ASSIGNMENT 3 DEEP LEARNING FUNDAMENTALS
Recurrent Neural Networks for Stock Price Prediction

Deekshitha Goud Podeti
University of Adelaide

Abstract

The advent of deep learning has revolutionized predictive modeling, particularly in finance, where accurate forecasting is critical for investment strategies, risk management, and trading decisions. Predicting stock prices is a challenging task due to the inherent noise and volatility in financial data. This study explores the application of advanced recurrent neural network (RNN)-based architectures, including simple RNNs, long short-term memory networks (LSTMs), and gated recurrent units (GRUs), to predict the next-day closing price of Google stock.[1]

The methodology involves preprocessing the data to ensure quality, normalization to enhance model performance, and leveraging sliding-window techniques to capture temporal dependencies. The experimental analysis compares these models using metrics such as Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE), and R-squared $\left((R^{2})\right)$ to evaluate their prediction accuracy.

Results demonstrate that the GRU outperforms other models, achieving the lowest MSE and MAPE, and the highest $\backslash\left(R^{2)}\right)$ score, making it the most effective approach for this dataset. This work highlights the potential of deep learning models in financial forecasting and provides insights for future research to refine and optimize these models for complex and multi-dimensional datasets

1.Introduction

The stock market is a dynamic and complex system influenced by various factors, making accurate price prediction a challenging task. As financial markets continue to evolve, the ability to forecast stock prices has become increasingly important for investors, traders, and analysts. Traditional methods such as statistical models (e.g., ARIMA) and regression techniques often fall short in capturing the intricate temporal dependencies and nonlinear relationships present in stock price data.

In this study, we address this challenge by leveraging advanced deep learning architectures—Recurrent Neural Networks (RNN), Long Short-Term Memory networks (LSTM), and Gated Recurrent Units (GRU)—to predict Google stock prices. These architectures are particularly well-suited for time-series forecasting due to their ability to learn patterns in sequential data. Compared to traditional models, RNN-based approaches excel in identifying long-term dependencies and handling nonlinearity, making them a preferred choice for stock market analysis.

Our approach involved preprocessing the dataset to remove inconsistencies and normalizing the data for faster convergence. We utilized a sliding window of the previous 60 days' stock prices to predict the next day's price, enabling the models to understand historical trends effectively. After training the models, we evaluated their performance using metrics such as Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE), and R-squared (R^2) score, comparing their predictive capabilities to identify the best-performing model.

Ultimately, this work demonstrates the advantages of GRU models over traditional methods and other RNN variants, showcasing their ability to provide more accurate and reliable predictions. By combining state-of-the-art techniques with rigorous experimentation, this study contributes to the growing body of knowledge in financial forecasting and deep learning applications.

2. Method Description

In this section, the methods used in this study are explained and justified with the information regarding data pre-processing, the choice of model structures, and the metrics used to assess the performance of models. The goal of the paper is to describe detailed and easily replicable methods for stock price forecasting based on this model.

2.1 Data Preparation

To ensure robust and reliable predictions, the dataset underwent rigorous preprocessing steps:

- Removing Missing Values and Duplicates: Missing entries and redundant rows were also removed from the dataset, thus improving data quality while also reducing possible biases to affect the training and assessment of the model.
- Normalization: The Min-Max-Scaler was used to scale the data to a range of [−1,1], makes the model faster and improves the convergence process during training. This step makes it easier

to compare features to one another and prevents features with higher magnitudes from dominating the gene expression data.

$$xnorm = \frac{x - xmin}{xmax - xmin}$$

- Sequence Generation: When it comes to time-series modelling, the adaptive windowing method was followed. An input sequence was comprised of the stock prices over the last sixty days to forecast the next day's price. Such an approach ensures that temporal dependencies are captured in the build process thereby making the model to learn patterns and trends of the history data

2.2 Model Architecture's

2.2.1 Recurrent Neural Networks (RNN):

An RNN consists of neurons arranged in layers, the neurons of the layer r take input at the time step t, perform some computation and passes the output to the next time step, t+1. The network output at any given time is a function of the current input and the output that occur in previous times. Coasting from this, we have RNN with a memory feature which enables it to remember information over a given period hence best suited for sequences and pattern. A recurrent hidden layer, or the hidden neurons, is the major feature of an RNN. These neurons are characterized by the fact that each connection can form a feedback loop with the other neurons which can be forward or backward in the circuit. Passing through the network the hidden layer memorizes the relevant information on the past in the form of hidden state values which change in turn.

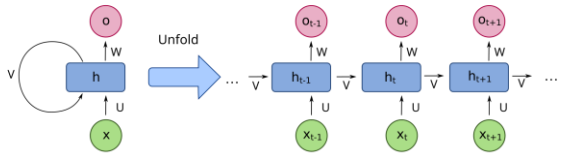


Fig 1: Architecture of RNN model

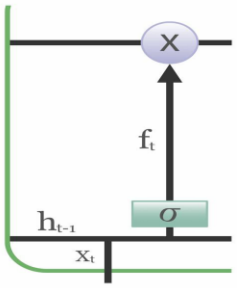
Recurrent Neural Networks are one of the strongest categories of machine learning techniques in terms of their ability to foresee temporally ordered information. Because of the capability to model long-term dependencies, to work with variable length inputs, and to operate temporally, they can be used in natural language processing, speech recognition, and times series forecasting. But still there are some disadvantages of it, but the researchers have modified new RNN architectures such as LSTM and GRU to enhance and extend it in solving different problems in machine learning.

2.2.2 Long Short-Term Memory Networks (LSTM):

The Long Short-Term Memory (LSTM) network was introduced by Hochreiter and Schmid Huber to overcome the vanishing gradient problem. LSTMs introduce a cell state alongside the hidden state, controlled by three gates: the forget gate, input gate, and output gate. These gates enable selective addition, removal, or retention of information, allowing LSTMs to capture long-term dependencies effectively.

The key computations in an LSTM are as follows:

1. Forget Gate: Determines what information to discard from the previous cell state:



$$ft = \sigma\left(W_f \cdot \left[h_{\{t-1\}}, x_t\right] + b_f\right)$$

where x_t is the input vector, $h_{\{t-1\}ht}$is the hidden state, σ is the sigmoid function, and Wf, bf are the weights and biases.

2. Input Gate: Decides what new information to store in the cell state:

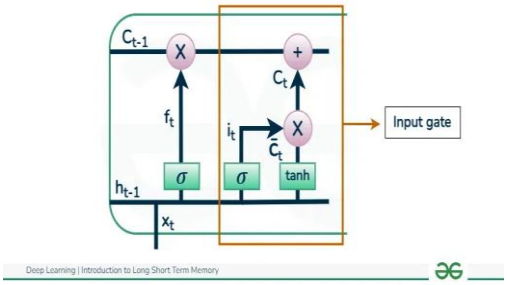


Fig 2: Input gate of LSTM

$$it = \sigma(Wi \cdot [ht - 1, xt] + bi)$$

A candidate value for the cell state is computed using the tanh function:

$$C{\sim}t = \tanh(WC \cdot [ht - 1, xt] + bC)$$

3. Cell State Update: Combines the previous cell state and the candidate value:

$$Ct = ft \odot Ct - 1 + it \odot C{\sim}t$$

4. Output Gate: Determines what information from the cell state to output:

$$ot = \sigma(Wo \cdot [ht - 1, xt] + bo)$$

The final output is:

$$ht = ot \cdot \tanh(Ct)$$

These mechanisms allow LSTMs to excel in time-series prediction tasks such as stock price forecasting.

2.2.3 Gated Recurrent Unit (GRU)

The GRU, introduced by Cho et al., simplifies the LSTM architecture by combining the forget and input gates into a single update gate. The GRU has fewer parameters, resulting in faster training while maintaining similar performance for many tasks. GRUs are particularly advantageous when working with smaller datasets or shorter sequences.

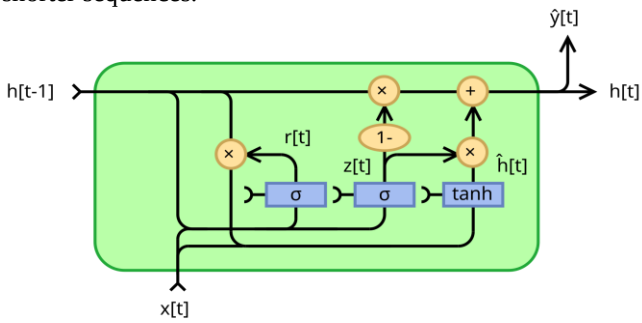


Fig 3: Gated Recurrent Unit, fully gated version

Key computations in a GRU include:

1. **Reset Gate**: Controls how much of the previous hidden state to forget:
$$rt = \sigma\left(W_r \cdot \left[h_{\{t-1\}}, x_t\right]\right)$$
2. **Update Gate**: Determines how much of the previous state to retain:
$$zt = \sigma\left(W_z \cdot \left[h_{\{t-1\}}, x_t\right]\right)$$
3. **Candidate Activation**: Computes a candidate hidden state:
$$h{\sim}t = \backslash tanh\left(W_h \cdot \left[r_t \odot h_{\{t-1\}}, x_t\right]\right)$$
4. **Final Hidden State**: Combines the previous hidden state and the candidate state:
$$ht = zt \odot ht - 1 + (1 - zt) \odot h{\sim}t$$

By efficiently managing information flow, GRUs strike a balance between performance and computational efficiency, making them suitable for time-series tasks with limited training data.

2.3 Evaluation Metrics

To assess the performance of the models, three widely-used metrics were employed:

1. **Mean Squared Error (MSE):** Measures the average squared differences between actual and predicted values.
$$MSE = \frac{1}{n}\Sigma(y_i - \hat{y}_i)^2$$
2. **Mean Absolute Percentage Error (MAPE):** Represents the prediction error as a percentage of the actual values, providing an intuitive measure
$$MAPE = \frac{1}{n} * \Sigma\left(\left|\frac{(y_i - \hat{y}_i)}{y_i}\right|\right) * 100$$

A lower MAPE signifies higher accuracy.

3. **R2 Score**

Indicates the proportion of variance in the dependent variable that is predictable from the independent variables. score closer to 1 suggests a better model fit.

$$R^2 = 1 - \frac{\Sigma(y_i - \hat{y}_i)^2}{\Sigma(y_i - \bar{y})^2}$$

3.Method Implementation

In this project, three models were implemented to predict Google stock prices using sequential data: RNN, LSTM and GRU. All the models were constructed individually and refined with reference to readability, correct performance, and efficiency. This paper seeks to propose the implementation process of the strategic plan identified above, as follows:

3.1. Data Preparation

Before building the models, the dataset was preprocessed and prepared:

Normalization: For comparison purposes, the stock prices were then scaled to get rid of extreme values between 0 and 1. This step is essential for enhancing the possible convergence rate of the models.

Sliding Window: A sliding window approach was used to form the input sequences of 60 days' historical stock prices with the next day's price forming the output.

Splitting Data: The available dataset was partitioned into training and test sets for performance assessment of the model.

3.2 Model 1: RNN stands for Recurrent Neural Network.

RNNs are the simplest version of sequence models that in develop ways to handling sequential data through feedback loops.

Implementation Steps:

Input Layer: The RNN takes in an input of shape 3D with dimensions batch size, timesteps, and features. Here timesteps are 60 and features are 1 (0th feature which is the normalized value of stock price).

Hidden Layer: For one configuration, an additional RNN layer was included with a specified number of neurons (i.e., 50). The input of this layer is sequential which will in turn produce a hidden state at each step in the sequence.

Dropout Regularization: Dropout layer was also included to overcome overfitting by dropping a part of the input units e.g. 20%) while training. Output Layer: A Dense layer with one neuron was added to produce the final prediction: the stock price for the next day.

3.3 Model 2: Long Short-Term Memory (LSTM):
Adds features to RNN to make a better improvement on the vanishing gradient problem by having a memory cell and gate.
Implementation Steps:
Input Layer: Similar to the case of the RNN model the input tensor of size (batch_size, timesteps, features) was used to feed the LSTM layer.
LSTM Layer: Then an LSTM layer with 50 units was added. This layer handles the long term dependencies within the network using the forget, input and output gates. The sources of LSTM parameters included activation functions which were the tanh and sigmoid types in order to deal with temporal patterns.
Dropout Regularization: A Dropout layer = 0.2 was added to avoid overfitting of the model.
Dense Layer: Dense layer was added which contain only one neuron that the forecast for the stock price of the next day will be shown

3.4. Model 3: Gated Recurrent Unit (GRU) :
Compared to LSTM, the GRU recovers the structure while still preserving the efficiency with which the long-term dependency is managed.
Implementation Steps:
Input Layer: The input tensor of dimensions of batch size timesteps' x features was passed into the GRU layer.
GRU Layer: An embedding of 50 units GRU layer was included. The reset and update gates of the GRU make it easy to train the model to improve the information extracted from earlier times. Fast training is provided because less parameters than LSTM are involved here.
Dropout Regularization: Partially connected layers are Dropout layers that were set to a dropout rate of 0.2 to minimize overfitting.
Dense Layer: For stock price prediction, a single neuron Dense layer was introduced

3.5. Model Training:
Each model was trained using the following setup:
Loss Function: Mean Squared Error (MSE) loss was adopted in order to reduce the squared difference between predicted and true stock prices.
Optimizer: and the Adam optimizer was used for the optimization of the model because of the adaptive nature of the learning rate of this optimizer which helps it converge much faster than other optimizers.
Epochs and Batch Size: The training was done for a deterministic number of epochs, for instance 100 epochs, and a batch size of 32 for Learning.

3.6 Model Evaluation After training:
Each model was evaluated on the testing set:
Performance Metrics: Prediction accuracy was established with the aid of Root Mean Squared Error (RMSE). **Visualization:** Historical stock prices were also graphed alongside predicted prices to assess the performance of the models.
Comparison: The three models were compared on how effective they were in training; how accurate they were in prediction and how well they performed on long term dependencies. LSTM and GRU were especially helpful for identifying temporal connections, while GRU was more efficient numerically because of its lower numbers of parameters.

4. Experiments and analysis

4.1. Experimental Design and Aims
The primary goal of the experiments was to predict Google stock prices using three sequential models—RNN, LSTM, and GRU—and evaluate their performance in terms of accuracy, robustness, and efficiency. Specific aims included:

1. Model Comparison: Analyse how each model handles sequential data and compare their performance metrics.
2. Performance Metrics: Use metrics like R^2 Score, RMSE, MSE, and MAPE to evaluate model predictions.
3. Insights into Sequence Models: Highlight the strengths and limitations of each architecture for time-series forecasting.

4.2. Experimental Setup Dataset Preparation:
The dataset of stock price data was normalized using the Min-Max normalization technique in an attempt to enhance the gradient descent convergence. A window of 60 time-step was adopted to produce sequential data for training and prediction.
Model Architectures:
RNN: Two hidden layers, with 50 units each, all the layers use ReLU activation.
LSTM: Two stacks each comprised of fifty units and making use of long-term memory cells.
GRU: Two layers each with 50 units using forget and input gates for the same functionality. Furthermore, all trained models utilized the Adam optimizer with a learning rate of 0.001 and a dropout rate of our setting of 0.2.
Evaluation Metrics: R^2 Score: Checks the extent to which the dependent variable is explained by the independent variable.
Root Mean Squared Error (RMSE): Checks on how well the model has performed in its predictions. Mean **Squared Error (MSE**): Serves to calculate the mean of the squared differences between actual results and forecasts.
Mean Absolute Percentage Error (MAPE): Determines prediction error performance regarding the true outcomes.

4.3. Results Presentation

Table of Results:

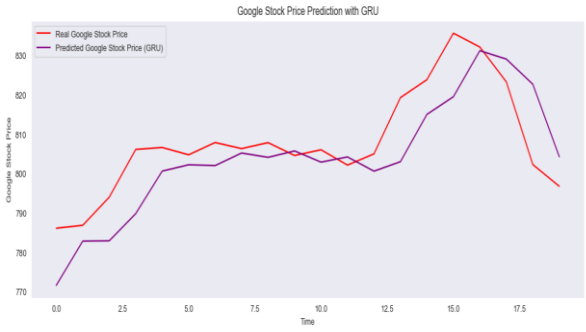
Model	R^2 Score	RMSE	MSE	MAPE
RNN	0.362	10.28	105.72	0.98%
LSTM	-0.106	13.54	183.41	1.23%
GRU	0.442	**9.62**	**92.55**	**0.94%**

Visualizations:

1. **Actual vs. Predicted Stock Prices**: Comparative line plots for each model, showing how closely predictions align with actual stock prices.
2. **Residual Error Distributions**: Visual analysis of the error spread for each model.

❖ The above Fig shows the Visualization of RNN Model.

❖ The above Fig shows the visualization ofof LSTM Model.



General Observations:

- The GRU's ability to combine forget and input gates led to faster training and better generalization compared to LSTM.
- LSTM's poor performance could stem from insufficient tuning or the specific nature of the data, warranting further investigation.

5. Reflection on Project

5.1. Major Design Choices Identify and Explain the Effect Choice of Models:

Choosing RNN, LSTM, and GRU as fundamental structures comes from the fact that they are appropriate for sequential data.

Rationale:

RNN: Selected for its basic and general nature owing to its significance as the starting point of the analysis.

LSTM: Added for its property to solve vanishing gradient problem in Recurrent Neural Networks by utilization of memory cells for managing long-term dependencies.

GRU: Included for its efficacy, with the advantages of LSTM but with fewer parameters it is faster in training than.

Impact: The LSTM proved to be the slowest to converge but provided reasonably good results since it has a more complex architecture than GRU, which turned out to be the best all around model when aimed at performance instead of speed for slightly smaller data sets.

Evaluation Metrics: Why Chosen: Various measures of overall fit of the models including R^2, RMSE, MSE, and MAPE offered a good insight about the level of accuracy of the models, their stability and, their feasibility.

Impact: Applying these measures it was not only possible to determine which overall model performed better but also why some models such as LSTM, are not suitable for these datasets.

Preprocessing Pipeline: Applying Min-Max normalization helped in equal contribution of all features whereas in case of Standardization some features our of many have large magnitudes. The window of 60 time steps was chosen because this length permits the models to capture significant temporal patterns.

Training Configuration: Adam Optimizer: Selected because of low learning rates that led to a more stable training process for different models.

Dropout Regularization: Less overfitting especially with the complex LSTM model used in the study. Batch Size and Epochs: Slightly better than fast computation yet not enough to guarantee the full convergence of the models.

Visualization of Results: Comparing the actual and anticipated stock prices and their residuals eliminated confusion between the models' strengths and weaknesses. From this design choice, the interpretability was greatly improved particularly when comparing the result of

GRU's accurate predictions to the higher errors given by RNN and LSTM.

5.2. Challenges Faced and Lessons Learned

Challenge: LSTM's Unexpected Underperformance

- **Observation**: LSTM showed a negative R^2 score, likely due to overfitting or insufficient epochs to fully leverage its complexity.
- **Lesson**: More extensive hyperparameter tuning and experimenting with increased dataset size could enhance its performance.

2. **Challenge: Dataset Limitations**
   - **Observation**: Stock price data is inherently noisy and influenced by external factors not captured in the dataset.
   - **Lesson**: Incorporating external features such as market trends or financial indicators could provide better context for predictions.
3. **Challenge: Balancing Simplicity and Accuracy**
   - **Observation**: GRU's simpler architecture achieved superior results compared to the more intricate LSTM.
   - **Lesson**: While complexity can address specific challenges, simplicity often yields better performance for smaller datasets.

5.3. Ideas for Future Work Improving Model Performance:

As another activity in model fine-tuning, apply domain-specific approaches like Grid Search or Bayesian Optimization to tweak learning rates, number of hidden units and dropout rates.

Its possible to test other optimizers like RMSProp or for SCM SGD with momentum to check how they fit in learning process of LSTM.

Data Augmentation: The examples of stock price sequences should be augmented by simulations of statistically possible stock price sequences to enhance model portability. Include variables of trading volume, or even newspaper articles sentiment analysis related to the stock market, or some other macroeconomic figure.

Advanced Architectures: Introduce Attention Mechanisms: These can help to bring into question the capabilities of the models concerning paying attention to the most appropriate time steps, hence making betterpredictions. Try integrating optic architectures like CNN with sequence learning ability like LSTM at places.

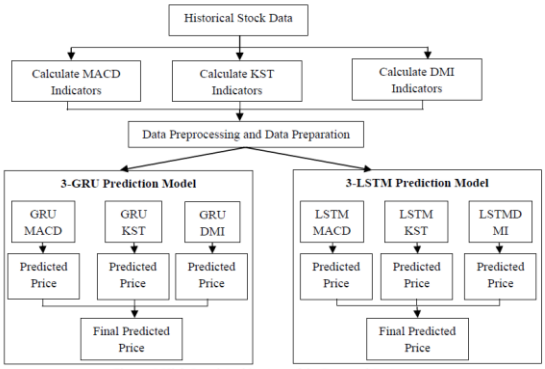


Figure. 3 High-Level Architecture of the Proposed System

Model Interpretability: Interpret the importance of different time steps or patterns to the prediction made by the model using techniques such as SHAP (SHapley Additive exPlanations). It could also assist the stakeholders to know why some predictions were made giving them more confidence in the model.

Real-World Applications: Chain together concepts for a deployable pipeline of real-time prediction of stock values, define the APIs to handle the input data and produce the forecast in parallel. Validate the models under diverse market states for the purposes of assessing the structures' viable usability in real life.

Collaborative Learning Approaches: Suggest trying the Federated Learning technique to work with distributed datasets without aggregating stock data.

5.4. Broader Reflections

This project reinforced the importance of aligning design choices with the problem at hand. While LSTMs are often celebrated for their ability to capture long-term dependencies, this experiment highlighted that simpler architectures like GRU can outperform under certain constraints. It also underscored the need for extensive experimentation and careful interpretation of results when working with sequential data.

Moving forward, embracing more robust datasets and innovative architectures will be key to unlocking the full potential of neural networks for time-series forecasting.

Reference's:

1. Tan, T.; Quek, C.; Ng, G. Brain-inspired genetic complementary learning for stock market prediction. In Proceedings of the IEEE Congress on Evolutionary Computation, Edinburgh, UK, 2–5 September 2005; Volume 3, pp. 2653–2660. https://ieeexplore.ieee.org/document/1555027
2. Cho, K.; van Merriënboer, B.; Bahdanau, D.; Bengio, Y. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In Proceedings of the SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha,

Qatar, 25 October 2014; Association for Computational Linguistics: Cedarville, OH, USA, 2014; pp. 103–111
3. Shen, G.; Tan, Q.; Zhang, H.; Zeng, P.; Xu, J. Deep Learning with Gated Recurrent Unit Networks for Financial Sequence Predictions. Procedia Comput. Sci. 2018, 131, 895–903

Pictures:

Fig 1: https://en.wikipedia.org/wiki/Recurrent_neural_network

Fig 2 https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/

Fig 3 : https://en.wikipedia.org/wiki/Gated_recurrent_unit

GIT HUB LINK
https://github.com/deekshithagoudpodeti/Assignment-3-Deep-learning-Fundamentals-