COS30018

Intelligent Systems

Option B: Stock Prediction

Task B.4 – Machine Learning 1

Name: Duc Thuan Tran
Student ID: 104330455
Tutor: Dr. Ru Jia
Tutorial: Friday 2:30 – 4:30

Table of Contents

Introduction

This report presents the development and enhancement of a stock prediction model using deep learning techniques, now updated to version v0.3 of the codebase. The primary goal of this task is to improve the flexibility and efficiency of the deep learning model, enabling it to predict stock prices based on historical data. The project makes use of advanced Recurrent Neural Networks (RNNs), specifically LSTM (Long Short-Term Memory), GRU (Gated Recurrent Units), and their bidirectional variants, to capture both short-term and long-term dependencies in time-series data.

In this version (v0.3), significant improvements have been implemented, including the introduction of flexibility in the model creation process, allowing the use of different RNN architectures. Additionally, error handling in the predict_next_day function has been refined, and the model's structure has been enhanced with the option to include bidirectional layers for better sequential data analysis.

This report details the changes made to the model_operations.py file, the exploration of various RNN architectures, and the impact of hyperparameter tuning on model performance. The execution of the model, along with its outputs and the source code, is documented and made available in the GitHub repository for further review and testing.

Overview of Previous Model Operations

In version v0.1 of the stock prediction model, the deep learning architecture was relatively straightforward, consisting of a sequential stack of LSTM layers. The model was designed to capture temporal dependencies in stock prices using the following structure:

1. **LSTM Layers:**
    - Three LSTM layers, each with 50 units, were used to model the sequence of stock prices.
    - The return_sequences=True parameter was applied to the first two LSTM layers, ensuring that the full sequence output was passed to the next LSTM layer.
    - The final LSTM layer returned only the last hidden state as input to the subsequent Dense layer.
2. **Dropout Layers:**
    - A Dropout layer with a rate of 0.2 was added after each LSTM layer to prevent overfitting and enhance the model's generalization ability.
3. **Dense Output Layer:**
    - The model concluded with a Dense layer with one unit to predict the final stock price for the next day.
4. **Compilation:**
    - The model was compiled using the Adam optimizer with a mean_squared_error loss function, suitable for regression tasks like stock price prediction.

Overall, this version of the model was effective but limited in its flexibility and adaptability to different deep learning architectures. The fixed use of LSTM layers and static parameters did not allow for experimentation with other RNN types, bidirectionality, or more advanced configurations, which were introduced in subsequent versions.

```python
def build_model(input_shape):
    model = Sequential()

    model.add(LSTM(units=50, return_sequences=True, input_shape=input_shape))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50, return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))

    model.compile(optimizer='adam', loss='mean_squared_error')

    return model
```

Image 1. build_model function version v01

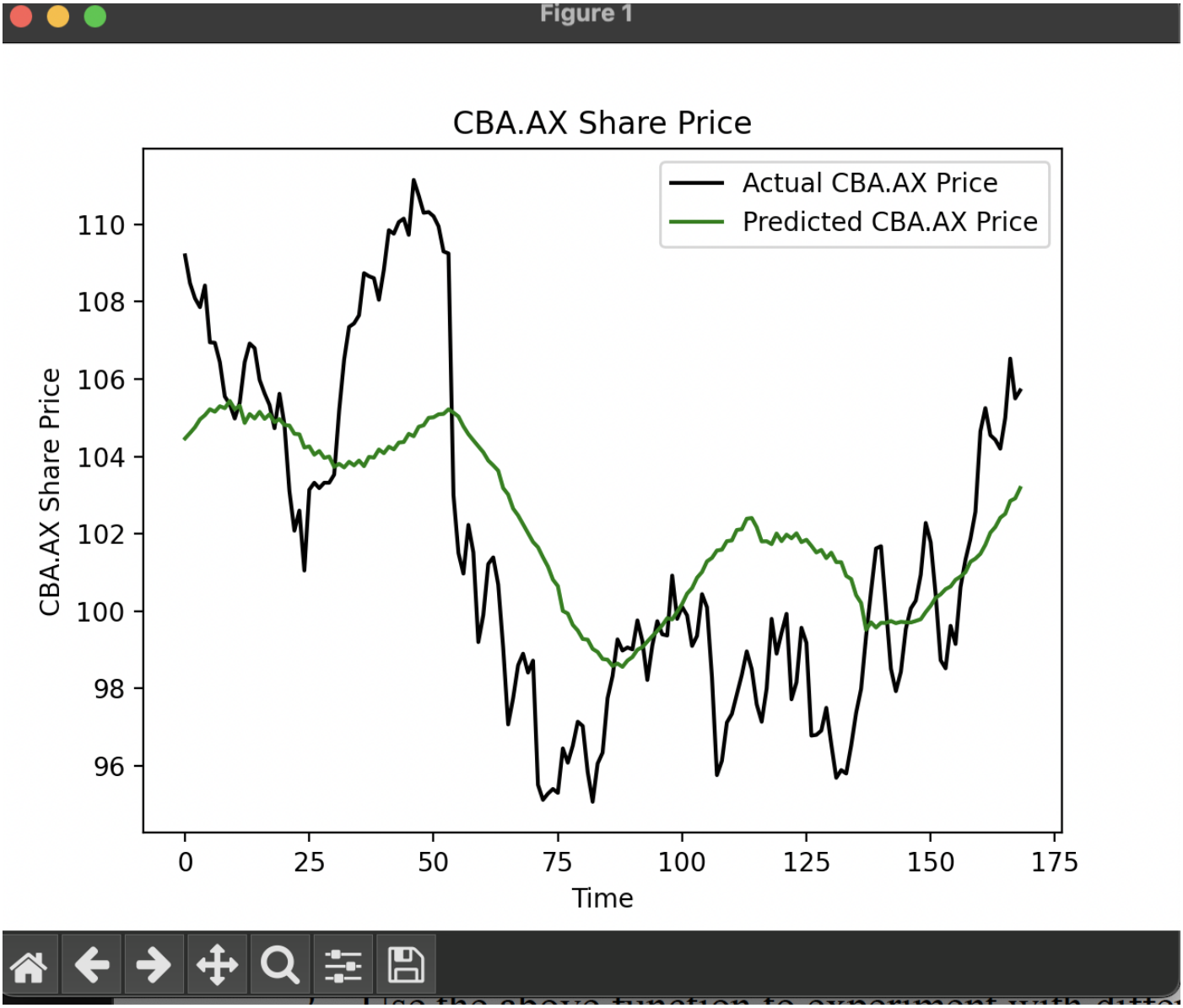

Image 2. Stock-prediction result version v01, v02

Enhancements in the Current Task

In version v0.3, the stock prediction model has undergone significant enhancements to increase its flexibility and adaptability. These improvements include support for different Recurrent Neural Network (RNN) types, such as LSTM, GRU, SimpleRNN, and their bidirectional variants. The enhanced model allows for better experimentation with various architectures and hyperparameters, improving its ability to capture the complexities in stock price prediction. Below are the specific changes introduced in this version:

1. Layer Type Flexibility:

- **Previous Version (v0.1):** The model was limited to using only LSTM layers, which, although powerful, did not allow for easy experimentation with other RNN types.

- **Current Version (v0.3):** The build_model function now includes a layer_type parameter, which allows the user to select between multiple RNN layer types:
    - **LSTM (Long Short-Term Memory):** Suitable for capturing long-term dependencies in the sequence.
    - **GRU (Gated Recurrent Unit):** A computationally efficient alternative to LSTM that still handles long-term dependencies well.
    - **SimpleRNN:** A basic RNN layer for simpler tasks.
    - **BiLSTM (Bidirectional LSTM):** Captures dependencies in both directions, enhancing performance for tasks where future context is important.
    - **BiGRU (Bidirectional GRU):** Similar to BiLSTM but with fewer parameters, making it faster to train.

- The inclusion of these options greatly enhances the model's versatility, allowing it to adapt to different types of sequential data and prediction tasks.
-

2. Conditional Layer Handling:

- **First Layer:** The build_model function checks if the num_layers parameter is greater than 1 to decide whether the first recurrent layer should return sequences. This ensures that the model can handle both single-layer and multi-layer architectures.
- **Subsequent Layers:** Additional layers are added based on the value of num_layers, allowing for dynamic depth in the model. Each of these layers inherits the specified layer_type and applies the return_sequences parameter conditionally.

```
def build_model(input_shape, num_layers=3, layer_type='LSTM', layer_size=50, dropout_rate=0.2):
    model = Sequential()
```

Image 3. Build model function with configurable compilation

3. Introduction of Bidirectional Layers:

- In addition to standard LSTM, GRU, and SimpleRNN layers, the enhanced model also supports **Bidirectional LSTM (BiLSTM)** and **Bidirectional GRU (BiGRU)** layers.
- **Bidirectional Layers:** Process the sequence both forward and backward, improving the model's ability to capture dependencies from both past and future time steps. This is particularly useful in tasks like text processing or time series prediction, where context from both directions can improve accuracy.

```
#first RNN layer
if layer_type == 'LSTM':
    model.add(LSTM(units=layer_size, return_sequences=(num_layers > 1), input_shape=input_shape))
elif layer_type == 'GRU':
    model.add(GRU(units=layer_size, return_sequences=(num_layers > 1), input_shape=input_shape))
elif layer_type == 'RNN':
    model.add(SimpleRNN(units=layer_size, return_sequences=(num_layers > 1), input_shape=input_shape))
elif layer_type == 'BiLSTM':
    model.add(Bidirectional(LSTM(units=layer_size, return_sequences=(num_layers > 1)), input_shape=input_shape))
elif layer_type == 'BiGRU':
    model.add(Bidirectional(GRU(units=layer_size, return_sequences=(num_layers > 1)), input_shape=input_shape))
else:
    raise ValueError(f"Unsupported layer_type: {layer_type}")

model.add(Dropout(dropout_rate))
```

Image 4. First Rnn layer

4. Dropout for Regularization:

- Dropout layers are included after each RNN layer to prevent overfitting. The dropout rate is controlled by the dropout_rate parameter, which remains at 0.2 by default. This ensures that the model remains generalizable even when dealing with complex datasets.

```
#remaining RNN layers
for _ in range(1, num_layers):
    if layer_type == 'LSTM':
        model.add(LSTM(units=layer_size, return_sequences=(_ < num_layers - 1)))
    elif layer_type == 'GRU':
        model.add(GRU(units=layer_size, return_sequences=(_ < num_layers - 1)))
    elif layer_type == 'RNN':
        model.add(SimpleRNN(units=layer_size, return_sequences=(_ < num_layers - 1)))
    elif layer_type == 'BiLSTM':
        model.add(Bidirectional(LSTM(units=layer_size, return_sequences=(_ < num_layers - 1))))
    elif layer_type == 'BiGRU':
        model.add(Bidirectional(GRU(units=layer_size, return_sequences=(_ < num_layers - 1))))

    model.add(Dropout(dropout_rate))
```

Image 5. Remaining Rnn layers

5. Output Layer and Model Compilation:

- The output layer remains a single-unit Dense layer, as in previous versions, suitable for the regression task of predicting stock prices.
- The model is compiled using the **Adam optimizer** and the **mean squared error (MSE)** loss function, which are standard choices for regression tasks and ensure stable training.

```
    #Output layer
    model.add(Dense(units=1))


    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')


    return model
```

Image 6. Output layer and Model Compilation

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, GRU, SimpleRNN, Bidirectional
import numpy as np
import pandas as pd
```

Image 7. New Neural models adding

Summary of Changes:

- **Layer Type:** Added support for LSTM, GRU, SimpleRNN, BiLSTM, and BiGRU.
- **Dynamic Depth:** The number of recurrent layers (num_layers) is customizable, allowing the model depth to scale based on task complexity.
- **Bidirectional Layers:** Introduced bidirectional variants (BiLSTM, BiGRU) for improved context understanding.
- **Improved Regularization:** Dropout layers help prevent overfitting, making the model more robust.
- **Output and Loss:** The model retains its single-unit Dense output for regression, with MSE as the loss function.

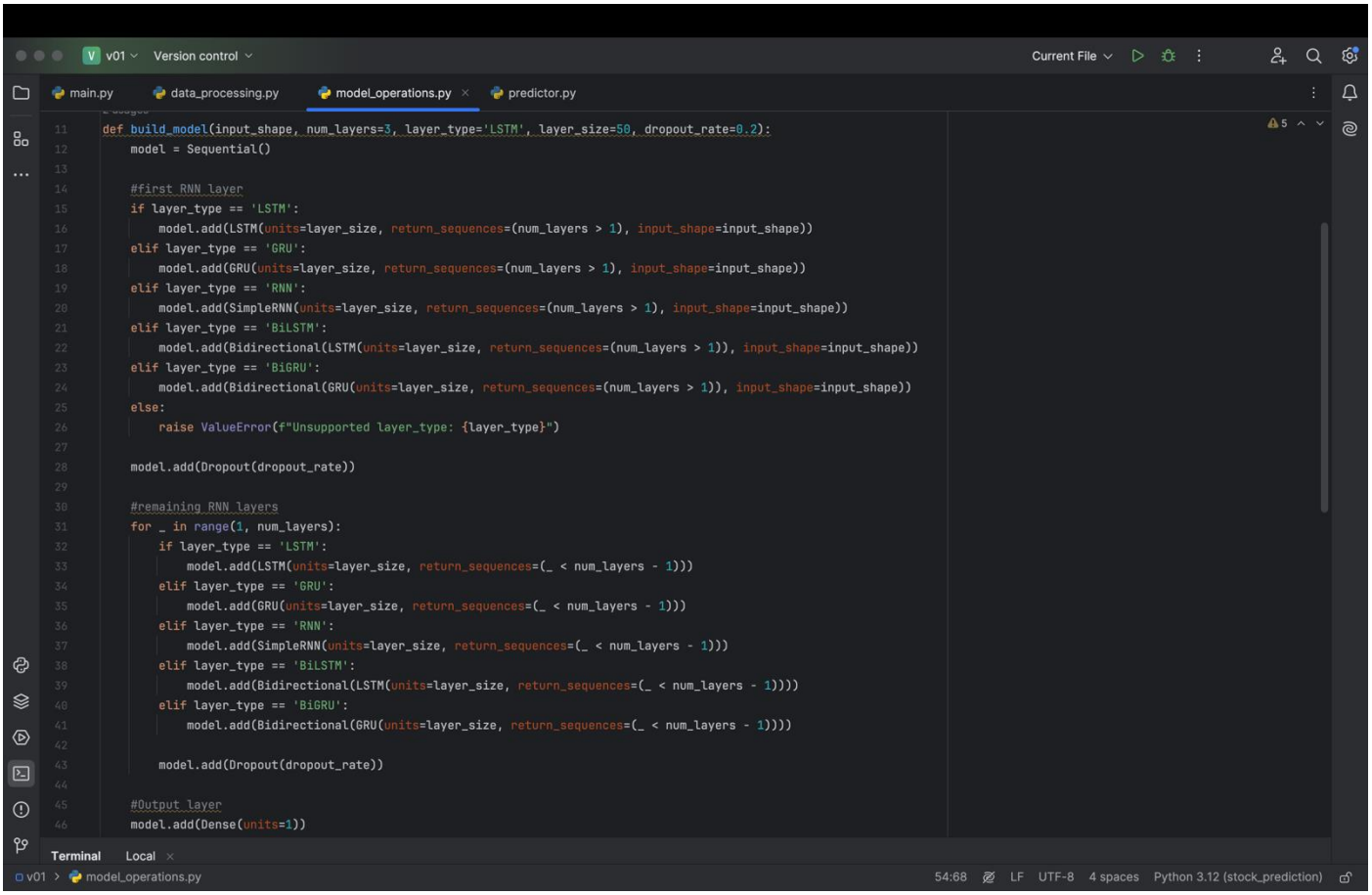

Image 8. build_model function version 3.0

These enhancements provide more flexibility for model experimentation, allowing users to better tailor the architecture to their specific needs. By offering options for different RNN types and bidirectional processing, the model is now better suited to handle a broader range of sequential prediction tasks.

Model Execution and Outputs

To execute the model and observe the stock prediction results, the script was run from the main.py file. This version of the model was set to run for 25 epochs, following the adjustments made to the deep learning architecture. These modifications were specifically made in the build_model function, which now includes flexibility for different recurrent neural network (RNN) types, such as LSTM, GRU, SimpleRNN, and their bidirectional variants.

The model was configured to use a **GRU** architecture with **4 layers**, **100 units** per layer, and a **dropout rate of 0.3**.

Key Changes in the Main Code (main.py):

1. **Model Configuration:**
   - The main code now includes the updated build_model function, allowing us to define the model with specific configurations. In this execution:
     - num_layers=4
     - layer_type='GRU'
     - layer_size=100
     - dropout_rate=0.3
   - This configuration was chosen to balance complexity with computational efficiency while still capturing both short-term and long-term dependencies in the stock price data.

```
# Build, train, and test model
model = build_model( input_shape: (x_train.shape[1], len(FEATURE_COLUMNS)), num_layers=4, layer_type='GRU', layer_size=100, dropout_rate=0.3)
train_model(model, x_train, y_train)
predicted_prices = model.predict(x_test)
predicted_prices = scalers["Close"].inverse_transform(predicted_prices)
```

Image 9. Model Execution with GRU Configuration

2. **Model Training:**
   - The model was trained on the preprocessed stock price data for **25 epochs**, which involved fitting the model on the training data (x_train, y_train). During each epoch, the model adjusted its weights using backpropagation, aiming to minimize the mean squared error (MSE) between predicted and actual stock prices.
     - The execution took some time to complete as the GRU-based model iterated over the dataset multiple times, adjusting its internal state based on the input sequence.

```
Epoch 10/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 0.0325
Epoch 11/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - loss: 0.0315
Epoch 12/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 0.0567
Epoch 13/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 0.0530
Epoch 14/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - loss: 0.0270
Epoch 15/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 86ms/step - loss: 0.0212
Epoch 16/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 0.0105
Epoch 17/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 0.0102
Epoch 18/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 0.0101
Epoch 19/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - loss: 0.0091
Epoch 20/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 0.0085
Epoch 21/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 0.0067
Epoch 22/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - loss: 0.0059
Epoch 23/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - loss: 0.0075
Epoch 24/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - loss: 0.0073
Epoch 25/25
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - loss: 0.0065
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step
```

Image 10. Model training

3. **Model Prediction:**
   - After training, the model was used to predict stock prices on the test dataset (x_test). The predicted prices were then inverse transformed using the previously fitted scaler to bring them back to the original scale for comparison with the actual prices.

4. **Stock Prediction Results:**
    - Once the model had finished training and predictions were made, the output graph displaying both **actual** and **predicted** stock prices was generated. Compared to the previous version of the model (v0.1), which used only LSTM layers, the results with this GRU-based model showed noticeable improvements.
    - The predicted stock prices more closely followed the trends of the actual prices, particularly in areas where the LSTM model struggled with sudden fluctuations. The GRU's ability to handle long sequences efficiently contributed to a smoother prediction curve.

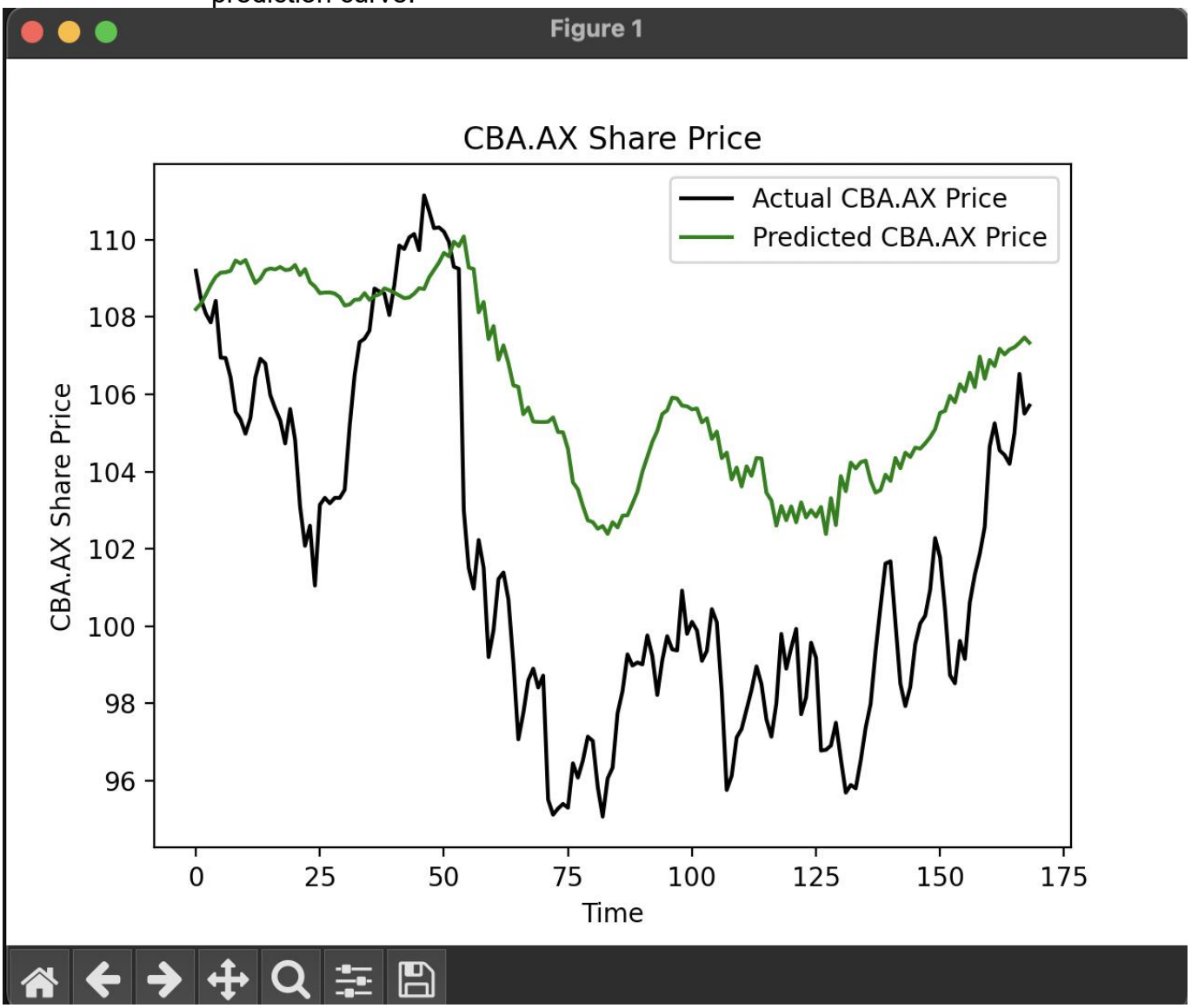

Image 11. Stock-prediction result verion v03

Comparison with Previous Versions:

- In the earlier version of the model, the predicted stock prices were more prone to lag behind the actual stock prices, especially when sudden market shifts occurred. The introduction of the GRU architecture with 4 layers, combined with the slightly higher dropout rate, helped prevent overfitting while improving the model's ability to track both upward and downward trends in stock prices.
- The graph now shows better alignment between the **actual** and **predicted** prices, with a reduction in the divergence between the two curves, particularly in volatile periods.

GitHub Repository

The project's code base is hosted on GitHub for version control and review. Everyone can access the repository via the following link: GitHub Repository. This repository contains all necessary files for the project and is available for the tutor to review the work.

References

- https://www.youtube.com/watch?v=UuBigNaO_18NeuralNine. (2022, October 1). *Stock Price Prediction using LSTM in Python*. [Video]. YouTube. Retrieved from
- https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/
- https://commtelnetworks.com/exploring-the-depths-unraveling-the-intricacies-of-machine-learning-and-deep-learning/
- https://www.tensorflow.org/api_docs/python/tf/keras/Sequential