CSC 561: Neural Networks and Deep Learning

Multivariate Time-series Forecasting on Stock Market Data

Alfred Timperley

University of Rhode Island
Department of Computer Science and Statistics
Project Report
Spring 2022

Contents

1 Abstract

Multivariate Time Series Forecasting (TSF) focuses on predicting future values based on historical contexts. Recent developments in the field utilize Transformer architectures to allow the models to scale to longer time frames. The current state-of-the-art Transformer in TSF is the *SpaceTimeFormer*. This architecture flattens the spatio and temporal aspects of the data into one long stream of variables. This allows the transformer to better learn the distinct temporal and spatial relationships between variables. This paper applies this architecture to the problem of stock price prediction on financial time series. Financial Time series forecasting is a particularly hard aspect of Time series forecasting due to the 'unpredictable' nature of the markets. We achieved interesting results that highlighted the shortcomings of our approach and the great potential for future work.

2 Introduction and Motivation

Time series forecasting is the attempt to predict future outcomes based on historical context. This task has applications in many domains whether that be science or business. Some applications exhibit obvious cyclic patterns while other applications exhibit more complex patterns or seemingly no patterns at all. Many forecasting applications also involve multivariate inputs that requires the models to jointly interpret a set of variables to make accurate predictions on future behavior. Finally, some application may also involve complex temporal relationships between and within variables which also need to be properly modeled.

Not every application exhibits all previously mentioned traits of time series data. This can allow for some liberties to be taken in the design and use of the TSF models. Whether that be designing for a single variable input, no inherent long term dependencies, or a well behaved pattern. Stock price forecasting however does not allow for such design choices. The complexity of the patterns and variety of potential input sources all but require multiple variables to attempt to properly model the problem. The temporal relationships between and within variables are also highly relevant. A change in one variable can have long lasting impacts on others, requiring accurate modeling of long term dependencies. These factors make predicting stock prices to any accuracy an extremely challenging problem in the field of finance and data science.

Methods for predicting stock prices can be split into two categories: (1) *Fundamental Analysis* or (2) *Technical Analysis*. Fundamental Analysis attempts to evaluate stocks by measuring their intrinsic value while Technical Analysis looks at the statistical trends in the stocks metrics to make predictions. Over time more and more researchers and financiers have been trying to solve this problem with Machine Learning. Recently, attempts using Deep Learning models have become popular. For example, some methods have been developed in the vein of Fundamental Analysis to analyze information such as news text, financial reports, and analyst reports. Methods deploying Technical Analysis have also been used, featuring models such as RNNs [1] and models in the LSTM [2] family. However problems have been encountered with capturing the extremely long term dependencies on financial time series. This has led to the use of adoption of the well known sequence-to-sequence models called Transformers [3] to model these Technical Analysis problems. In this paper we explore the use of state-of-the art TSF transformers in the pursuit of making accurate stock price predictions via Technical Analysis.

3 Related Work

3.1 Deep Learning in Financial Analysis

Deep learning has a history of success in stock price prediction with technical and fundamental analysis. With the advent of the internet there was an explosions in financial and alternative financial information. This supported the development of machine learning and deep learning techniques in financial predictions via fundamental analysis. One of the early attempts of this new approach was in 2009 with a predictive machine learning approach that extracted textual representations from financial news articles [4]. Another novel method was used to predict future price movements by combining data from different sources [5]. A further method used a stochastic recurrent model with an additional discriminator and attention mechanism to try to address the inherent adaptability in stocks [5].

On the other side there has been a long history of using machine learning and deep learning for technical analysis as well. These methods typically extract price-volume information for historical data. One example is an SVM-based approach for stock price prediction [6]. There have also been LSTM models [2] used to predict stock movements based solely on technical analysis indicators [7]. Some LSTM models have also been trained on historical data to discover multi-frequency trading patterns [8]. There has also been a ConvLSTM-based sequence-to-sequence framework used for stock movement prediction [9]. A further method used an Attentive-LSTM model with an attention mechanism to predict stock price movement [10] and was further developed by adding a data augmentation approach with the idea of adversarial training [11]. However an issue encountered by these LSTM-based models is that they struggle to capture extremely long-term dependencies inherent in the stock time series data. LSTM models can only distinguish about 50 positions nearby with an effective context size of about 200. This issue plagued not only technical analysis but time series analysis in general. One method to solve this is to use Transformer based models so as to better find the intrinsic long-term and complex patterns in financial time series.

3.2 Deep Learning in Time Series Analysis

Previously, the most common class of TSF models where combinations of Recurrent Neural Networks (RNNs) and one-dimensional convolutions (Conv1Ds). However with the difficulty found in improving RNNs auto-regressive training and in interpreting long-term patterns, Transformers have began to take over as the most popular class of TSF model. The most notable model out of these developments is the Informer [12]. The Informer is a transformer architecture that encodes the time series inputs with a learned temporal embedding and appends a start token to the decoding sequence of zeros that needs prediction. The mechanism of a transformer have been discussed at length in many papers so for the details of it's operation we would point you to the original paper [3]. What is more relevant and novel for this analysis however is the development and use of the SpaceTimeFormer (STF) [13].

3.3 Development of the SpaceTimeFormer

3.3.1 Long Range Transformers

Transformers attention mechanism is centered around the multi-head self attention (MSA) mechanism. Through combining linear transformations between the query, key, and value vectors, the MSA layer learns weighted relationships between its input in order to pass information between tokens. Due to the MSA matching each query to the keys of the entire sequence its

run time and memory use grows quadratically with the length of its input. This becomes a problem in domains where a long sequence length is common and the use of self-attention is promising, such as financial analysis. There have been developments in MSA-variants to deal with the long sequence problem. Many of these solutions involve creating heuristics to try to increase the sparsity the attention matrix. One such method attends primarily to the adjacent input tokens [14], or just a select few global tokens [15], or increasing distant tokens [16] [17] or a combination of the methods [18] [19]. However, these approach rely on inductive biases of the attention matrix structure and does not always apply outside of tasks in NLP.

Another approach to this problem is through approximating the MSA in sub-quadratic time while retaining its flexibility [20] [12]. The STF uses a variant of this called the Performer [21]. The Performer approximates the MSA in linear space and time with a kernel of random orthogonal features. This enables the STF to operate on long sequence data and makes it applicable to tasks with financial time series.

3.3.2 Spatiotemporal Forecasting

In multivariate TSF there are two axes of the complexity: (1) the forecast sequence duration, L, and (2) the number of variables, N, used at every time-step. As the size of N grows it becomes increasingly important to model the cross-spatial relationships between the variables. There have been methods of addressing this problem in the domain of RNN-based attention [22] [23]. In these architectures the models alternated between spatial and temporal attention layers. There has also been further work in extending this idea [24] [25] but these works rely on a predefined dependency graph akin to a GNN.

The STF deals with the difficulties of representing multiple variables per token by flattening the variables into a single token per variable, resulting in a sequence which is N times longer. The idea behind this is to revert back to the classical NLP thought of representing just one idea per token. So now the sequences are LN tokens long and the attention mechanism needs to be able to handle such an increase. This is where the approximation of the MSA from the Performer Attention layer helps the model scale to the additional length. So now with the longer input sequence the model can learn unique relationships for every variable. In addition the STF does not have different spatial and temporal attention layers and instead combines them into one. This single attention mechanism is a simpler solution and allows for the spatial relationships to change dynamically overtime, rather than relying on inductive biases to define the relationships.

3.3.3 Positional Embedding

Transformers are permutation invariant and thus need a positional embedding or other method of encoding the positional information of the input into the transformer. In traditional NLP contexts the positional embedding would be some sort of sinusoidal pattern [3]. For the STFs flattened input they would want all N tokens to have to same temporal information and could accomplish this through a modified embedding to account for the flattened sequence. However, a fixed position embedding only accounts for relative order and would not capture the intricate long term dependencies of TSF or financial time series. This is remedied by learning the positional embedding based on a *Time2Vec* [26] layer. *Time2Vec* passes a representation of absolute time through sinusoidal patterns of learned offsets and wavelengths. This helps represent periodic relationships. STF also appends a relative order index to the time input values to help distinguish between data taken at such close intervals that their *Time2Vec* embedding are

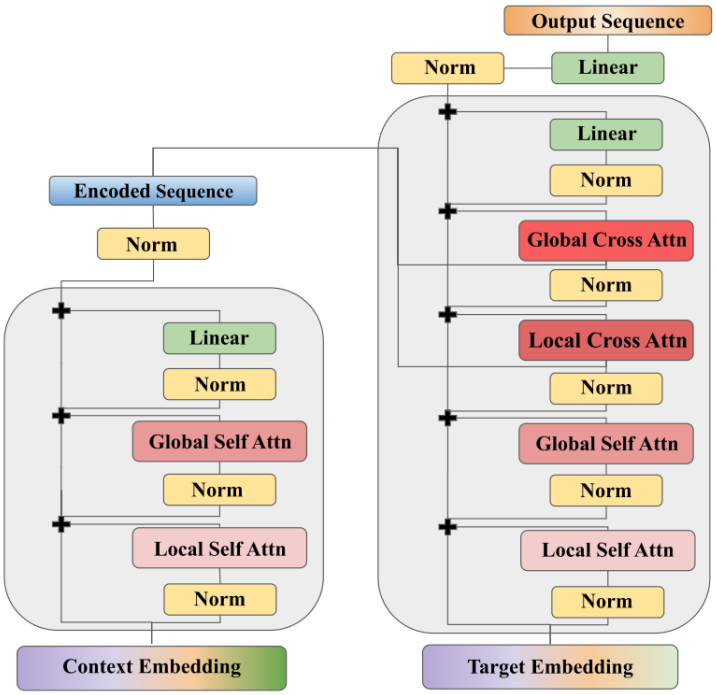

Figure 1: The SpaceTimeFormer architecture [13]

identical. In the STF they call this layer the "Value & Time Embedding" and apply it to each token in the sequence.

3.3.4 Scaling the Model

With the new sequence length of LN the MSA layer will need additional modifications to be able to handle the length. Vanilla MSA layers have a maximum sequence length of less than 1,000 tokens and aside from the highest end hardware we are limited to lengths closer to 500. This quickly becomes a problem with TSF where sequences can be thousands of time-steps long and multivariate. In particular with financial time series there can easily be tens of thousands of multivariate time-steps in a sequence. Luckily there has been work done in fast-attention mechanisms for long-sequence transformers. The Informer model [12] proposes a ProbSparse attention mechanism. However, this mechanism only allows for a sequence length of about 1k-2k tokens. Thus the STF opted for the Performer FAVOR+ attention mechanism [21] which uses a linear approximation of the soft-max attention using a random kernel method. They found that this attention mechanism paired with a multi-GPU setup was sufficient for all but the longest spatiotemporal sequences.

3.3.5 Locality

It was found that just using a global attention layer eliminated local biases that hurt performance in sequences with a large N. Thus the STF implemented implemented local attention layers that had every token attend to every token in it's own sequence and then to every token in the spatiotemporal global sequence. This can be seen in figure 1. This is different than the other architectures who split temporal and spatial attention, as here they are still computed together.

There have also been work done to improve locality specifically in the context of financial time series data. One such method is to enhance locality by implementing a multi-scale Gaussian prior [27]. The idea behind this is to implement an inductive bias for financial time series by setting the window size for heads of the MSA mechanism to different sizes. These sizes would correspond to typical temporal stock divisions. So there would be window sizes that show the last 5-days, 10-days, 20-days, or 40-days to coincide with popular trading strategies.

3.4 Sharpness Aware Minimizer Optimizer

Typically transformer models are characterized by large amounts of pre-training on extensive data-sets. This can be limiting both by resources consumption and also limit the potential tasks that transformers can be applied to. Recent work in vision transformers has shown that the Sharpness Aware Minimizer (SAM) [28] optimizer can be sufficient to allow transformers to outperform ResNets [29] without the need for extensive pre-training [30]. This development was of interest to this work due to the limited resources available to adequately pre-train a transformer on time series data.

4 Methodology

4.1 Description of Data

The data-set used to train the Transformer for this work are stocks from the Standard and Poor's 500 index (S&P 500). The S&P 500 is a stock market index tracking the performance of 500 large companies listed on stock exchanges in the United States [31]. The index ranges over all industries of the US economy from Information Technology to Materials. This data-set was chosen due to it's diversity in types of companies, large volume of transactions, and prominence in the financial world. Out of this data-set the first 100 stocks in alphabetical order were chosen to pre-train the transformer. The alphabetical order of the stocks in the list has little to do with the industry, time listed on the market, or total trade volume so it was deemed a fine metric to use to create a subset of the list.

The target stock chosen was IBM. It was removed from the training data-set for pre-training. It was chosen as the target due to it's long history of being a public company (Publicly traded since 1911 but capped at 1962 due to *yfinance*), the large volume of transactions, and it's popularity.

The data-sets were acquired from the python library *yfinance* [32]. This library provides a wrapper for accessing the Yahoo Finance public API however the package is not affiliated, endorsed or vetted by Yahoo. It's historical data goes back to 1962 so it does not include the full public history of every company. The finest granularity of it's long term historical data is daily data. *Yfinance* provides seven variables for every daily time step:

- Open: The open price that day
- Close: The close price that day
- High: The highest price that day
- Low: The lowest price that day
- Volume: The total amount of shares bought and sold that day
- Dividends: The amount paid out by dividends that day

- Stock Split: The stock split ratio that day

The Dividend and Stock Split data is exceedingly sparse in all stock data-sets but was still included in the training. This was decided due to the importance of the data points on price and the ability of the STF architecture to showcase such long term dependencies. The amount of time steps in each stocks data-set can range from 538 to 15,191 time-steps, with the high-end being capped due to yfinance data only going back to 1962.

4.2 Description of Methods

The architecture used in this project is the SpaceTimeFormer [13] with a minor modification by adding in an option for the SAM [28] optimizer. The models were trained on two Titan X GPUs from the Knuth Research Server at the University of Rhode Island. The models were pre-trained on 100 stocks from the S&P 500 data-set and each was run sequentially. Every stock ran until an early stopping condition was reached. We used Early Stopping with a patience of 10 epochs and reduced learning rates on validation loss plateaus. The training was managed using the WeightsAndBiases Machine Learning DevOps platform [33].

The following hyper-parameters was used to train both models, besides the difference in loss functions:

Hyper-parameter	Value	Hyper-parameter	Value	Hyper-parameter	Value
Model Dim	200	Local Self Attn	performer	Dropout FF	0.2
FF Dim	800	Local Cross Attn	performer	Dropout Q/K/V	0
Attn Heads	8	Global Self Attn	performer	Dropout Token	0
Enc Layers	2	Global Cross Attn	performer	Dropout Emb	0.1
Dec Layers	2	Normalization	batch	L2 Weight	0.01
Initial Convs	0	Start Token Len	8	Loss Function	NLL/MSE
Inter. Convs	0	Time Emb Dim	12	Batch Size	64

5 Experiments, Results, and Analysis

We conducted two pre-training sessions on the same 100 stocks from the S&P 500 data-set. The two sessions used different loss functions, either Negative Loss Likelihood (NLL) or Mean Squared Error (MSE). Once the pre-training was complete the two models were tested on the IBM data-set with their respective loss functions. We also tested applying the SAM optimizer to both models to see the effect.

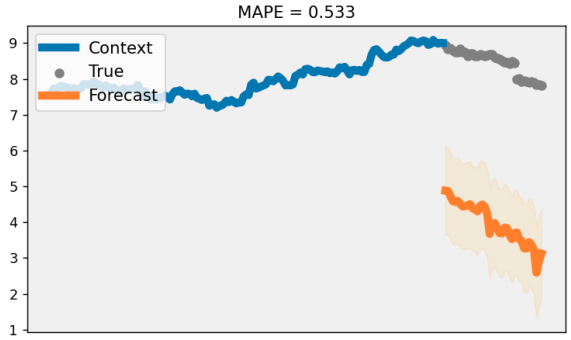

Figure 2: IBM Open Price forecast with MSE loss

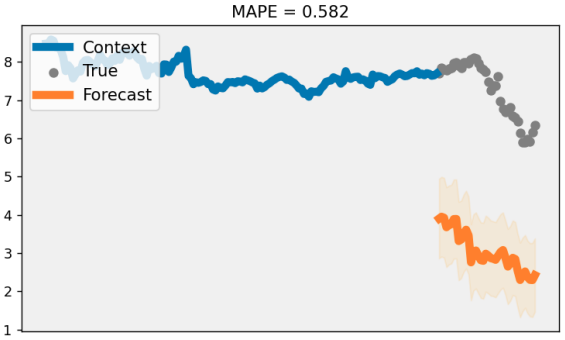

Figure 3: IBM Open Price forecast with MSE loss and SAM optimizer

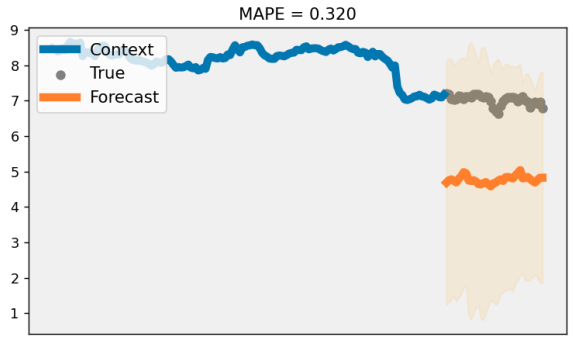

Figure 4: IBM Open Price forecast with NLL loss

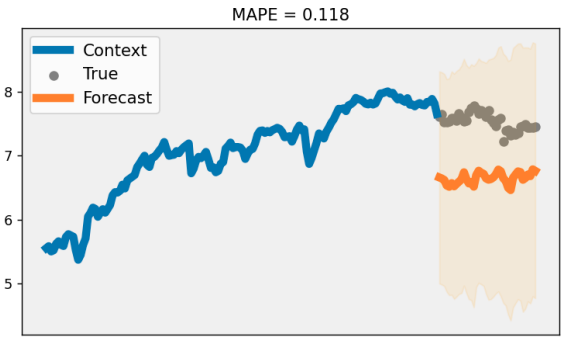

Figure 5: IBM Open Price forecast with NLL loss and SAM optimizer

Figures 2-5 show the models forecasts for the Open price of IBM stock. Each plot showcases 200 time-steps with 160 time-steps for context and 40 time-steps for prediction. Each section of the plot was chosen at random from the historical data.

As can be seen for the figures the model does not perform well in general in it's forecasting task. The model seemed to have been learning an offset during it's training that is not accounted for. The drastic offset only seems to occur in the Open, High, Low, and Close forecasts and does not appear in the Volume, Dividend, or Stock Split forecast. Figure 6 showcases one of the models forecasts for dividends while Figure 7 showcases one of the models forecasts for volume. Notice the lack of a severe offset.

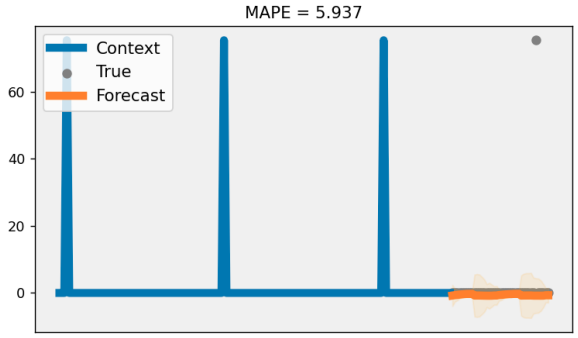

Figure 6: IBM dividends forecast with NLL loss

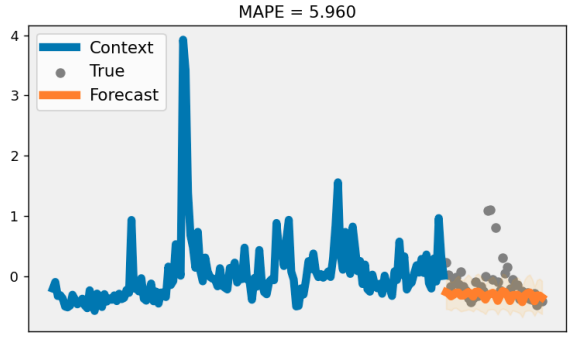

Figure 7: IBM volume forecast with NLL loss and SAM optimizer

A possible explanation for the offset can be a larger portion of the data is centered around the mean of the inputs for Open, High, Low, and Close variables as opposed to Volume, Dividends, and Stock Splits. This can be seen in the following table as the ratio of the standard deviation over the range is much smaller for the Volume, Dividends, and Stock Splits than the Open, High, Low, and Close variables. One standard deviation accounts for roughly 30% of the range in the Open, High, Low, Close inputs as opposed to less than 6% for Volume, Dividends, and Stock Splits. Showing that the data is much more centered around the mean.

Metric	Open	High	Low	Close	Volume	Dividends	Stock Split
Mean	39.41	39.78	39.07	39.43	$5.10\text{x}10^6$	0.005	0.0009
Stan. Dev.	44.27	44.62	43.94	44.29	$4.65\text{x}10^6$	0.07	0.04
Min	0.9	0.95	0.88	0.9	0	0	0
25th Perct.	4.53	4.56	4.51	4.53	$1.45\text{x}10^6$	0	0
50th Perct.	12.69	12.81	12.58	12.72	$4.33\text{x}10^6$	0	0
75th Perct.	65.82	66.65	64.78	65.85	$7.11\text{x}10^6$	0	0
Max	145.12	145.47	144.39	145.4	$7.26\text{x}10^7$	1.64	4
Range	144.21	144.52	143.5	144.49	$7.26\text{x}10^7$	1.64	4
Stan. Dev. / Range	0.307	0.308	0.306	0.306	0.064	0.044	0.011

The model may be making an 'average' prediction for the magnitude of the forecast for those variables. That would make sense as to why the offset appears at a similar magnitude in all predictions. It is also important to note the relative shape of the predictions follows the actual data well. That would continue to support the idea that the magnitude of the predictions seems to be a factor of the data distribution across the global spatial space. The shape of the forecast seems to be more influenced by the context points that immediately proceed the forecast. We believe exploring the global and local attention layers more deeply may provide more insight about this observation. The model may also need to put more weight on the temporal aspects of price as from one year to another the magnitude of a price can vary greatly, while the behavior of the price may not. We also believed that increase pre-training will probably decrease the drastic offset as well.

There are also significant differences in quality of the models forecasts based on the loss function and optimizer used. The models with NLL loss perform much better across all metrics as compared to the models trained with MSE loss. The NLL loss maximizes the probability of the target sequence according to the output distribution and makes use of uncertainty estimates. The model learns an intuitive prediction strategy that generally increases uncertainty along the target sequence. It also adds uncertainty to inflection points.

Metric	MSE	MSE w/ SAM	NLL	NLL w/ SAM
MSE	16.899	22.721	12.334	**12.221**
MAE	2.138	2.919	1.048	**0.9734**
RSE	0.8374	0.9717	0.714	**0.7109**
Class. Acc.(%)	25.63	42.76	41.21	**54.99**

As seen in the above table NLL w/ SAM performs the best out of all the models with marginal gains compared to NLL. The Classification Accuracy Metric is a feature of the STF architecture. At the end of the encoder and decoder output sequences they added a soft-max classification layer to output a prediction for the time series variable the token originally came from. The classification loss is detached from the forecast loss and does not effect the models training.

Figures 8-11 show the attention matrix of the Open Price variable across the different models. The darker blue an area the greater the attention. The axes for the figure can be interpreted as follows. From 0-160 is Open, 160-320 is High, 320-480 is Low, 480-640 is Close, 640-800 is Volume, 800-960 is Dividends, 960-1120 is Stock Splits. Every variable ranges from 0-160 which corresponds to the number of context time-steps the model is fed before it has to make its forecasts. You can see the forecast plots in Figures 2-7.

An interesting observation from the plots is the effect of Dividends on attention. The Divi-

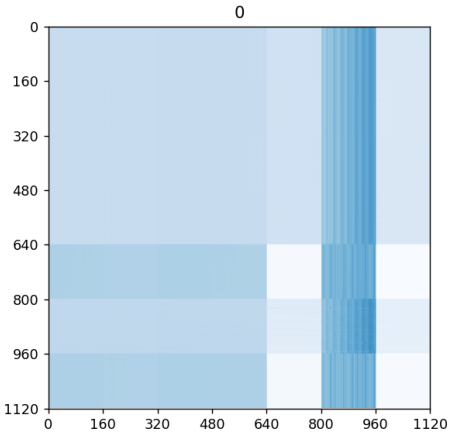

Figure 8: IBM Open Price Attention Matrix with MSE loss

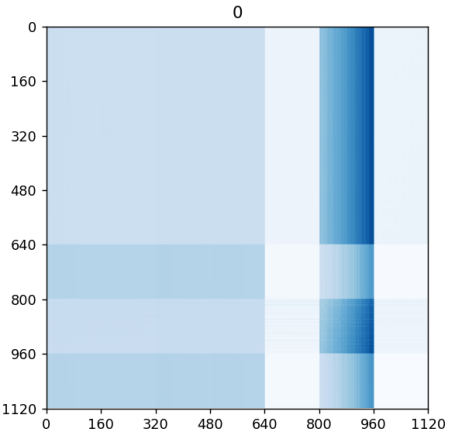

Figure 9: IBM Open Price Attention Matrix with MSE loss and SAM optimizer

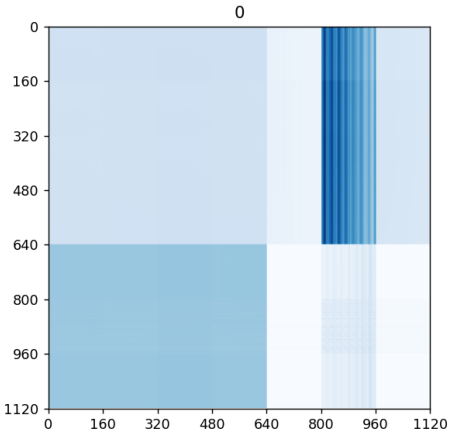

Figure 10: IBM Open Price Attention Matrix with NLL loss

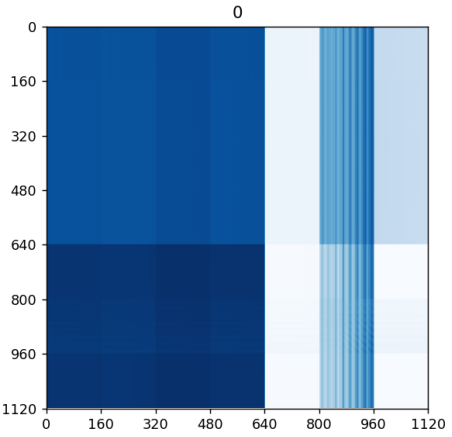

Figure 11: IBM Open Price Attention Matrix with NLL loss and SAM optimizer

dends for IBM have been consistently occurring over the entire time frame and the model seems to pay particular attention to the dividends around the time they should be due. This can be seen in the distinct bars seen in the dividend column in the figures. Dividends seems to have a greater effect on price than other aspects. This seems to mimic human behavior as well due to the fact people will pay attention to the dividends around the time they should be due, and if a company misses a dividend payment than that will surely be reflected in their share price.

One thing to note in the differences between the NLL plots and the MSE plots. The MSE matrices have lower attention scores across the board while the NLL matrices have a lot more contrast between the low attention and high attention portions. This may also be reflected in the performance of the models as NLL models also outperformed MSE models. It appears that with the greater contrast in attention the NLL models were able to create allowed them to give more weight to the 'consequential' variables.

Another interesting aspect in the attention matrices is the effect of SAM. In both the NLL and MSE models SAM appears to have smoothed out the contrasts within variables. This is most easily seen in the dividends column as the inter-variable contrast is most noticeable there. This is also an expected effect of the SAM optimizer as it is suppose to smooth loss geometries and provide a more general model. It's also very interesting to note the increase in attention the NLL with SAM model puts on the 'consequential' variables. The effect of this is not substantially shown in the results but perhaps with more experimentation this will be more clear.

Finally it is interesting to see how across all of the models the price pays the most attention to the Volume, Dividends, and Stock Splits. This can be seen in the blue corners in all the plots. The intuition behind this must be that due to the low variability of those variables (as seen in their summary statistics) that any change from the norm would have a big effect on price. This makes sense as any day with abnormally high volume seems to indicate that the stock must be going through a period of volatility. It also coincides with how consequential Stock Splits and Dividends are too investors. It also makes sense that a substantial amount of attention is paid in and between the price variables as well as the past price does seem to have an effect on the future price.

6 Conclusions

The primary purpose of this project was to explore Time Series Forecasting with Transformers and get up to speed with the state-of-the-art in the field. The models did not perform as well as hoped on the data-sets, however we do believe we have reached some interesting explanation as to why the performance fell short. We do not believe that the results shown in this report are representative of the predictive power the SpaceTimeFormer can have on Financial Time Series primarily due to the resource restrictions faced in this exploration. Especially with the unique architecture of the STF the memory demands are great and the training is expensive. We were also not able to train long enough, or find a pre-trained model to utilize transfer learning, so we were not able to exploit the potential power of pre-training for transformers.

We believe the paths for future work in this area are clear from this exploration. A deep dive into the temporal aspects of the attention mechanisms of this model will help shed some light on the large offsets seen in the price predictions. The use of a SAM optimizer also appears to be beneficial and more experimentation with it will be needed to see the true effect. Further modifications of the architecture to improve it's specific performance on Financial Time Series would also be beneficial, such as incorporating the Multi-Scale Gaussian Prior into the MSA of the model.

Bibliography

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[4] R. P. Schumaker and H. Chen, "Textual analysis of stock market prediction using breaking financial news: The azfin text system," *ACM Transactions on Information Systems (TOIS)*, vol. 27, no. 2, pp. 1–19, 2009.

[5] B. Weng, M. A. Ahmed, and F. M. Megahed, "Stock market one-day ahead movement prediction using disparate data sources," *Expert Systems with Applications*, vol. 79, pp. 153–163, 2017.

[6] Y. Lin, H. Guo, and J. Hu, "An svm-based approach for stock market trend prediction," in *The 2013 international joint conference on neural networks (IJCNN)*, pp. 1–7, IEEE, 2013.

[7] D. M. Nelson, A. C. Pereira, and R. A. De Oliveira, "Stock market's price movement prediction with lstm neural networks," in *2017 International joint conference on neural networks (IJCNN)*, pp. 1419–1426, IEEE, 2017.

[8] L. Zhang, C. Aggarwal, and G.-J. Qi, "Stock price prediction via discovering multi-frequency trading patterns," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 2141–2149, 2017.

[9] J. Wang, T. Sun, B. Liu, Y. Cao, and H. Zhu, "Clvsa: A convolutional lstm based variational sequence-to-sequence model with attention for predicting trends of financial markets," *arXiv preprint arXiv:2104.04041*, 2021.

[10] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell, "A dual-stage attention-based recurrent neural network for time series prediction," *arXiv preprint arXiv:1704.02971*, 2017.

[11] F. Feng, H. Chen, X. He, J. Ding, M. Sun, and T.-S. Chua, "Enhancing stock movement prediction with adversarial training," *arXiv preprint arXiv:1810.09936*, 2018.

[12] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proceedings of AAAI*, 2021.

[13] J. Grigsby, Z. Wang, and Y. Qi, "Long-range transformers for dynamic spatiotemporal forecasting," *arXiv preprint arXiv:2109.12218*, 2021.

[14] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan, "Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[15] Q. Guo, X. Qiu, P. Liu, Y. Shao, X. Xue, and Z. Zhang, "Star-transformer," *arXiv preprint arXiv:1902.09113*, 2019.

[16] Z. Ye, Q. Guo, Q. Gan, X. Qiu, and Z. Zhang, "Bp-transformer: Modelling long-range context via binary partitioning," *arXiv preprint arXiv:1911.04070*, 2019.

[17] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[18] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, *et al.*, "Big bird: Transformers for longer sequences," *Advances in Neural Information Processing Systems*, vol. 33, pp. 17283–17297, 2020.

[19] H. Zhang, Y. Gong, Y. Shen, W. Li, J. Lv, N. Duan, and W. Chen, "Poolingformer: Long document modeling with pooling attention," in *International Conference on Machine Learning*, pp. 12437–12446, PMLR, 2021.

[20] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *arXiv preprint arXiv:2006.04768*, 2020.

[21] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, *et al.*, "Rethinking attention with performers," *arXiv preprint arXiv:2009.14794*, 2020.

[22] S.-Y. Shih, F.-K. Sun, and H.-y. Lee, "Temporal pattern attention for multivariate time series forecasting," *Machine Learning*, vol. 108, no. 8, pp. 1421–1441, 2019.

[23] T. Gangopadhyay, S. Y. Tan, Z. Jiang, R. Meng, and S. Sarkar, "Spatiotemporal attention for multivariate time series prediction and interpretation," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3560–3564, IEEE, 2021.

[24] M. Xu, W. Dai, C. Liu, X. Gao, W. Lin, G.-J. Qi, and H. Xiong, "Spatial-temporal transformer networks for traffic flow forecasting," *arXiv preprint arXiv:2001.02908*, 2020.

[25] L. Cai, K. Janowicz, G. Mai, B. Yan, and R. Zhu, "Traffic transformer: Capturing the continuity and periodicity of time series for traffic forecasting," *Transactions in GIS*, vol. 24, no. 3, pp. 736–755, 2020.

[26] S. M. Kazemi, R. Goel, S. Eghbali, J. Ramanan, J. Sahota, S. Thakur, S. Wu, C. Smyth, P. Poupart, and M. Brubaker, "Time2vec: Learning a vector representation of time," *arXiv preprint arXiv:1907.05321*, 2019.

[27] Q. Ding, S. Wu, H. Sun, J. Guo, and J. Guo, "Hierarchical multi-scale gaussian transformer for stock movement prediction.," in *IJCAI*, pp. 4640–4646, 2020.

[28] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," *arXiv preprint arXiv:2010.01412*, 2020.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[30] X. Chen, C.-J. Hsieh, and B. Gong, "When vision transformers outperform resnets without pre-training or strong data augmentations," *arXiv preprint arXiv:2106.01548*, 2021.

[31] "Standard and poor's 500." `https://www.spglobal.com/spdji/en/indices/equity/sp-500/#overview`. Accessed: 2022-05-09.

[32] "yfinance library." `https://pypi.org/project/yfinance/`. Accessed: 2022-05-09.

[33] "Weights and biases devops." `https://wandb.ai/site`. Accessed: 2022-05-09.