STOCK PRICE PREDICTION

Comparative Analysis of RNN and LSTM for Stock Price Prediction

Project Technical Report

Dong, Thi Kieu Trang

Martucci, Giuseppe

1 Introduction

In the fast-paced world of financial markets, leveraging advanced machine learning techniques to predict market trends is increasingly essential for maintaining a competitive edge. This project is focused on developing a robust AI-based trading model tailored to navigate the complexities of global financial markets effectively.

Our strategy utilizes two advanced predictive models: Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) networks. RNNs are selected for their proficiency in processing sequential time series data, which is crucial for analyzing the temporal patterns found in stock prices. LSTMs, on the other hand, are favoured due to their ability to retain long-term dependencies and handle sequential data effectively, crucial traits for grasping the temporal dynamics of financial markets.

A significant innovation of our methodology is the formulation and application of a custom loss function. Unlike traditional regression metrics such as Root Mean Square Error (RMSE), which primarily focus on the magnitude of errors, our custom loss function is crafted to prioritize the accuracy of directional predictions, a vital element in trading strategies. This approach aims to address the limitations of conventional metrics by focusing more acutely on aspects critical to effective trading.

This project not only evaluates these models using the standard RMSE criteria but also explores their performance with our novel loss function. Through this comparative analysis, we seek to demonstrate the superiority of our customized approach, enhancing predictive accuracy in a real-world trading context.

2 Methods

2.1 Covariates Construction

In the development of our models, we worked on datasets organized weekly, where variables were crucial for conducting temporal and financial analysis that are:

- **Open, High, Low Prices (OHL)**: These variables represent the opening, highest, lowest, and closing prices within each week.
- **Volume**: This metric denotes the total number of shares or contracts traded during the week for a particular stock or index, reflecting market activity and liquidity.
- **Adjusted Close Price**: Adjusted for corporate actions like dividends, stock splits, and rights offerings, this measure offers a more accurate reflection of the stock's or index's value over time.
- **Date/Time Stamp**: Each record is associated with a specific week, facilitating the temporal analysis of financial metrics.

To augment our model's predictive capabilities, we also incorporated some common technical indicators:

- **SMA (Simple Moving Average)**: A Simple Moving Average (SMA) is an arithmetic moving average calculated by adding recent closing prices and then dividing that by the number of periods in the calculation average.

- **EMA (Exponential Moving Average)**: An Exponential Moving Average (EMA) is a type of moving average that places a greater weight and significance on the most recent data points. It's more responsive to new information compared to the simple moving average.
- **Stochastic Oscillator (STOCH)**: The Stochastic Oscillator is a momentum indicator comparing a particular closing price of a security to a range of its prices over a certain period of time. The sensitivity of the oscillator to market movements is reducible by adjusting that time period or by taking a moving average of the result.
- **RSI (Relative Strength Index)**: The Relative Strength Index (RSI) is a momentum oscillator that measures the speed and change of price movements. RSI oscillates between zero and 100.
- **MACD (Moving Average Convergence Divergence)**: The Moving Average Convergence Divergence (MACD) is a trend-following momentum indicator that shows the relationship between two moving averages of a security's price.

2.2 Target Construction

To effectively model and forecast financial outcomes, we focused on forecasting stock returns and used it as a target variable for our models. Logarithmic returns were used for their statistical properties, such as quasi-normal distribution and constant volatility, which simplify calculations and improve model stationarity (Martucci 2024). The formula used to calculate logarithmic returns is:

$$\text{Log Return} = \log\left(\frac{\text{Price}_t}{\text{Price}_{t-1}}\right) \tag{1}$$

Where P_t represents the current price and P_{t-1} the previous one.

2.3 A new loss function

We realized that the RMSE results were not satisfactory, especially in our case of financial forecasts where the direction of changes is as important as their magnitude. Thus, we decided to create our custom loss function to try to improve the results of our model.

Our custom loss function imposes a heavier penalty for incorrect predictions about the direction of price changes, not just the size of the error. This means that if the model predicts an increase when the price actually decreases, or vice versa, the error is considered more significant. The loss for each prediction takes into account both the size of the error and whether the prediction correctly captured the direction of change.

2.4 Models

2.4.1 RNNs

The essential components of **RNN architecture** include:

- **Input Layer**: Responsible for receiving sequential data, where each element may represent a word or character in tasks like natural language processing.
- **Hidden Layer**: Maintaining an internal state that evolves as the network processes each element in the sequence, capturing information from previous time steps.
- **Recurrent Gate**: A crucial feature involving weights and connections looping back to the hidden layer from the previous time step, allowing the RNN to update its hidden state and remember past information.
- **Output Layer**: Produces predictions based on the information in the hidden state.

Specifically, we trained our model with the Adam optimizer with other hyper-parameters which are listed in Table 1.

Table 1: RNN Architecture for both loss functions

Hyperparameters	RNN with RMSE	RNN Custom loss function
Number of Units per Layer (Hidden Size)	20	20
Number of Layers	1	1
Learning Rate	0.001	0.001
Number of Epochs	100	100
Sequence Length	10	10
Activation Function	tanh	tanh
Batch Size	32	32
Loss Penalty	–	10000

2.4.2 LSTM

The essential components of **LSTM architecture** include:

- **Input Layer**: Takes in sequential data, with each element representing a time step in the input sequence, often used in tasks like time series prediction or natural language processing.
- **Forget Gate**: Decides what information is discarded from the cell state. It looks at the previous hidden state and the current input, and outputs a number between 0 and 1 for each number in the cell state.
- **Input Gate**: Updates the cell state by deciding which values to update and a vector of new candidate values that could be added to the state.
- **Cell State**: The component that carries relevant information throughout the sequence processing, allowing information to be retained or removed at each time step through its gates.
- **Output Gate**: Decides what the next hidden state should be, which contains information that will be used for predictions, based on the current input, the previous hidden state, and the current cell state.
- **Output Layer**: Generates the final output of the LSTM for a given time step which can be a prediction that feeds into another layer or the final prediction of the model.

Table 2: Comparison of LSTM RMSE and LSTM custom loss function

Hyperparameters	LSTM with RMSE	LSTM Custom loss function
Number of Units per Layer (Hidden Size)	50	50
Number of Layers	2	2
Learning Rate	0.001	0.001
Number of Epochs	100	100
Sequence Length	10	10
Activation Function	tanh	tanh
Batch Size	32	32
Loss Penalty	–	10000

We trained our models with Adam optimizer. Learning rate values and other hyper-parameters are listed in Table 2.

3 Experimental Designs

3.1 Experimental Setup

We conducted our experiments using historical stock price data of IBM from 1994-03-06 to 2024-03-03. The dataset was preprocessed to normalize the data and to calculate relevant financial indicators like moving averages and volatility indices. We split the data into training, validation, and testing sets, respectively 70%, 20% and 10%, to ensure robust model evaluation.

3.2 Custom loss function implementation

Our custom loss function was applied to both models to compare its effectiveness against traditional loss functions like RMSE. This function imposes heavier penalties for incorrect directional predictions, aligning more closely with real-world trading objectives where the direction of movement is more critical than the magnitude.

3.3 Evaluation

The outputs of models which are predicted Log_Return then are compared signs with the actual Log_Return. If they are not negative, the classification value is 1; otherwise 0. Results were quantitatively analyzed using accuracy, precision, recall, and F1 scores, and qualitatively discussed to draw conclusions about each model's performance under different experimental conditions. Comparisons were made not only between the models but also between the different loss functions used.

- **Accuracy**: The proportion of total correct predictions (both true positives and true negatives) out of all predictions made.

$$\text{accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2}$$

- **Precision**: The ratio of correct positive predictions (true positives) to the total predicted positives (both true positives and false positives).

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \tag{3}$$

- **Recall**: The ratio of correct positive predictions to the actual positives (both true positives and false negatives).

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \tag{4}$$

- **F1**: The harmonic mean of precision and recall, a measure that balances both metrics.

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{5}$$

4 Results

Our analysis yielded insightful comparisons between the RNN and the LSTM across different datasets and loss functions. LSTM consistently shows a higher precision compared to RNN across both loss functions. RNN tends to perform better in terms of recall when using RMSE, but this trend does not hold with the Custom loss function where LSTM has a higher recall.

Table 3: Performance Evaluation of Models

Loss Function	RNN				LSTM			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
RMSE	0.5118	0.5556	0.4412	0.4918	0.5197	0.5854	0.3529	0.4404
Custom	0.5354	0.5738	0.5147	0.5426	0.5433	0.5893	0.4853	0.5323

The custom loss function generally leads to better performance metrics for both models compared to RMSE, indicating that it might be better suited for the specific characteristics or distribution of the dataset used. Also, accuracy and F1 Score are higher when using the Custom loss function for both models, indicating a more balanced trade-off between precision and recall, which is crucial for many practical applications.

5 Conclusions

In conclusion, the comparative analysis of RNN and LSTM models using RMSE and Custom loss functions highlights distinct performance characteristics under varying conditions. The LSTM model generally exhibits higher precision, while the RNN tends to have superior recall, particularly when utilizing the RMSE loss function. The introduction of a Custom loss function appears to enhance performance across all metrics for both models, suggesting its effectiveness in aligning model outputs more closely with the specific requirements of the dataset. This evaluation underlines the importance of choosing appropriate model architectures and loss functions to optimize performance for specific tasks and datasets. However, it is important to note the limitation of this analysis due to the absence of any optimizer technique for hyperparameter selection. Without such techniques, there might be a risk of not fully capitalizing on the models' potential, as manually selected hyperparameters may not be optimal. Employing a systematic approach to hyperparameter tuning could further enhance model performance and provide more definitive insights into the comparative strengths and weaknesses of each model configuration.

Acknowledgement: This is one part of the project at LUISS.

References

Martucci, Giuseppe (2024). “Benchmarking econometrics and deep learning methodologies for mid-frequency forecasting”. In: *Available at SSRN 4773344*.