Stock Price and Trend analysis with Transformer-based architectures

Benjamin Irving
irving.b@northeastern.edu

December 2022

Abstract

The automation of stock market analysis will play an increasingly large role in the future of the economy. Funds at the corporate and private level are influenced by automated trades, by decision and because other agents deploy such methods. Stock market data is largely sequential, which thus gives the transformer architecture potential to be used to great effect in analysing trends due to its ability to create long range dependencies. The inputs used in the following paper consist of price and Tweet data, employing a general encoder architecture with an LSTM, followed by a temporal attention mechanism. The model below achieves state of the art results with binary classification, predicting 86.77% of the price changes on my test set correctly. The code is available at https://github.com/Lysander-curiosum/michinaga

1 Introduction

Stock movement prediction and analysis have long had active interest in the research community, largely due to the potential financial benefits [1]. Initially, research focused on time-series coupled with historic price data [2]. The problems of price prediction and trend identification proved to be enormously complex, seemingly erratic, the sum of the decisions of millions of people, made for a plethora of reasons, some consciously and others not. A human agent is not necessarily rational. Often, stocks are purchased in response to the change or trend in a technical indicator. On the other hand, buyers and sellers commonly make trades for reasons beyond the mathematical, among which include superstition, random impulse, or a growing trend in an online forum such as Twitter [3].

To comprehend such fluctuations, several studies have employed natural language techniques to financial markets, giving birth to the field of natural language-based financial forecasting (NLFF). Many of these studies have focused on public news [4][5][6]. Social media presents more time-sensitive information from active investors. Thus, for short term analysis, many researchers

have begun to focus on Tweets for feature extraction [3][7][8], through which some have combined NLP techniques with traditional analysis on price data [9][10]. Since Tweets often correspond to events as they happen in real time, such data is better suited for smaller windows, a 5 or 10 day period of price information.

Combining the features extracted through NLP methods with price data has shown promising results [9]. However, it is ineffective to feed the concatenated information to the model without encoding temporal dependencies [11]. It is important to know when the tweets and prices occurred in relation to each other to avoid losing their expressive power. Closing prices and tweets which occur shortly before the target day will have a greater effect then the other auxilary values. Thus, in the model that I present below, the key component, along with the the concatenation processor which combines the tweet and price data, is the temporal attention mechanism. I focus on an input where each target day is preceded by a mere 5 day lag period, to account for the lack of long term foresight presented by tweets. The feature extraction of the Tweets themselves is processed by a traditional Transformer encoder [12]. I also employ the use of an LSTM [13], combining all of these components into a hybrid-transformer architecture. The goal of the primary experiment is to make a binary prediction about a stock, whether or not it will increase on the following day.

The paper is organized as thus: in the following section, I describe the data set that I used, which is followed by a description of the model itself in section 3. In section 4, I discuss my experiments and the results that I found, followed by a conclusion which discusses challenges with the current model, and some directions that future research could take. This paper largely builds upon the TEANet model [11].

2 Data

In finance, stocks are organized into 9 categories, which include Basic Materials, Consumer Goods, Healthcare, Services, Utilities, Conglomerates, Financial, Industrial Goods and Technology. High volume tickers are often discussed more on Twitter, so the data set that I employed pulls the all of the stocks from the Conglomerates category and the top 10 stocks in terms of capital size from the other 8 categories, over a two year period from 1/1/2014 to 1/1/2016 Following the manner in which the creators of the data set delineated increases and decreases, days which have a movement percent greater than 0.55 are labeled as 1, or an increase, while stocks which have a movement percent below -0.5 are labeled as 0, or a decrease. Any data point which has a movement percent between these two values are discarded. In terms of the Tweets that accompany the price data, they were gathered based on the NASDAQ ticker symbols present in the tweets themselves. The dataset[1] was created for the Stock Movement Prediction from Tweets and Historical Prices paper [9].

[1] https://github.com/yumoxu/stocknet-dataset

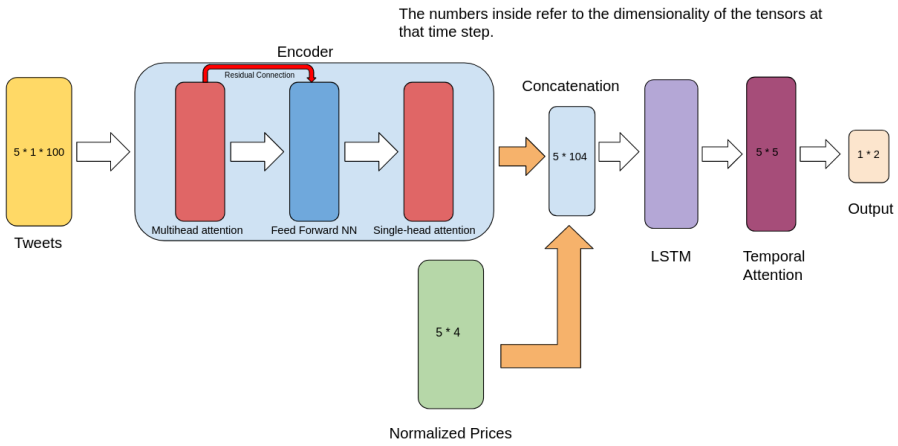

Figure 1: Above is the complete model. As stated in the image, the numbers inside each stage represent the dimension of the input at that stage. Of course, these numbers fall under the assumption that a lag period of 5 is being used, along with 1 averaged Tweet for each auxiliary trading day. They also do not account for batch size, which would simply add another dimension to the inputs.

3 Model Architecture

The primary aim of the model is to predict whether or not a stock will experience a positive or negative movement percentage on a given target day. To inform this prediction, I employ the use of a 5-day lag period, meaning that the input to the model consists of the 5 previous days of price information, with a various number of Tweets for each of those days. If td represents the target day, then the range of the input can be labeled as follows:

$$[td - \Delta td, td - 1] \tag{1}$$

Δtd represents the lag period. Each day of price data is represented by a vector p of length 4, where the values are:

$$p = [adjc, open, high, low] \tag{2}$$

The adjusted closing price simply reflects corporate actions and other factors which may not be represented by the closing price. These vectors are then normalized using the following equation

$$p_{new} = \frac{p}{p^c_{td-1}} - 1 \tag{3}$$

where p^c_{td-1} represents the raw adjusted closing price for the previous day. For the Tweet data, I used the Twitter embedding from the FLAIR NLP library [14]. These embeddings are then processed using a traditional Transformer encoder [12].

3.1 The Transformer

The Transformer is an attention based architecture introduced by the Google team in 2017 [12]. The aim of the Transformer was to extract detailed features from sequential data without the use of recurrence, instead relying completely on attention. The main advantages of the Transformer model is that it can create long range dependencies between disparate parts of a given input, while also being resistant to severe effects from anomalous data. This is important to consider regarding the stock market, where patterns could be found across many days, or even weeks [5]. Furthermore, the architecture is extremely parallizable, and can utilize distributed computation over many GPUs to fine-tune its weights at a much faster pace. Of primary interest, regarding my model, is the attention mechanism.

3.2 Attention

The attention mechanism takes its biological inspiration from the fovea, the human field of vision is centered. In the mathematical context, it can be described as mapping a set of key-value pairs to an output, where the query, key, and values are also represented by vectors. Specifically, I use the mechanism known as scaled-dot product attention, which can be defined as follows. Let Q, K, and V be the query, key, and value matrices, respectively, where each matrix has dimensions $n \times d$, and d is the dimension of the query, key, and value vectors, and n represents the number of input tokens to be processed. The scaled dot product attention function is then defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{4}$$

In this equation, the dot product of the query matrix and the transpose of the key matrix is calculated, and the result is divided by the square root of the dimensionality of the query and key vectors ($\sqrt{d}$). This scaled dot product is then passed through the softmax function, which normalizes the values to sum to 1, creating a set of weights that can be used to scale the values in the value matrix. Finally, the scaled values from the value matrix are multiplied by the weights to produce the final output of the attention function.
However, with a single attention function, we are limited in the parts of the input that the model can build specific dependencies for. Thus, we employ multiple scaled dot-product attention mechanisms in parallel, in a process known as multihead attention, which is defined as:

$$\text{Multihead}(Q, K, V) = \text{concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{5}$$

where head_i is the output of the i-th attention head, which is defined as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{6}$$

Multihead attention creates Q, K, and V matrices which become specialized at

processing vectors from different parts of the input.
In the model, I use 5 attention heads to process the Tweet information. The dimension of the embedded Tweets is $\mathbb{R}^{100}$, so to capture detailed feature information I use a dimension of $\mathbb{R}^{100}$ in the multi-head attention mechanism.

3.3 Feed Forward Neural Network

After the embedded Tweets are processed by this initial step, they are fed into a fully-connected feed forward neural network with a residual connection, meaning that they are fed into the network alongside the original embedded Tweets. This connection serves to alleviate the vanishing gradient problem [15]. The FFN consists of two linear layers, along with a ReLU connection in between.

$$\mathrm{FFN}(x) = \max\left(0, xW_1 + B_1\right) W_2 + B_2 \tag{7}$$

3.4 Concatenation

Before the Tweet and price data are concatenated, the output of the FFN is processed by a self-attention layer, allowing the model to identify important information from the already prioritized results of the multi-head attention mechanism. The output is then concatenated to the normalized price vectors, each trading day represented by the vector

$$c = [t_d, p_d] \tag{8}$$

where t_d represents the Tweets processed by the encoder on an arbitrary trading day d, and p_d represents the normalized price values for that same trading day d. The group of concatenated input vectors for the auxilary trading days are then fed into the LSTM.

3.5 LSTM

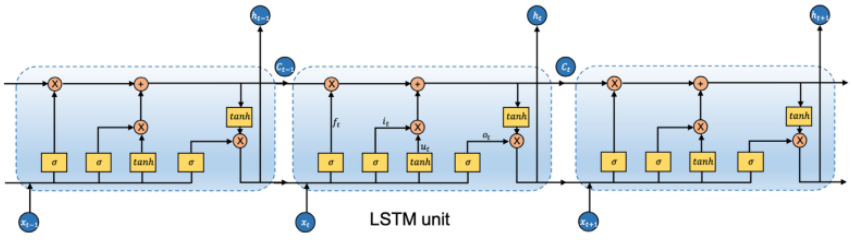

Figure 2: The LSTM [11]

LSTM (Long-Term Short-Term memory) networks are variations of RNNs, that solve the vanishing gradient problem. RNNs are an essential area of study pertaining to sequential data. Introduced by Hochreiter [13] in 1991, they proved their effectiveness at creating long range dependencies in data. An LSTM is made up of cell states which are controlled by 3 gates, the input gate, output

gate, and the forget gate. Looking at the figure above, x_t represents the input data for the current time step t, while h_t and c_t represent output value and cell state at the current time step. f_t, o_t, and i_t correspond to the forget gate, output gate, and input gate respectively. The network first gets rid of information, using the forget gate. Then, new information is added to the cell state using the input gate, after which the final cell state is determined by the output gate.

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{9}$$

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{10}$$

$$Cprime_t = \tanh(W_c * [h_{t-1}, x_t] + b_c) \tag{11}$$

$$C_t = f_t * C_{t-1} + i_t * Cprime_t \tag{12}$$

$$O_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) \tag{13}$$

$$h_t = O_t * \tanh(C_t) \tag{14}$$

The output of the LSTM is then concatenated to the LSTM input (again, the residual nature of the input alleviates issues with the vanishing gradient). This value is finally fed into the last portion of the model, to which we now must turn.

$$l_t = [x_t, h_t] \tag{15}$$

3.6 Temporal Attention

Temporal attention allows the model to create dependencies in the input across time. Specifically, my mechanism creates predictions for the auxiliary trading days, which are used in the final prediction of the target day. Th concatenated inputs are first fed into a linear layer with a hyperbolic tangent activation function, to avoid the vanishing gradient problem.

$$g_t = tanh(W_g * l_t + b_g) \tag{16}$$

These temporal input values are then processed into two different scores, an information score and a dependency score. The informational seeks to extract relevant features from the input, and project each auxiliary day into 1 value.

$$v_i = w_i^t * tanh(W_{g,i} * g + b_{g,i}) \tag{17}$$

The dependency score seeks to prioritize information in the input based off a relationship to the final auxiliary trading day, which is the closest information temporally to our prediction. Thus, dependencies between this value and previous days is of importance. w_i is a weight matrix $\mathbb{R}^{dim \times 1}$, while $W_{g,i}$ is a matrix $\mathbb{R}^{dim \times dim}$.

$$v_d = g_{td}^T * tanh(W_{g,d} * g_{g,d} + b_{g,d}) \tag{18}$$

g_{td}^T represents the temporal input for the last auxiliary trading day, which is of the dimension $\mathbb{R}^{dim \times 1}$. $W_{g,d}$ is a matrix of the dimension $\mathbb{R}^{dim \times dim}$. The

information and dependency score are then multiplied in a point wise fashion to produce a feature extraction vector to be used in the final prediction.

$$v_f = v_i \odot v_d \tag{19}$$

For the auxiliary trading days, I project the initial temporal inputs into 2 dimensions, which represent the two classes. Then I apply a Softmax function to get the probabilities for the two classes, which is further projected into 1 dimension to produce a feature extraction vector for the final prediction.

$$u = W_n * softmax(W_u * g_t + b_u) + b_n \tag{20}$$

$W_u \in \mathbb{R}^{dim \times 2}$, $W_n \in \mathbb{R}^{dim \times 1}$. To make the final auxiliary prediction, the auxiliary trading prices are multiplied by the feature extractions, which is then concatenated to the final auxiliary trading day temporal input. This value is then fed into a linear projection into 2 dimensions, where the softmax function is applied to produce a final binary prediction.

$$q = softmax(W_q * [u * v_f, g_{td}] + b_q) \tag{21}$$

$W_q \in \mathbb{R}^{dim \times 2}$. q will be a tensor of length 2, where the first value represents the probability of the prediction being an increase, and the second value represents the probability of the prediction being a decrease.

3.7 Deviations from the TEANet Model

The architecture described above is largely based off the TEANet Model[11]. There are some notable differences. I use layer normalization between the linear layers, and my text encoder use a traditional self-attention mechanism rather then the variation proposed in the paper. The paper itself has some confusing notation, particularly how they use the price values for the target day in their inferences. This would seem to defeat the purpose of the model, for the eventual purpose would be to perform inferences on lag periods up to the current day, predicting the price value of the next day. In such a scenario, one would obviously not have detailed price information about the price of the next day. Furthermore, in their temporal attention mechanism, they do not describe how the use the auxiliary trading days. I project them into a single dimension. There are virtually no implementation details listed in the original paper. The main point of inspiration comes from their temporal attention mechanism, which they actually took from another paper [10].

4 Experiment

The central goal of this paper was to compute accurate binary price predictions based on the auxiliary trading days. I used variations of the data set above, in terms of how the training and test sets were organized. All of the runs for binary price prediction were completed on the above runs.

4.1 Experimental setup

The model and training loop relied heavily on the PyTorch[16] library. I used the Adam optimizer with a learning rate of 0.001, and a cosine learning rate scheduler. I divided the original data set into a 75/25 train-test split, which resulted in 49873 training points, and 16625 test points. Many previous papers[9][11] employed the use of the Matthews Correlation Coefficient [17] (MCC), which can be defined as

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{22}$$

MCC takes true positives, false positives, true negatives, and false negatives into account. It measures the quality of binary predictions, returning a value between -1 and 1. -1 represents a completely incorrect prediction, 0 and random prediction, and 1 a correct prediction. MCC is more accurate then basic accuracy, because it takes false negatives and false positives into account. I measured the performance of the model on these two metrics. In terms of the base-model specifics, I used 6 stacked text encoders, with only one LSTM and one temporal attention mechanism. The deeper-model had 30 stacked text encoders, with 5 LSTMS. The model-without tweets had one LSTM, while the model without price had 6 stacked text encoders. All of the models were run on GPU, with 300 epochs.

4.2 Performance

	Accuracy %	MCC
base-model	86.77	0.54
deeper-model	94.3	0.76
model-without-tweets	64.3	0.12
model-without-prices	74.57	0.03

Table 1: Here are the results from the first binary experiment with a 5-day lag period, measured on the test set.

Looking above, the deeper-model which utilizes both Tweets and price values performs the best, guessing 94.3% of the test set correctly. The MCC for this model is 0.54. For the base model, 86.7% is a largely successful accuracy total. The original TEANet model [11] had slightly different specifications, and only achieved 65.16% on the same test set. The authors did not specify the number of epochs that they used, but my strategy centered around maximizing performance, and it seems safe to presume that they used far fewer training loops. Comparing the performance of my model to some other generative strategies on the same data set, CAPTe net was the second most perform-ant deep model behind TEAnet [11][7], and it achieved a 64.22% accuracy on the

same data set. Again, in the face of my results, there appears to be a vast fall off in performance, which is most likely due to the number of epochs over which I trained my model. To give perhaps a better idea of the true performance of my model, after 1 epoch my base-model achieved a performance of 60.35%. The deep-model performed far worse, achieving an accuracy of 52.03%. Furthermore, the model performs better on the 5-day lag period than the 10-day, with a performance drop off of around 10%. The base-model achieved a 75.98% accuracy on the test set with a 10 day lag period, which shows us that the Tweets present short-term value in terms of their predicative power. This makes sense, given that people do not often Tweet with foresight[10].

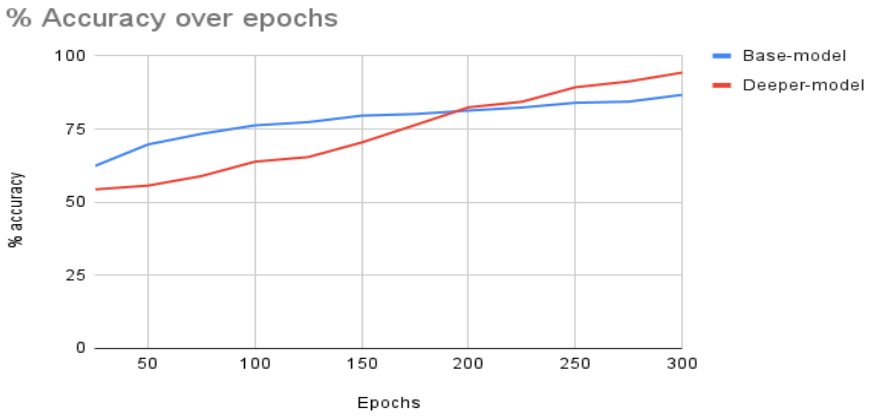

Figure 3: Here is a graph showing the accuracy as it develops over the 300 epoch period.

Looking at Figure 3 above, the deeper model only passed the base-model in accuracy at around 200 epochs or so. This exemplifies the reason that I trained the models over so many epochs. Transformers, by nature, are low-pass filters, meaning that they are resistant to change. The deeper the model, the higher the training requirement [12]. To show what the architecture could really do, I wanted to run a longer experiment. Examining Table 1 more closely, what stands out as most interesting is the accuracy of the model without Tweets, versus the accuracy of the model without price values. The model without Tweets performed well, achieving 74.57% accuracy on the test set, with an MCC value of 0.12. The model without price values achieved only a 64.3% accuracy on the test set, with an MCC value of 0.03. This discrepancy emphasizes the importance of the semantic data in the effectiveness of the model. The Tweets play a huge role in the effectiveness of the model, appearing to be more important to the overall performance then the price data. This is an extremely surprising result, contrary to the assumptions of the traditional technical analyst. It indicates that the sentiment found on Twitter is more of an accurate guide to price prediction than the previous prices themselves.

5 Conclusion

The model proved to be very successful on the data set, surpassing the paper from which it was inspired in performance by a good margin (albeit, with likely far more training). That it was able to outperform SOTA models speaks to the powerful nature of Tweets as they pertain to stock market prices. There could be many reasons for this, one of which could be that the casual investor looks to social media when making a decision. Another could be that many quantitative algorithms performing large amounts of trades take Tweets into account. Whatever the case, semantic feature extraction is clearly a potent tool in the analysis of the stock market.

5.1 Challenges

One obstacle that I struggled with was the notation in the paper that inspired my model architecture. Many of the matrix dimensions were not clearly labeled, so in the implementation I had to make my own decisions based on the logic of information extraction. Furthermore, the data was not organized at all. I had to write the scripts to process them, which felt like a frustrating waste of time. The largest challenge that I faced came with the training itself. I was experiencing huge performance drop offs after a certain number of epochs usually at around 70% training accuracy on the training set. The reason for this was saddle points, where the back prop would lead the gradients to a local minimum from which they could not escape. To fix this issue, I had to change the architecture of the model, adding more layer normalizations, along with changing the temporal attention process (specifically which the auxiliary predictions). I also added a learning rate scheduler to the training process. I really wanted to test the model on a data set compiled from more recent Tweets, perhaps even those in live time. However, I was never able to get elevated Twitter API access, even though I met the requirements. These challenges prevented me from running the wide breadth of experiments that I had in mind.

5.2 Extensions

I believe that this model could really shine in the area of trend identification, where it could be deployed on other indicators, such as the RSI index and MACD (Moving Average Convergence Divergence), to identify stocks which are bearish and bullish. This area of technical analysis is far more interesting than price-prediction, because its more resistant to chaotic factors which pop up randomly. In other words, it is easier to predict with certainty that a stock is gaining momentum in a bullish direction, rather than that it is going to experience a positive increase on some particular day. This task also hold far more practical utility. Most investors, be they private funds, quantitative algorithms, or individuals, want to hold stocks over a larger period of time then a few days in order to make a real profit. This is the next step for this project, and I plan to explore it in the coming months. In terms of research extensions, one could use

a vision transformer on graphs corresponding to technical indicators, or even to pictures pertaining to the companies in general. I also want to try some different temporal architectures, and perhaps remove the LSTM all-together to make the model more light weight. The text encoder could also be bi-directional, to attend to the inputs in a more detailed fashion and perhaps find more confirmable trends in the semantic data. I could do this with a BERT architecture. It would be interesting to play with the model on a more robust data set as well, using semantic information from more sources then Twitter to better capture long-standing information that may inform the direction of the stock over a longer period.

References

[1] Listening to chaotic whispers: A deep learning framework for news-oriented stock trend prediction. *CoRR*, abs/1712.02136, 2017. Withdrawn.

[2] Hirotugu Akaike. Fitting autoregressive models for prediction. *Annals of the Institute of Statistical Mathematics*, 21:243–247, 1969.

[3] Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models. *CoRR*, abs/1908.10063, 2019.

[4] Huy Huynh, L. Minh Dang, and Duc Duong. A new model for stock price movements prediction using deep neural network. pages 57–62, 12 2017.

[5] Mathias Kraus and Stefan Feuerriegel. Decision support from financial disclosures with deep neural networks and transfer learning. *CoRR*, abs/1710.03954, 2017.

[6] Xiaodong Li, Pangjing Wu, and Wenpeng Wang. Incorporating stock prices and news sentiments for stock market prediction: A case of hong kong. *Information Processing Management*, 57:102212, 02 2020.

[7] Jintao Liu, Hongfei Lin, Xikai Liu, Bo Xu, Yuqi Ren, Yufeng Diao, and Liang Yang. Transformer-based capsule network for stock movement prediction. In *Proceedings of the First Workshop on Financial Technology and Natural Language Processing*, pages 66–73, Macao, China, August 2019.

[8] Sahar Sohangir and Dingding Wang. Finding expert authors in financial forum using deep learning methods. In *2018 Second IEEE International Conference on Robotic Computing (IRC)*, pages 399–402, 2018.

[9] Huizhe Wu, Wei Zhang, Weiwei Shen, and Jun Wang. Hybrid deep sequential modeling for social text-driven stock prediction. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 1627–1630, New York, NY, USA, 2018. Association for Computing Machinery.

[10] Hongfeng Xu, Lei Chai, Zhiming Luo, and Shaozi Li. Stock movement predictive network via incorporative attention mechanisms based on tweet and historical prices. *Neurocomputing*, 418:326–339, 2020.

[11] Qiuyue Zhang, Chao Qin, Yunfeng Zhang, Fangxun Bao, Caiming Zhang, and Peide Liu. Transformer-based attention network for stock movement prediction. *Expert Systems with Applications*, 202:117239, 2022.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[14] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649, 2018.

[15] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

[16] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[17] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.