# Transformers: Time-Series in Disguise

Edward Celella
*2118317*
Supervisor: Dr Shan He

School of Computer Science, University of Birmingham,
Birmingham B15 2TT, UK
emc918@student.bham.ac.uk

**Abstract.** In recent years transformer networks have shown their prowess in the field of natural language, becoming the new state of the art model. This paper outlines a method in which transformers can be repurposed for the use of time-series forecasting. Specifically, the network is applied to the task of stock market prediction. The results obtained from this project show that for the task of regression, transformers have the ability to produce accurate forecasts. These results are further improved through the use of technical analysis methods, which provide different representations of the data. In addition to this, the transformer model is also applied to the task of generating trading signals, in an effort to combine two systems which usually remain separate. The result of this still remains inconclusive, with results obtained in this study, suggesting transformers have a poor performance when applied in this way.

**Keywords:** Transformers · Stock Prediction · Technical Analysis · Time-Series

⋆ *Thesis submitted to the University of Birmingham in partial fulfilment for the degree MSc in Advanced Computer Science (September 2020).*

# Table of Contents


Table of Contents . . . . . i

List of Figures . . . . . iii

List of Tables . . . . . iv

Acknowledgements . . . . . v

1 Introduction . . . . . 1

2 Literature Review . . . . . 3

2.1 Sequence to Sequence Architectures . . . . . 3

2.1.1 Encoder-Decoder Models . . . . . 3

2.1.2 Attention . . . . . 4

2.2 Transformers . . . . . 6

2.3 Time-Series Forecasting . . . . . 8

2.3.1 Convolution Neural Networks . . . . . 9

2.3.2 Recurrent Neural Networks . . . . . 9

2.3.3 Attention Mechanisms . . . . . 10

2.3.4 Other Models . . . . . 11

2.4 Financial Forecasting . . . . . 11

2.4.1 Moving Averages . . . . . 12

2.4.2 Chart Patterns . . . . . 13

3 Methodology . . . . . 15

3.1 Embedding Systems . . . . . 16

3.2 Output Layer . . . . . 17
3.3 Data sets . . . . . 18
3.4 Loss Function . . . . . 19
4 Implementation . . . . . 20
4.1 Transformer . . . . . 20
4.2 Embedding Systems . . . . . 21
4.3 Recursive Neural Network . . . . . 22
4.4 Hardware . . . . . 23
5 Results and Analysis . . . . . 24
5.1 Regression . . . . . 24
5.2 Classification . . . . . 26
5.3 Comparison to a Recursive Neural Network . . . . . 28
6 Concluding Remarks and Discussion . . . . . 32
7 Future Work . . . . . 34
8 Reflection . . . . . 35
Bibliography . . . . . 35
A Chart Patterns . . . . . 39
B Running the Project . . . . . 41

# List of Figures

1 Transformer architecture proposed by Vaswani et al. (2017) . . . 7

2 Head and shoulders chart pattern Schwager (1999) . . . . . 13

3 Moving averages produced from S&P 500 data. Green line N=20, red line N=100. . . . . 21

4 Extrema detected on S&P 500 data. Area around day 25 shows a head and shoulders pattern. . . . . 22

5 S&P 500 regression validation loss for a five day prediction using normalised price data for each epoch. . . . . 24

6 S&P 500 regression validation loss for a five day prediction using normalised price data for each epoch. . . . . 25

7 S&P 500 regression validation loss for a five day prediction using normalised weighted moving average data for each epoch. . . . . 26

8 S&P 500 regression validation loss for a five day prediction using normalised weighted moving average data for each epoch. . . . . 27

9 S&P 500 classification training loss for a five day prediction using normalised weighted moving average data for each epoch. . 28

10 S&P 500 classification validation loss for a five day prediction using normalised weighted moving average data for each epoch. . 29

11 S&P 500 classification training loss for a five day prediction using chart price patterns for each epoch. . . . . 30

12 S&P 500 classification validation loss for a five day prediction using chart price patterns for each epoch. . . . . 31

# List of Tables

1 Test losses obtained for the transformer and RNN networks for recursive tasks. . . . . 29

2 Test losses obtained for the transformer and RNN networks for classification tasks. . . . . 30

## Acknowledgements

I would like to thank Dr Shan He, for his help and support not just for this project, but since are first project January. Thank you for guiding me through the world algorithmic trading.

# 1 Introduction

The task of forecasting future stock price movements, is a topic which has been studied over a wide range of disciplines for many years. There are many reasons for this, including the financial incentive. From the perspective of Computer Science however, the main factor which attracts researchers is the high complexity of the data. In addition to this, the structure of stock price data means that techniques developed for this task can be easily applied to any problem whose data is a chronologically ordered sequence. However, unlike many other machine learning tasks, stock price forecasting adds an additional requirement for models to be effective. Namely, this requirement is speed. Any model produced not only needs to be accurate, but must also be able to keep-up with sudden market changes, as well as beating competitive systems.

Due to all these reasons, it is therefore no surprise that many researchers are attracted to this problem. Over the years, a wide variety of machine learning techniques have been applied to the problem. These include but are not limited to: Linear models (Jansen 2018, p.175) (Fama & French 1996), Non-linear models (Qi 1999), Bayesian systems (Jansen 2018, p.268), Ensemble methods (de Prado 2018, p.100), and Evolutionary algorithms (Nenortaite & Simutis 2004, Kaboudan 2000).

One active field of study is the application of deep learning models to the stated problem. This area of research is of particular interest, due to the fact that stock price data, by its very nature, encapsulates all market information at each time step (Fama 1970). Thus, any relationships to be discovered using solely past stock prices (known as technical analysis), will require a large number of variables to model.

In recent years natural language processing has been dominated by one form of architecture, known as sequence to sequence learning. This is owed to the fact they allow variable length inputs and outputs. Thus, for tasks such as translation, these models are well suited to the data. In the recent years these models have been enhanced by using an attention mechanism (Bahdanau et al. 2015), which allows the relationship between all elements in the sequence to be modelled.

Vaswani et al. (2017) used this idea of attention to develop a new deep learning architecture called a transformer. Unlike previous models applied to sequence learning, transformers rely entirely on the use of an attention mechanism, whereas previous models required the use of recursion. The removal of recursion allows for parallelisation whilst training, which due to recursions very nature was not previously achievable. Not only does this model increase the training speed of the system, but also obtains a greater level of accuracy over at the time state of the art models.

As previously mentioned, the structure of financial data means developed models can be applied to any chronologically sequenced data. This also works in reverse. For example, due to the sequential nature of text, methods developed for natural language can be applied to financial forecasting. This relationship is important to understand, as the relatively new transformer architecture can be applied to the stock market.

Their are several key benefits to this. Firstly, when applied to natural language data, transformers obtain scores of significant improvement. Recent developments have shown a further increase in the achieved accuracy. By merely increasing the size of the model to include 1.5 billion parameters, researchers at open AI managed to achieve state of the art results. These results were so accurate they did not release the model due to potential "malicious applications" (Radford et al. 2019). So, if this increase in accuracy can be applied to financial forecasting, this could help investors make more informed decisions.

In addition to this, as transformers are parallelizable, this would offer a significant speed advantage over the current systems utilised. As this increase occurs during training, this is an even greater benefit. The reason for this is due to efficient market hypothesis, which technical analysis is built on. The theory states that not only is each stock price an encapsulation of all market information at that time, but each companies stock is a different representation of this information (Fama 1970). Therefore, in order to forecast each companies stock price, a new model needs to be trained on said data. Thus, an increase in training rate gives a competitive advantage once a stock has been identified to forecast.

Lastly, the attention mechanism utilised by the system provides the exact modelling required for stock data (i.e. the relationship between each price in the sequence will be individually modelled). This point-wise modelling of relationships in the sequence, once analysed, may help the development of the domain.

The objective of this paper is to evaluate the performance of transformers when applied to stock market forecasting. This will be achieved by augmenting the structure of the transformer architecture to perform regression tasks. The task of financial forecasting will also be restructured into a classification task, in which the goal of the standard transformer will be to generate buy or sell signals. By evaluating the transformer as both a regressive and classification system for time series data, the hope is that this will prove the effectiveness of attention when applied to financial forecasting.

## 2 Literature Review

This section details the different fields and papers studied. It is divided into three sections, which represent the three main areas of research conducted for this project. Namely, these are: an overview of sequence to sequence architectures, a review of transformer networks, an overview of time-series analysis, and an introduction to technical analysis.

### 2.1 Sequence to Sequence Architectures

#### 2.1.1 Encoder-Decoder Models

Over the past six years, sequence to sequence architectures (SSA) have rose to become one of the most dominate forms of deep neural network (DNN) structures. First proposed by Sutskever et al. (2014) and Cho et al. (2014), SSAs solve one of the main problems presented by DNNs. Namely, this is the fact that DNNs can only be applied to problems whose inputs can be encoded into a fixed length vector. Thus, tasks which contain a variable length input and/or output (e.g. natural language), can not be efficiently or effectively applied to DNNs.

The solution provided by SSAs to this problem, is the use two recursive neural networks (RNNs). The former of this pair acts as an encoder, whilst the latter acts as a decoder. The encoder reads the variable length sequence one step (token) at a time. Due to the formulation of RNNs, each node in the network contains a hidden state, which is updated after every input (1).

$$h_t = \text{sigm}(W^{hx}x_t + W^{hh}h_{t-1}) \tag{1}$$

Once the entire input sequence has been passed through the encoder, the final hidden state $h_T$ (known as the context, $c$) can be considered to be a fixed length representation of the variable length input. This hidden state is used to initialise the decoder's hidden state, which produces a variable length output.

This architecture, when applied to the task of neural machine translation (NMT), managed to outperform previous DNN models. Sutskever et al. (2014) utilised two long short-term memory (LSTM) networks to apply this architecture to the task of NMT, whilst Cho et al. (2014) used a novel gated RNN. Both of which saw an improvement in BLEU (Bilingual Evaluation Understudy) scores when compared to previous models.

However, this method does contain two limitations. Firstly, due to the process of updating the hidden state, tokens near the start of the sequence will be less

represented than those nearer to the end. Both Sutskever et al. (2014) and Cho et al. (2014) try to mitigate this problem through the use of gated RNNs, which selectively update the hidden state. However, this merely manages, and does not solve the problem. Interestingly, Sutskever et al. (2014) also found that reversing the input sequence further increased the accuracy of their model. This is assumed to be caused by the reduction in short-term dependencies, between the start of the input and output sequences.

The second problem with this method is due to the compression of the input sequence. As the input sequence is represented by a fixed length vector, there is a compression of the data which takes place. The consequence of this means that data which could potentially be useful could be overwritten in the representation, or simply under-represented. This is especially prevalent as the input sequences become larger.

### 2.1.2 Attention

Bahdanau et al. (2015) proposed a partial solution to both of the problem presented by SSAs. Instead of solely using the encoders final hidden state, Bahdanau et al. (2015) uses a series of hidden states produced at each step from the encoder. Specifically, two series of contexts $\{\overrightarrow{h_i}\}_{i=1}^{T}$ and $\{\overleftarrow{h_i}\}_{i=1}^{T}$ (where $T$ is the length of the input sequence) were produced using a bidirectional RNN. The annotation for each token, $x_i$ in the input is simply given by concentrating on the hidden states at step $i$ (2).

$$h_i = [\overrightarrow{h_i}; \overleftarrow{h_i}] \tag{2}$$

For prediction, the decoder uses these annotations to produce a unique context for each token in the output series. This is achieved by taking a weighted sum of all annotations, whose weights are in turn produced by an alignment model (implemented as a feed-forward neural network), which is trained jointly with the system (3).

$$\begin{aligned} c_i &= \sum_{j=1}^{T} \alpha_{ij} h_j \\ \alpha_{ij} &= \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})} \\ &\text{where,} \\ e_{ij} &= a(s_{i-1}, h_j) \end{aligned} \tag{3}$$

This method solves both problems presented by SSAs. Firstly, as the input sequence is inputted in both forward and reverse order, this means that all tokens in the input are equally represented within the annotations. This is intuitive as on the forward pass, tokens near the start will be under-represented, whilst those at the end will be over-represented. The converse is true on the backwards pass. Thus, when summed together, each annotation holds a balanced representation. Furthermore, as a series of hidden states is produced, each hidden state at step $i$ is concentrated on capturing token $x_i$.

Secondly, as the system produces an annotation for each token $x_i$, the problem of compression is made redundant. By using a series of annotations, the decoder can instead focus on the sections of the input which are most relevant for the current output token being produced.

Although this method developed by Bahdanau et al. (2015) is not named within their paper, is has been since coined as attention. This attention system, when applied to NMT, saw a 7.25 increase in BLEU score over the originally proposed encoder-decoder architecture. Furthermore, the attention system even scored higher (on known words) than the at the time state of the statistical machine translation (SMT) model, Moses, by 0.52 BLEU.

The attention mechanism proposed by Bahdanau et al. (2015) can be considered a global search over all the annotations. Luong et al. (2015) formalised this definition, by considering the weights as an alignments vector. For a global search, the alignment vector is produced by comparing the current annotation to all other annotations. The issue with this approach is that it is costly to iterate over all the annotations.

To solve this issue, Luong et al. (2015) proposed the use of local search attention mechanism. In this case only a small subset of the annotations are considered for each context. The creation of this subset can either simply be a monotonic window, meaning at position $p_t$ only the subset $\{p_{t-D}, \ldots, p_{t+D}\}$ is considered (where $D$ is the size of the window). Alternatively, a more robust method is through the use of a predictive alignment technique. This method takes into consideration past alignment decisions, by concatenating the previous attention decision with the next time step. Thus, the system is aware of previous alignment decisions. Due to this the alignment vectors must be calculated jointly. Surprisingly, this latter method not only reduced the search cost when compared to the global variant, but also managed to outperform its global counterpart by 0.9 BLEU.

### 2.2 Transformers

Currently, the SSAs discussed rely on the recursion of RNNs in order to process input sequences. Transformers however, rely solely on attention. Proposed by Vaswani et al. (2017), transformers use an encoder-decoder architecture. The key difference being that RNNs are not utilised at any stage.

As previously discussed, RNNs produce hidden state $h_t$ by utilising a function $f(h_{t-1}, x_t)$. This means that the system is sequential by nature. Attention on the other hand is parallelizable, and can model dependencies in the input no matter the distances between tokens in the sequence. Thus the benefit of relying solely on attention is clear.

Although transformers share the same encoder-decoder structure as previous sequence to sequence models, the lack of any RNN means their composition is entirely different. Both are built using the same three sub layers, which are defined as:

- *Self-Attention Layer*
  The self-attention layer performs a scaled dot product attention on the given query ($Q$), keys ($K$), and values ($V$). This is defined in equation 4. This sub-layer also linearly projects the inputs into $h$ different spaces. This is known as multi-headed attention, with each head being computed in parallel. The benefit of this is that each head gives a different representation of the data at each position. Each head produces an output of dimensionality $d_v$, which are then all concatenated.

$$\begin{aligned} \text{Attention}(Q, K, V) &= \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \\ \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{4}$$

- *Feed-Forward Neural Network*
  A two-layer fully connected feed forward neural network (FFN) is applied to each position. Formally this is defined as:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{5}$$

- *Layer Normalisation*
  Normalisation, as proposed by Ba et al. (2016), is applied after every operation.

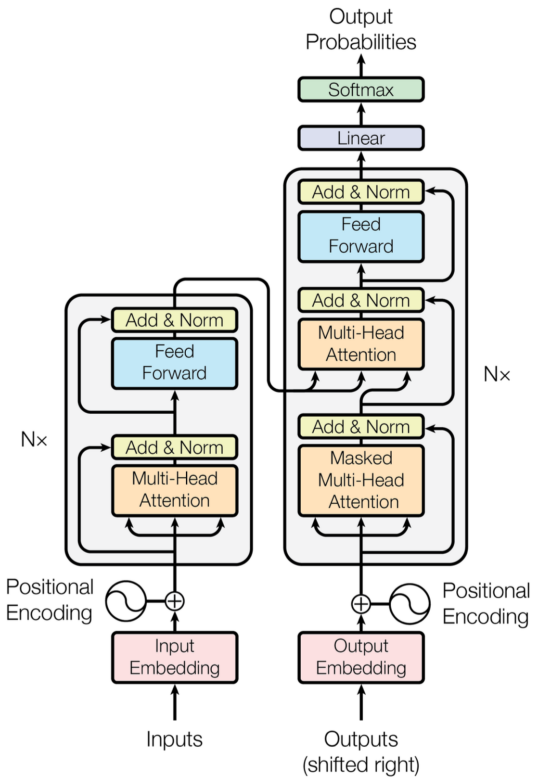


Fig. 1: Transformer architecture proposed by Vaswani et al. (2017)

As shown in figure 2, the described sub-layers are stacked to create the encoder and decoder. These are in turn stacked $N$ times to produce the transformer.

In addition, a positional encoder is applied to the input of the encoder and decoder. This allows for the position of each element in the sequence to be accounted for. Specifically, the positional encoder works by producing a vector of the same dimensionality as the input, and adding this vector to the input. The positional vector values are given by the equation 6.

$$\begin{aligned} PE_{pos,2i} &= sin(pos/10000^{2i/d_{\text{model}}}) \\ PE_{pos,2i+1} &= cos(pos/10000^{2i/d_{\text{model}}}) \end{aligned} \tag{6}$$

The class embedding simply produced a word embedding for each of the input sentences, and the softmax layer merely gives a probability of the next output token being each word in the vocabulary.

By utilising the self-attention mechanism in this way, a transformer can better model long-term dependencies within the data, as it will compute the relationship between all tokens in the sequence. However, this comes at a heavy computational and memory cost, which is especially prevalent in long sequences.

Sukhbaatar et al. (2019) aimed to solve this problem by applying an adaptive memory span. Each head in the transformer assumes the same attention span is required, however on investigation, Sukhbaatar et al. (2019) found that this is not the case. With some heads focusing on recent history, whilst others focus on the entire sequence. The solution proposed in this work is the application of a masking function to each head. This function controls the attention span of each head, thus allowing each head to only concentrate on the parts which are relevant. The addition of this masking function meant that the transformer could accurately produce a context for sequences of up to eight thousand in length, whereas before the maximum was one thousand.

The ability for transformers to model long sequences was further improved by Child et al. (2019), who developed a sparse transformer model. Like the previously mentioned adaptive memory span, sparse transformers also aim to reduce the required number of calculations for each attention matrix. The model achieves this by utilising sparse attention patterns, which only updates a subset of each attention matrix for each output. This subset is relatively small, specifically $\sqrt{N}$. This therefore reduces the required number of calculations from $\mathcal{O}(N^2)$ to $\mathcal{O}(N\sqrt{N})$. The use of this system allows the transformer to identify patterns in sequences thirty times longer than it was previously capable of. This increase allowed Child et al. (2019) to generate images, and even audio data. Interestingly, the utilisation of the sparse matrices not only increased the speed of the transformer, but also saw a decrease in loss with shorter sequences.

### 2.3 Time-Series Forecasting

A time series is any set of numerical data points which are organised in chronological order (e.g. weather or financial data). The driving goal of time series forecasting is to predict the next step(s) in the sequence. More formally, given a set of data points $\mathcal{D} = \{x_1, x_2, \ldots, x_{T-1}, x_T\}$, $\mathcal{D}$ is a time series if and only if each $x_t$ occurs one time step after $x_{t-1}$. Framing a time series this way, allows a clear comparison to be made between time-series analysis and NLP. It is therefore of no surprise that many of the models and techniques developed for NLP, have been applied to time series analysis.

$$\underbrace{\begin{array}{cccccccccc} \text{The} & \text{quick} & \text{brown} & \text{fox} & \text{jumps} & \text{over} & \text{the} & \text{lazy} & \text{dog} \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \end{array}}$$

Each word in the sentence is the next time-step in the series.

As formally defined by Lim & Zohren (2020), the goal of time series forecasting is to produce a function $f$ which satisfies:

$$\hat{y}_{i,t+1} = f(\boldsymbol{y}_{i,t-k:t}, \boldsymbol{x}_{i,t-k:t}, \boldsymbol{s}_i) \tag{7}$$

Where $\hat{y}_{i,t+1}$ is the forecast for the next time step, $\boldsymbol{y}_{i,t-k:t} = \{y_{i,t-k}, \ldots, y_{i,t}\}$, $\boldsymbol{x}_{i,t-k:t} = \{x_{i,t-k}, \ldots, x_{i,t}\}$ the targets and inputs in a window $k$ respectively, and $\boldsymbol{s_i}$ is any meta data required. The methods to produce the function $f$ vary, with the focus of this work being the application of deep learning models to the problem. Lim & Zohren (2020) conducted a survey of the common deep learning architectures applied too said task. The main techniques as detailed by the survey are detailed within this section.

### 2.3.1 Convolution Neural Networks

As described by Lim & Zohren (2020), convolution neural networks (CNNs) have been effectively applied to time-series forecasting through the use of layered casual convolutions. These casual convolutions act as filters, which hides future information from the network, thus upholding the temporal nature of time-series data. Given $A(\cdot)$ is an activation function, $l$ is the current layer, $t$ is the current time step, and $\boldsymbol{W}(l, i)$ is the filter weights at layer $l$. A convolution filter for time-series forecasting is defined as:

$$\boldsymbol{h}_t^{l+1} = A\left(\sum_{i=0}^{k} \boldsymbol{W}(l,i)\boldsymbol{h}_{t-i}^{l}\right) \tag{8}$$

As Lim & Zohren (2020) explain in their paper, CNNs by nature assume any relationships are time-invariant. Furthermore they are limited by the window size $k$, which due to the application of filter weights must be fixed.

### 2.3.2 Recurrent Neural Networks

As previously discussed, the developments in the application of RNNs to NLP make the models prime candidates for time series forecasting. This is due to the

sequential nature of inputs required by RNNs, lending themselves seamlessly the structure of time-series data. Furthermore, RNNs allow an infinite window, as it processes one element in the sequence at a time. Thus, RNNs are widely applied to a range of time-series forecasting tasks (Rangapuram et al. 2018, Lim et al. 2019, Wang et al. 2019).

It is worth noting that the encoder-decoder models defined in 2.1.1 have also been applied time-series forecasting (Qin et al. 2017, Malhotra et al. 2016).

### 2.3.3 Attention Mechanisms

Attention mechanisms are another method highly suited to time-series forecasting. As previously described in 2.1.2, attention allows the model to focus on key points in a sequence. For time-series data, this allows the system to isolate key time-steps when making a prediction.

One such example of this is the work conducted by Qin et al. (2017), who applied an encoder-decoder architecture with attention to forecast climate and stock data. Specifically, the model used an LSTM for the encoder and decoder, but applied different attention layers to each. The encoder used an input attention layer, which produced weights for the current hidden state based on the input and the previous hidden state. The decoder used a temporal attention layer, which produces a context based on the previous decoder hidden state, and a weighted sum of all the encoder hidden states. The output of the temporal attention layer is used as the input for the decoder. When applied to the NASDAQ 100 data set, this system managed to achieve better results than all previous state of the art methods. The same results were found for the SML 2010 data set.

More recently, Wallbridge (2020) used casual convolution layers to extract information, and then fed this into a transformer block which applied multi-head attention. When tested on the FI-2010 data set, this system managed to outperform a standard LSTM and CNN.

In addition to this, Li et al. (2019) proposes the use a transformer with a convolution based attention mechanism. The reason for this is that the point-wise calculation used in the standard transformer may not utilise the shape of the data. The proposed attention mechanism simply works by applying a casual convolution operation to the input, using kernel size $k$ and a stride of one, to produce queries and keys. The motivation for this is that the convolution operation may allow the queries and keys to represent the shape of the data. This model was used to forecast electricity, traffic, solar, and wind over different windows and shows similar results to state of the art RNNs.

### 2.3.4 Other Models

The application of deep learning models to time-series forecasting is constantly developing. Recently, researchers have begun combining traditional quantitative methods with deep learning. The reasoning for this is that machine learning models are prone to over fitting, and so by combining domain knowledge the risk of this occurring is mitigated. Although this research is in early development, preliminary studies have show these hybrid models can outperform both traditional and solely deep learning systems (Lim & Zohren 2020).

## 2.4 Financial Forecasting

One of the most widely studied time-series problems is the ability to predict future stock price movements. The motivation for this task is simple. The more accurate the model, the bigger the profits.

Technical analysis is one of the two main methods of stock price prediction. It is built upon the efficient market hypothesis, which states that the price of the stock "fully reflect all available information" (Fama 1970). This therefore means that models only require the past prices of a company, to forecast future movements.

As discussed in my previous work (Celella 2020), the efficient market hypothesis also works against the idea that past stock prices can be used in this way. This is due to the idea of the random walk. Proposed by Fama (1970), the theory suggests that due to the very fact the market price encapsulates all information, this inherently means that the movement of prices is random, as each step is a reflection of independent information.

Proponents of technical analysis disagree with this however, stating that prices move in clear trends (Lo & MacKinlay 1999). A explanation for this behaviours was proposed by John M. Keynes, who hypothesised the idea that even if the underlying data is random, investors tend to follow the crowd. This theory is the castle-in-the-sky theory, with suggests that movements in the market are caused by the psychological behaviour of investors, which in itself is predictable (Malkiel 1973). In addition to this (Schwager 1999, p.1), states that even if the system on a whole is random, this does not mitigate the claim that there are windows of predictable behaviour.

Due to the inherent volatility in technical data, analysts have developed a range of techniques which attempt to remove noise. Two of these techniques are outlined in this section.

### 2.4.1 Moving Averages

Moving averages are a method of analysing trends in stock price movements. The idea behind this technique is that by smoothing out the price movements, trends become easier to identify. This however comes at a cost of lagging the data, meaning if a sudden change occurs the system may not react fast enough. It is worth noting that usually the closing prices are used to produce averages.

There are several ways to calculate the moving average of a price. The most basic method is the simple moving average (SMA), which merely sums the price of the past $N$ and divides by $N$ (9). More sophisticated methods do exist, such as the weighted moving average (WMA), which weights each price by the amount of time steps it is behind the current (10). The benefit of this is that the more recent prices will have a greater reflection in the average(Schwager 1999).

$$SMA = \frac{1}{N}\sum_{t=0}^{N} p_t \tag{9}$$

$$WMA = \frac{1}{N}\sum_{t=0}^{N} (N-t)p_t \tag{10}$$

Another average calculation which is widely used is an exponentially weighted moving average (EWMA). This method aims to encapsulate all previous stock price information, by recursively calling the previous value (11). The result of this means that each data-point contains more information, as all previous days are reflected albeit at an exponentially decreasing weighting. This latter point means that although information is retained, the produced value is still highly reflective of the recent data (Schwager 1999).

$$EWMA_t = \alpha \cdot p_t + (1-\alpha) \cdot EWMA_{t-1} \qquad 0 \leq \alpha \leq 1 \tag{11}$$

As the calculations for WMA and EWMA both weight recent prices higher than the older, they are more reactive to sudden changes in the market. Thus, this helps to mitigate the problem of lagging caused by moving averages.

Technical analysts not only use moving averages to reduce local noise in the data, but also to generate buy or sell signals. This is achieved by calculating a slow moving average (e.g. $N = 100$), and a fast moving average (e.g. $N = 20$). When the fast moving average crosses its slow counterpart, this a signal to sell. The converse is true when the fast moving line crosses below the slow.

### 2.4.2 Chart Patterns

Chart patterns are predefined, recurring patterns in price charts. They are used by technical analysts of indications of future movements. These are generally not useful in isolation, however when multiple patterns occur, these signals can be combined to form a more accurate prediction of the change in trend (Schwager 1999).

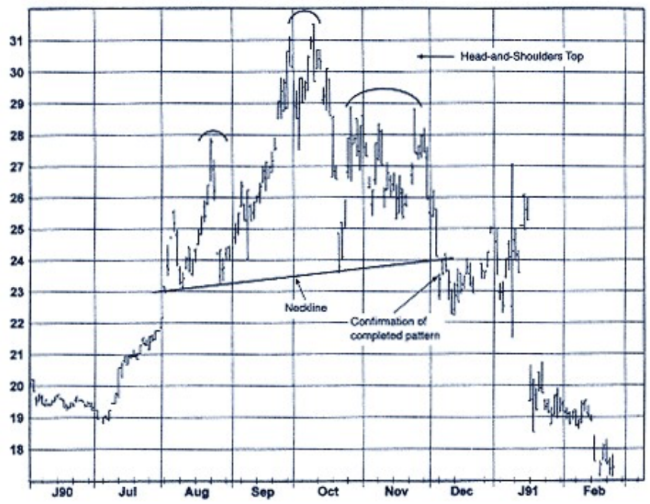


Fig. 2: Head and shoulders chart pattern Schwager (1999)

Some of the most common chart patterns are defined as follows Schwager (1999):

- *Head and Shoulders*: One large peak is nested between two smaller peaks. An inverse head and shoulders occurs when a large trough is nested between two smaller ones. The smaller peaks/trough form a neckline that if broken indicates a reversal in the current price direction.
- *Broadening wedge*: A broadening wedge occurs when consecutive extrema bounce between increasing highs and lows. This pattern shows that the market is highly volatile.
- *Triangles*: A set of consecutive relative highs/lows, followed by a set of relative lows/highs. Symmetrical triangles indicate the market will continue on the current trend. If not, it will follow the hypotenuse.
- *Rectangles*: Consecutive extrema all fall within a boxed boundary. If the price breaks the top line, this is an indication the price will rise. The converse is true if the bottom line is broken.

These patterns were formally defined by their geometric properties by Lo et al. (2000). Equation 22 shows the formal definition of the head and shoulders pattern. The definitions for the other patters can be found in appendix A. All definitions require a set of five extrema $\{E_1, \ldots, E_5\}$, which are labelled either maxima or minima.

$$HS = \begin{cases} E_1 & \text{Is a maximum.} \\ E_3 > E_1, E_3 > E_5 & \\ E_1 \ \& \ E_5 & \text{Within 1.5\% of thier average.} \\ E_2 \ \& \ E_4 & \text{Within 1.5\% of thier average.} \end{cases} \tag{12}$$

The reasoning for using chart patterns in computerised systems is that like moving averages, it can be used to reduce local noise in price data. Chart patterns are more effective at this however, as they provide an abstract geometric representation of the data. The obvious downside to this is loss of granular details (which is somewhat preserved by moving averages).

## 3 Methodology

As previously described, the structure of natural language data is highly similar to serialised price data. This is due to the structure of both being a chronological sequence. Therefore, the internal mechanisms of a standard transformer can be directly applied to financial data. However, there are two key aspects of the model which must be augmented.

The first change which must be made is to the class embeddings. Financial data is already numerically represented, thus does not need any representation (unlike text). However, in order to test how the volatility of price data effects the results, the model will be tested using not only the raw data, but also using moving average and chart pattern representations. Thus, two embedding systems will be produced. In addition to this, the raw price data will be normalised.

The second change to the model will be the output layer of the transformer. In the standard model, the transformer uses a softmax layer to produce a probability of the next token being each word in the given vocabulary. This means that currently the transformer can only be utilised as a classification tool. Hence, this will require replacing with a layer capable of producing a continuous values.

The changes made to the transformer architecture are detailed within this section.

In addition to this, the results obtained from the augmented transformer will be compared to a standard RNN, which will be trained using the same data sets. This is so the performance of the transformer network can be measured against a standard approach.

It is worth noting that for this project, only the standard transformer proposed by Vaswani et al. (2017) will be implemented, without any of the augmentations suggested by Sukhbaatar et al. (2019) and Child et al. (2019). This decision was made as all of these methods aim to increase the length of the input sequence a transformer can handle. For this project, the input and target sequences will be kept relatively short due to project time constraints, and so these problems will not be encountered. Furthermore, there is currently very little work which evaluates the capabilities of a standard transformer when applied to time-series forecasting. Thus, it would be useful for this project to evaluate how well a transformer can be applied to the task. This is especially true for financial forecasting, in which the only work which has applied a transformer found was Wallbridge (2020), who relied heavily on CNNs for feature extraction, and didn't use a full transformer network. Thus, this work will aim to evaluate the performance of the transformer network, using domain knowledge data representations.

### 3.1 Embedding Systems

As previously discussed, different representations of the financial price data will be used to train the transformer network. These can be considered different embeddings of the data, in the same way numerical word embeddings represent text.

The first data set which will be utilised, will be the normalised raw price values. However, due to the nature of transformers requiring the source and target sequence as inputs during training, standard normalisation will not be sufficient. This is due to the fact if the data was normalised across the source and target, whenever the price is rising the source will consist of low values. With the converse being true if the price drops. Therefore, the transformer will learn this pattern, which will not occur in the test set or real-world application. Instead, the source must be normalised independently of the target, whilst the target normalisation must in turn depend on the source. More formally, given a source sequence $\boldsymbol{s} = \{s_1, \ldots, s_N\}$, a target sequence $\boldsymbol{t} = \{t_1, \ldots, t_N\}$, the normalisation of each sequence is as follows:

$$\begin{aligned} \boldsymbol{s}_{normalised} &= \frac{\boldsymbol{s} - s_{minimum}}{s_{maximum} - s_{minimum}} \\ \boldsymbol{t}_{normalised} &= \frac{\boldsymbol{t} - s_{minimum}}{s_{maximum} - s_{minimum}} \end{aligned} \tag{13}$$

This methods has the additional benefit of allowing easy classification of the data, as any values above one in the target sequence will indicate a price rise from the maximum in the source. This means that the outputted values can be considered as buy or sell signals, depending if the values are above or below one respectively.

The second embedding system which will be used, will be a moving average representation of the data. As previously discussed, the use of moving averages will reduce the local noise within the data, meaning it the transformer may be able to more accurately model trends. Specifically, the weighted moving average will be utilised, due to its ability to react quickly to sudden price movements (10). The source and target sequences produced will be normalised using the equations specified in 13.

The final method of representing the price data will be through the use of chart patterns. This will be achieved by breaking the input sequence into $k$ windows,

with each window containing $N$ close prices. The extrema for each of these windows will be identified, to produce a set of extrema $E$. As each geometric definition of chart patterns defined by Lo et al. (2000) require five extrema, the set $E$ will be iterated along in a sliding window of five with a stride of one. Each of the definitions will then be applied, detecting whether the current window of extrema satisfy the properties of said pattern. The $k$ windows of the input sequence will then be represented by a fixed length vector, in which each element represents a specific pattern. If the pattern is detected, the corresponding element in the vector will be set to 1. If the pattern is not present, a value of 0 will be used at the corresponding position.

$$\begin{matrix} \{ & HS & IHS & BT & BB & TT & TB & RT & RB & \} \\ \{ & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & \} \end{matrix} \tag{14}$$

This embedding will give the transformer an abstract representation of the shape of the data. Allowing it to concentrate of the shape of the graph, instead of the actual values.

### 3.2 Output Layer

The goal of a time-series forecast is to produce a prediction of the next time step(s) in the sequence (7). It is therefore inherently a regression task, and thus requires a linear layer to project the output of the decoder to the desired values. To achieve this, a simple feed-forward neural network will be utilised. This network will consist of three layers, with each layers output being passed through the ReLU activation function. This activation function was selected due to the fact a price can never fall below 0, but can increase to any positive value, making it suitable for the task at hand. This output layer is formally defined in equation 15.

$$\text{Output}(x) = \max(\max(0, xW_1 + b_1)W_2 + b_2)W_3 + b_3 \tag{15}$$

In addition to this, the transformer will also be used to attempt to directly classify a sequence as a buy or sell opportunity. This can be approached in one of two ways.

The first approach to be tested will be to simply augment the standard soft max layer in the original transformer. The only change which would be required, would be a reshaping of the output from $d_{model}$ to a size of three. These new vectors of length three will represent the buy, sell and hold signals respectively.

With a value of 1 at the corresponding element position indicating which operation to enact.

$$\hat{y}_{i,t+1} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{matrix} \text{Buy} \\ \text{Hold} \\ \text{Sell} \end{matrix} \tag{16}$$

The second approach is to reshape the output the of the encoder, using a preliminary layer between the encoder and decoder. This will simply be a feed-forward neural network, and will be of the same implementation as the first classification method. The key difference between this method and the previously described classification method is the role of the encoder. In the first method, the encoder will produce a representation of the input sequence, so that the decoder can produce a sequence which is of the same representation as the input. This secondary idea instead uses the encoder to represent the data as a series of buy or sell signals. Thus, the decoder will be producing a different representation of the data (whereas in the first method the representation is only changed in the output layer).

For this project the former approach will be taken. This decision was made due to preliminary tests with the latter method proving to yield no significant results. Thus, it was not taken further due to the time-constraints of the project. After some analysis, the reason why this method did not seem to produce valid results is due to the fact the preliminary layer compressed the encoder output. This resulted in too much data loss for the decoder to make any accurate prediction.

Both classification tasks will require a soft max layer for the output, as defined in the standard transformer architecture.

It is worth noting at this point that not all the representation systems outline in section 3.1 will be used for both the regression and classification task. The reasoning for this is that the moving average, and chart pattern representations were initially developed to generate buy and sell signals (Schwager 1999). Because of this only these two representations will be used to test the classifier. Conversely, the raw price data and moving average representations will be used for the regression task.

### 3.3 Data sets

Two separate data sets are used to train this system. Namely, these are the Standard and Poors' 500 (S&P), and the Dow Jones Industrial Average (DJIA). Both data sets were retrieved from the Wall Street Journals historical prices

archive (Journal n.d.*b*,n). The S&P 500 set consists of all daily price values from 9/1/1978 to 3/8/2020, whilst the DJIA set contains all daily price values from 2/1/1990 to 1/9/2020.

These data sets were used for three key reasons. Firstly both of these data sets represent a wide range of industries, meaning they encapsulate a large majority of the market information. Secondly, both the DJIA and S&P indexes are well established, dating back to 1896 and 1957 respectively. This they both contain a large amount of historic data, and thus a sufficient data set to train the proposed network. Lastly, both these indexes are widely used in other machine learning projects tackling the same task. Thus, using these data sets allows easy comparisons between other peoples work.

### 3.4 Loss Function

In order to evaluate the accuracy of the transformer for regression, the mean squared error loss function (17) will be utilised. This loss function was selected due to the fact Lim & Zohren (2020) survey specified it as the common method utilised for time series point estimation. Thus, this will allow easy comparison between the results obtained in this work and others.

$$\mathcal{L}_{regression} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{17}$$

For classification, the cross entropy loss function will be used. This was again chosen for the same reason as mean squared error, as Lim & Zohren (2020) specified the function as the standard.

$$\mathcal{L}_{classification} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i}) \tag{18}$$

# 4 Implementation

This section details how each of the models and methods were implemented, as well as detailing the software libraries and packages utilised. Appendix B provides a guide on how to utilise the system.

## 4.1 Transformer

The transformer network was implemented by following Vaswani et al. (2017) paper, with the Pytorch (Paszke et al. 2017) being used to implement all stages of the network. Pytorch was used due to the backwards propagation feature inbuilt into its tensors. This means that the weight matrices in self-attention mechanism, which was built from scratch, would also undergo gradient descent. Other packages such as Keras (Chollet et al. 2015), do not contain the ability to back propagate through novel code, meaning they were unsuitable for this task.

The Adam optimiser was used with an adjustable learning rate, as described by Vaswani et al. (2017). This followed the formula:

$$lrate = d_{model}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \tag{19}$$

By using an adjustable learning rate, the chance of over fitting is reduced. This is because the learning rate on later epochs will adjust the weights by an ever decreasing amount. The parameters used to initialise the Adam optimiser are the same as used by Vaswani et al. (2017). These are shown in equation 20.

$$\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 10^{-9}, warmup\_steps = 4000 \tag{20}$$

In addition to this, every stage of the network includes a dropout layer. This is again to decrease the chance of over fitting, by only training each batch on a sub-network of the graph.

The implementation of the transformer, produced for this project, follows a top down object-oriented approach. This therefore allows different components on the model to be changes, thus allowing the different experiments outlined in the methodology to be run easily. Furthermore, as the sub-layers are independent classes, this allows for future experiments to be run using the code base. For example the self-attention mechanism can be changed to the sparse-attention mechanism defined by Child et al. (2019).

The transformer used to produce results was set to $N = 2$. Meaning that it consisted of two encoder layers, and two decoder layers. The reason for this reduction in model size from the $N = 6$ used by Vaswani et al. (2017), is simply due to the fact the dimensionality of the data for this project is far lower than that used in the NLP task. This reduction provides a decrease in training time of the model (allowing more tests to be run), whilst ensuring the model is not over complicated for the data.

### 4.2 Embedding Systems

The three embedding systems described in section 3.1 were implemented using the Pandas and Scipy libraries (Wes McKinney 2010, Virtanen et al. 2020). The normalisation method outlined by equation 13 was directly applied as described in section 3.1. This is also true for the weighted moving average representation, which was implemented as a lambda function and applied to each data point in the data set:

$$\lambda\, \boldsymbol{x}.\left(\sum_{i=1}^{N} x_i * i\right)/N \tag{21}$$

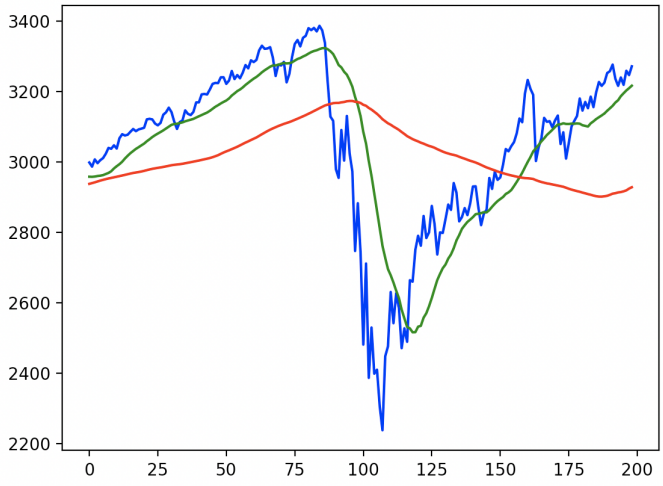


Fig. 3: Moving averages produced from S&P 500 data. Green line N=20, red line N=100.

It is worth noting that both the SMA and EWMA formulas have also been implemented within the code base. This is to allow for future experimentation.

For the chart pattern embeddings, each of the geometric properties were implemented as an independent function. The extrema were extracted from each of the $k$ windows, by utilising the *arglemin* and *arglemax* signal functions in the Scipy library. Each set of five extrema, which were produced as described in section 3.1, were then individually applied to each of the pattern functions.

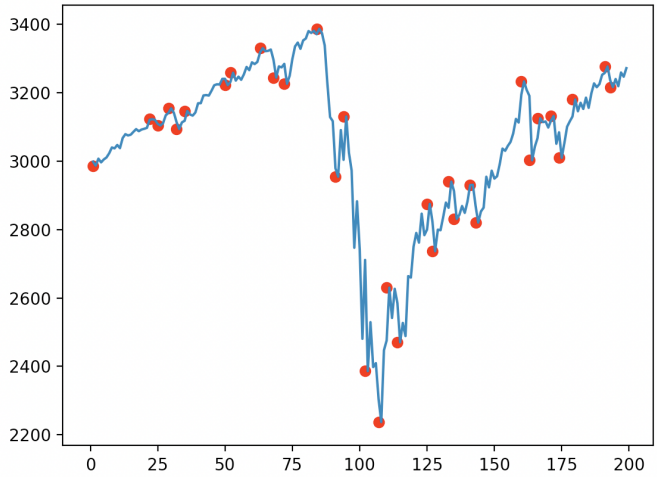


Fig. 4: Extrema detected on S&P 500 data. Area around day 25 shows a head and shoulders pattern.

### 4.3 Recursive Neural Network

The baseline RNN model was also implemented in Pytorch. The network consists of four hidden layers, each of which of dimension 2048. These number of layers and dimensions were chosen due to the fact the transformer network consists of four layers (two encoders, two decoders). This means that within the transformer there are four feed-forward neural networks. Each of which consisting of one hidden layer of dimension 2048. Therefore, the RNN consists of the same number of nodes as there are within the transformer. This will allow for a direct comparison between the transformers self-attention mechanism, and the RNNs hidden state mechanic.

The same adjustable Adam optimiser described in equation 19 and 20 will be used to train the model. Each layer will also be regularised using dropout.

### 4.4 Hardware

The models will be trained a Nvidia K80, with 12GB of memory and RAM, all of which has been provided by Google Collaboratory.

# 5 Results and Analysis

This section details the results obtained from the tests performed on the described network. All tests were run for 50 epochs, with a validation set used to measure the performance of the models after each epoch. To produce a final result after training, an test set consisting of unseen data was used in order to calculate the loss.

## 5.1 Regression

To test the normalised price data, five consecutive days worth of price data was used to predict the next five days prices. As figures 11, and 12 show, the model does in fact learn over time a function for prediction. One interesting thing of note is the initial spike and jagged behaviour in figure 12. From analysis of the hyper parameters, this is caused by the adjustable learning rate, which was suggested by Vaswani et al. (2017). Unfortunately, an optimal set of these hyper parameters could not be found, as each change came with its own drawback. Lowering the variable *warmup_step* for example, decreased the initial spike, but increased the volatility in the rest of the training.

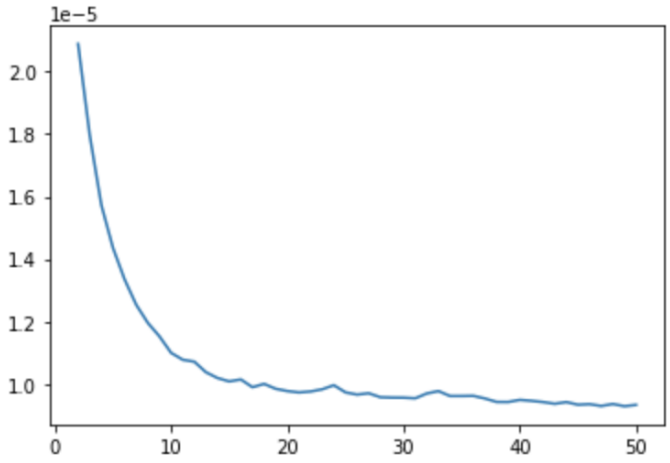


Fig. 5: S&P 500 regression validation loss for a five day prediction using normalised price data for each epoch.

The normalised price data was further tested with a longer sequence. Specifically the network was set to intake a 20 day sequence, and predict the next 20

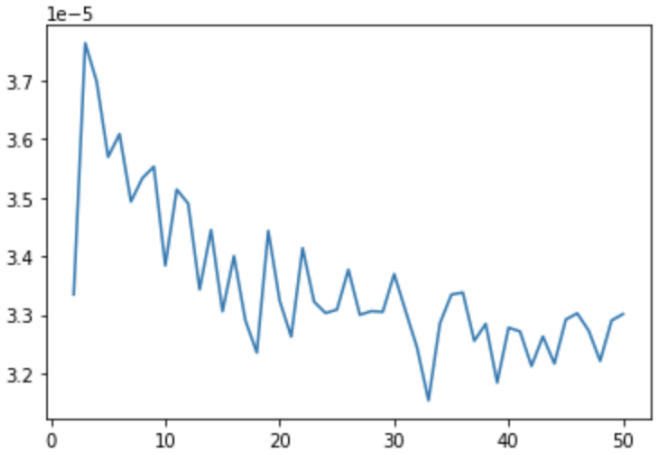


Fig. 6: S&P 500 regression validation loss for a five day prediction using normalised price data for each epoch.

days. Although the training loss decreased in the same way as with the shorter sequence, as shown in figure **??** the validation results obtained were less than ideal. These results show that the data clearly over fitted. A remedy to this would be to increase the size of the model, however due to hardware limitations this was not possible for this project.

For the S&P 500 data set, a test loss obtained for the 5 day normalised price data was, $3.6 \times 10^{-5}$, whilst for the 20 day period a loss of $8.9 \times 10^{-}5$ was produced. Similar results were obtained for the DJIA, scoring $3.9 \times 10^{-5}$ and $8.1 \times 10^{-}5$ respectively.

These same tests were the performed on the moving average data. For this data set, each day was represented by six values. These values were the calculated weighted moving averages using an increasing window. Specifically these were $N = 20, 40, 60, 80, 100, 120$. The idea behind this test was to see if by smoothing local noise, the system would do achieve more accurate predictions. The values for the windows were based off the standardised technical analysis model which uses a moving average of $N = 20$ and $N = 100$ to generate buy a sell signals. Due to the high dimensionality of the model, more windows were therefore added in an effort to increase the accuracy. A shown in figures **??** and **??**, the model again does learn a forecasting system. The volatility of figure **??** is again caused by the optimiser.

A test score of $2.37 \times 10^{-6}$ was obtained from this system using the S&P 500 data set, showing a significant improvement over using the raw data. The DJIA also saw an decrease in loss, obtaining $2.9 \times 10^{-6}$.

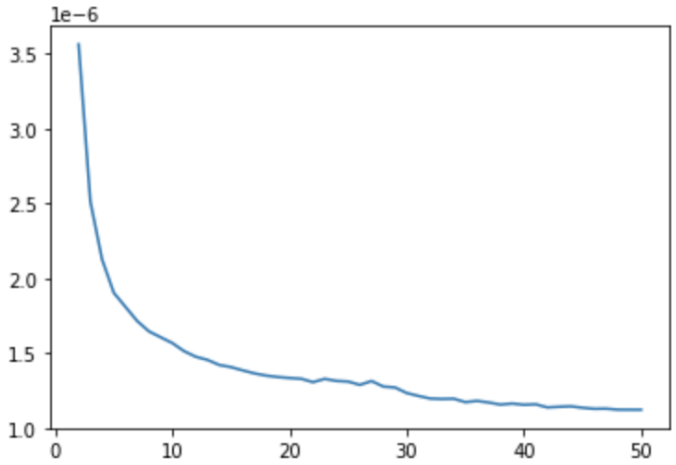


Fig. 7: S&P 500 regression validation loss for a five day prediction using normalised weighted moving average data for each epoch.

In order to evaluate whether the number of moving average windows caused this increase (due to a higher dimensional representation of the date), or whether the reduction in local noise was simply the key, a secondary test was performed. This test merely increased the amount of moving averages calculated from the original set, to $N = 20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 220$. This test was conducted using the S&P 500 data base, and obtained a test loss of $1.4 \times 10^{-6}$. Thus, this is an indication that the a higher dimensional representation of each day is more useful for the transformer.

To further determine whether the increase in dimensionality is the sole reason for this decrease, another moving average test was run using the S&P 500, using $N = 20, 40, 60, 80$. The reasoning behind this is that this representation has the same dimensions as the raw financial data, which used the open, close, high and low price of each day. Thus, a direct comparison can be made. This test produced a test loss of $6.3 \times 10^{-6}$, indicating that the dimensionality, and the local noise reduction both play a factor in the decrease in loss.

### 5.2 Classification

The classification tests were run in a similar fashion as to the regression. Specifically, the first test run used a five day moving average representation of $N = 20, 40, 60, 80, 100, 120$, to produce buy and sell signals for the five days. As shown in figures **??** and **??**, the results obtained from this experiment are not encour-

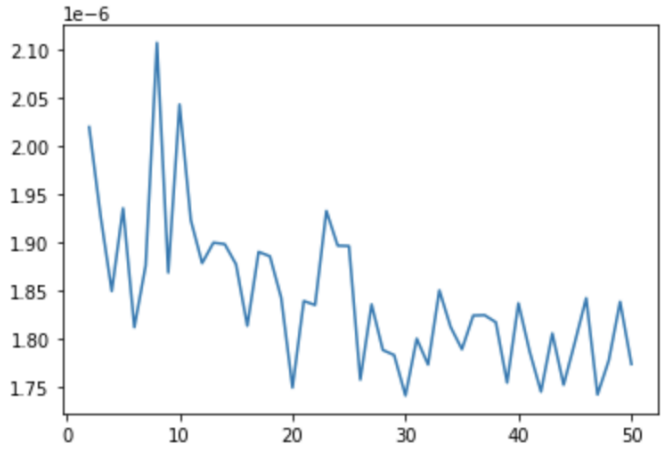


Fig. 8: S&P 500 regression validation loss for a five day prediction using normalised weighted moving average data for each epoch.

aging, due to the divergence in the training and test loss. This is a sign of clear over fitting of the data, and the same occurred for the DJIA data.

Originally, the idea behind this experiment was that because technical traders use the crossing points of different rate moving averages, the transformer would be able to simulate this behaviour. Instead, after inspecting the results, it seems as though the transformer simply tries to perform a regression on the data, predicting the next 5 days moving average. The error then occurs in the output layer, which attempts to map the moving average representation to the classification system defined in section 3.1.

The validation loss obtained for the S&P data set was $8.4 \times 10^{-5}$, whilst the DJIA achieved $6.5 \times 10^{-5}$. These are the highest loss values obtained in the project.

For the chart pattern representations, a window size of 30 days was utilised for each time step, with a stride of one day between each step in the sequence. The large window size was required in order to ensure a subset of chart patterns occurred in each time step. This allowed for a diverse set of representation to occur in the data set. Furthermore, as Schwager (1999) mentions, a single chart pattern is not sufficient to generate a trend prediction, thus multiple were required to be present in each representation.

Figures **??** and **??** show the training and loss values for an input sequence of 10, predicting the next 10 days buy and sell signals (after the last day in the final elements window). An initial assessment of this data would indicate that

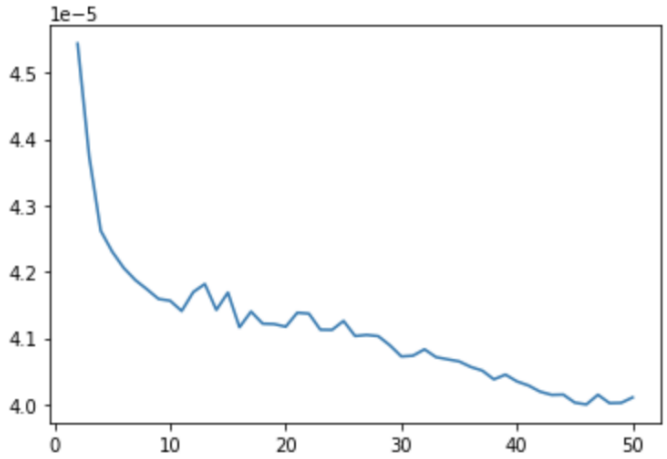


Fig. 9: S&P 500 classification training loss for a five day prediction using normalised weighted moving average data for each epoch.

there is a leakage between data in the training and validation set. However, the validation and testing days are removed before any prepossessing begins. Thus, a further investigation was conducted.

On analysis of the data, although a large window size was selected, the diversity produced in the data set was still lacking. This resulted in the system converging on a loss value within the first few epochs, as after there was nothing else to learn. Furthermore, the large window size meant a lot of overlap between sequences in the training set, and the validation set. This further enhanced the quick convergence on the loss. Due to these factors, this paper can make no conclusion on the validity chart patterns for this purpose, as a larger data set is required.

The S&P 500 and DJIA data sets achieved a validation loss of $1.9 \times 10^{-5}$ and $2.3 \times 10^{-5}$ for this task respectively.

### 5.3 Comparison to a Recursive Neural Network

The RNN developed for this project was trained on the four main tests conducted by this project. Namely, these are:

– *Raw_Rec_5*: Normalised raw price data, of sequence length five, used to forecast the next five days prices. Each time step is represented by the open, close, high and low price.

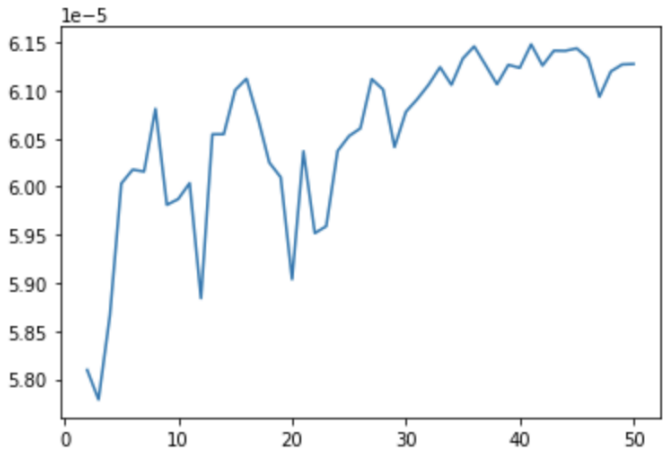


Fig. 10: S&P 500 classification validation loss for a five day prediction using normalised weighted moving average data for each epoch.

- *WMA_Rec_5*: Normalised weighted moving average data, of sequence length five, used to forecast the next five days. Each time step is represented by the weighted moving average using the windows $N = 20, 40, 60, 80, 100, 120$.
- *WMA_Class_5*: Normalised weighted moving average data, of sequence length five, used to forecast the next five days trading signals. Each time step is represented by the weighted moving average using the windows $N = 20, 40, 60, 80, 100, 120$.
- *PC_Class_10*: A sequence of 10, used to forecast the next 10 days trading signals. Each time step is a 30 day window, which is represented by a vector of length eight, in which each element indicates if a chart pattern occurs.

Tables 1 and 2 shows the test loss achieved by the RNN, compared to the transformer.

| Model | RNN | Transformer |
|---|---|---|
| *Raw_Rec_5* - S&P 500 | $3.67 \times 10^{-5}$ | $3.62 \times 10^{-5}$ |
| *Raw_Rec_5* - DJIA | $3.89 \times 10^{-5}$ | $3.92 \times 10^{-5}$ |
| *WMA_Rec_5* - S&P 500 | $2.74 \times 10^{-6}$ | $2.38 \times 10^{-5}$ |
| *WMA_Rec_5* - DJIA | $3.21 \times 10^{-6}$ | $2.99 \times 10^{-5}$ |

Table 1: Test losses obtained for the transformer and RNN networks for recursive tasks.

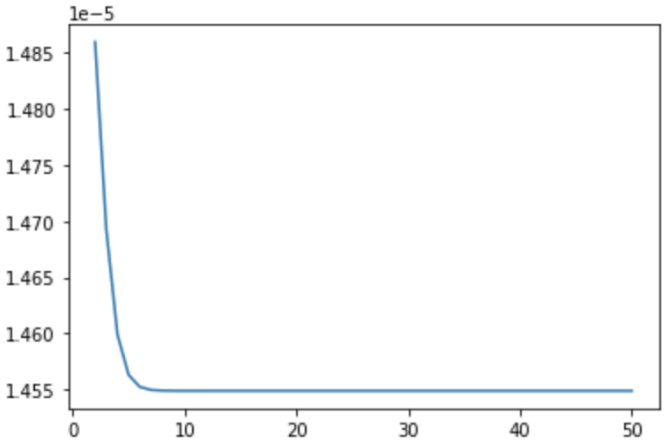


Fig. 11: S&P 500 classification training loss for a five day prediction using chart price patterns for each epoch.

| Model | RNN | Transformer |
|---|---|---|
| *WMA_Class_5* - S&P 500 | $8.64 \times 10^{-5}$ | $8.41 \times 10^{-5}$ |
| *WMA_Class_5* - DJIA | $8.31 \times 10^{-5}$ | $6.52 \times 10^{-5}$ |
| *PC_Class_10* - S&P 500 | $3.65 \times 10^{-5}$ | $1.87 \times 10^{-5}$ |
| *PC_Class_10* - DJIA | $3.87 \times 10^{-5}$ | $2.34 \times 10^{-5}$ |

Table 2: Test losses obtained for the transformer and RNN networks for classification tasks.

As shown the losses achieved by both systems are fairly similar. With the transformer only really seeing improvements for classification. However, there are two aspects which should be mentioned. Firstly, transformer networks have been shown to scale well to longer sequences. Thus, only because the RNN can match the accuracy for smaller sequences, this does not necessarily mean they are equally suited for the task. Secondly, when comparing the training time for the RNN against the transformer, the transformers features truly shine. In nearly all tests the transformers training time was nearly half when compared to the RNN. This is due to the models ability to parallelise whilst training.

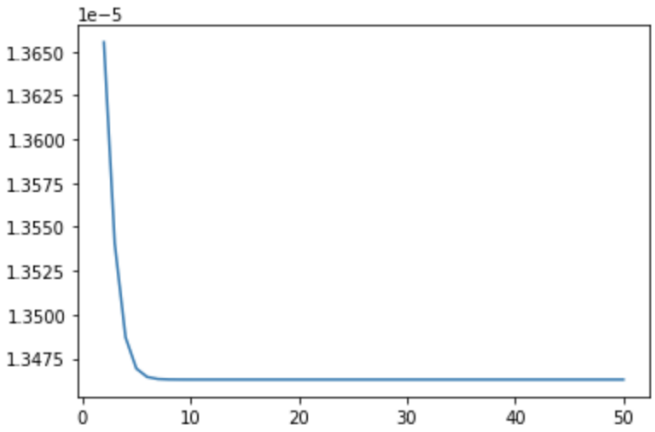


Fig. 12: S&P 500 classification validation loss for a five day prediction using chart price patterns for each epoch.

## 6 Concluding Remarks and Discussion

As this paper demonstrates, the ability of a transformer network to model dependencies in a sequence, stretches beyond the field of natural language. The main objective of this paper was to show, that with little augmentation, these capabilities could be applied to a complex time-series forecasting problem. Namely, stock market prediction.

The results obtained from this project highlight a transformers ability to produce an accurate model of forecasting, when applied to the regressive task of forecasting stock prices. As previously mentioned, due to the high training speed given through its parallelised architecture, the application of transformers at a commercial level could introduce a new level of high frequency trading. This is especially true when considering the work of Radford et al. (2019), which shows a transformers ability to accurately model dependencies only increases with the size of the network.

In addition to this, the data used in this project was of a limited, and arguably inadequate quality. This latter point is due to the already discussed random walk hypothesis, which if true means the use of technical data will never produce accurate results (Malkiel 1973). Thus, if institutions with access to high quality fundamental data, take advantage transformers, this may set a new standard for state of the art.

Although the classification results obtained in this study were inconclusive, a clear argument can be made for continued study. Firstly, the primary objective of this paper was to evaluate the regressive abilities of the transformer. Thus the classification systems developed could be improved. For example, log returns could be used as a classification. In addition, better systems which utilise more factors such as oscillators, or a wider variety of chart patterns, may prove more fruitful.

One aspect which should be noted is the effect of the adjustable learning rate optimiser, proposed by Vaswani et al. (2017), on the validation results of regression. It is clear that for smaller sequences and dimensional representations, the optimiser is not well suited. This is especially clear when working with normalised data, as this project did, in which the adjustments required are small. Thus, it is my recommendation that simpler decay based optimisers are used, when working with lower dimensional data. By doing so, the initial spike in loss can be avoided, and so convergence may occur on more optimal parameters. Of course this does come at the risk of being stuck in a local optima, however, the results obtained by this project show that the initial spike produced takes too long to recover from.

To conclude, the results obtained from this study prove the main hypothesis that transformers have the ability to accurately forecast stock series data. These results have been verified, by using two independent data sets, and two different representations. Furthermore, the increase in accuracy gained from using moving average data shows, that if applied correctly, transformers and domain standard analysis techniques can become a powerful predictor. Although this statement is not verified by the classification data, this merely shows that the right techniques need to be applied.

## 7 Future Work

There are three key aspects of this project which require further study. Firstly, given the correct hardware resources, an evaluation of larger transformer networks should be conducted. This should also include an increase in the sequence length of financial data, to detect whether the model can isolate any long term dependencies not considered within this work. Secondly, the ability of the transformer to produce trading signals should be further investigated. It is my belief that given fundamental indicators, this network could produce accurate buy and sell signals. Chart pattern signals should also be further looked into, given a larger, high frequency trading data set.

The last area of study, which I personally wish to conduct in the future, is a revisitation to the idea of a preliminary layer between the encoder and decoder. This idea was initially abandoned as a result of poor results on initial synthetic data. Due to the time constraints of the project, I therefore elected to abandon this model, in favour of the other two augmentations. However, as described in my analysis, the main problem encountered with the classification data was that the output layer. Specifically, the layer could not accurately generate the desired signals from the prediction. Thus, this method may have the ability to solve this problem. As if the encoders job was to produce an encoding of the input in terms of buy and sell signals, this would leave the decoder to learn how to map said signals into predictions.

# 8 Reflection

On reflection I am pleased with the end result of this project. The final product is something I am overall proud of, especially as this is the first time I have implemented code directly from a paper.

One aspect I have noted needs improving, is knowing when I have done enough research. When this project started, I felt very out my depth, as I had never even encountered the word transformer within the field of machine learning before. This meant that I arguably wasted more time than I needed to reading the background around this model. Including learning about encoder-decoder architectures, attention mechanism etc. On reflection, although this has given me an in depth understanding on how transformers operate, and the work they are built on, in a time constrained project such as this I should have simply focused on the transformer. This would have given me more time to program, as I spent the first couple weeks simply reading, still feeling out of my depth.

The upside to this however, is that I not only produced a project I am happy with, but also gained a lot of knowledge and understanding about a field I was before unacquainted with.

In general, I feel as though this project has been managed well. I set aside enough time to generate a solid code-base, and produce my report. Thus, I consider this summer a personal success.

# Bibliography


Ba, J. L., Kiros, J. R. & Hinton, G. E. (2016), 'Layer normalization'.

Bahdanau, D., Cho, K. & Bengio, Y. (2015), Neural machine translation by jointly learning to align and translate, *in* Y. Bengio & Y. LeCun, eds, '3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings'.
**URL:** *http://arxiv.org/abs/1409.0473*

Celella, E. (2020), 'Algorithmic application to stock trading methods'.

Child, R., Gray, S., Radford, A. & Sutskever, I. (2019), 'Generating long sequences with sparse transformers', *CoRR* **abs/1904.10509**.
**URL:** *http://arxiv.org/abs/1904.10509*

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. & Bengio, Y. (2014), Learning phrase representations using RNN encoder – decoder for statistical machine translation, *in* 'Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)', Association for Computational Linguistics, Doha, Qatar, pp. 1724–1734.
**URL:** *https://www.aclweb.org/anthology/D14-1179*

Chollet, F. et al. (2015), 'Keras'.
**URL:** *https://github.com/fchollet/keras*

de Prado, M. (2018), *Advances in Financial Machine Learning*, Wiley.

Fama, E. F. (1970), 'Efficient capital markets: A review of theory and empirical work', *The Journal of Finance* **25**(2), 383–417.

Fama, E. F. & French, K. R. (1996), 'Multifactor explanations of asset pricing anomalies', *The Journal of Finance* **51**(1), 55–84.

Jansen, S. (2018), *Hands-On Machine Learning for Algorithmic Trading: Design and implement investment strategies based on smart algorithms that learn from data using Python*, Packt Publishing.

Journal, W. S. (n.d.*a*), 'Dow jones industrial average historical prices'.
**URL:** *https://www.wsj.com/market-data/quotes/index/DJIA/historical-prices*

Journal, W. S. (n.d.*b*), 'Sp 500 index historical prices'.
**URL:** *https://www.wsj.com/market-data/quotes/index/SPX/historical-prices*

Kaboudan, M. A. (2000), 'Genetic programming prediction of stock prices', *Computational Economics* **16**(3), 207–236.

Klein, G., Kim, Y., Deng, Y., Senellart, J. & Rush, A. M. (2017), Opennmt: Open-source toolkit for neural machine translation, *in* 'Proc. ACL'.
**URL:** *https://doi.org/10.18653/v1/P17-4012*

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X. & Yan, X. (2019), Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting, *in* H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox & R. Garnett, eds, 'Advances in Neural Information

Processing Systems 32', Curran Associates, Inc., pp. 5243–5253.
**URL:** *http://papers.nips.cc/paper/8766-enhancing-the-locality-and-breaking-the-memory-bottleneck-of-transformer-on-time-series-forecasting.pdf*

Lim, B. & Zohren, S. (2020), 'Time series forecasting with deep learning: A survey'.

Lim, B., Zohren, S. & Roberts, S. (2019), 'Recurrent neural filters: Learning independent bayesian filtering steps for time series prediction', *arXiv preprint arXiv:1901.08096* .

Lo, A. W. & MacKinlay, A. C. (1999), *A Non-Random Walk Down Wall Street*, Princeton University Press.

Lo, A. W., Mamaysky, H. & Wang, J. (2000), 'Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation', *The journal of finance* **55**(4), 1705–1765.

Luong, M., Pham, H. & Manning, C. D. (2015), 'Effective approaches to attention-based neural machine translation', *CoRR* **abs/1508.04025**.
**URL:** *http://arxiv.org/abs/1508.04025*

Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P. & Shroff, G. (2016), 'Lstm-based encoder-decoder for multi-sensor anomaly detection'.

Malkiel, B. G. (1973), *A Random Walk Down Wall Street*, Norton, New York.

Nenortaite, J. & Simutis, R. (2004), Stocks' trading system based on the particle swarm optimization algorithm, *in* M. Bubak, G. D. van Albada, P. M. A. Sloot & J. Dongarra, eds, 'Computational Science - ICCS 2004', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 843–850.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. & Lerer, A. (2017), 'Automatic differentiation in pytorch'.

Qi, M. (1999), 'Nonlinear predictability of stock returns using financial and economic variables', *Journal of Business  Economic Statistics* **17**(4), 419–429.

Qin, Y., Song, D., Chen, H., Cheng, W., Jiang, G. & Cottrell, G. W. (2017), A dual-stage attention-based recurrent neural network for time series prediction, *in* 'Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17', pp. 2627–2633.
**URL:** *https://doi.org/10.24963/ijcai.2017/366*

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. & Sutskever, I. (2019), 'Language models are unsupervised multitask learners', *OpenAI Blog* **1**(8), 9.

Rangapuram, S. S., Seeger, M. W., Gasthaus, J., Stella, L., Wang, Y. & Januschowski, T. (2018), Deep state space models for time series forecasting, *in* S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi & R. Garnett, eds, 'Advances in Neural Information Processing Systems 31', Curran Associates, Inc., pp. 7785–7794.
**URL:** *http://papers.nips.cc/paper/8004-deep-state-space-models-for-time-series-forecasting.pdf*

Schwager, J. (1999), *Getting Started in Technical Analysis*, Getting Started In..., Wiley.

Sukhbaatar, S., Grave, E., Bojanowski, P. & Joulin, A. (2019), Adaptive attention span in transformers, *in* 'Proceedings of the 57th Annual Meeting of the

Association for Computational Linguistics', Association for Computational Linguistics, Florence, Italy, pp. 331–335.
**URL:** *https://www.aclweb.org/anthology/P19-1032*

Sutskever, I., Vinyals, O. & Le, Q. V. (2014), Sequence to sequence learning with neural networks, *in* Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence & K. Q. Weinberger, eds, 'Advances in Neural Information Processing Systems 27', Curran Associates, Inc., pp. 3104–3112.
**URL:** *http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u. & Polosukhin, I. (2017), Attention is all you need, *in* I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan & R. Garnett, eds, 'Advances in Neural Information Processing Systems 30', Curran Associates, Inc., pp. 5998–6008.
**URL:** *http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf*

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Jarrod Millman, K., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, İ., Feng, Y., Moore, E. W., Vand erPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P. & Contributors, S. . . (2020), 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python', *Nature Methods* **17**, 261–272.

Wallbridge, J. (2020), 'Transformers for limit order books'.

Wang, Y., Smola, A., Maddix, D. C., Gasthaus, J., Foster, D. & Januschowski, T. (2019), 'Deep factors for forecasting', *arXiv preprint arXiv:1905.12417* .

Wes McKinney (2010), Data Structures for Statistical Computing in Python, *in* Stéfan van der Walt & Jarrod Millman, eds, 'Proceedings of the 9th Python in Science Conference', pp. 56 – 61.

# A Chart Patterns

This section details the geometric properties of chart patters described by Lo et al. (2000).

Head and shoulders:

$$HS = \begin{cases} E_1 \text{ is a maximum.} \\ E_3 > E_1, E_3 > E_5 \\ E_1 \text{ \& } E_5 \text{ within 1.5\% of their average.} \\ E_2 \text{ \& } E_4 \text{ within 1.5\% of their average.} \end{cases}$$

Inverse head and shoulders

$$IHS = \begin{cases} E_1 \text{ is a minimum.} \\ E_3 < E_1, E_3 < E_5 \\ E_1 \text{ \& } E_5 \text{ within 1.5\% of their average.} \\ E_2 \text{ \& } E_4 \text{ within 1.5\% of their average.} \end{cases}$$

Broadening top:

$$BTOP = \begin{cases} E_1 \text{ is a maximum.} \\ E_1 < E_3 < E_5 \\ E_2 > E_4 \end{cases}$$

Broadening bottom:

$$BBOT = \begin{cases} E_1 \text{ is a minimum.} \\ E_1 > E_3 > E_5 \\ E_2 < E_4 \end{cases}$$

Triangle top:

$$TTOP = \begin{cases} E_1 \text{ Is a maximum.} \\ E_1 > E_3 > E_5 \\ E_2 < E_4 \end{cases}$$

Triangle bottom:

$$TBOT = \begin{cases} E_1 \text{ Is a minimum.} \\ E_1 < E_3 < E_5 \\ E_2 > E_4 \end{cases}$$

Rectangle top:

$$RTOP = \begin{cases} E_1 \text{ Is a maximum.} \\ \text{tops are within 0.75\% percent of their average.} \\ \text{bottom are within 0.75\% percent of their average.} \\ \text{lowest top } > \text{highest bottom,} \end{cases}$$

Rectangle bottom:

$$RBOT = \begin{cases} E_1 \text{ Is a minimum.} \\ \text{tops are within 0.75\% percent of their average.} \\ \text{bottom are within 0.75\% percent of their average.} \\ \text{lowest top } > \text{highest bottom,} \end{cases}$$

## B Running the Project

The following section details how to run the code developed for this project. The code base can be found at the following repository address:

*https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2019/emc918*

The project is written in Python 3. Thus in order to run the code Python 3 must be available on the machine as well as the following libraries:

- Jupyter Notebook
- Scipy (Pandas, Matplotlib etc.)
- Pytorch

All code can be found in the code directory. Within this directory are two sub directories called Transformer and Data. The former directory contains all code developed to build the transformer network, and the latter contains the data sets as well as all code for prepossessing. The file names in each of these directories explain the purpose of each files code.

To run the project a Jupyter notebook server requires running in the Code directory, in order to launch the *main.ipynb*. From this file, all project code can be run. In order to run a specific test, the following steps must be taken:

1. Run the Imports cell.
2. Uncomment the filename for the data set you wish to use.
3. Choose a type of prepossessing to perform on the data from the regression or classification data set subsections. Only run one of these cells.
4. Run all cells within the Prepare Set subheading.
5. For the Transformer, run the first cell in the Transformer section. Then run the cell commented either regression model, or classification model. The type of model you select must match the prepossessing type ran (e.g. if a cell in the Regression Dataset subheading was run, the regression model must be selected). Once a model has been selected run the last cell in the Transformer section. Once this has all be completed run the cell in the Transformer training subsection to run the selected test.
6. To run a test using the baseline RNN model, follow the same method described in step 5 of this guide within the Baseline model section. Note that due to this model using a pre-built RNN from the Pytorch library, training may need to be manually set. This varies depending on which device the training takes place.

All code was produced independently. The work of Klein et al. (2017) was used to compare transformer implementations, to ensure correctness, with some inspiration drawn from their method. Specifically this was the idea to create a class to store each batch, masking techniques, and the adjustable optimiser.