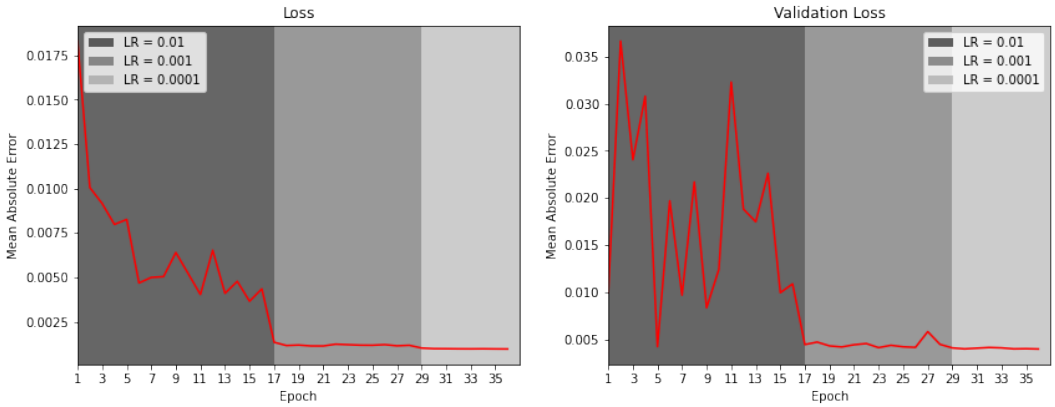

Figure 16: Training History

After training we can pass our test set into the model to get predictions for each day. Before plotting the predictions against the actual prices to compare, we must undo our preprocessing by applying the reverse of the normalisation function and then the reverse of the standardisation function:

```
# Undo normalisation
y_test = (y_test * anotherScaler.data_range_[0]) + anotherScaler.data_min_[0]
y_hat = (y_hat * anotherScaler.data_range_[0]) + anotherScaler.data_min_[0]

# Undo standardisation
y_test = (y_test * scaler.scale_[0]) + scaler.mean_[0]
y_hat = (y_hat * scaler.scale_[0]) + scaler.mean_[0]
```

Figure 17: Reverse Scaling

The unscaled predictions and actual prices are then plot before the performance metrics are printed out for both the training and testing set and the trading strategies are run as outlined ealier.

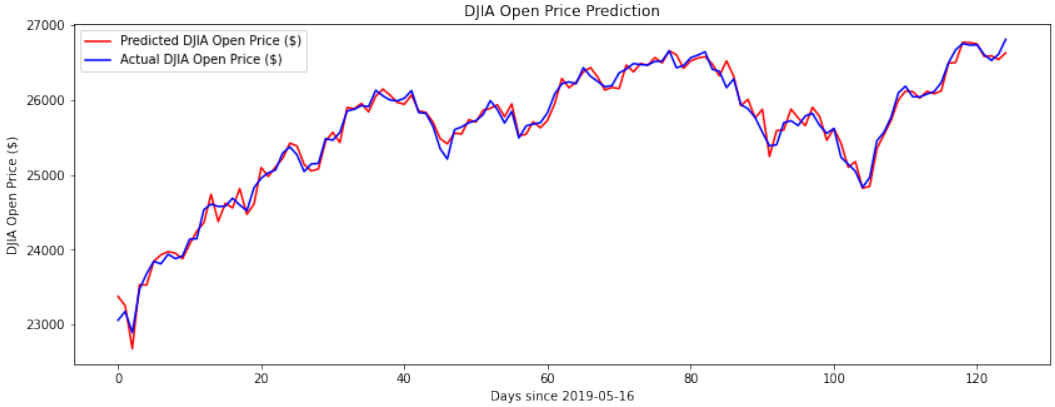

Figure 18: Prediction Plot

6.3 Wavelet Transform

To implement the wavelet transform, I augmented a function for removing high-frequency noise using discrete wavelet transforms from a guide [27]. I changed the function to transform each column of a data frame individually and to take the levels of decomposition as a parameter. Here, PyWavelets is used to perform the wavelet functions such as decomposition and recomposition.

```
def WaveletTransform(data, levels, threshold=0.63, wavelet='coif3'):
    reconstructedData = pd.DataFrame()
    for i in range(data.shape[1]):
        threshold = threshold * np.nanmax(data[:,i])
        coefficients = pywt.wavedec(data[:,i], wavelet, mode='per', level=levels)
        coefficients[1:] = (pywt.threshold(i, value=threshold, mode='soft') for i in coefficients[1:])
        reconstructedColumn = pywt.waverec(coefficients, wavelet, mode='per')
        reconstructedData = pd.concat([reconstructedData, pd.DataFrame(reconstructedColumn)], axis=1)
    return reconstructedData
```

Figure 19: Wavelet Transform Method

The first set of models I built using wavelet transforms only apply them to the inputs of the training set. This is because signals that have been recomposed from wavelet coefficients have spikes at the head and tail leading to poor performance in these ranges.

The second set of models I built that use wavelet transforms apply them to the inputs of both the training and testing set. This time however, the model inputs contain all five columns of the data both before the wavelet transform and after the wavelet transform leaving 10 input features. The idea behind this being that the signal reconstructed from wavelets will add additional information about the time series rather than replace it.

6.4 Attention Mechanism

The attention mechanism that I use in my models is was implemented as a class inheriting from the Keras Layer object so that the attention can be added to the models exactly how any other layer would be [8]. Since the attention mechanism requires all hidden states to build the context vector, any LSTM layer preceding an attention layer must have the return_sequences parameter set to true which means that the layer will return all of its hidden states.

6.5 Sentiment Analysis Model

The sentiment analysis model starts by loading and cleaning a dataset just as before. The initial preprocessing applied to the sentences in the dataset is the simple_preprocess method from the Gensim library [28]. This puts all the characters into lowercase and converts the sentences into arrays of tokens. Then, the text labels of 'positive', 'neutral', and 'negative' are mapped to the integers 0, 1, and 2.

Now we must translate the words to vectors so that we may build our input and output series for the LSTM. Word to vector (word2vec) translation is a highly researched field of machine learning filled with nuance. As it is not the focus of this project, the word2vec model was implemented using a preset architecture from the Gensim natural language processing library and is by no means optimal. It is trained on all the sentences from our dataset over 100 epochs.

Once the word to vector mappings have been learned, we can build our input and output series by replacing each word in each sentence by it's learned vector representation. Each input sequence of word vectors are then padded with zeros to account for sentence length variability in the fixed-shape LSTM. Finally, the integer sentiment labels are one hot encoded such that the model outputs a confidence score for each of the three classes. Now the inputs and outputs can be split into training and testing sets before the model is defined and compiled.

For this project I have implemented four different LSTM classifiers for sentiment analysis each with a slightly different architecture. The first of which is simply an LSTM layer with 32 memory cells and a sigmoid dense layer with 3 outputs while the second architecture is the same but with an attention layer. The final two models use a bidirectional LSTM layer, one with attention and one without. An LSTM layer can become bidirectional using the Keras Bidirectional wrapper around the layer.

All sentiment analysis LSTMs were trained for 10 epochs using the Adam optimiser with the learning rate fixed to 0.01. Once training is complete we can predict on the test set, decode the predictions, and evaluate our model.

```
# Decode sentiment predictions
for i in range(len(y_hat)):
    for j in range(len(y_hat[i])):
        yHatAbsolute[i, j] = round(y_hat[i, j])
    if (yHatAbsolute[i, 0] == 1):
        yHatText.append("positive")
    elif (yHatAbsolute[i, 1] == 1):
        yHatText.append("negative")
    elif (yHatAbsolute[i, 2] == 1):
        yHatText.append("neutral")
```

Figure 20: Prediction Decoding

6.6 Combination of Models

In the file that combines the price prediction regression LSTM and the sentiment analysis classification LSTM into one trading strategy, there are no new implementation details to note. It starts off by building and training the sentiment model then building and training the price prediction model. The predictions from each are then made and the trading strategies described above are run.

6.7 Seq2Seq Model

My sequence to sequence (seq2seq) model is the first model I built that uses the yfinance library to get historical price data up to and including the current day. This is achieved using the library's Ticker object which has a history attribute containing all historical data in Yahoo Finance's database.

Here, the input series are built by concatenating the standardised and normalised data with the wavelet transformed data. The output targets however, are the opening price n days ahead of the final input date. This trains the model to predict days into the future. It follows that n is inversely proportional to the model performance.

My sequence to sequence model follows an Encoder-Decoder architecture. Here, the first LSTM builds a fixed-length representation of the data which is then repeated to fit the second LSTM input which acts as a decoder. The decoder LSTM returns all the hidden states which are passed into a single-output dense layer in a TimeDistributed wrapper. This wrapper ensures that the same output layer is used for each time step.

Unfortunately, every permutation of hyperparameters for this model resulted in a negative coefficient of determination (R2) on the test set. This means that there is no statistical correlation between the learned network weights and the expected output. In other words, the predictions have no validity. On the positive side, building this model taught me how to use the Yfinance library to access data through Yahoo Finance APIs.

6.8 Magic Window

As an addition to my research recreating and evaluating the model from Qiu et al., I implemented an application that allows users to build, train, and use their own LSTM neural networks based on my best performing architecture. The graphical user interface is built using the Tkinter library which allows objects such as Labels and Buttons to be defined and organised into windows.

When the user enters valid parameter values and clicks run, the UpdateVariables method is called which gets the values from each of the entry fields and updates the global variables. Before any preprocessing begins, the application will check if all the parameters have been entered and, if not, will alert the user accordingly. If successful, the historical data of the stock of choice is loaded from Yahoo Finance and split into training and testing sets. The data is then standardised, normalised, and wavelet transformed before the input and output series are built. In this case, the inputs are the concatenated pre- and post-wavelet transform historical financial data giving the model 10 input features. The corresponding output is the normalised open price x number of days into the future where x is defined by the user in the interface.

The model architecture that the application uses is a single LSTM layer with 50 memory cells, an attention layer with 16 units, and finally a single-output dense layer. I selected this architecture as it showed to have the best balance between maximum performance and consistency of results.

The training procedure varies the learning rate as it did in the previous models, however, as the number of epochs are determined by the user, the three training stints are split evenly across the epochs.

Once training is complete, the training report and plots are displayed. Unlike the plots in the other models, these plots each appear in their own windows and have a set of tools for 'exploring' the plot. The user can both scale to examine and save different parts of the plot using matplotlib's backend object NavigationToolbar2Tk.

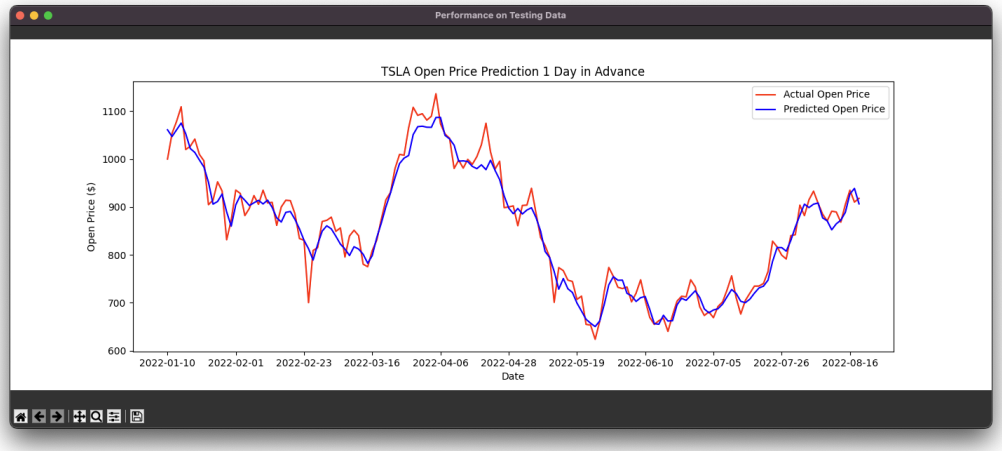

Figure 21: Tesla Price Prediction from Magic Window App

7 Experimental Results

7.1 Price Prediction Regression

The percentage profit column shows the amount earned over the test set by trading the price predictions. For the first two datasets, the test set spans a period of 125 days while for HSI, this period is 119 days.

S&P 500	MAE	MSE	RMSE	R^2	% Profit
Initial Model	0.003954	2.688E-05	0.005185	0.9319	24.78
Wavelet Model	0.007190	7.766E-05	0.008812	0.8035	26.39
New Wavelet Model	0.003873	3.035E-05	0.005509	0.9288	24.03
Attention Model	0.004549	3.219E-05	0.005673	0.9171	23.77
Wavelet and Attention Model	0.005775	5.256E-05	0.007236	0.8656	25.32
New Wavelet and Attention Model	0.05055	3.971E-05	0.006302	0.9005	26.83
Stacked Model	0.006714	7.164E-05	0.008464	0.8109	26.09
New Stacked Model	0.004349	3.197E-05	0.005638	0.9222	24.03

DJIA	MAE	MSE	RMSE	R^2	% Profit
Initial Model	0.004132	2.766E-05	0.005295	0.9296	25.12
Wavelet Model	0.007889	9.624E-05	0.009810	0.7467	23.56
New Wavelet Model	0.003959	2.993E-05	0.005471	0.9301	24.25
Attention Model	0.004383	2.932E-05	0.005414	0.9232	25.12
Wavelet and Attention Model	0.005628	4.703E-05	0.006858	0.8696	24.29
New Wavelet and Attention Model	0.004425	3.350E-05	0.005788	0.9200	25.73
Stacked Model	0.008362	0.0001092	0.01045	0.6957	25.32
New Stacked Model	0.004817	3.787E-05	0.006154	0.9061	20.35

HSI	MAE	MSE	RMSE	R^2	% Profit
Initial Model	0.005583	6.015E-05	0.007756	0.9118	18.33
Wavelet Model	0.01013	0.0001578	0.01256	0.7684	27.34
New Wavelet Model	0.005495	5.174E-05	0.007193	0.9171	24.38
Attention Model	0.005932	6.628E-05	0.008141	0.9001	16.24
Wavelet and Attention Model	0.009283	0.0001366	0.01169	0.8039	14.75
New Wavelet and Attention Model	0.005895	5.727E-05	0.007568	0.9080	24.57
Stacked Model	0.009902	0.0001670	0.01292	0.7336	12.75
New Stacked Model	0.005660	5.467E-05	0.007394	0.9153	23.67

7.2 Sentiment Analysis Classification

	Training Categorical Cross-Entropy	Testing Categorical Cross-Entropy	Training Accuracy	Testing Accuracy
Initial Model	0.8072	0.9019	0.6477	0.5915
Attention Model	0.2746	0.9778	0.8620	0.7077
Bidirectional Model	0.3851	0.9877	0.8252	0.6342
Bidirectional Attention Model	0.2581	0.9564	0.8693	0.7128

7.3 Combined Model for Trading

SP 500 only due to time limitations staging news headlines:

```
------------------------------------------------------------------
Trading over 125 days
Initial Balance: $1000000
------------------------------------------------------------------
Random Trading Strategy
Profit: $151609.9973341804
Balance Percentage Change: 15.16099973341804%
------------------------------------------------------------------
Buying on day one and holding
Balance Percentage Change: 19.96196784708542%
------------------------------------------------------------------
Trading strategy based on price predictions
Profit: $251505.31948059984
Balance Percentage Change: 25.150531948059985%
------------------------------------------------------------------
Trading based on price predictions and sentiment predictions
Profit: $252164.50822731666
Balance Percentage Change: 25.216450822731666%
------------------------------------------------------------------
Trading strategy based on price predictions and sentiment label
Profit: $284077.6311124705
Balance Percentage Change: 28.407763111247053%
------------------------------------------------------------------
```

Figure 22: Trading Strategy Results

8 Evaluation and Critical Appraisal

The coefficient of determination, R2, represents the proportion of the variation in the dependent variable (open price) that is predictable from the independent variables (network weights) [29]. For this reason, we will use it as the primary metric for comparing the regression models. In the paper underpinning this report [2], the highest R2 reported for each dataset are as follows:

- SP 500 - 0.9470
- DJIA - 0.9621
- HSI - 0.8783

Compared to my results:

- SP 500 - 0.9319
- DJIA - 0.9301
- HSI - 0.9171

We can see that the Chinese research team were able to achieve slightly better results on the SP 500 and DJIA datasets while I was able to improve the coefficient of determination on the HSI dataset. One reason for this is that their models were trained for 600 epochs contrasting to my 36 total epochs.

From my experimental results we can also see that the 'New Wavelet' models were the highest performing. These models passed both the pre- and post-wavelet transform data into a model with a single 50 unit LSTM layer and no attention. Since this pipeline proved to be the most consistent, it is also the fundamental design for the Magic Window app.

Interestingly, by only showing a model wavelet transformed data, it is able to learn general trends but cannot accurately predict day by day price changes although we see an anomalous result on the HSI dataset. After seeing the disappointing trading results on these models, I decided to concatenate the pre-and post-wavelet transform data to maximise the information that can be learned about the time series. I theorised that maybe the noise in the time series helps the model to generalise to unseen data.

Another conclusion to be made is that the attention mechanism did not improve the results of my regression models but did make significant enhancements to the sentiment analysis models. Perhaps this is a consequence of the specific implementation but it also appears consistent with the fact that attention mechanisms were designed for natural language processing rather than time series analysis. Furthermore, the sentiment models were also improved by switching to a Bidirectional architecture and the best performing model was indeed a bidirectional LSTM with attention boasting an accuracy of 71.28%.

As part of this project I have built various LSTM neural networks with and without attention mechanisms in an attempt to recreate the results from Qiu et al. I also implemented wavelet transforms which I used in two different ways in an attempt to improve upon the existing model. Additionally, I built a sentiment analysis model which I used in conjunction with my price prediction regression model to improve the performance of a custom trading strategy. It follows that I have satisfied each of the primary requirements outlined in the requirements specification of the document.

To test the deployability of the researched models, I decided to also build a application with a simple user interface that can build, train, and use LSTM neural networks for price prediction. This satisfies the secondary requirements of this project.

When collecting my experimental results, since I was evaluating the performance between architectures, I kept all hyperparameters constant. These parameters were found through a period of trial and error and must be consistent across the models for a fair comparison. Additionally, the training procedure was also uniform to minimise external influence on the model performance.

To further improve the validity of my experimental results, I could have re-run all the models multiple times and taken an average of the performance metrics. I would like to have done this but did not have the time or computational resources to complete before the deadline.

9 Conclusions

To conclude, I reproduced the results obtained by Qiu et al. by decoding their research paper and learning how to build the described models myself. I also extended their research by producing a trading strategy, a sentiment analysis model, and combining the two. To test the deployability of my discovered implementation I built Magic Window, the user-facing application enabling both easy trading recommendations and machine learning research.

The most exciting outcome of this project is proof that a sentiment analysis model can improve the performance of a trading algorithm that uses price prediction regression models. This was an extension to the research published by Qiu et al. From Figure 22 we can see that, given daily sentiment labels, the trading performance improved by up to 3.5% over the test set period. Unfortunately, the results were not as impressive when using the labels predicted by the sentiment analysis model. Although we still see an improvement, the classification accuracy is not high enough to consistently contextualise market sentiment.

Given more time and resources, there are a few directions I would like to take this research. The first of which would be to design and implement a state-of-the-art sentiment analysis model to run on daily news headlines pulled from Yahoo Finance. I think that this would significantly improve the trading performance beyond what we have seen in this work. Perhaps extending the context of the model could also improve performance, by exposing it to tweets through the Twitter API for example.

Another research direction I propose is to build another classification model that outputs Buy, Sell, or Hold for each time step based on the price prediction and sentiment score. It would be interesting to see how a model like this would compete with the current algorithm based on an action threshold. I tried implementing such a model but reached a roadblock when trying to design a custom loss function representing profits.

I also think that this research can be applied to more than just mainstream stock price data. In practice, the models I have designed can be applied to any time series prediction problem from ECG analysis to modelling population growth.

10 Appendix 1 - Testing Summary

For each regression model:

- The MAE, MSE, RMSE, and R2 was measured for both the training and testing data.
- The loss and validation loss are then both plot with respect to epochs to visualise the model learning.
- The price predictions over the test set are then passed into the trading algorithm which outputs how it would've performed over that period.

For each classification model:

- The categorical cross-entropy and classification accuracy are measured for training and testing sets.

11 Appendix 2 - User Manual

11.1 Prerequisites

Python 3.10 - https://www.python.org/downloads/release/python-3106/

11.2 Installing

Install the source code either from MMS or from the GitHub repository [https://github.com/Jynxer/Stock-Price-LSTM-RNN-with-Attention] then install the packages listed in the 'requirements.txt' file:

```
$ pip3 install -r requirements.txt
```

11.3 Executing

All Jupyter notebooks have already been run and have saved outputs for each cell. Should you wish to run these again, simply run all of the cells in order of appearance.

To run the Magic Window application, run the following command from a terminal:

```
$ python3 .../MagicWindow.py
```

References

[1] ALAIN P. CHABOUD et al. "Rise of the Machines: Algorithmic Trading in the Foreign Exchange Market". In: *The Journal of Finance* 69 (Sept. 2014), pp. 2045–2084. DOI: 10.1111/jofi.12186.

[2] Jiayu Qiu, Bin Wang, and Changjun Zhou. "Forecasting stock prices with long-short term memory neural network based on attention mechanism". In: *PLOS ONE* 15 (Jan. 2020). Ed. by Tao Song, e0227222. DOI: 10.1371/journal.pone.0227222.

[3] Felix A. Gers, Douglas Eck, and Jürgen Schmidhuber. "Applying LSTM to Time Series Predictable through Time-Window Approaches". In: *Artificial Neural Networks — ICANN 2001* (2001), pp. 669–676. DOI: 10.1007/3-540-44668-0_93. (Visited on 05/09/2021).

[4] Divyanshu Thakur. *LSTM and its equations*. Medium, July 2018. URL: https://medium.com/@divyanshu132/lstm-and-its-equations-5ee9246d04af.

[5] Kamilya Smagulova and Alex Pappachen James. "Overview of Long Short-Term Memory Neural Networks". In: *Modeling and Optimization in Science and Technologies* (Apr. 2019), pp. 139–153. DOI: 10.1007/978-3-030-14524-8_11. URL: https://link.springer.com/chapter/10.1007%2F978-3-030-14524-8_11 (visited on 05/03/2021).

[6] Jason Brownlee. *A Gentle Introduction to RNN Unrolling*. Machine Learning Mastery, Sept. 2017. URL: https://machinelearningmastery.com/rnn-unrolling/.

[7] *Coiflets 3 wavelet (coif3) properties, filters and functions - Wavelet Properties Browser*. wavelets.pybytes.com. URL: http://wavelets.pybytes.com/wavelet/coif3/ (visited on 08/19/2022).

[8] Philippe Rémy. *Keras Attention Mechanism*. GitHub, Aug. 2022. URL: https://github.com/philipperemy/keras-attention-mechanism (visited on 08/19/2022).

[9] "Qiu et al. Model Architecture Diagram". In: (). DOI: 10.1371/journal.pone.0227222.g003. (Visited on 08/19/2022).

[10] *SP 500 (^{S}PX)HistoricalData – YahooFinance*. finance.yahoo.com. URL: https://finance.yahoo.com/quote/%5ESPX/history/.

[11] *Dow Jones Industrial Average (^{D}JI)HistoricalData – YahooFinance*. @YahooFinance, 2019. URL: https://finance.yahoo.com/quote/%5EDJI/history?p=%5EDJI.

[12] *HANG SENG INDEX (^{H}SI)HistoricalData – YahooFinance*. finance.yahoo.com. URL: https://finance.yahoo.com/quote/%5EHSI/history?p=%5EHSI.

[13] *Financial Sentiment Analysis*. www.kaggle.com. URL: https://www.kaggle.com/datasets/sbhatti/financial-sentiment-analysis (visited on 08/19/2022).

[14] *NumPy — NumPy*. Numpy.org, 2009. URL: https://numpy.org.

[15] *Python Data Analysis Library — pandas: Python Data Analysis Library*. Pydata.org, 2019. URL: https://pandas.pydata.org.

[16] *matplotlib.pyplot — Matplotlib 3.4.3 documentation*. matplotlib.org. URL: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html.

[17] *seaborn: statistical data visualization — seaborn 0.9.0 documentation*. Pydata.org, 2012. URL: https://seaborn.pydata.org.

[18] *Home - Keras Documentation*. Keras.io, 2019. URL: https://keras.io.

[19] scikit-learn. *scikit-learn: machine learning in Python*. Scikit-learn.org, 2019. URL: https://scikit-learn.org/stable/.

[20] Filip Wasilewski. *PyWavelets - Wavelet Transforms in Python — PyWavelets Documentation*. pywavelets.readthedocs.io. URL: https://pywavelets.readthedocs.io/en/latest/.

[21] Philippe Rémy. *Keras Attention Mechanism Class*. GitHub, Aug. 2022. URL: https://github.com/philipperemy/keras-attention-mechanism/blob/master/attention/attention.py (visited on 08/19/2022).

[22] *gensim: topic modelling for humans*. radimrehurek.com. URL: https://radimrehurek.com/gensim/.

[23] Python Software Foundation. *tkinter — Python interface to Tcl/Tk — Python 3.7.2 documentation*. python.org, 2019. URL: https://docs.python.org/3/library/tkinter.html.

[24] *Yfinance: Yahoo! Finance market data downloader*. GitHub. URL: https://github.com/ranaroussi/yfinance.

[25] *Word2Vec Models*. radimrehurek.com. URL: https://radimrehurek.com/gensim/models/word2vec.html.

[26] *Custom R2 Metric Function in Keras*. jmlb.github.io. URL: https://jmlb.github.io/ml/2017/03/20/CoeffDetermination_CustomMetric4Keras/ (visited on 08/19/2022).

[27] admin. *A guide for using the Wavelet Transform in Machine Learning*. ML Fundamentals, Dec. 2018. URL: https://ataspinar.com/2018/12/21/a-guide-for-using-the-wavelet-transform-in-machine-learning/.

[28] *gensim.utils.simple$_p$reprocess*(). tedboy.github.io. URL: https://tedboy.github.io/nlps/generated/generated/gensim.utils.simple_preprocess.html (visited on 08/19/2022).

[29] Wikipedia Contributors. *Coefficient of determination*. Wikipedia, Feb. 2019. URL: https://en.wikipedia.org/wiki/Coefficient_of_determination.