**SCHOOL OF
OPERATIONS RESEARCH AND INFORMATION ENGINEERING
COLLEGE OF ENGINEERING
CORNELL FINANCIAL ENGINEERING MANHATTAN
CORNELL UNIVERSITY**

ORIE-5255
Project 2:
Machine Learning Techniques for Stock Price Forecasting

By:

Vishakha Korde (vk369) and Luis Alonso Cendra Villalobos (lc2234)

Professor:
Irene Aldridge

November 2024

1 Introduction

Convolutional Neural Networks (CNNs), traditionally employed in image recognition tasks, have demonstrated significant potential for time-series forecasting, particularly in stock price prediction. Stock prices are sequential and exhibit local temporal dependencies, which CNNs are well-equipped to capture through their convolutional layers. These layers automatically extract relevant features from time-series data, such as peaks, troughs, and trends, without requiring explicit feature engineering. This makes CNNs highly efficient for stock price prediction, especially when working with large datasets spanning many time steps. Additionally, CNNs are more computationally efficient compared to other models like Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks, making them suitable for the large and dynamic nature of financial data.

In our model development, we initially used LSTM networks as a baseline for predicting stock price movements. LSTM networks are a type of RNN designed to learn from sequences of data, and they are particularly useful for time series forecasting. By implementing a sequential model with three LSTM layers and a Dense layer for the final predicted price, this model was optimized using Adam and a mean squared error loss function. The model was trained on features such as Open, High, Low, Close, and Adjusted Close, as well as rolling averages. While this methodology demonstrates a systematic approach to stock price prediction, the moderate accuracy suggests the need for further refinement.

This is where CNNs offer an advantage. The architecture of CNNs, coupled with an encoder-based design, enables the extraction of hierarchical features, allowing the model to understand multi-scale relationships in the stock price data. This is crucial, as stock prices are influenced by factors at different time scales, such as daily trends or weekly patterns. Incorporating an attention mechanism further enhances the model by allowing it to prioritize key events, such as earnings reports or macroeconomic announcements, which can significantly impact stock prices. Dropout regularization is also employed to prevent overfitting, ensuring the model generalizes well even with noisy and non-stationary financial data. Furthermore, the model's ability to perform multi-step forecasting provides valuable insights for short-term trading decisions. In summary, this CNN-based model with an encoder architecture is highly effective for stock price prediction, offering the ability to capture complex patterns, prioritize important features, and deliver reliable forecasts.

2 Models

All data for the models below is sourced from Yahoo! Finance and it comprises data for Apple Inc. (ticker: AAPL) for the period of 1st Jan. 2014 to 1st Jan. 2024.

2.1 Long Short-Term Memory (LSTM)

In our model development, we initially used Long Short-Term Memory (LSTM) networks as a baseline for predicting stock price movements. LSTM networks are a type of Recurrent Neural Network (RNN) designed to learn from sequences of data, and they are particularly useful for time series forecasting.

By implementing a sequential model with three LSTM layers and a Dense layer for the final predicted price, this model is optimized using Adam and a mean squared error loss function. The model is trained on features such as Open, High, Low, Close, and Adjusted Close and rolling averages.

The dataset was partitioned into an 80-20 split for training and test data, and scaled using MinMaxScaler to normalize the values between 0 and 1. A sliding window approach was utilized to generate input sequences of 100 time steps, preparing the data for sequential learning. The training process involved feeding the preprocessed sequences into the classifier. Hyperparameters were optimized. The training loss for the model is 0.0000564.

Once the model is trained, we can use it for forecasting. A 30-day stock price forecast is generated by using the model to predict the next day's price iteratively. Figure 1 shows the forecasted and actual values from the dataset. The In-Sample (IS) Root Mean Squared Error (RMSE) is 1.33 and the Out-Of-Sample (OOS) RMSE is 5.48.

The methodology demonstrates a systematic approach to stock price prediction, however, the moderate accuracy suggests the need for further refinement, CNNs are more computationally efficient compared to RNNs, making them suitable for exploring further in our modeling, as explained below.

2.2 CNN with Hyperparameter Optimization

We explore building a CNN for stock price prediction as an improvement on the last model. The first step in the workflow involves data preprocessing. We normalize the stock prices to a range between 0 and 1 using the MinMaxScaler from scikit-learn. This is crucial in neural networks because it ensures that all

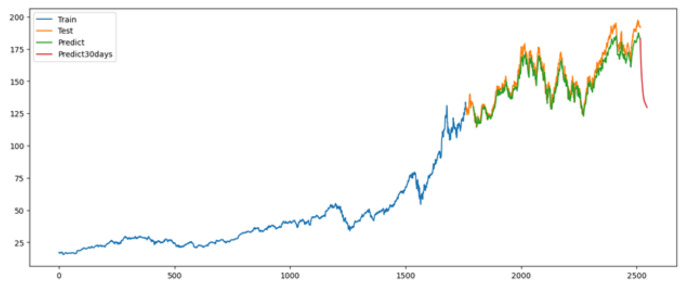

Figure 1: LSTM model performance

input features are on a similar scale, improving model performance.

We proceed to create the time-series dataset by slicing the data into sequences using a rolling window of 30 days. Thus, the model learns from 30 days of past data to produce a single closing price. The dataset is split into training (80%) and testing (20%) sets.

We then proceed to setup the CNN model, see below the main components:

1. Conv1D Layers: The core of the model consists of Conv1D layers. The first Conv1D layer is initialized with tunable hyperparameters such as the number of filters and kernel size. These parameters control the number of feature maps (filters) and the size of the convolution window used to extract features from the time series data. The kernel size is optimized within a range of 2 to 5, allowing the model to capture both short-term and slightly longer-term trends. The number of filters (ranging from 32 to 128) controls the depth of the feature maps, affecting how much information is extracted from the input sequence.

2. MaxPooling Layers: After each Conv1D layer, a MaxPooling1D layer is added to help reduce the dimensionality and reducing overfitting.

3. Dense Layers: The flattened output from the convolutional layers is passed through Dense layers. These layers help the model learn higher-level representations of the data and make predictions. The number of units in these Dense layers is also a tunable hyperparameter, optimized between 50 and 200 units.

4. Output Layer: The final Dense layer has a single unit, as the model aims to predict a single continuous value (the next day's stock price).

5. Loss Function: The model uses mean squared error as the loss function, a standard choice for regression problems, as it minimizes the squared difference between predicted and actual values.

Hyperparameter optimization is implemented using keras tuner library which performs hyperparameter search using the Hyperband algorithm, which is a highly efficient algorithm that performs early stopping and bandwidth exploration to quickly find the best hyperparameters. The hyperparameters optimized include: number of filters for the Conv1D layers, kernel size for the Conv1D layers, number of units in the Dense layer and optimizer (Adam or RMSProp). The search space for each of these hyperparameters is defined within a specific range defined heuristically by the authors.

During training, the validation data is also used to track the model's performance on the test set. Once trained, the model is evaluated on the test data to determine its loss. The loss represents the difference between the predicted stock prices and the actual values, providing a measure of the model's accuracy. The training loss for the model is 0.0013935.

Once trained, the model can be used for forecasting. A 30-day stock price forecast is generated by using the model to predict the next day's price iteratively. Figure 2 shows the performance of the CNN-based model for stock price prediction, with actual stock prices (blue line), predicted stock prices (red line), and a 30-day forecast (green line).

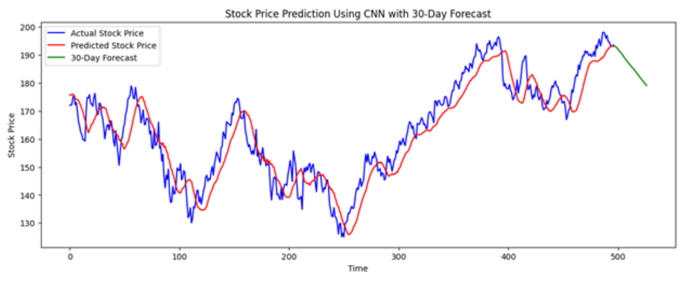

Figure 2: CNN model performance

2.3 CNN + Encoder Model

We further explore modifications by combining our CNN model with an encoder architecture. The approach utilizes a unique combination of CNN layers with attention-like operations, dropout regularization, and a dense output layer.

The model architecture is implemented using TensorFlow and Keras, with the design as follows:

1. Initial Convolutional Layer: The model begins with a single 1D convolutional layer (Conv1D) with a kernel size of 3 and a specified number of filters.

2. Encoder Blocks: The core of the model consists of multiple convolutional layers stacked together. Each encoder block applies a convolutional operation on the previous layer's output to capture patterns.

3. Attention-like Mechanism: An attention-like mechanism is implemented using two dense layers, which are applied to the output of the encoder. The mechanism enhances certain features by weighting them, allowing the model to focus on more significant patterns.

4. Dropout Regularization: To avoid overfitting, a Dropout layer is applied after the encoder blocks. This regularization technique randomly drops a fraction of neurons during training, forcing the model to learn more robust features.

5. Flatten and Output Layer: After the encoder and attention layers, the output is flattened to convert it into a 1D vector. Finally, a dense layer is used to predict the next day's stock price (the output size is 1).

The model is trained using the Adam optimizer with mean squared error as the loss function. The learning rate is specified as a hyperparameter. Training and testing, as well as forecasting, are designed as in previous models.

Figure 3 shows the performance of the CNN-based model for stock price prediction, with actual stock prices (blue line), predicted stock prices (red line), and a 30-day forecast (green line).

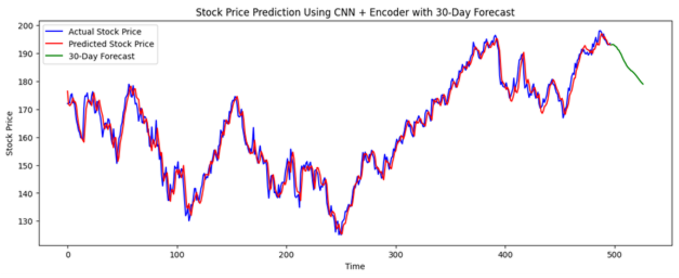

Figure 3: CNN+Encoder model performance

2.4 Comparison between models

We select RMSE as the measure of forecasting performance. In Table 1, we show a comparisson between models.

The LSTM model performs exceptionally well on the in-sample (IS) data, with an RMSE of 1.33, but experiences a significant drop in performance on the out-of-sample (OOS) data, where the RMSE rises to 5.48. This suggests that the model may be overfitting, capturing noise or irrelevant patterns that do not generalize well. Similarly, the CNN+Encoder model, performs well in-sample (RMSE = 3.21), but its performance drops significantly out-of-sample (RMSE = 7.24). Suggesting a similar overfitting problem.

Table 1: Root Mean Squared Errors Across Models

Model	**LSTM**		**CNN**		**CNN+Encoder**	
	IS	**OOS**	**IS**	**OOS**	**IS**	**OOS**
RMSE	1.33	5.48	6.72	6.77	3.21	7.24

The CNN model shows relatively consistent performance across both in-sample (IS) and out-of-sample (OOS) data, with RMSE values of 6.72 and 6.77, respectively. The minimal difference between the IS and OOS RMSE values indicates that the CNN model is not overfitting, making it the most reliable model for forecasting out of the three we considered.

3 Conclusions and Possible Improvements

The CNN + Encoder model demonstrated in this code offers a powerful approach to stock price prediction by combining convolutional layers, an encoder architecture, and attention mechanisms. By leveraging historical stock data, the model effectively learns patterns and trends in the price movement, making it well-suited for short-term forecasting. The use of convolutional layers helps capture local temporal dependencies, while the encoder blocks extract hierarchical features, allowing the model to adapt to complex relationships within the data. The attention mechanism further enhances the model's ability to focus on significant events, improving its predictive accuracy. The inclusion of dropout regularization prevents overfitting, ensuring the model generalizes well to unseen data. Furthermore, the ability to forecast stock prices for multiple future days provides valuable insights for traders and investors, aiding in decision-making processes. Overall, this model represents an advanced method for financial forecasting, showcasing the potential of deep learning techniques to predict stock price movements. While no model can guarantee absolute accuracy in financial markets, this approach, when fine-tuned and applied in conjunction with other factors, can offer a valuable tool for understanding market trends and making informed investment choices.

To improve prediction accuracy, several avenues can be explored. Regularization techniques such as L1/L2 regularization and weight decay could be employed to prevent overfitting. Enhancing the input data through feature engineering, such as including financial indicators like volatility indexes, trading volume, or sentiment analysis from news data, could provide richer context for predictions. Incorporating moving averages and exponential moving averages as features may also boost model performance. Additionally, ensemble methods could combine the strengths of LSTM, CNN, and CNN + Encoder models, potentially improving both in-sample and out-of-sample accuracy. Optimizing hyperparameters with advanced techniques such as Bayesian Optimization or Grid Search can help identify ideal configurations for each model. Exploring hybrid architectures, such as combining LSTM and CNN layers, may capture both temporal dependencies and local patterns more effectively. Attention-based models like the Transformer or Temporal Fusion Transformer (TFT) offer another promising direction for improving generalization. Finally, testing dynamic window sizes for input sequences could provide deeper insights and enhance robustness. By pursuing these strategies, the models' ability to predict stock prices reliably can be significantly improved.