Bitcoin Price Prediction Using TSMixer
AIST4010 - Foundation of applied deep learning
Project Report

Name: Wong Shing Lok
SID: 1155156680

Abstract

This work centers on the TSMixer model, a state-of-the-art architecture that leverages mixer layers to analyze both temporal and feature-related dynamics within multivariate time series data. It was re-implemented in PyTorch, diverging from its original TensorFlow framework. The research also benchmarks TSMixer against advanced models such as the Temporal Convolutional Network (TCN) and the Temporal Fusion Transformer (TFT) to evaluate its performance in this specific domain. Utilizing the Neuralforecast library, comprehensive hyperparameter tuning was conducted to optimize model configurations. The key findings reveal that while TSMixer provides insightful predictions, TCN and TFT outperform TSMixer in terms of Mean Squared Error (MSE). The results contribute to the understanding of how deep learning can be applied to forecast complex financial time series, as well as the potential for further improvements in model architectures tailored to forecast economic data.

1 Introduction

This work focuses on the prediction of Bitcoin prices, employing a cutting-edge deep learning model known as TSMixer[1]. TSMixer introduces an innovative all-MLP (Multi-Layer Perceptron) architecture tailored for time series forecasting. Unlike traditional models, TSMixer exploits mixing operations along both time and feature dimensions, enhancing its ability to utilize cross-variate and auxiliary data. This capability is particularly crucial for handling the intricate dynamics associated with multivariate time series data.

The challenge of forecasting Bitcoin prices lies in their volatile and unpredictable nature, making accurate predictions essential yet difficult. These predictions hold significant value for investors and traders by potentially offering insights that can inform investment decisions. Also, TSMixer has demonstrated promise in various domains, its efficacy in financial environments like stock or cryptocurrency markets still remains less explored. This work aims to adapt and evaluate TSMixer within the context of financial time series data, an area not covered in the original work.

The model architecture incorporates two principal components: a mixer layer that disaggregates the input time series into patches for individual MLP processing—capturing both intra-series and inter-series relationships and a temporal projection that maintains the output tensor's shape for effective forecasting or representation learning. This approach is depicted in Figure 1.

By extending TSMixer to Bitcoin forecasting, this work seeks to contribute to the understanding of its scalability and effectiveness in the high-stakes environment of financial markets, where predictive accuracy is crucial.

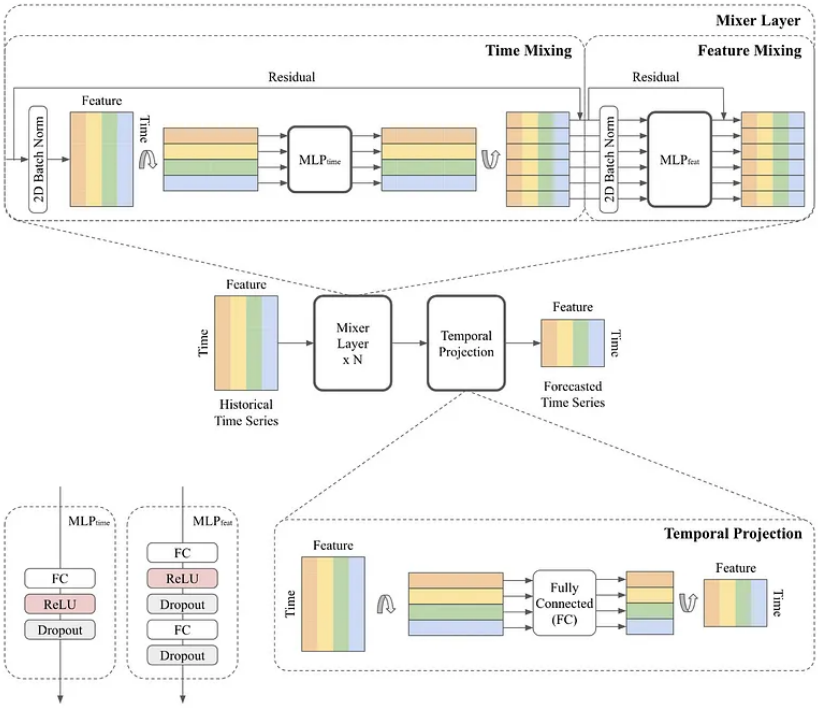

Figure 1: TSMixer's Architecture[1].

2 Related Works

Time series forecasting is a crucial analytical task that involves predicting future values of a dataset based on its historical patterns. Traditional statistical methods like ARIMA[2] and Prophet[3] have long been standards in this domain. However, the dynamic and often volatile nature of real-world market data poses significant challenges to these classical models, especially when dealing with non-linear patterns inherent in financial time series.

In the realm of machine learning, deep learning approaches have been adopted for this task for quite some time[4]. RNN-based methods such as LSTM[5], GRU[6] and TCN[7] have been popular due to their proficiency in capturing temporal dependencies. In recent years, transformer-based models have become increasingly popular for this task due to their ability to model complex sequential data[8]. However, transformer-based models do not always perform as expected. Interestingly, a simple univariate linear model, which treats multivariate features as different univariate sequences, can outperform multivariate transformer-based models on long-term forecasting tasks[9]. In addition to the main time series data, there may also be auxiliary data, such as static features and future time-varying features. Several deep learning models, including DeepAR[10] and TFT[11], have been introduced to handle such scenarios. While these models perform well, they tend to be quite complex. The TSMixer model presents a novel approach to this problem. It employs an architecture based on the MLP Mixer for time series forecasting. This approach effectively addresses the challenges posed by the complexity of the models and the need to incorporate auxiliary data.

3 Dataset

The source of the dataset is available on Kaggle under the title "Cryptocurrency Historical Prices"[12]. It consists of multiple prices of top cryptocurrencies such as BTC, ETH, ADA and BNB. In this task, I will just focus on BTC which is the `coin_Bitcoin.csv` file. It is approximately 382.27 kB. The dataset covers daily price information from April 28, 2013, to July 7, 2021. The dataset contains the following fields:

- **Date:** The date of observation.
- **Open:** Opening price of Bitcoin on the given day.
- **High:** Highest price of Bitcoin on the given day.
- **Low:** Lowest price of Bitcoin on the given day.
- **Close:** Closing price of Bitcoin on the given day.

- **Volume:** Volume of Bitcoin transactions on the given day.
- **Market Cap:** Market capitalization in USD.

4 Interpretation of TSMixer

The paper discusses the forecastability of time series data in three aspects:

1. **Persistent Temporal Patterns**: These include trends and seasonal patterns, such as long-term inflation and day-of-week effects.
2. **Cross-Variate Information**: This refers to correlations between different variables, such as an increase in blood pressure associated with a rise in body weight.
3. **Auxiliary Features**: These are static features and future information, such as product categories and promotional events.

The paper mentions that transformer-based models can capture both complex temporal patterns and cross-variate dependencies. However, multivariate models are not always superior to simple univariate temporal linear models. They may suffer from overfitting, especially when the target time series is not correlated with other covariates. For temporal linear models, their time-step-dependent characteristics make temporal linear models excellent candidates for learning temporal patterns.

4.1 TSMixer Model

The paper presents three model variants:

1. **TMix-Only (Univariate Model)**: A simplified variant of TSMixer that only employs time-mixing.
2. **TSMixer (Multivariate Model)**: The Time-Series Mixer with time-domain and feature-domain operations representing time-mixing and feature-mixing operations. **This is the variant I will use for the experiment.**
3. **TSMixer-Ext (Multivariate Model + Auxiliary Information)**: An extended TSMixer for Time Series Forecasting with auxiliary information. In our task, the dataset does not contain any auxiliary features as they are all cross-variate information which are interrelated and observed over time.

The TSMixer model consists of several components:

1. **Temporal Projection (TP)**: A linear layer that adjusts the length of the input time series.
2. **Time Mixing (TM)**: Applies transformations across all columns of the input matrix, capturing complex temporal patterns in the data.
3. **Conditional Feature Mixing (FM)**: Works on rows of the input matrix and captures interactions between different features of the data. This is only for **TSMixer-Ext**.
4. **Residual Connections**: Both TM and FM layers have residual connections, which help in training deep networks.
5. **Conditional Mixer Layer**: A composition of TM and FM layers that combines the information captured by both temporal and feature transformations. This is only for **TSMixer-Ext**.

5 Approach

There is an official implementation available on Github using Tensorflow[1], which I used it as a starting point. I reimplemented it using Pytorch, starting from constructing it layer by layer to get a full understanding of its inner workings. The inputs for the model were multiple features as multivariate inputs. The final prediction targeted on the **closing price** only, which is an univariate output.

5.1 Model Implementation

This section describes the architecture and functionalities of the implementation of TSMixer model components, which follows the same design shown in Figure 1.

[1]Basic Implementation by Google Research: https://github.com/google-research/google-research/tree/master/tsmixer/tsmixer_basic

5.1.1 Reversible Instance Normalization (RevIN)

As per the original TSMixer design, for longterm forecasting task, they apply reversible instance normalization on the model[13] to normalize the input and denormalize the output of the model. It dynamically normalizes the input features based on the instance-specific statistics, allowing for reversible normalization. This reversible aspect is particularly useful for preserving the original feature distribution, which is essential in tasks involving detailed feature recovery.

5.1.2 Mixer Layer

The MixerLayer is a composite layer that performs both temporal and feature mixing. Temporal mixing manipulates the input across the time dimension, promoting the model's ability to learn temporal dependencies. Feature mixing operates across the feature channels, enabling the model to capture complex feature interactions.

- Temporal Mixing: Temporal mixing is achieved through a series of operations including normalization (using BatchNorm), a linear transformation across the time dimension, a ReLU activation function, and dropout. This process allows the model to capture and learn from temporal patterns in the data.
- Feature Mixing: Feature mixing follows a similar pattern of operations but is applied across the feature channels. This process enables the model to understand and utilize interactions between different features for improved forecasting accuracy.
- Output Aggregation: The outputs from the temporal and feature mixing processes are aggregated to produce the final output of the MixerLayer. This aggregation allows the model to combine learned temporal patterns and feature interactions effectively.

5.1.3 Temporal Projection

Temporal Projection is employed to align the length of the processed sequence with the target prediction length. It uses a linear transformation to modify the temporal dimension of the input sequence. This transformation enables the model to output sequences that match the desired prediction length, facilitating accurate forecasting over specified horizons.

5.1.4 TSMixer Model

The TSMixer model assembles the aforementioned components into a cohesive architecture. Initially, the input data undergoes reversible normalization (RevIN), followed by a series of MixerLayers for temporal and feature mixing. Finally, the Temporal Projection adjusts the sequence length to match the prediction length, and the output undergoes reverse normalization to produce the final forecast.

5.2 Data Preprocessing

5.2.1 Data Cleaning

For the purpose of forecasting, irrelevant columns such as **SNo**, **Name**, and **Symbol** are removed, to ensure that only the necessary data is retained for analysis.

5.2.2 Data Loader

The data is then scaled using the **StandardScaler** from **scikit-learn**. This scaler removes the mean and scales the data to unit variance. The preprocessed data is converted into sequences to be used as input for the TSMixer model. This involves creating sequences of a fixed length **seq_len** for the historical data that the model will use to make predictions. The target sequences of length **pred_len** are also prepared, which the model aims to predict.

The dataset is then split into training, validation, and testing sets, with a ratio of $7:2:1$. The training set is used to train the model, the validation set is used to see how the model performs on the unseen data, and the testing set is used to evaluate the final model's performance.

5.3 Model Training

The model training was configured with mean squared error (MSE) loss function, Adam optimizer[14]. For other hyperparameters setting, such as learning rate, please refer to the section Experiments. The illustration of the process is shown in Figure 2.

5.4 Inference and Visualization

In the inference phase, the trained model was utilized to predict future time series values using the test dataset. This involved a forward pass of the input data through the model to generate output sequences. These sequences represent the model's forecasts for the target variable over the specified prediction horizon. For a fair comparison between the model's predictions and the actual data, I reversed the standard scaling transformations that were initially applied in data loader to ensure that the predicted and actual values were in the same scale.

For each prediction date, I aggregated the forecasts and actual values to compute the average predicted value and the average actual value. By visual inspection, we could know the model's forecasting ability and detect periods where the model's performance deviated significantly from expectations.

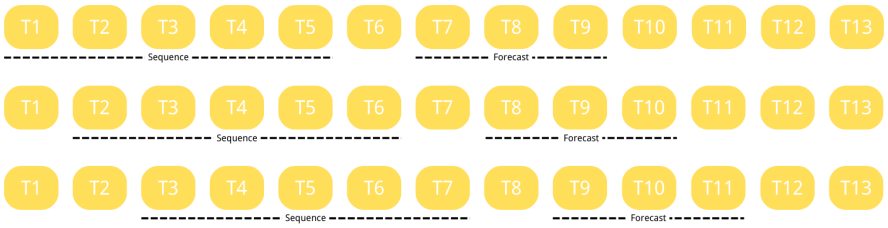

Figure 2: Illustration of Training Process

6 Experiments

To evaluate the model across various hyperparameter configurations, I utilized Ray Tune[15], a scalable library for experiment execution and hyperparameter tuning. The hyperparameter search spaces are shown in Tables 1 and 2. I segmented the search spaces into two sets of experiments to specifically assess the impact of sequence and prediction lengths on model performance.

The outcomes of these experiments are summarized in Tables 3 and 4. Notably, the best-performing configuration from the short-sequence experiment yielded a significantly lower validation loss compared to that from the long-sequence experiment. The corresponding predictive results are depicted in Figures 3 and 4. It is observed that the model with long sequence parameters did not capture the underlying patterns as effectively as the model trained with shorter sequence lengths.

Table 1: Hyperparameter Search Space of Short Sequence and Prediction Length

Parameter	**Values**
Learning Rate	$[1e-6, 1e-3]$ (log-uniform sampling)
Batch Size	$\{32, 64\}$
Sequence Length	$\{5, 10\}$
Prediction Length	$\{10, 20, 30\}$
Number of Blocks	$\{2, 4\}$
Dropout Rate	$\{0.1, 0.3, 0.5, 0.9\}$
Feedforward Dimension	$\{64, 128, 256\}$
Epochs	$\{100, 500, 1000\}$

Table 2: Hyperparameter Search Space of Long Sequence and Prediction Length

Parameter	Values
Learning Rate	$[1e-6, 1e-3]$ (log-uniform sampling)
Batch Size	$\{32, 64\}$
Sequence Length	$\{64, 100\}$
Prediction Length	$\{100, 200, 300\}$
Number of Blocks	$\{2, 4\}$
Dropout Rate	$\{0.1, 0.3, 0.5, 0.9\}$
Feedforward Dimension	$\{64, 128, 256\}$
Epochs	$\{100, 500, 1000\}$

Table 3: Hyperparameter Tuning Results for Short Sequence and Prediction Length

lr	batch_size	seq_len	pred_len	n_block	dropout	ff_dim	epochs	loss
0.000279503	64	5	30	2	0.5	128	1000	0.0122564
0.000833078	32	10	30	4	0.9	256	100	0.0121871
0.000668957	32	10	30	4	0.1	256	100	0.0146586
4.44007e-05	64	5	30	4	0.9	64	500	0.0120028
3.12162e-06	32	10	30	2	0.9	256	1000	0.0134914
0.000454967	32	5	30	2	0.9	64	100	0.0117904
9.68671e-05	32	5	30	2	0.9	128	100	0.0122023
0.000288237	64	10	20	2	0.5	128	1000	0.0093525
1.56529e-06	32	5	20	2	0.3	256	100	0.00897801
5.45189e-05	**32**	**5**	**20**	**4**	**0.5**	**128**	**1000**	**0.00877985**

Table 4: Hyperparameter Tuning Results for Long Sequence and Prediction Length

lr	batch_size	seq_len	pred_len	n_block	dropout	ff_dim	epochs	loss
0.000386735	32	100	100	4	0.3	64	100	0.0757575
0.000687456	32	100	300	2	0.5	128	1000	0.988087
2.41146e-05	**64**	**64**	**100**	**4**	**0.5**	**256**	**500**	**0.0465351**
4.68935e-05	64	100	300	4	0.5	64	1000	0.896512
0.000306605	64	64	300	2	0.9	256	100	0.329785
1.31196e-06	32	100	200	4	0.5	128	500	0.100276
2.68342e-06	64	100	100	4	0.9	64	500	0.05771
0.000340795	32	64	300	4	0.3	256	100	2.41132
1.57758e-06	64	64	100	2	0.9	64	1000	0.0501884
0.000152858	64	100	100	4	0.5	128	500	0.0728992

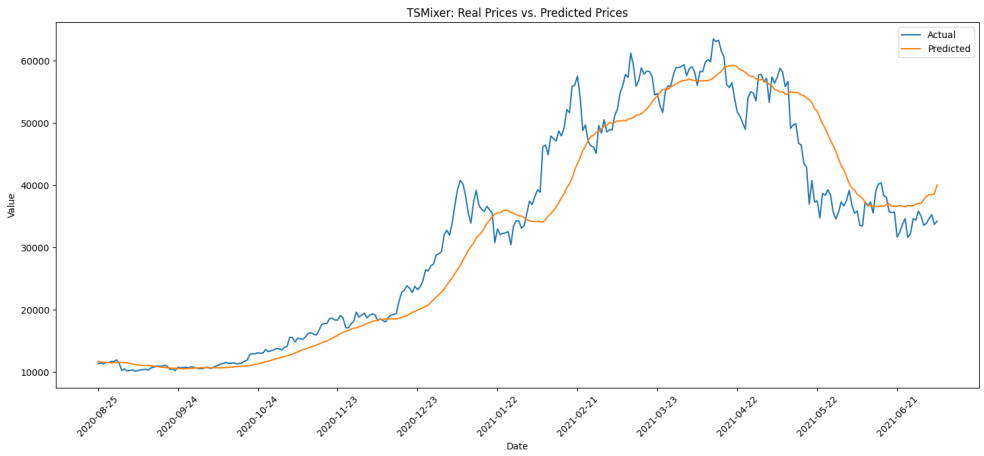

Figure 3: Short Length Prediction Result with MSE:23938874

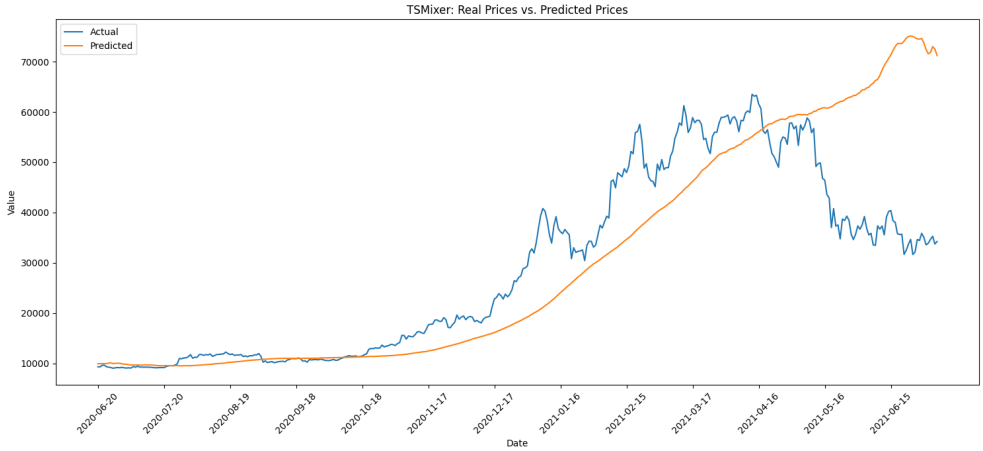

Figure 4: Long Length Prediction Result with MSE:182010480

6.1 Models Comparison

In this subsection, I compare the performance of the TSMixer with the following models:

- **Temporal Convolutional Network (TCN):** A variant of convolutional neural networks which has been proven effective for sequence modeling and forecasting tasks. TCNs address the vanishing gradient problem commonly encountered in RNNs and have been shown to capture complex temporal patterns efficiently [7].
- **Temporal Fusion Transformer (TFT):** An advanced model specifically designed for interpretable time series forecasting. It incorporates various forms of temporal dynamics, leveraging them to achieve state-of-the-art performance on numerous benchmark datasets[11].

For a fair comparison, I utilized the optimal short sequence length identified in prior experiments for all models. The Neuralforecast library[16], which includes a collection of models such as TCN, TFT, and TSMixer, was employed to ensure consistency in hyperparameter tuning and model evaluation. Each model was tasked with forecasting the closing price (univariate output) of a financial time series, using a prediction horizon of 5 days.

The hyperparameter tuning search spaces for each model are shown in Tables 5, 6, and 7. Comparative results and Mean Squared Errors (MSE) are illustrated in Figure 5 and Table 8, respectively.

Surprisingly, the TSMixer from Neuralforecast did not perform as well as expected, potentially due to overlooked nuances in configuration or specific optimizations within the library. Notably, the TCN model achieved the lowest MSE, indicating its robust capability in handling complex time series data like cryptocurrency markets. This result suggests that TCN might be well-suited for tasks requiring the capture of intricate temporal dependencies. Meanwhile, TFT also performed commendably, slightly trailing behind TCN. In contrast, both implementations of TSMixer lagged behind. This could be caused by the model's inability to handle the highly volatile and speculative nature in cryptocurrency data. This insight might explain the absence of such datasets in the original TSMixer's paper.

Table 5: Search Space Configuration of Temporal Convolutional Networks (TCN) in Neuralforecast

Parameter	Values
Encoder Hidden Size	20, 50, 100, 200
Context Size	5, 10, 50
Decoder Hidden Size	64, 128
Learning Rate	10^{-5} to 10^{-1} (log-uniform sampling)
Max Steps	500, 1000
Batch Size	16, 32

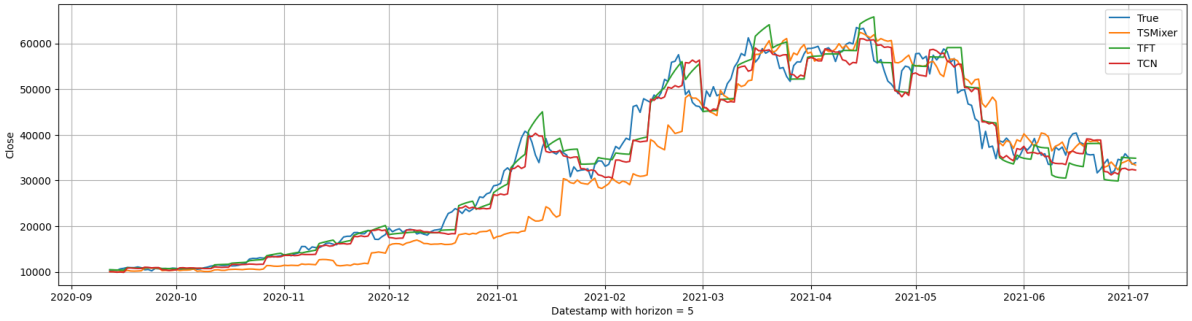

Figure 5: Comparison of The Models

Models	MSE
TCN	10097831.320
TFT	10126363.741
TSMixer(self-implemented best)	23938874
TSMixer(Neuralforecast)	40657197.901

Table 8: MSE of The Models

Table 6: Search Space Configuration of Temporal Fusion Transformer (TFT) in Neuralforecast

Parameter	Values
Sequence Length	20, 30, 60, 90, 150, 300
Hidden Size	64, 128
Learning Rate	10^{-5} to 10^{-2} (log-uniform sampling)
Max Steps	1000
Batch Size	16, 32
Scaler Type	standard
Val Check Steps	100
Early Stop Patience Steps	5

Table 7: Search Space Configuration of TSMixer in Neuralforecast

Parameter	Values
Sequence Length	15, 20, 30, 60, 90, 150, 300
Max Steps	500, 1000
Val Check Steps	100
Early Stop Patience Steps	5
Learning Rate	10^{-5} to 10^{-2} (log-uniform sampling)
N Blocks	1, 2, 4, 6, 8
Dropout	0.1, 0.3, 0.5, 0.9
FF_Dim	32, 64, 128

7 Conclusion

The investigation included a comprehensive exploration of various hyperparameter configurations using Ray Tune, which allowed for systematic and scalable experimentation.

The experiments revealed that shorter sequence lengths generally led to better forecasting performance. This insight suggests that for the highly volatile cryptocurrency market, capturing more immediate temporal dependencies is more critical than integrating longer historical data.

Moreover, a comparative analysis with established models like TCN and TFT demonstrated that while the self-implemented TSMixer performed robustly, it was outperformed by the TCN and TFT when using the Neuralforecast library. This outcome implies that the TSMixer is may not be able to handle complex time series data such as cryptocurrency prices, which may require capturing intricate interdependencies and external influences that the current TSMixer configuration might not fully account for.

The varying performance of the TSMixer model when implemented independently versus its deployment through the Neuralforecast framework could be due to differences in implementation details, optimization techniques, or modeling approach.

8 Future Work

Given the complex nature of financial markets, which are influenced by a lot of unpredictable factors, future work could extend in several key areas:

- **Incorporation of External Data:** Future models could benefit significantly from the integration of external datasets that capture financial news, political events, and macroeconomic indicators. These data sources could provide crucial context that affects market behavior, potentially enhancing the model's ability to anticipate market movements. Advanced text analysis techniques, such as sentiment analysis, could be employed to quantify the impact of news articles or social media on market prices.
- **Advanced Hyperparameter Optimization:** While this study utilized Ray Tune for systematic hyperparameter tuning, there remains a vast space of configurations to explore. Future work could implement broader search space and more sophisticated optimization algorithms, such as Bayesian optimization[17], to fine-tune the models more precisely and explore deeper and more complex network architectures.

9 Lesson Learned

This project provided me a comprehensive learning experience, from theoretical understanding to practical application of complex models in the area of time series forecasting. Here are some of the key lessons I have gathered throughout this process:

- **Understanding Time Series Data:** Working with time series data, especially in the context of financial markets like cryptocurrency, was enlightening. I learned about the intricacies of dealing with univariate and multivariate series and how different features can interact within a model. This project emphasizes the importance of selecting appropriate model inputs and data preprocessing techniques.
- **Model Construction from Research Papers:** Translating a model from a research paper into a working implementation was a tough exercise that enhanced my technical skills like writing the code using PyTorch. It taught me how to bridge the gap between theoretical research and practical application, a valuable skill in any data scientist's toolkit.
- **Hyperparameter Tuning with Ray Tune:** It was my first hands-on experience using such an advanced tool. I learned how to define search spaces and interpret the results of the tuning process. This experience has equipped me with the knowledge to use Ray Tune in future projects for efficient model optimization.
- **Documentation:** Documenting the process and results in a structured report helped me understand the importance of clear communication in projects.

Overall, this project was not only about applying what I had previously learned but also about pushing the boundaries of my understanding and capabilities. It consolidated the theoretical foundation while also providing practical skills that I will carry forward into my future work.

References

[1] S.-A. Chen, C.-L. Li, N. Yoder, S. O. Arik, and T. Pfister, *Tsmixer: An all-mlp architecture for time series forecasting*, 2023. arXiv: 2303.06053 [cs.LG].

[2] S. Ho and M. Xie, "The use of arima models for reliability forecasting and analysis," *Computers Industrial Engineering*, vol. 35, no. 1, pp. 213–216, 1998, ISSN: 0360-8352. DOI: https://doi.org/10.1016/S0360-8352(98)00066-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835298000667.

[3] S. Taylor and B. Letham, "Forecasting at scale," Sep. 2017. DOI: 10.7287/peerj.preprints.3190v2.

[4] N. Kourentzes, "Intermittent demand forecasts with neural networks," *International Journal of Production Economics*, vol. 143, no. 1, pp. 198–206, 2013. DOI: 10.1016/j.ijpe.2013.01.00. [Online]. Available: https://ideas.repec.org/a/eee/proeco/v143y2013i1p198-206.html.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[6] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, 2014. arXiv: 1412.3555 [cs.NE].

[7] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, *Temporal convolutional networks for action segmentation and detection*, 2016. arXiv: 1611.05267 [cs.CV].

[8] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL].

[9] A. Zeng, M. Chen, L. Zhang, and Q. Xu, *Are transformers effective for time series forecasting?* 2022. arXiv: 2205.13504 [cs.AI].

[10] D. Salinas, V. Flunkert, and J. Gasthaus, *Deepar: Probabilistic forecasting with autoregressive recurrent networks*, 2019. arXiv: 1704.04110 [cs.AI].

[11] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, *Temporal fusion transformers for interpretable multi-horizon time series forecasting*, 2020. arXiv: 1912.09363 [stat.ML].

[12] S. Rajkumar, *Cryptocurrency price history*, 2021. [Online]. Available: https://www.kaggle.com/datasets/sudalairajkumar/cryptocurrencypricehistory?select=coin_Bitcoin.csv.

[13] T. Kim, J. Kim, Y. Tae, C. Park, J.-H. Choi, and J. Choo, "Reversible instance normalization for accurate time-series forecasting against distribution shift," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=cGDAkQo1C0p.

[14] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[15] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[16] K. G. Olivares, C. Challú, F. Garza, M. M. Canseco, and A. Dubrawski, *NeuralForecast: User friendly state-of-the-art neural forecasting models.* PyCon Salt Lake City, Utah, US 2022, 2022. [Online]. Available: https://github.com/Nixtla/neuralforecast.

[17] N. D. Sanders, R. M. Everson, J. E. Fieldsend, and A. A. M. Rahat, *Bayesian search for robust optima*, 2021. arXiv: 1904.11416 [cs.LG].