Reinforcement Learning for Financial Time-Series Forecasting

Naman Choudhary
Carnegie Mellon University
namancho@andrew.cmu.edu

Karthik Subramaniam
Carnegie Mellon University
ksubram2@andrew.cmu.edu

Nimish Jindal
Carnegie Mellon University
nimishj@andrew.cmu.edu

Sarah Li
Carnegie Mellon University
sarahli@andrew.cmu.edu

Abstract

This paper presents RLFinNet, a novel reinforcement learning-based model tailored for the efficient and rapid forecasting of financial time-series data. Focused on the specific dynamics of individual company stock prices, RLFinNet addresses the need for models that are not only accurate but also quick to adapt to new data, avoiding the influence of irrelevant market factors. Our approach enhances the classical deep reinforcement learning framework by incorporating an innovative dual forecasting model, adapted from the OneNet architecture, which leverages both cross-time and cross-variable dependencies to predict future stock prices effectively. We describe the adaptations made to our model to handle the multivariate financial data, detailing the updates to our data loader and the modifications to the training loop to accommodate sequential processing across various datasets. The model was tested using a financial dataset featuring a range of metrics including open, high, low, and closing prices, among others. We conducted a series of experiments with varying prediction and sequence lengths to optimize our model for different forecasting horizons. Our findings demonstrate that RLFinNet can achieve significant improvements in prediction accuracy with a mean squared error (MSE) goal of less than 0.4, aligning with the performance metrics of established models. The results highlight the model's potential in transforming financial market analytics by providing robust, scalable, and efficient predictive insights. This work not only advances the field of financial forecasting but also opens up new avenues for real-time data processing in other domains.

1 Introduction

The financial portfolio management problem is the process of constant redistribution of a fund into different financial products. Algorithmic trading has thus received increasing spotlight as the volume of financial data grows. Algorithmic trading involves the use of mathematical models and statistical analysis to exploit market opportunities and make financial trading decisions. Recently, neural-network based trading has been gaining attention, as price predictions are not market actions, and converting them into actions requires additional logic for real-world applications. Due to its unsupervised nature, we expand upon model-free reinforcement learning for this research project.

Our project goal is to create an efficient and quick network – **RLFinNet** – that is small in terms of architecture parameters. This is beneficial to traders as a model specialized to a specific company's stock prices will not be influenced by irrelevant companies' stock prices, and retraining the model with custom or tweaked hyper-parameters can be done in a matter of seconds. The model is ultimately designed to train on real-time data and make predictions more rapidly than more saturated and generalized models.

1.1 Model Description

Deep Reinforcement Learning (DLR) has been an effective candidate of time-series forecasting for years. DRL as a stochastic problem can represent an environment, in which an a model “agent” is able to choose a course of actions given state policies, with the ultimate goal of maximizing accumulated rewards. For this project, we aim to show the usefulness of Deep Reinforcement Learning for online and rapid training of time-series forecasting.

The base model OneNet that we have adapted is made of Convolutional layers. The entire model is an ensemble of 2 model– one cross-variable and one cross-time. The individual models are trained separately using MSE loss and the models are ensembled using reinforcement learning to adjust the weight of the combination of the two models. This will be further elaborated in *section 5*.

1.2 Dataset

We used the **ACL18** financial dataset from the previous STOTA model, StockNet [23], which features historical prices from a range of companies. Specifically, we intend to predict the future closing price of a stock, based on its *Open*, *High*, *Low*, *Volume*, *Volume*, and *Adjusted Close* prices traded on the current day. Including the date, the input is 7-Dimensional.

2 Literature Review

The Stock Market is a financial ecosystem involving over 100 trillion dollars in 2021, according to the world bank [22]. As a result, the financial industry generates enormous data every single day. Ranging from customer information to transaction records, the data are not only kept confidential but also provided quickly to customers for strategic planning and decision-making. With the advancement of these digital technologies, enormous datasets can be efficiently handled, analyzed, and recorded. Quantitative stock investment is a fundamental financial task that highly relies on accurate prediction of market status and profitable investment decision making [1]. Different investors may approach the market from different angles, for example, by studying market behavior, identifying influential factors, trading stocks, forecasting market directions, making asset recommendations for portfolio management, etc. Despite any and all of these goals, the investors continuously face adversity with time constraints.

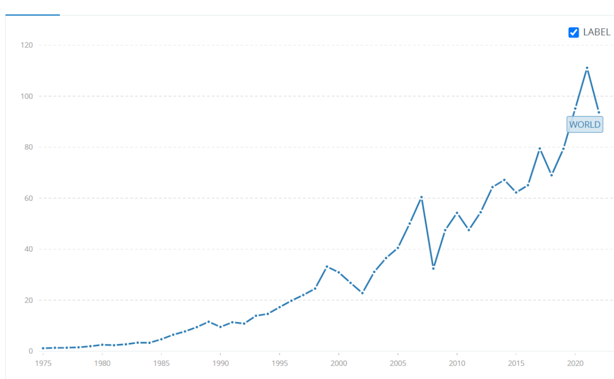


Figure 1: Value of the global stock market [22]

We reference several research papers for our project. In "Enhancing Stock Movement Prediction with Adversarial Training [9]," FuLi *et al* used adversarial training to improve the generalizability of Neural Network predictions on stock data by training the model under intentional perturbations to stochastic price variables, which had conventionally been treated as static features in prediction tasks. This novel approach has demonstrated an improvement in accuracy compared to the previous state-of-the-art model StockNet developed by [Xu and Cohen, 2018] to forecast temporally dependent data.

Shaban *et al* [20] compared two novel models - One with LSTM layers and one with BiGRU layers to existing DL-based approaches for stock prediction. The new recurrent architectures proved to be computationally more efficient and proved better at real-time trading. They were able to predict stock prices 10 minutes and 30 minutes into the future. While our project is not based on real-time trading, we do believe that recurrent approaches could be much more efficient that standard time-series forecasting models.

Applying RL to portfolio optimization is a fairly old idea, but has never caught on in the industry. Ralph Neunier [18] employed 'Adaptive Dynamic Programming' (The lexicon of Machine Learning Engineers has changed greatly over the years) in 1995. He showed the feasibility of QL learning as a function approximator as a Markov Decision problem. Moody *et al* [16] also implemented direct reinforcement with great success. These researchers paved the way for Reinforcement Learning in Finance and Portfolio Management.

In 2021, Weiwei [11] summarized the recent progress of deep learning models in stock price prediction. He concluded that Graph Neural Network (GNN) and DRL models held promise in the field but were yet to be explored in depth. Cui *et al* [7] were one of the first to use convolutional neural networks (CNNs) for time-series forecasting of stock prices. It leverages the strength of CNN to automatically learn good feature representations in both time and frequency domains. In particular, MCNN contains multiple branches that perform various transformations of the time series, which extract features of different frequency and time scales, addressing the limitation of many previous works that they only extract features at a single timescale. Cheng *et al* [24] also attempted an attention-based LSTM model in stock prediction. While the results are undoubtedly impressive, it still leaves room for improvement in terms of computational efficiency.

Zhipeng Liang *et al* demonstrated superiority of adversarial models for portfolio mangement [13]. They implemented three state-of-art continuous reinforcement learning algorithms, Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO) and Policy Gradient (PG) in portfolio management. All of them are widely-used in game playing and robot control. The implemented algorithms were extremely sophisticated and took into account the volatility of the stock market and the assets (which would have influenced the investors). Chaouki *et al* [5] also explored deep deterministic learning for portfolio optimization, but also compared their approach to the known optimal strategies of the time. They concluded that their DDPG-RL approach was always able to recover the established baselines (which even investors deem difficult). It did not beat the optimal strategies but was very close to the optimal. However, one specificity of their approach was the reward function was assumed unknown. Thus, more efficient algorithms could be made that address this concern of theirs and could potentially take the model past the other 'optimal' strategies that they considered.

Similarly, Cannelli *et al* [4] employed Q Learning for hedging. If an agent is trained solely on simulated data, the run-time performance will primarily reflect the accuracy of the simulation, which leads to the classical problem of model choice and calibration. However, their approach was not as successful as what we saw above. Chen *et al* [6] and Liang *et al* [13] explored adversarial reinforcement learning with similar motivations as Feng *et al*.

In 2020, Liu *et al* [14] proposed a library of three layers: environments, agents and applications. The three layers of FinRL library are stock market environment, DRL trading agent, and stock trading applications. The agent layer interacts with the environment layer in an exploration-exploitation manner, whether to repeat prior beneficial decisions or to make new actions hoping to get greater rewards. The lower layer provides APIs for the upper layer, making the lower layer transparent to the upper layer. The environment would be a real-time time-driven trading simulator. By interacting with the environment, the trading agent will derive a trading strategy with the maximized rewards as time proceeds.

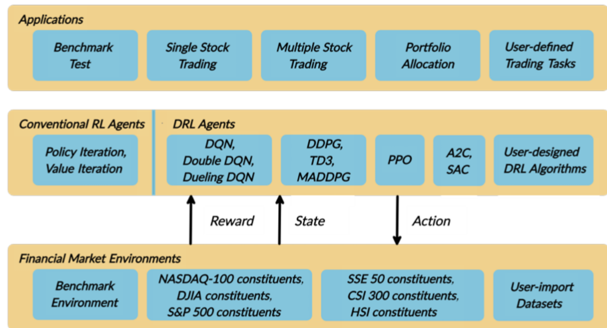


Figure 2: An overview of the FinRL Library's 3 layers

In the realm of online time series forecasting, addressing concept drift—where the underlying patterns of data change over time—is crucial. A recent notable work in this domain is "OneNet" by Yi-Fan Zhang *et al* [24]. OneNet innovatively tackles the challenge of concept drift by dynamically updating and combining two models: one focused on temporal dependencies and the other on cross-variable dependencies. This approach leverages the strengths of both models, enabling more accurate and robust forecasting in the face of evolving data patterns. OneNet incorporates a reinforcement learning-based strategy to adjust the weights of these models dynamically, showcasing a significant improvement in forecasting accuracy, reducing mean-squared errors by over 50% compared to the STOTA methods. This method's success underscores the potential of adaptive, multi-model strategies in handling the complexities of online forecasting, presenting a promising direction for our exploration in financial portfolio management with deep reinforcement learning.

3 Baseline Selection

In the literature, we encountered many issues with choosing a reinforcement learning model trained on finance data. While Chaouki *et al*'s theory was sound, the results were not stellar and the heuristics to evaluate the model were not standardized with the rest of the literature. Liu *et al*'s FinRL, while very informing, is more of a library of models than a novel model in itself. The implementation of these older architectures described in 2018 [14] may not be very practical as Reinforcement Learning has changed and adaptive within recent years. Azhikodan *et al* [3], Jia *et al* [10], and Meng *et al*'s [17] models also followed a similar approach using outdated reinforcement learning methods.

We used Adv-ALSTM, as suggested in [9], as the baseline implementation and run the model on two benchmarks on stock movement prediction. This adversarial learning approach helped them create a very generalized model. We were interested to see how it works since the goal of our project was to create hyper-specific models. Using a state of the art generalized model as our baseline would help validate or invalidate our hypothesis.

This baseline trained on two sets of data from gathered by StockNet [23]. The **ACL18** sources historical stock data from Jan-01-2014 to Jan-01-2016 of 88 high-trade-volume-stocks in NASDAQ and NYSE, while **KDD17** includes a longer range from Jan-01-2007 to Jan-01-2016 of 50 stocks in U.S. markets. This paper introduces adversarial training into the realm of financial market predictions, a domain characterized by high uncertainty and noise. This improves the model resilience against market volatility. The adversarial training method suggested by the paper is still referenced by many papers to this day. The paper offers a comprehensive experimental setup, including datasets, evaluation metrics, and model configurations, which can be easily used to benchmark our model's performance. The heuristics and evaluation metrics are simple to understand and compare with other architectures, the generalization is a great advancement towards incorporating AI in portfolio management, and their approach of normalizing the stock prices to explicitly capture the interaction seems most practical.

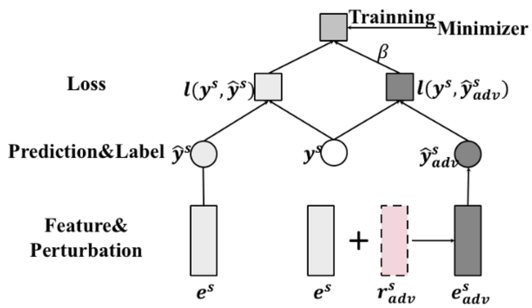


Figure 3: Illustration of the Adverserial Attentive LSTM

4 Baseline Implementation

During our research, we referenced a baseline model described in "Enhancing Stock Movement Prediction with Adversarial Training" [9] and were able to successfully replicate its resulting accuracies as demonstrated in Table 1. For the implementation and execution of our machine learning model, we utilized Google Cloud Platform (GCP). The model itself was constructed using TensorFlow and Keras. We first ran the model without the saved checkpoint and then with the checkpoint to verify the results. The run without the saved checkpoint was able to achieve the same accuracy when compared to one with the saved checkpoint after 150 epochs. This is corroborated in Figure 4 and Figure 13. Performance over each epoch was recorded on Wandb. The accuracy at the start of pre-trained model is very close to the accuracy achieved at the end of 150 epochs of the model run without saved checkpoints.

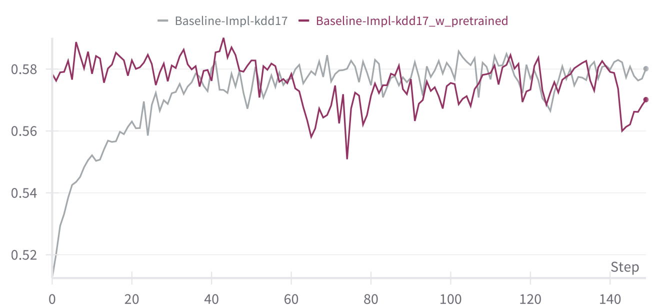


Figure 4: Comparing model run with and without saved checkpoint for KDD17 Data

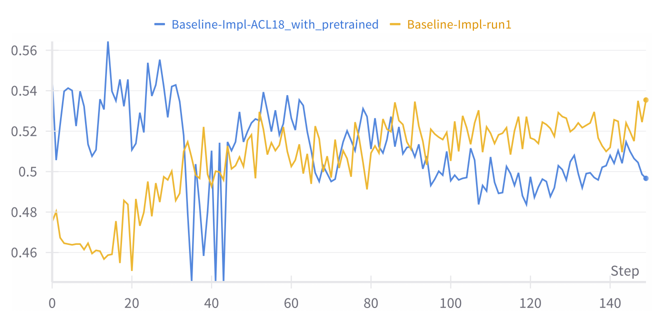


Figure 5: Comparing model run with and without saved checkpoint for KDD17 Data

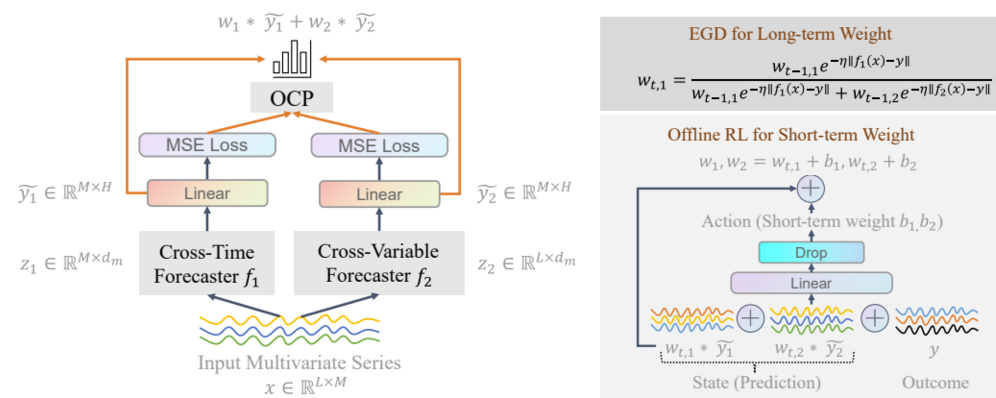
Table 1: Performance comparison on the two datasets.

Method	ACL18		KDD17	
	Acc	MCC	Acc	MCC
Adv-ALSTM	57.20	0.1483	53.05	0.0523
Our implementation	**58.72**	**0.1752**	**53.05**	**0.05292**
Difference	1.52	0.0269	0	0.00062

5 Model Description

5.1 OneNet Model for Online Time Series Forecasting

In addressing the challenge of concept drift in time series forecasting, OneNet stands out for its online ensembling approach, dynamically balancing model predictions to accommodate changing patterns in data over time. Figure 2 illustrates the dual architecture of OneNet, which processes multivariate data through two specialized branches: the Cross-Time Forecaster and the Cross-Variable Forecaster, each tailored to capture temporal and cross-variable dependencies, respectively. The core innovation of OneNet lies in its Online Convex Programming (OCP) block that integrates the predictions of both forecasters. This OCP block not only draws upon the long-term historical data through Exponentiated Gradient Descent (EGD) but also adapts rapidly to recent trends via offline reinforcement learning. This dual-weighting system allows OneNet to maintain robust forecasting while dynamically adapting to concept drift, which is often seen in real-world scenarios such as financial markets.



(a) **The overall OneNet architecture.** (b) **OCP block.**

Figure 6: (a) The Overall OneNet architecture (b) OCP block [24]

Key components of OneNet Architecture:

- **Cross-Time Forecaster(f1):** Focuses on the temporal correlation within the data, projecting the input series into a representation that captures time-dependent variations.
- **Cross-Variable Forecaster (f2):** Emphasizes the interdependencies between different variables within the data, using a last-step representation to predict future values.

The OCP block's weight adjustment is depicted in part (b) of Figure 2. The long-term weights managed by EGD are designed to capitalize on the overall historical performance, while the short-term weights generated by an offline RL policy target recent changes, allowing for agile adjustment to new patterns. The combined weight $w_{t,i}$ is then applied to the forecasters' predictions to produce the final ensemble forecast.

OneNet employs a decoupled training strategy, where each forecaster is trained independently to predict future values. The OCP block is then trained to find the optimal ensemble weights that minimize the prediction error when combining the forecasters' outputs. This approach ensures that both forecasters are adequately trained even if one initially outperforms the other, a crucial feature for maintaining performance under concept drift conditions.

5.2 Model Framework

We've adapted OneNet to predict individual companies' future stock prices. As discussed earlier, the most novel aspect comes from the combination of using EGD for online learning and long-term weights, and RL for offline learning and short-term weights. EGD had been traditionally used to make long-term predictions, whereas reinforcement learning has been a more recent concept. In general, RL describes a state space, in which an agent interacts with its environment and gathers rewards by learning policies in a trial and error manner. This algorithm deals with sequential decision making in a wide range of fields in both natural and social sciences, and engineering [21] but has gained more attention within financial applications. Mathematically, it can be represented as an optimization problem -

$$\max_{\pi} \quad \mathbb{E}_{t=0}^{T-1}\left[R_t(s_t, a_t, s_{t+1}, x_t)\right] \tag{1}$$

subject to:

$$s_{t+1} = f_t(s_t, a_t, h_t) \tag{2}$$

where $a_t \in A$ indicates the actions, $s_t \in S$ the state of the system at time t, ζ_t and η_t are noise variables, and R_t is the reward received at every time step. In RL the second line in the above equation is usually referred to as the 'environment'. The 'agent' intends to choose its actions a_t, given the state s_t, so as to maximize the total expected accumulated reward. Figure 7 shows this schematically.

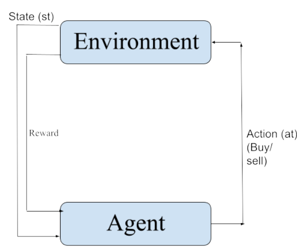


Figure 7: Schematic representation of Environment-agent interactions

In the context of our model, at time step *t*, the target aims to learn short-term weights conditioned on long-term weights $\mathbf{w}$ and experts' performances during a short period of history $I = [l, t]$. The agent then chooses actions using a policy $\pi_{\theta_{rl}}(\mathbf{b}_t | \{w_{t,i}\tilde{y}_{i}{}_{i=l}^{d}\}_{t \in I}; \mathbf{y})$ parameterized by θ_{rl}. During training, the product between each prediction and expert weight $(w_{t,i} * \tilde{\mathbf{y}}_i)$ is concatenated with the outcome y for the input. In reference to RvS [24], the policy network is implemented as a two-layer MLP $f_{rl} : \mathbb{R}^{H \times M \times (d+1)} \to \mathbb{R}^d$, with the short-term weight calculated as:

$$\mathbf{b_t} = f_{rl}(w_{t,1}\mathbf{y}_1 \otimes \cdots \otimes w_{t,d}\tilde{\mathbf{y}}_d \otimes \mathbf{y}) \tag{3}$$

and the final ensembling weight calculated as:

$$\tilde{w}_{t,i} = (w_{t,i} + b_{t,i}) / \left(\sum_{i=1}^{d}(w_{t,i} + b_{t,i})\right) \tag{4}$$

The network is then trained with the objective of minimizing the forecasting error incurred by the new weight: $\min_{\theta_{rl}} \mathcal{L}(\tilde{\mathbf{w}}) := ||\sum_{i=1}^{d} \tilde{w}_{t,i} fi(\mathbf{x}) - \mathbf{y}||^2$. As concept drift changes incrementally during inference, the expression $\mathbf{w}_{t-1} + \mathbf{b}_{t-1}$ is used to generate the prediction and train the networks after the ground truth outcome is observed:

$$\tilde{\mathbf{y}} = \tilde{w}_1 * \tilde{\mathbf{y}}_\mathbf{1} + \tilde{w}_2 * \tilde{\mathbf{y}}_\mathbf{2} \tag{5}$$

Updates to the long-term and short-term weights are as follows from the OneNet algorithm [24]:

$$w_i = w_i \exp(-\eta \|\tilde{\mathbf{y}}_\mathbf{i} - \mathbf{y}\|^2) / \sum_{i=1}^{2} w_i \exp(-\eta \|\tilde{\mathbf{y}}_\mathbf{i} - \mathbf{y}\|^2) \tag{6}$$

$$f_{rl} \leftarrow \text{Adam}(f_{rl}, \mathcal{L}(\tilde{w}_1 * \tilde{\mathbf{y}}_\mathbf{1} + \tilde{w}_2 * \tilde{\mathbf{y}}_\mathbf{2}, \mathbf{y})) \tag{7}$$

And, lastly, we update the two forecasters to be:

$$f_1 \leftarrow \text{Adam}(f_1, \mathcal{L}(\tilde{\mathbf{y}}_\mathbf{1}, \mathbf{y})), f_2 \leftarrow \text{Adam}(f_2, \mathcal{L}(\tilde{\mathbf{y}}_\mathbf{2}, \mathbf{y})) \tag{8}$$

5.3 Model Architecture

Table 2: OneNet architecture

Layer	Shape	Params
Sequential-21	[-1, 64]	9,856
Linear-22	[-1, 64]	1,344
Linear-23	[-1, 64]	12,352
SiLU-24	[-1, 64]	0
SamePadConv-25	[-1, 64, 20]	0
Linear-26	[-1, 3]	195
Linear-27	[-1, 1]	65
Linear-28	[-1, 1]	65
CosineSimilarity-29	[-1]	0
ConvBlock-30	[-1, 64, 20]	0
Conv1d-31	[-1, 64, 20]	12,288
Sequential-32	[-1, 64]	9,856
Linear-33	[-1, 64]	1,344
Linear-34	[-1, 64]	12,352
SiLU-35	[-1, 64]	0
SamePadConv-36	[-1, 64, 20]	0
Linear-37	[-1, 3]	195
Linear-38	[-1, 1]	65
Linear-39	[-1, 1]	65
CosineSimilarity-40	[-1]	0
ConvBlock-41	[-1, 64, 20]	0
Conv1d-42	[-1, 64, 20]	12,288
Sequential-43	[-1, 64]	9,856
Linear-44	[-1, 64]	1,344
Linear-45	[-1, 64]	12,352
SiLU-46	[-1, 64]	0
SamePadConv-47	[-1, 64, 20]	0
Linear-48	[-1, 3]	195
Linear-49	[-1, 1]	65
Linear-50	[-1, 1]	65
CosineSimilarity-51	[-1]	0
ConvBlock-52	[-1, 64, 20]	0
Conv1d-53	[-1, 64, 20]	12,288
Sequential-54	[-1, 64]	9,856
Linear-55	[-1, 64]	1,344
Linear-56	[-1, 64]	12,352
SiLU-57	[-1, 64]	0
SamePadConv-58	[-1, 64, 20]	0
Linear-59	[-1, 3]	195
Linear-60	[-1, 1]	65
Linear-61	[-1, 1]	65
CosineSimilarity-62	[-1]	0
ConvBlock-63	[-1, 64, 20]	0
ConvNet-64	[[-1, 20, 64], [-1, 20, 64], [-1, 20, 64], [-1, 20, 64]]	0
Total		225,600
Trainable		225,600
Non-trainable		0

6 Evaluation Metric and Loss Function

As mentioned in the introduction, the Loss function we have employed is MSE loss for training the component models of RLFinNet. We have found in the literature that common evaluation metrics for general time-series forecasting models is the MSE Loss (Mean Squared Error) in the training as well as the MAE of the testing portion. For MSE, we are aiming for a mean MSE loss of all models to be lower than 0.4, which is in line with OneNet's original implementation. While we would like to ensure that the MSE loss for all specialized models would be less than 0.4, we were not able to target this due to time constraints. Since the MAE (Mean Absolute Error) is closely linked to the MSE loss, we don't have a specific target for the MAE, but will compare the MAE for each of our experiments. For comparable MSEs, we expect to see similar MAEs. MSE is a common loss function used in regression tasks. It measures the average squared difference between the actual values and the predicted values. For a dataset with n samples, where y_i are the actual values and $\hat{y}_i$ are the predicted values, the MSE is calculated as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The Mean Absolute Error (MAE) is another loss function used in regression tasks. It measures the average absolute difference between the actual values and the predicted values. For a dataset with n samples, the MAE is calculated as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

We are also plotting the predicted closing price vs. the true closing price for 3 random companies – AFGS, BA, and MDT, chosen randomly. We have also plotted the training time for each companies that we recorded in one of our experiments. All experiments were performed on a Nvidia Tesla T4 GPU on a Google Cloud virtual machine with 26GB of RAM.

7 RLFinNet Adaptations and Experiments

7.1 Data Loader Update for Finance Data

To accommodate the financial dataset described in the Dataset section, we first updated our data loader to handle multiple features including the *Open*, *High*, *Low*, *Volume*, and *Adjusted Closing* prices, along with our target forecasting variable *Close*, or closing price. The input data thus became 7-dimensional, reflecting the multivariate nature of financial time series.

7.2 Training Loop Modification

Given the extensive data across various companies, it was imperative to modify the training loop to enable sequential processing. This setup ensures that once the model completes training and making predictions for one company's data, it automatically proceeds to the next, effectively managing multiple datasets in a single training regime. This loop not only facilitates extensive training across different datasets but also streamlines the process of validation and prediction for each company.

7.3 Implementation of Data Loader and Training Mechanism

Implementing the modified data loader involved ensuring that each company's data is loaded, processed, and fed into the model in succession. The training mechanism was adapted to this setup, allowing for a seamless transition between training sessions for different companies without manual intervention.

7.4 Visualization and Post-Processing

Post-training, it was crucial to set up a robust framework for visualizing and post-processing the results. This step included plotting the predicted vs. actual closing prices and assessing model performance across different metrics such as the MSE, MAE, and prediction vs. truth plots.

7.5 Experimental Setup

With the pipeline established, we conducted a series of experiments to explore the impact of varying prediction lengths and sequence lengths on the model's performance. We tested three different prediction lengths: 1, 20, and 40 days, alongside three sequence lengths: 10, 20, and 40 days. These experiments were designed to provide insights into the optimal configurations for both short-term and long-term forecasting accuracy.

7.6 Insights and Observations

The experimental results, detailed in the subsequent section, shed light on how different configurations influence the forecasting capabilities of our model. These findings are crucial for understanding the trade-offs between responsiveness and accuracy in financial time series forecasting. Finding optimal configurations will require many more ablations.

8 Results and Discussion

8.1 MAE for varying sequence lengths

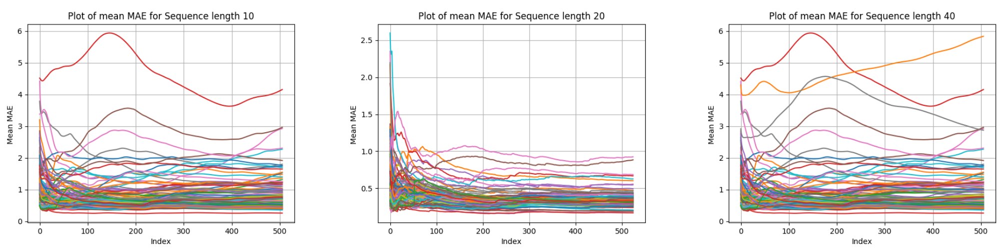


Figure 8: MAE for each company in the dataset when trained with sequence lengths of 10, 20 and 40

8.2 MSE loss for varying sequence lengths

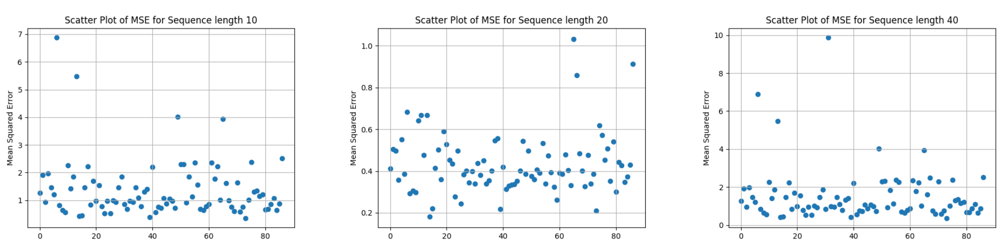


Figure 9: MSE for each company in the dataset when trained with sequence lengths of 10, 20 and 40

8.3 Predictions vs. Ground Truth for companies AGFS, BA, MDT

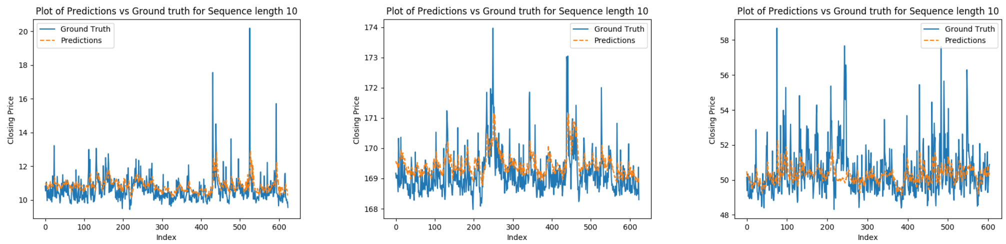


Figure 10: Predictions vs. Ground Truth with encoder sequence length=10

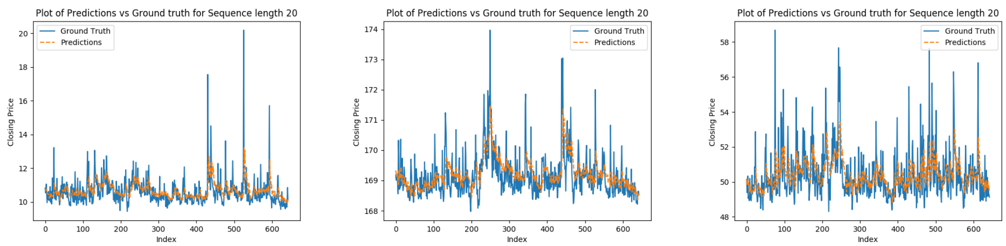


Figure 11: Predictions vs. Ground Truth with encoder sequence length=20

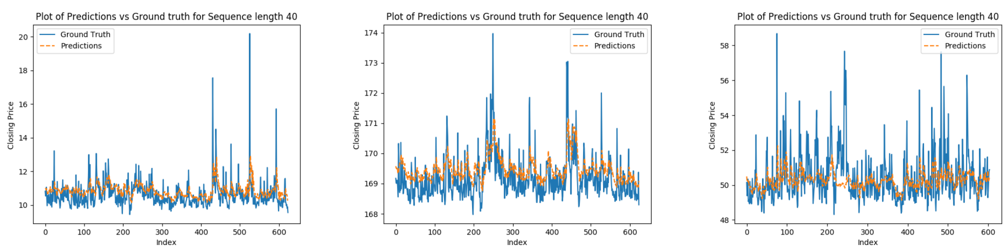


Figure 12: Predictions vs. Ground Truth with encoder sequence length=40

8.4 Training Time

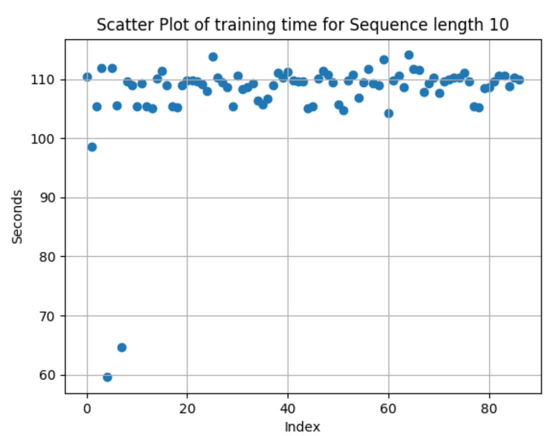


Figure 13: Training time for each specialized model

The original OneNet performed best with a sequence length of 40 on their hourly and monthly training on the electricity transformer temperature (ETT) data, while ours did best with a sequence length of 20. We observe from figure 8 that the encoder sequence length of 20 had the smoothest configuration for the companies, while lengths 10 and 40 exhibit some anomalies. We suspect that 10 days is not enough context to make an accurate prediction, whereas 40 leads to an overfitting of the model. Figure 9 demonstrates a similar pattern, where a sequence length of 20 in general had a more uniform average $MSE < 0.4$, while there exists some outliers for the other two plots. This average, from out literature reviews, is relatively normal within research in forecasting time-series data, though the original OneNet had an MSE values around 0.35 for their ablations on the ETT dataset. Following, our most insightful comparisons come from the ground truth vs. prediction plots of three random companies, AGFS, BA, and MDT. All 3 companies in all three lengths were able to predict the trends of the market conditions pretty closely though did not forecast the extreme variances that exist in the stock market. This is expected since our training tries to model the distribution of the data and cannot precisely predict when spikes occur in the sequence of data. Since there is scarce research done on forecasting specifically stock price data using MSE or MAE as metrics, we consider our model relatively new within the domain and omit comparisons with previous baseline models. To assess the computational efficiency of RLFinNet, we plot the training times with encoder sequence length 10 as a reference in Figure 13. and observe that each company less than 2 minutes to train on, making our model quick and lightweight.

9 Conclusion and Future Works

From our experimentation, we have observed that RLFinNet effectively deals with concept drifts and visually performs relatively well in forecasting performances of stock prices. At only 225k parameters with our current implementation, it is very lightweight easily accessible for each company. Our prediction vs. ground truth plots demonstrate that the predictions are able to model the volatility of the stock market but not to the extreme variances, but this is to be expected due to the non-stationary nature of the financial market. While RLFinNet is designed to train on real-time data, we have only worked with public stock price datasets for 87 different companies. We aim to configure our network on a real trading floor and streamline financial data from the online market through various APIs. Additionally, we hope to automate the process of hyper-parameter tuning to be tailored for trading users in their companies of interest.

References

[1] Ahmadi, S. (2024). A Comprehensive Study on Integration of Big Data and AI in Financial Industry and its Effect on Present and Future Opportunities. International Journal of Current Science Research and Review, 07 (01), 66-74.

[2] Alemansour, M.M., & Rastgarpour, M. (2021). Improving the Prediction of Cryptocurrencies Movement Using Adversarial Training. In M. Rastgarpour (Ed.), Progress in Intelligent Decision Science: Proceeding of IDS 2020 (pp. 282-293). Springer International Publishing.

[3] Azhikodan, A.R., Bhat, A.G.K., & Jadhav, M.V. (2019). Stock trading bot using deep reinforcement learning. In Innovations in Computer Science and Engineering: Proceedings of the Fifth ICICSE 2017 (pp. 41-49). Springer Singapore.

[4] - Cannelli, Loris, Giuseppe Nuti, Marzio Sala, and Oleg Szehr. "Hedging using reinforcement learning:331 Contextual k-armed bandit versus Q-learning." The Journal of Finance and Data Science 9 (2023): 10010

[5] Chaouki, A., Hardiman, S., Schmidt, C., Sérié, E., & De Lataillade, J. (2020). Deep deterministic portfolio optimization. The Journal of Finance and Data Science, 6 (1), 16-30.

[6] Chen, Y.-Y., Chen, C.-T., Sang, C.-Y., Yang, Y.-C., & Huang, S.-H. (2021). Adversarial attacks against reinforcement learning-based portfolio management strategy. IEEE Access, 9, 50667-50685.

[7] Cheng, L.-C., Huang, Y.-H., & Wu, M.-E. (2018). Applied attention-based LSTM neural networks in stock prediction. In 2018 IEEE International Conference on Big Data (Big Data) (pp. 4716-4718). doi: `10.1109/BigData.2018.8622541`.

[8] Emmons, S., Eysenbach, B., Kostrikov, I., & Levine, S. (2022). Rvs: What is essential for offline rl via supervised learning? ICLR.

[9] Feng, F., Chen, H., He, X., Ding, J., Sun, M., & Chua, T.-S. (2019). Enhancing Stock Movement Prediction with Adversarial Training. In IJCAI, 19, 5843-5849.

[10] Jia, W. U., Chen, W. A. N. G., Xiong, L., & Hongyong, S. U. N. (2019). Quantitative trading on stock market based on deep reinforcement learning. In 2019 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE.

[11] Jiang, W. (2021). Applications of deep learning in stock market prediction: recent progress. Expert Systems with Applications, 184, 115537.

[12] Jiang, Z., Xu, D., & Liang, J. (2017). A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*.

[13] Liang, Z., Chen, H., Zhu, J., Jiang, K., & Li, Y. (2018). Adversarial deep reinforcement learning in portfolio management. *arXiv preprint arXiv:1808.09940*.

[14] Liu, X.-Y., Yang, H., Chen, Q., Zhang, R., Yang, L., Xiao, B., & Wang, C.D. (2020). FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance. *arXiv preprint arXiv:2011.09607*.

[15] Liu, X.-Y., Xiong, Z., Zhong, S., Yang, H., & Walid, A. (2018). Practical deep reinforcement learning approach for stock trading. *arXiv preprint arXiv:1811.07522*.

[16] Moody, J., & Saffell, M. (2001). Learning to trade via direct reinforcement. IEEE transactions on neural Networks, 12 (4), 875-889.

[17] Meng, T.L., & Khushi, M. (2019). Reinforcement learning in financial markets. Data, 4 (3), 110.

[18] Neuneier, R. (1995). Optimal asset allocation using adaptive dynamic programming. Advances in neural information processing systems 8.

[19] Scott, E., Eysenbach, B., Kostrikov, I., & Levine, S. (2022). Rvs: What is essential for offline rl via supervised learning? ICLR.

[20] Shaban, W.M., Ashraf, E., & Slama, A.E. (2024). SMP-DL: a novel stock market prediction approach based on deep learning for effective trend forecasting. Neural Comput & Applic, 36, 1849–1873.

[21] Sutton, R.S., & Barto, A.G. (1999). Reinforcement learning: An introduction. Robotica, 17 (2), 229-235.

[22] World Bank. "Market Capitalization of Listen Companies." Retrieved from `https://data.worldbank.org/indicator/CM.MKT.LCAP.CD/`.

[23] Xu, Y., & Cohen, S. (2018). Stock movement prediction from tweets and historical prices. In ACL, 1, 1970–1979.

[24] Wen, Q., Chen, W., Sun, L., Zhang, Z., Wang, L., Jin, R., & Tan, T. (2024). Onenet: Enhancing time series forecasting models under concept drift by online ensembling. Advances in Neural Information Processing Systems 36.

Work Division: [1] **GitHub Repository**: [2]

[1] Naman Choudhary: Research OneNet + RLFinNet adaptions, Karthik Subramaniam: Research OneNet + RLFinNet ablations/adaptions, Nimish Jindal: Research OneNet/Adv-ALSTM baseline + RLFinNet adaptions, Sarah Li: Research OneNet + RLFinNet adaptions

[2] GitHub