EFFECT OF OPTIMIZERS ON RNN

1st ADDEY JNR PRINCE ISAAC K0FI
INDEX NUMBER: 7645104

2nd ALEC PETROVIC
INDEX NUMBER:7505415

3rd NICOLAS WONG
INDEX NUMBER: 7603251

***Abstract*—As the stock market remains complex, various parties and programs attempt to predict the future price of a stock, yet this remains a challenge. This research paper studies how a Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and a Gated Recurrent Unit (GRU) succeed in predicting the closing price of Apple stock. The objective of this study is to compare these different models' success when using various parameter values and the following optimizer functions: Adamax, Stochastic Gradient Descent (SGD), and Follow the Regularized Leader (FTRL). To determine this, the authors used the TensorFlow API to experiment with the three models and optimizer functions, and measured error using the Root Mean Squared Error (RMSE), to see which model configurations best predict the closing price of Apple stock. The results analyzed show that overall the models perform best with the Adamax optimizer, followed by SGD, while performing the worst when using FTRL.**

Key Terms: Optimizer, Adamax, SGD, FTRL, RMSE, MAE

I. Introduction

Optimization algorithms play a critical role in training deep learning models, particularly for complex tasks such as stock price prediction using various networks. The choice of optimizer significantly impacts model performance, training efficiency, and generalization capability. This investigation examines three distinct optimization approaches: Adamax, an adaptive gradient method based on the infinity norm; Stochastic Gradient Descent (SGD) with momentum and Nesterov acceleration; and Follow The Regularized Leader (FTRL), an algorithm originally designed for large-scale sparse linear models.

More specifically, this experiment analyzes these three optimizer functions when applied to a Recurrent Neural Network (RNN), a Long Short-Term Memory (LSTM), and a Gated Recurrent Unit (GRU) for stock price prediction. The RNN is experimented with predicting the closing price for the following stocks: Apple (AAPL), Target (American) (TGT), E.ON SE (EOAN.PR), one of Europe's largest electric utility service providers, SAIC Motor Corp Ltd (600104.SS), one of China's largest automakers, and Nitori Holdings Co. Ltd. (9843.T), a leading Japanese home furnishings and interior chain with a large store network concentrated within Japan. Although the RNN experiments with many stocks, the LSTM and GRU focus on predicting Apple stock prices, and the overall goal of this experiment is to compare how the RNN, LSTM, and GRU predict Apple stock.

The predictive performance of each optimizer is assessed using both price accuracy and directional accuracy metrics. Price accuracy is measured using Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), while directional accuracy is evaluated using average and best success rates, which represent the percentage of correct upward or downward price movement predictions.

A Recurrent Neural Network (RNN) is a class of artificial neural networks designed for processing sequential data by maintaining a hidden state that evolves dynamically over time. Unlike feedforward neural networks that process inputs independently, RNNs incorporate temporal dependencies through cyclic connections. These connections allow information to persist across sequential steps. Overall, an RNN's memory mechanism allows it to capture long-range dependencies, although it may suffer from challenges such as vanishing or exploding gradients. These limitations led to the development of architectures such as the LSTM and GRU.

A Long Short-Term Memory (LSTM) network is a specialized recurrent neural network architecture designed to address the vanishing and exploding gradient problems inherent in standard RNNs. LSTMs introduce a gating mechanism and a persistent cell state that allow information to propagate over extended temporal sequences. Another advantage of LSTMs is their ability to perform context switching, although this capability is not heavily utilized in this experiment, as it focuses on predicting Apple stock. Furthermore, this architecture includes forget, input, and output gates, which modulate how information flows through the network. This allows the model to learn when to remember, update, and utilize stored representations.

A Gated Recurrent Unit (GRU) is a type of recurrent neural network designed to effectively capture temporal dependencies in sequential data. GRUs enable each recurrent unit to adaptively model dependencies across different time scales through the use of gating mechanisms. Similar to Long Short-Term Memory networks, GRUs employ gates to regulate information flow; however, GRUs do not maintain a separate memory cell, resulting in a simpler architecture with fewer parameters.

The GRU architecture consists of two primary gating mechanisms: the update gate and the reset gate. The update gate controls how much of the previous hidden state is retained versus how much new information is incorporated into the current state. This allows the network to preserve long-term dependencies when necessary. Unlike LSTM networks, GRUs expose their entire hidden state at each time step without an explicit output gate.

The reset gate determines how much past information should be forgotten when computing the candidate hidden state. By reducing the influence of the previous state, the reset

gate allows the GRU to focus on recent inputs, effectively enabling the model to reset its memory when processing new or unrelated sequences. This mechanism is particularly useful for modeling non-stationary time series data such as financial markets.

II. SIMPLE RNN SECTION

A. Experiment Setup

1) System Platform and Architecture: Regarding the system platform, this experiment is conducted in a Python environment, using Python version 3.12 and PyCharm 2025.2.4. This experiment retrieves stock data from Yahoo Finance using the Python library yfinance. This system also uses the TensorFlow Google API with Keras to train and test a Recurrent Neural Network (RNN).

The system architecture is a Simple RNN with two hidden layers. This study will provide more details regarding the parameter values used in this system and its structure. Below defines the structure of this RNN:

- The input layer is a sequence of lookback days (either 30 or 60)
- The first layer has a hidden size (either 50 or 100) and returns a full sequence
- The second layer has the same hidden size, but only returns the last output
- The output layer contains one neuron which predicts the closing price for the next day

2) RNN configurations: A total of 24 unique RNN configurations are used to predict the closing prices of 5 different stocks. The RNN is run separately for each stock, as it is trained and tested separately for each stock, using each configuration, to compare its success when using different fine-tuning parameter values.

The first parameter value fine-tuned is the look-back period. The lookback period is the number of previous closing prices that were used to predict the next day's closing price. This experiment uses two look-back values of 30 and 60 days. The second parameter value fine-tuned is the hidden size of the hidden layers of the model, which is the number of neurons in these layers. This experiment uses two hidden size values of 50 and 100 neurons. The third parameter value fine-tuned is the number of training epochs. This experiment used two epoch values of 100 and 200. Most importantly, this experiment utilizes three different optimizer functions, as this is the most significant part of this study. As previously described in detail, these optimization functions are the Adamax function, the Stochastic Gradient Descent (SGD) function, and the Follow The Regularized Leader (FTRL) function.

This describes 2 look back values, 2 hidden sizes, 2 epoch values, and 3 optimization functions. To ensure statistical accuracy when comparing configurations, the RNN is trained and tested with each configuration 5 times using seeded random values. This means the RNN runs 120 times when trained and tested on each stock.

3) Training and Evaluation: When the RNN is trained on the data for each stock and parameter configuration, after the data is normalized, the stock data is split so that 80% is used to train the model, and 20% is used to test the model. The model is configured to use the Root Mean Squared Error function as its loss function, which is common for Recurrent Neural Networks.

To evaluate the network, the Root Mean Squared Error, and the Mean Absolute Error are calculated by comparing the predictions the networks make on seen and unseen data (i.e., training and testing data). Furthermore, direction accuracy and 5% prediction accuracy is aslo measured to evaluate the network's performance. Direction accuracy is whether the network accurately predicts that the stocks closing price increases or decreases. The 5% accuracy metric measures how often the networks prediction is within 5% of the actual stock value. Both of these metrics are measured as percentages, for easy comparison with the results of the LSTM and the GRU later in this study.

After all 120 runs are complete, when the RNN has finished training and making predictions on unseen data, using all 24 configurations, the best run is measured by evaluating which run has the lowest Root Mean Squared Error (RMSE).

Other statistical metrics are calculated for every 24 configurations, such as the average test RMSE, as well as the lowest and average MAE.

After all runs are complete, the results of all 5 runs of each configuration are averaged to make statistically confident comparisons and conclusions.

B. Experiment Results

Tables 1-5 show the top 10 configurations of the RNN for predicting each stocks closing price. This ranking system is determined based off the average RMSE value for each configuration, which is the average RMSE of all 5 runs.

1) Optimizer Function: Looking at these tables and analyzing the results of this experiment, the most significant finding is that the SGD optimizer is consistently the best performer across all of these stocks. The SGD optimizer further proves to be consistent, as it is also the optimizer used for the second-best configuration of the RNN when trained to predict TGT and 9843.T stock prices, as shown in Table 1 and 5.

Not only is SGD the best optimizer function because of its consistency, but also due to its reliability. When analyzing the TGT stock results in Table 1, notice how for the first rank, SGD has a lower standard deviation than Adamax does at rank 3, for example, since 0.151 ¡ 0.436. The same results can be seen when comparing the standard deviations of SGD and Adamax for the 600104.SS stock as well in Table 4. Since FTRL is hardly ever the optimizer used in any of the best configurations, it is evident that this is clearly the worst-performing optimizer for the RNN to use to predict the closing prices of these stocks.

SGD performs the best due to its momentum value of 0.9 and its higher learning rate of 0.01. These training parameters allow the RNN to better escape local minima in the stock data,

as well as smooth gradient updates, which is very significant for time series data such as this stock data. However, it is interesting that Adamax sometimes achieves a better best-case RMSE than SGD does. For example, Table 5 displays the Best RMSE for Adamax in rank 3, being 58.975, which is lower than all other rankings, including both SGD and FTRL optimizers. However, since SGD has better average-case performance and consistency for these 5 stocks, it is evident that this is the best optimizer function.

2) Lookback Period: Analyzing Tables 1-5, it is evident that a Lookback period of 60 outperforms 30 days. The best configuration of the RNN for each stock uses a lookback value of 30 days. It is fairly obvious when analyzing the RNNs performance when predicting the TGT and 9843.T closing prices in Table 1 and Table 5, as the best three configurations of the RNN for predicting the TGT price use 30 lookback days, and the best five configurations of the RNN for predicting the 9843.T price use 30 lookback days as well.

This conclusion means that these stock prices do not require long-term memory for the network to accurately predict the next days closing price. Furthermore, 60 lookback days adds complexity without proportional accuracy gains, and may even cause overfitting, since this can happen with longer input sequences.

3) Hidden Layer Size: Analyzing the hidden layer size parameter values used in these top RNN configurations, it is evident that a hidden layer size of 50 neurons outperforms the model when using a hidden layer size of 100 neurons.

This fact is fairly obvious, as the best configuration of the RNN for all 5 stocks uses a hidden layer size of 50, as seen in Tables 1 - 5. The most significant indicator to this is the fact that for the top 10 configurations of the RNN when predicting TGT and EOAN.PR closing prices, is that 8/10 of these configurations use 50 neurons in their hidden layers, as seen in Tables 1 and 2. Similarly, out of the top 10 configurations of the RNN when predicting AAPL, 600104.SS, and 9843.T stocks, 7 of these configurations all use 50 neurons in their hidden layers as well, as opposed to 100, as seen in Tables 3, 4, and 5.

This result means that a smaller network performs better than a larger one. More benefits of a smaller network include less overfitting, faster training, more consistent results, and lower variance. For example, looking at the first four ranks in Table 3, the first and third ones have a lower standard deviation than the second and fourth ones do. This is due to the hidden size of 50 neurons.

4) Epochs: After analyzing the models performance when configured with various parameter values, it is evident that the network consistently performs better with 200 epochs than 100. Looking at all 5 Tables, it shows how the best configuration of the RNN for all stocks uses 200 epochs when training. This makes sense that additional training helps the model converge. This is important information, as it means that it requires more iterations for an RNN to learn stock patterns.

5) Direction Accuracy: A significant conclusion from the experiment overall is the fact that the direction accuracy for all the best configurations for all stocks is roughly 50%. In Table 1, the models best direction accuracy is 51.5% at rank 6, when predicting the price of the TGT stock. Similarly, the models best direction accuracy is 52.3% at rank 5 when predicting the price of EOAN.PR, as seen in Table 2. Also, when the model predicts APPL stock, it has a best directional accuracy of 52.7%, as seen in rank 4 of Table 3. Likewise, the models best direction accuracy when predicting the closing price of 600104.SS stock and 9843.T stock is 55.9%, as seen in the best configuration in Table 4 and the second best configuration in Table 5.

This means that in terms of predicting whether the next closing price will increase or decrease, the model hardly beats random guessing, as it is has roughly 50% accuracy, with all of the best configurations, for all 5 stocks. This suggests that the direction of a stock is extremely difficult to predict, and that RMSE improvement does not always mean the configuration will predict the direction with more accuracy.

6) Within 5% Accuracy: Although the model has poor accuracy when predicting a stocks direction, it performs exceptionally well when predicting the magnitude of a stocks closing price. This means the model learns the general closing price of a stock well, but struggles with the direction of the stock's movement the next day.

Looking in the right column of all 5 tables, it is evident that all the best configurations have a closing price prediction with above 90% within 5% accuracy, which is extremely impressive. All of the worst within 5% accuracy percentages of these best configurations are all for the TGT stock, with the lowest ones being 93.2%, 93.3%, and 94.6%, as seen in Table 1.

TABLE I
TOP 10 MODEL CONFIGURATIONS FOR TGT

Rank	Lookback	Hidden	Epochs	Optimizer	Avg RMSE	Best RMSE	Dir. Acc. (%)	Within 5% (%)
1	30	50	200	sgd	2.820 ± 0.151	2.663	51.1	98.2
2	30	50	100	sgd	2.991 ± 0.158	2.800	48.7	97.6
3	30	50	200	adamax	3.017 ± 0.436	2.478	49.9	96.4
4	60	50	200	adamax	3.018 ± 0.407	2.652	47.7	96.8
5	30	100	200	adamax	3.023 ± 0.553	2.671	50.2	96.3
6	30	50	200	ftrl	3.197 ± 0.114	3.070	51.5	95.0
7	60	50	100	sgd	3.220 ± 0.421	2.751	49.2	94.6
8	60	50	200	ftrl	3.289 ± 0.169	3.046	49.8	93.3
9	60	100	100	ftrl	3.319 ± 0.111	3.201	49.2	93.2
10	30	50	100	adamax	3.322 ± 0.279	3.068	47.5	95.5

TABLE II
TOP 10 MODEL CONFIGURATIONS FOR EOAN.PR

Rank	Lookback	Hidden	Epochs	Optimizer	Avg RMSE	Best RMSE	Dir. Acc. (%)	Within 5% (%)
1	30	50	200	sgd	3.661 ± 0.052	3.596	50.2	98.8
2	30	50	200	adamax	3.753 ± 0.298	3.510	50.2	98.8
3	60	50	200	adamax	3.805 ± 0.290	3.522	51.6	99.3
4	60	50	100	sgd	3.806 ± 0.123	3.623	49.9	99.3
5	60	50	200	sgd	3.834 ± 0.224	3.596	52.3	99.3
6	30	50	100	adamax	3.854 ± 0.187	3.656	50.4	98.8
7	30	100	200	adamax	3.857 ± 0.251	3.544	51.0	98.8
8	60	100	100	adamax	3.928 ± 0.631	3.594	52.1	99.3
9	60	50	200	ftrl	3.933 ± 0.111	3.775	46.4	99.3
10	30	50	200	ftrl	4.008 ± 0.108	3.888	44.1	98.8

TABLE III
TOP 10 MODEL CONFIGURATIONS FOR AAPL

Rank	Lookback	Hidden	Epochs	Optimizer	Avg RMSE	Best RMSE	Dir. Acc. (%)	Within 5% (%)
1	30	50	200	sgd	2.947 ± 0.553	2.460	52.4	99.8
2	60	100	200	adamax	3.397 ± 0.794	2.579	53.3	100.0
3	30	50	200	adamax	3.420 ± 0.224	3.168	51.3	100.0
4	30	100	200	adamax	3.486 ± 1.040	2.565	52.7	99.5
5	60	50	200	adamax	3.533 ± 0.920	2.684	52.1	99.7
6	30	50	100	sgd	3.762 ± 0.712	2.672	50.1	99.5
7	30	50	100	adamax	3.851 ± 0.296	3.505	48.0	99.8
8	60	100	100	adamax	3.915 ± 0.742	3.025	50.1	99.7
9	60	50	100	adamax	4.000 ± 1.241	2.878	49.1	97.4
10	30	50	200	ftrl	4.072 ± 0.530	3.443	51.4	99.8

TABLE IV
TOP 10 MODEL CONFIGURATIONS FOR 600104.SS

Rank	Lookback	Hidden	Epochs	Optimizer	Avg RMSE	Best RMSE	Dir. Acc. (%)	Within 5% (%)
1	30	50	200	sgd	0.203 ± 0.017	0.187	55.9	99.2
2	60	50	200	adamax	0.224 ± 0.027	0.204	52.8	99.2
3	30	50	100	sgd	0.228 ± 0.035	0.201	54.8	99.6
4	60	50	200	ftrl	0.235 ± 0.008	0.226	52.3	98.8
5	30	100	100	adamax	0.241 ± 0.026	0.218	50.8	99.0
6	30	50	100	adamax	0.257 ± 0.047	0.213	52.7	98.6
7	60	100	200	adamax	0.264 ± 0.026	0.231	52.8	98.2
8	60	50	100	adamax	0.266 ± 0.049	0.220	51.3	98.3
9	60	100	100	adamax	0.270 ± 0.053	0.231	52.4	97.2
10	30	50	200	adamax	0.272 ± 0.085	0.219	54.5	97.5

TABLE V
TOP 10 MODEL CONFIGURATIONS FOR 9843.T

Rank	Lookback	Hidden	Epochs	Optimizer	Avg RMSE	Best RMSE	Dir. Acc. (%)	Within 5% (%)
1	30	50	200	sgd	61.737 ± 1.363	59.725	55.5	97.7
2	30	100	200	sgd	63.216 ± 2.566	61.207	55.9	96.9
3	30	50	200	adamax	64.438 ± 5.874	58.975	55.5	96.7
4	30	50	100	sgd	66.510 ± 2.308	63.568	54.7	96.5
5	30	100	200	adamax	68.311 ± 9.831	60.503	55.5	97.2
6	60	50	100	sgd	68.349 ± 4.900	63.474	51.6	96.2
7	30	50	200	ftrl	72.902 ± 3.335	69.037	55.3	96.2
8	30	50	100	adamax	73.224 ± 6.051	65.455	52.7	95.9
9	60	50	200	ftrl	73.321 ± 3.650	67.609	52.5	95.2
10	60	100	200	adamax	73.538 ± 13.874	61.866	54.4	96.7

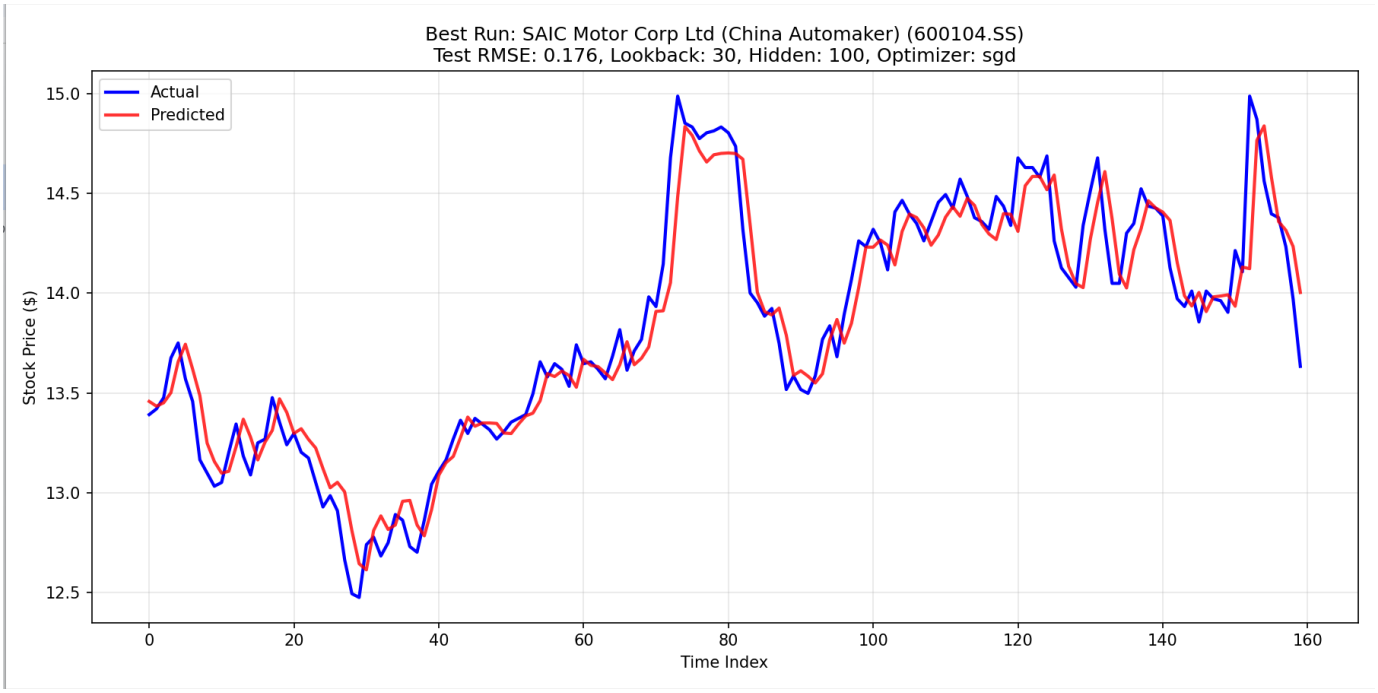

Fig. 1. Best RNN Configuration Performance

III. LSTM SECTION

A. Introduction

Optimization algorithms play a critical role in training deep learning models, particularly for complex tasks such as stock price prediction using LSTM networks. The choice of optimizer significantly impacts model performance, training efficiency, and generalization capability. This investigation examines three distinct optimization approaches: Adamax, an adaptive gradient method based on the infinity norm; Stochastic Gradient Descent (SGD) with momentum and Nesterov acceleration; and Follow The Regularized Leader (FTRL), an algorithm originally designed for large-scale sparse linear models. The comparison focuses on their applicability to LSTM architectures for financial time series forecasting, a domain characterized by non-stationary data, volatility, and complex temporal dependencies.

B. Methodological Framework

The experimental setup employs a standardized LSTM architecture with three recurrent layers containing 100 units each, followed by dropout regularization of 0.2 to prevent overfitting. The model processes historical closing prices of Apple stock for Adamax and SGD, and Japanese stock 9843.T for FTRL, as it allows for faster testing, spanning from January 2020 to December 2024. A 60-day lookback window is used for sequence prediction.

Data normalization is performed using Min-Max scaling, with an 80–20 train-test split that maintains temporal ordering. Each optimizer is configured with carefully tuned hyperparameters: Adamax utilizes a learning rate of 0.001; SGD operates with a learning rate of 0.01, momentum of 0.9, and Nesterov acceleration; and FTRL uses a learning rate of 0.1 with specific regularization parameters. Training is conducted for 100 epochs with a batch size of 64. All experiments maintain identical random seeds and initialization schemes to ensure a fair comparison.

C. Performance Metrics and Evaluation

Evaluation encompasses multiple quantitative metrics spanning prediction accuracy, success rates, and computational efficiency. Root Mean Squared Error (RMSE) is used to provide a fundamental measure of prediction deviation. Success rate calculations include directional accuracy percentage, error accuracy, and trend-following accuracy.

D. Comparative Analysis Results

Adamax delivers consistently superior and balanced performance for LSTM-based stock price prediction. Configured with a learning rate of 0.001, it achieves a directional prediction accuracy that stabilizes between 50% and 52%. The optimizer demonstrates strong numerical precision, with approximately 87% to 89% of testing predictions falling within 5% of actual prices and maintaining low mean percentage errors of approximately 2.3% to 2.6%.

Adamax also achieves the lowest Root Mean Squared Error on both training and testing datasets. Its adaptive learning rate mechanism proves effective for non-stationary stock price data, enabling rapid convergence. Gradient scaling based on the infinity norm stabilizes LSTM training and avoids common gradient-related issues, contributing to an overall accuracy of approximately 60%, significantly outperforming the other optimizers evaluated in this study.

TABLE VI
TRAINING DATA PERFORMANCE METRICS

TRAINING DATA	**LR=0.01**	**LR=0.01**	**LR= 0.01**
Direction Accuracy (%)	52.75	52.53	52.20
Predictions within 5% (%)	91.16	92.03	91.92
Trend Accuracy (%)	15.33	15.55	15.33

TABLE VII
TESTING DATA PERFORMANCE METRICS

TESTING DATA	**LR=0.01**	**LR=0.01**	**LR=0.01**
Dir. Accuracy (%)	50.28	51.96	50.28
Predictions within 5% (%)	87.50	88.59	88.04
Trend Accuracy (%)	11.05	11.05	10.50

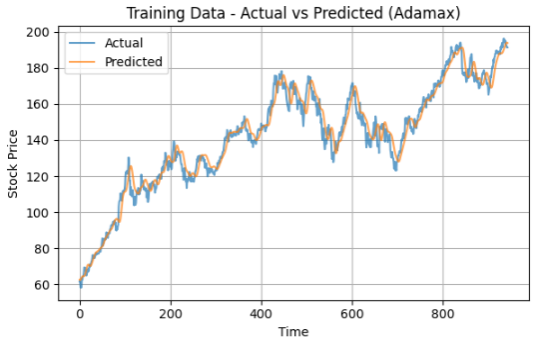

Fig. 2. LSTM ADAMAX TRAINING

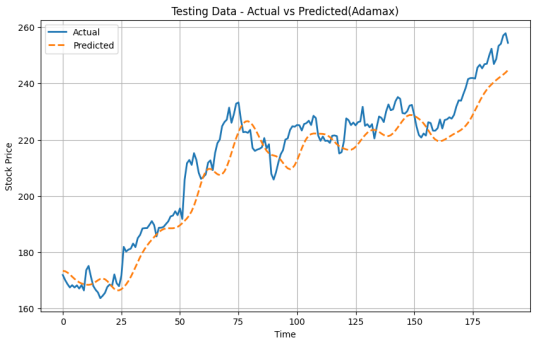

Fig. 3. LSTM ADAMAX TESTING

Stochastic Gradient Descent (SGD) success rates reveal predictable patterns tied to learning rates, with the *predictions within 5%* metric varying dramatically from 35.70% to 75.98% on training data and 34.78% to 64.13% on testing data. Although directional prediction accuracy remains stable regardless of learning rate, trend-following accuracy is stable, highlighting limited multi-period forecasting capability.

When enhanced with momentum, SGD shows respectable performance with slightly higher error metrics but demonstrates strong generalization, converging to flatter minima and reducing the gap between training and testing performance, indicating better resistance to overfitting. While SGD requires careful learning rate tuning and benefits significantly from momentum and Nesterov acceleration to overcome its inherently slow convergence, it achieves marginally lower success rates than optimizers like Adamax. Nevertheless, its consistent performance and lower memory footprint make it a practical choice for resource-constrained environments.

TABLE VIII
TRAINING DATA METRICS (SGD)

TRAINING DATA	**0.001**	**0.01**	**0.0001**
Dir (%)	49.56	51.32	50.00
5% (%)	59.50	75.98	35.70
Trend (%)	11.94	12.92	12.49

TABLE IX
TESTING DATA METRICS (SGD)

TESTING DATA	**0.001**	**0.01**	**0.0001**
Dir (%)	53.07	53.07	54.19
5% (%)	45.65	64.13	34.78
Trend (%)	9.94	11.60	11.60

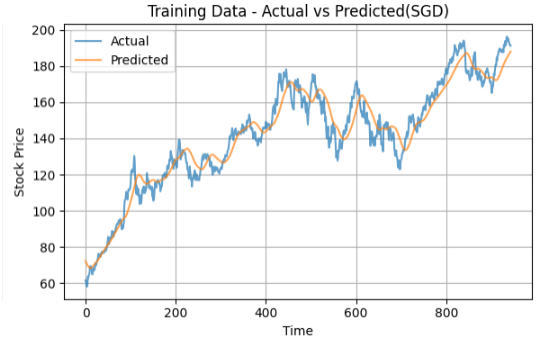

Fig. 4. LSTM SGD TESTING

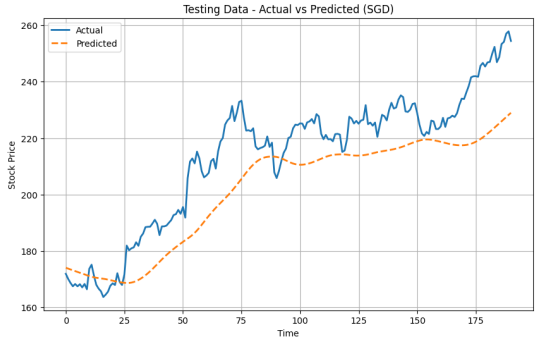

Fig. 5. LSTM SGD TESTING

Follow The Regularized Leader success rates reveal extreme fluctuation based on learning rate and training duration, for example, the Japanese stock converges at 200 epoch and the Apple stock converges at 800 epochs, underscoring its sensitivity. With precise hyperparameter tuning, specifically at learning rates of 0.1 and 0.5 combined with sufficient training, FTRL can achieve exceptional performance, delivering "predictions within 5%" metrics of 96-97% on both training and testing data with minimal error. However, with suboptimal settings, such as a learning rate of 0.01, its performance collapses dramatically to as low as 21.72% training accuracy, demonstrating that FTRL requires meticulous calibration and extended training to function effectively with LSTM architectures. Given this extreme sensitivity and the fundamental mismatch between FTRL's design for sparse linear problems and the dense, sequential nature of LSTM networks, it should generally be avoided for LSTM-based time series prediction. While future research could explore hybrid approaches, such as using FTRL to optimize specific feature extraction components while adaptive methods handle the recurrent layers. Such architectures fall outside conventional LSTM implementations. Therefore, applying FTRL to deep recurrent networks typically yields suboptimal results despite parameter adjustments.

TABLE X
FTRL TRAINING DATA PERFORMANCE

Metric	LR=0.1	LR=0.01	LR=0.0001
Dir Accuracy (%)	52.53	52.64	52.53
Predictions within 5%	97.05 (1.66%)	21.72 (11.58%)	96.62 (1.70%)
Trend Accuracy	13.47	13.69	13.36

TABLE XI
FTRL TESTING DATA PERFORMANCE

Metric	LR=0.1	LR=0.01	LR=0.0001
Dir Accuracy (%)	49.16	49.72	49.16
Predictions within 5%)	96.20 (1.91)	40.22 (7.93)	96.20 (1.96)
Trend Accuracy	11.05	11.05	11.05

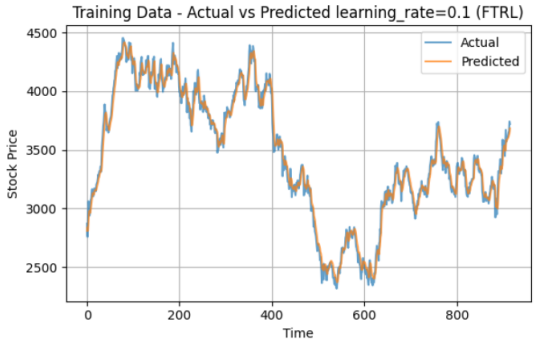

Fig. 6. LSTM FTRL TRAINING WITH LEARNING RATE= 0.1

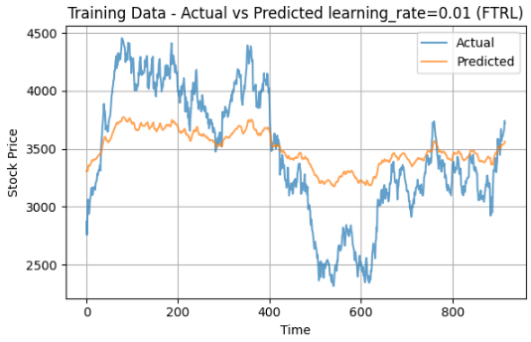

Fig. 7. LSTM FTRL TRAINING WITH LEARNING RATE 0.01

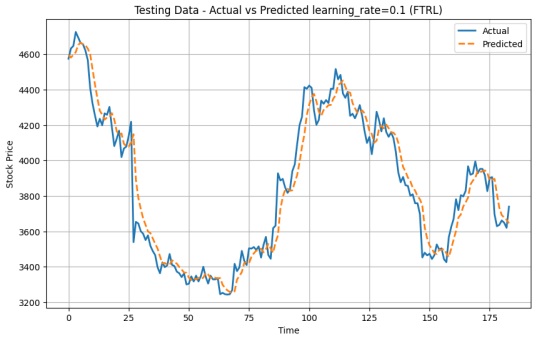

Fig. 8. LSTM FTRL TESTING WITH LEARNING RATE=0.1

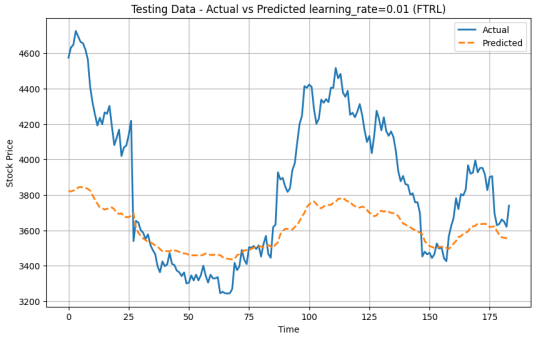

Fig. 9. LSTM FTRL TESTING WITH LEARNING RATE=0.01

IV. GRU SECTION

A. Introduction

The purpose of this experiment is to evaluate the performance of different optimization algorithms—namely Adamax, Stochastic Gradient Descent (SGD), and Follow-The-Regularized-Leader (FTRL)—when applied to a Gated Recurrent Unit (GRU) neural network for stock price prediction. The models are trained to predict the closing price of Apple Inc. (AAPL) stock using historical data spanning from January 1, 2020 to December 31, 2024.

The predictive performance of each optimizer is assessed using both price accuracy and directional accuracy metrics. Price accuracy is measured using Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), while directional accuracy is evaluated using average and best success rates, which represent the percentage of correct upward or downward price movement predictions.

A Gated Recurrent Unit (GRU) is a type of recurrent neural network designed to effectively capture temporal dependencies in sequential data. GRUs enable each recurrent unit to adaptively model dependencies across different time scales through the use of gating mechanisms. Similar to Long Short-Term Memory (LSTM) networks, GRUs employ gates to regulate information flow; however, GRUs do not maintain a separate memory cell, resulting in a simpler architecture and fewer parameters.

The GRU architecture consists of two primary gating mechanisms: the update gate and the reset gate. The update gate controls how much of the previous hidden state is retained versus how much new information is incorporated into the current state. This allows the network to preserve long-term dependencies when necessary. Unlike LSTM networks, GRUs expose their entire hidden state at each time step without an explicit output gate.

The reset gate determines how much past information should be forgotten when computing the candidate hidden state. By reducing the influence of the previous state, the reset gate allows the GRU to focus on recent inputs, effectively enabling the model to reset its memory when processing new or unrelated sequences. This mechanism is particularly useful for modeling non-stationary time series data such as financial markets.

B. Experiment

This study investigates the effect of various optimization algorithms on the performance of a **Gated Recurrent Unit (GRU)** network applied to stock price forecasting. The core objective is to determine how the selected optimizer impacts two key metrics: **accuracy in predicting the price value** and **effectiveness of directional prediction** (up or down).

For this experiment, we used historical stock price data for **Apple Inc. (AAPL)** obtained using the Yahoo Finance API. The dataset spans from **January 1, 2020, to December 31, 2024** and includes daily closing prices. Only the closing price is used in this study to focus on **univariate time series prediction**.

Before training, the data is normalized using **Min–Max scaling** to transform values into the range $[0, 1]$, which improves neural network convergence. The dataset is then divided into **80% training data** and **20% testing data** to evaluate generalization performance.

1) Input Preparation: A **sliding window approach** with a **lookback period of 60 days** is used to create input sequences. Each input sample consists of closing prices from the previous 60 trading days, and the target value is the closing price of the following day.

2) GRU Architecture: The GRU network architecture consists of:

- Two stacked GRU layers, each with 50 hidden units

- A fully connected output layer with one neuron for predicting the next day's closing price

Training is performed for **100 epochs** with a **batch size of 64**.

3) Optimizers Used: In neural network training, the **optimizer** is an essential algorithm. Its primary role is to minimize the loss function by adjusting the model's weights. This process directly influences the convergence speed, stability, and prediction performance of the model.

For this experiment, we focus on three optimizers: **Adaptive Moment Estimation (Adam)**, **Follow The Regularized Leader (FTRL)**, and **Stochastic Gradient Descent (SGD)**.

Adaptive Moment Estimation (Adam)

Adamax is an adaptive optimizer that combines **momentum (first-order moment)** and **adaptive learning rates (second-order moment)**. It computes individual learning rates for each parameter, making it especially effective for deep neural networks like GRUs.

Follow The Regularized Leader (FTRL)

FTRL is designed for **large-scale and sparse data problems**, commonly used in online learning and recommendation systems. Instead of updating parameters step-by-step, FTRL accumulates gradients, applies L1 and L2 regularization, and encourages sparse solutions.

Stochastic Gradient Descent (SGD)

SGD is the most basic and fundamental optimizer. It updates model parameters using the gradient of the loss function computed on each mini-batch. Major characteristics of this algorithm include:

- Using a fixed learning rate
- Simple and computationally efficient
- Slower convergence on complex or noisy data

C. Analysis

In this section, we analyze the results obtained from the experiment. The analysis is based on **Root Mean Square Error (RMSE)**, **Mean Absolute Error (MAE)**, **best success rate**, and **average success rate**.

1) Error Metrics: **Mean Absolute Error (MAE)** measures the average absolute difference between predicted and actual values. It treats all errors equally and is more robust to outliers. **Root Mean Square Error (RMSE)** calculates the square root of the average squared error. It penalizes large errors more heavily and is sensitive to outliers.

TABLE XII
RMSE AND MAE COMPARISON FOR DIFFERENT OPTIMIZERS

Optimizer	Train RMSE	Test RMSE	Train MAE	Test MAE
Adam	2.7196	3.1062	2.0456	2.3228
FTRL	72.3826	139.0835	66.6184	137.0053
SGD	3.7663	5.3985	2.9306	4.4021

2) Price Prediction Accuracy: RMSE and MAE are the evaluation metrics used to assess each optimizer's performance. These metrics quantify the accuracy of predicted stock prices against actual prices.

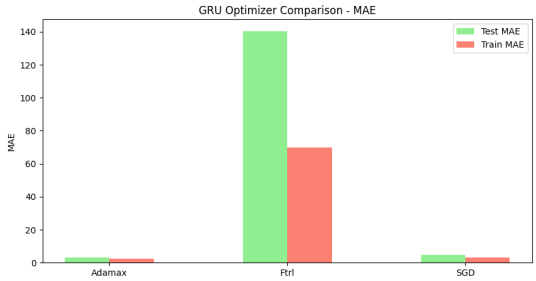

Fig. 10. MAE BAR GRAPH

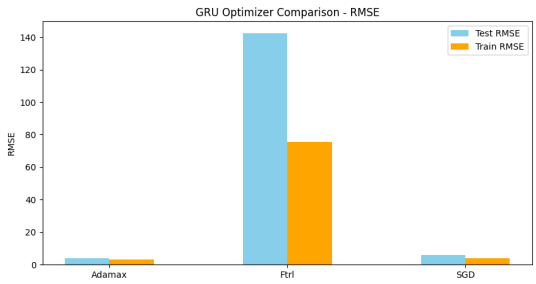

Fig. 11. RMSE BAR GRAPH

Based on the data, **Adamax** had the lowest RMSE and MAE in both training and test data. The small difference between the test RMSE and train MAE indicates that the model generalizes well. Additionally, since the difference between RMSE and MAE is small, although there might be some large errors (since RMSE is slightly greater than MAE), most errors are consistent. This shows that Adam is the best optimizer to use with GRU for stock prediction.

The next best optimizer was **Stochastic Gradient Descent (SGD)**. Similar to Adamax, the small difference between test RMSE and train MAE suggests good generalization. The small difference between RMSE and MAE also indicates that errors are mostly consistent. However, SGD has higher RMSE and MAE values compared to Adamax, making it the second-best optimizer for this experiment.

The worst optimizer was **Follow The Regularized Leader (FTRL)**. Unlike Adamax, FTRL had the highest RMSE and MAE values for both training and test data. The large difference between test RMSE and train MAE suggests that the model did not generalize well. One reason for FTRL's poor performance may be that it works best for **sparse, large-scale data**, while stock prices are a **dense dataset**. Additionally, FTRL cannot inherently adjust per-parameter learning rates in the same adaptive way as Adamax.

TABLE XIII
AVERAGE AND BEST DIRECTIONAL SUCCESS RATES (%)

Optimizer	Train Avg	Train Best	Test Avg	Test Best
Adamax	48.038	78.947	51.579	78.947
FTRL	39.873	73.684	46.842	73.684
SGD	48.887	78.947	54.737	84.211

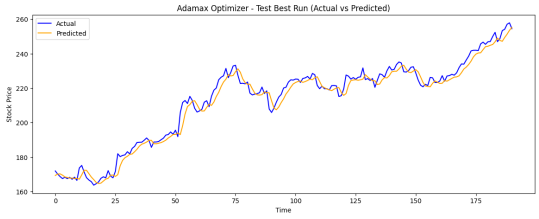

Fig. 12. ADAMAX BEST RUN

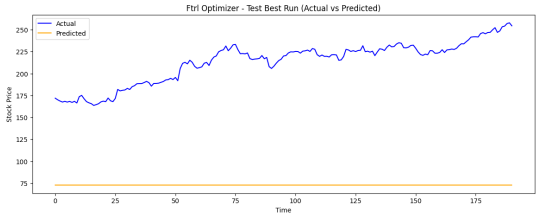

Fig. 13. FTRL BEST RUN

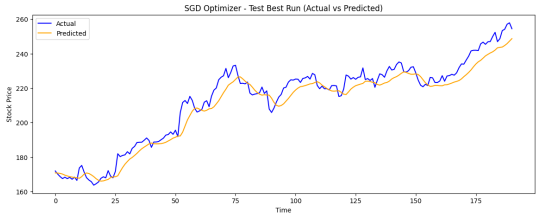

Fig. 14. SGD BEST RUN

3) Directional Accuracy: Directional accuracy measures how often the model correctly predicts the **upward or downward movement** of stock prices. Two metrics are used: **average success rate** and **best success rate**.

From the experiment data, the overall **average directional accuracy** is around 50%, which is slightly better than random chance but still poor. The **best directional success rates** show that SGD occasionally outperforms Adam in capturing short-term trends, achieving up to 84% on the test set. This indicates that while SGD may be less precise in predicting exact values, its simpler update mechanism sometimes allows it to follow local upward or downward trends more effectively than Adam.

Adamax maintains strong directional accuracy, but slightly lower best rates, possibly because its aggressive weight updates smooth out short-term fluctuations. FTRL, consistent with its poor price prediction performance, demonstrates the lowest directional accuracy. This confirms that FTRL's **sparse-update and heavy-regularization approach** is not suitable for dense time series data like stock prices.

D. Conclusion

This experiment investigated the performance of a Gated Recurrent Unit (GRU) network for stock price prediction using different optimizers: Adamax, SGD, and FTRL. The evaluation was based on price prediction accuracy, measured by RMSE and MAE, and directional accuracy, measured by average and best success rates.

From the results, Adamax consistently achieved the lowest RMSE and MAE, indicating it provides the most accurate price predictions. SGD performed moderately well, with slightly higher errors, but occasionally achieved higher best directional accuracy, suggesting it can capture short-term trends effectively. FTRL consistently underperformed in both price prediction and directional accuracy, likely due to its design for sparse, large-scale datasets rather than dense financial time series.

Overall, the findings suggest that Adamax is the best optimizer for GRU-based stock price prediction, balancing accurate price forecasting with stable generalization. While SGD may be useful for short-term directional trend predictions, FTRL is not suitable for dense time series data. These results highlight the importance of selecting the right optimizer when training recurrent neural networks for financial forecasting tasks.

V. Conclusion

A. Overall Performance Summary

Across all architectures, Adamax consistently demonstrated the best overall balance between prediction accuracy, stability, and generalization. SGD proved to be a reliable and often competitive alternative, particularly when enhanced with momentum, while FTRL was consistently the least suitable, except in rare, meticulously tuned scenarios.

B. Adamax

Adamax emerged as the top-performing optimizer for LSTM and the Price Prediction Accuracy part of GRUs. Its adaptive learning rate mechanism, which computes individual rates for each parameter, proved highly effective for non-stationary, volatile stock data. In the RNN experiment, Adamax was competitive but slightly outperformed by SGD in average RMSE. However, Adamax occasionally have better results than SGD, depending on the parameters, indicating strong peak performance. In the LSTM experiment, Adamax delivered the most balanced performance, with the lowest RMSE, stable directional accuracy, and high "within 5% accuracy". Its gradient optimization and normalization helped stabilize LSTM training. In the GRU experiment, Adamax achieved the best results in the Price Prediction Accuracy, as it handles noisy gradients and sparse updates effectively, making it ideal for deep recurrent networks trained on dense, sequential financial data.

C. Stochastic Gradient Descent (SGD)

SGD, especially when augmented with momentum and Nesterov acceleration, performed respectably across all architectures. It was notably the best optimizer for the Simple RNN, and competitive in GRU and LSTM experiments. In the RNN experiment, SGD was the top performer in average RMSE across all five stocks, with lower standard deviation than Adamax, indicating greater consistency. In the LSTM experiment, with proper tuning, SGD showed strong generalization and reduced overfitting, converging to flatter minima.

Its performance was stable but slightly lower than Adamax in accuracy metrics. And in the GRU experiment, SGD had higher RMSE/MAE than Adamax but occasionally achieved higher best directional accuracy, suggesting it can capture short-term trends effectively.

D. Follow The Regularized Leader (FTRL)

FTRL consistently underperformed across RNN, GRU, and LSTM architectures, except in highly specific, finely tuned LSTM scenarios where it achieved exceptional within-5% accuracy. In the RNN experiment, FTRL was rarely among the top configurations and showed poor average RMSE. In LSTM experiments, with optimal hyperparameters, FTRL achieved remarkable accuracy, but performance collapsed dramatically with suboptimal settings, highlighting extreme sensitivity. In the GRU experiment, FTRL had the highest RMSE/MAE and poorest directional accuracy, struggling to generalize.

E. Stock Specific Patterns

Across all the models, RNN and LSTM reacts differently with different stocks.

Overall in RNN, it is obvious that the model performs best when predicting the 600104.SS stock. The model has its lowest RMSE by far, with a value of 0.203. This means this stock is easier to predict in general, compared to the other four. This could be due to various international business factors, such as different market dynamics.

It is also evident that 9843.T has the largest RMSE value of these best configurations, with a value of 73.538. It is very obvious, looking at the RMSE in Table 5, compared to those in Tables 1-4, that the model performs the worst when predicting the Japanese stock 9843.T, compared to the other 4 stocks. It is possible that there are other tuning parameters, such as learning rate and momentum, that can cause the model to perform better.

In terms of these top 10 configurations, the model performs the most consistently with the EOAN.PR stock, as seen in Table 2, where all of the within 5% accuracy values are either 98.8% or 99.3%.

The best configuration for this RNN is using the SGD optimizer function, a 30-day lookback period, 50 neurons in each hidden layer, and 200 epochs when training. This is the best configuration of the RNN when predicting all 5 stocks, as seen in all 5 tables.

Furthermore, Figure 1 shows the best run the RNN had of all configurations, for all stocks. This was when the network was predicting the closing price when given unseen 600104.SS data, and it performed with an RMSE of 0.176.