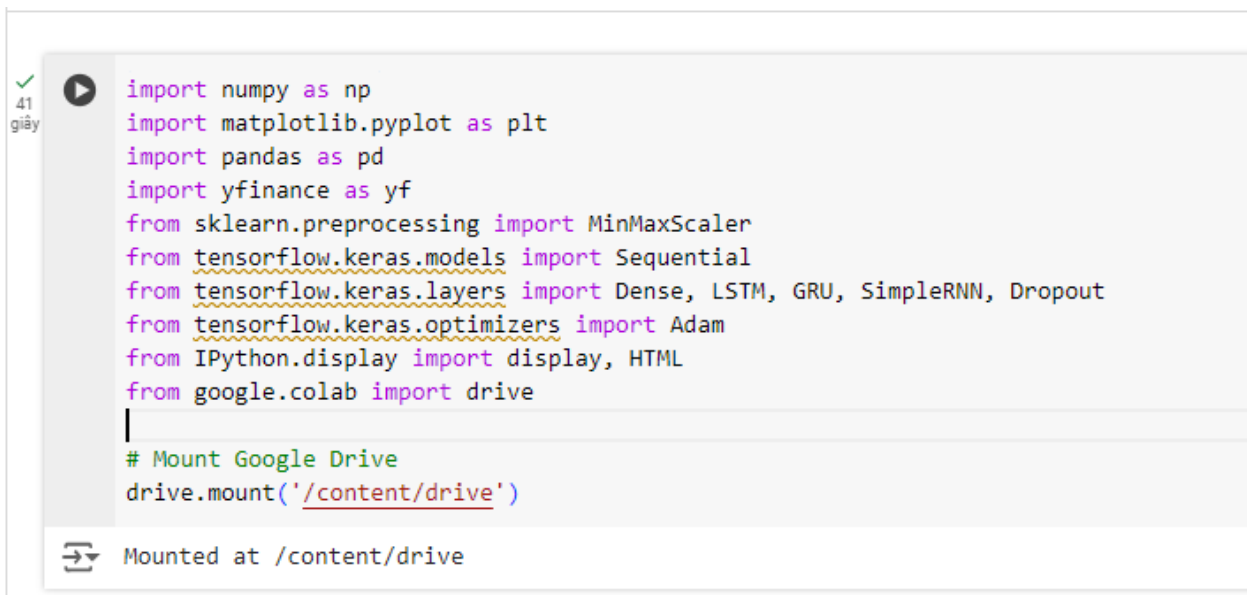# Task Report Cos30018 Option B

B.4: Machine Processing 1

Name: Le Bao Nguyen

Student Id: 104169837

I. Importing libraries (From the task B.2):

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU, SimpleRNN, Dropout
from tensorflow.keras.optimizers import Adam
from IPython.display import display, HTML
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')
```

Mounted at /content/drive

Figure 1: Importing libraries to run the code.

- The script import previous libraries and new libraries:
- "Sequential": A linear stack of layers in Keras.
- "Dense": A fully connected neural network layer.
- "LSTM", "GRU", "SimpleRNN": Various kinds of recurrent neural network layers applied to forecast time series.
- "Dropout": A regularization technique to prevent overfitting.
- "Adam": An optimization technique that uses training data to iteratively update the network weights.

II. Data loading and processing (From the task B.2):

```
[ ]     # Download data from Yahoo Finance
        data = yf.download(ticker, start=start_date, end=end_date)

        # Ensure the index is a DateTimeIndex
        data.index = pd.to_datetime(data.index)

        # Fill NaN values with previous values
        data.fillna(method='ffill', inplace=True)

        # Sanity check: Ensure high is not less than low
        if (data['High'] < data['Low']).any():
            raise ValueError("Inconsistent data: High value is less than Low value for some periods.")

        # Default to scaling the 'Close' column if no columns are specified
        if columns_to_scale is None or not columns_to_scale:
            columns_to_scale = ['Close']

        # Create a DataFrame for scaled data
        scaled_data = data.copy()
        scalers = {}

        # Scale specified columns
        for column in columns_to_scale:
            scaler = MinMaxScaler(feature_range=(0, 1))
            scaled_column = scaler.fit_transform(data[column].values.reshape(-1, 1))
            scaled_data[f'Scaled_{column}'] = scaled_column
            scalers[column] = scaler

        # Extract close prices and scale them
        close_prices = data['Close'].values.reshape(-1, 1)
        scaler = MinMaxScaler(feature_range=(0, 1))
        scaled_close_prices = scaler.fit_transform(close_prices)

        # Determine split date based on split_ratio or split_by_date
        if split_by_date:
            split_date = pd.Timestamp(split_ratio)
        else:
            split_date = pd.to_datetime(start_date) + (pd.to_datetime(end_date) - pd.to_datetime(start_date)) * split_ratio
```

Figure 2: Loading and processing data (1).

```
        scaler = MinMaxScaler(feature_range=(0, 1))
        scaled_close_prices = scaler.fit_transform(close_prices)

        # Determine split date based on split_ratio or split_by_date
        if split_by_date:
            split_date = pd.Timestamp(split_ratio)
        else:
            split_date = pd.to_datetime(start_date) + (pd.to_datetime(end_date) - pd.to_datetime(start_date)) * split_ratio

        # Split data into train and test sets
        if split_by_date:
            train_data = scaled_close_prices[data.index < split_date]
            test_data = scaled_close_prices[data.index >= split_date]
        else:
            train_data = scaled_close_prices[:int(len(scaled_close_prices) * split_ratio)]
            test_data = scaled_close_prices[int(len(scaled_close_prices) * split_ratio):]

        # Save data to a local file, replacing any existing file
        if local_file:
            file_path = f"/content/drive/My Drive/Cos30018/{local_file}"  # Change to your desired path in Google Drive
            data.to_csv(file_path)

        return train_data, test_data, scalers, data, scaled_data
```

Figure 3: Loading and processing data (2).

- We still use the same data loading and processing just like B.2.

III. Displaying data in a custom table (From the task B.2):

```
[ ] def display_custom_table(df, num_rows=5):
        """
        Display the first few and last few rows of the DataFrame with ellipses in between.

        Parameters:
        - df: DataFrame to display.
        - num_rows: Number of rows to display from the start and end of the DataFrame.
        """
        if len(df) <= 2 * num_rows:
            # Display the entire DataFrame if it's small enough
            display(df)
        else:
            # Display the first few and last few rows with ellipses in between
            head = df.head(num_rows)
            tail = df.tail(num_rows)
            ellipsis_row = pd.DataFrame([['...'] * len(df.columns)], columns=df.columns, index=['...'])
            df_display = pd.concat([head, ellipsis_row, tail])
            display(HTML(df_display.to_html(index=True)))
```

Figure 4: Displaying the data from csv file.

- We still use the same displaying data function just like B.2.

IV. Model Creation:

```
def create_dl_model(input_shape, layers_config):
    """
    Create a deep learning model based on the provided configuration.

    Parameters:
    - input_shape: Shape of the input data.
    - layers_config: List of dictionaries where each dictionary specifies the type and parameters of a layer.

    Returns:
    - model: Compiled Keras model.
    """
    model = Sequential()
    for i, layer in enumerate(layers_config):
        layer_type = layer.get("type")
        units = layer.get("units", 50)
        activation = layer.get("activation", "relu")
        return_sequences = layer.get("return_sequences", False)
        dropout_rate = layer.get("dropout_rate", 0.0)

        if layer_type == "LSTM":
            if i == 0:
                model.add(LSTM(units, activation=activation, return_sequences=return_sequences, input_shape=input_shape))
            else:
                model.add(LSTM(units, activation=activation, return_sequences=return_sequences))
        elif layer_type == "GRU":
            if i == 0:
                model.add(GRU(units, activation=activation, return_sequences=return_sequences, input_shape=input_shape))
            else:
                model.add(GRU(units, activation=activation, return_sequences=return_sequences))
        elif layer_type == "RNN":
            if i == 0:
                model.add(SimpleRNN(units, activation=activation, return_sequences=return_sequences, input_shape=input_shape))
            else:
                model.add(SimpleRNN(units, activation=activation, return_sequences=return_sequences))

        if dropout_rate > 0:
            model.add(Dropout(dropout_rate))

    model.add(Dense(1))  # Final layer for output
    model.compile(optimizer=Adam(), loss='mean_squared_error')
    return model
```

Figure 5: Code to create the model.

- We go over the dictionaries in the "layers_config" list, which define the type, units, activation function, and other parameters for every layer.
- We add each layer to the model after identifying if it is an LSTM, GRU, or RNN. An "input_shape" that specifies the form of the input data is needed for the first layer.
- When stacking RNN layers, the "return_sequences" argument is used. By doing this, the layer makes sure to return the entire output sequence rather than just the end result.
- The addition of dropout layers eliminates overfitting. A Dropout layer is created after the current RNN layer if "dropout_rate" is larger than 0.
- The predicted stock price is output by the Dense layer, which is the last layer and has a single unit. The mean squared error (MSE) loss function and Adam optimizer are used in the construction of the model, making it appropriate for regression applications like stock price prediction.

V. Experimentation with Different Configurations:

```python
def experiment_with_models(train_data, test_data, scaler, layers_configs, epochs=50, batch_size=32):
    """
    Experiment with different DL networks and configurations.

    Parameters:
    - train_data: Scaled training data.
    - test_data: Scaled testing data.
    - scaler: Scaler used for normalization.
    - layers_configs: List of different configurations to test.
    - epochs: Number of epochs for training.
    - batch_size: Batch size for training.
    """
    results = []
    time_steps = 60
    input_shape = (time_steps, 1)  # Input shape for the model

    # Reshape data for the model
    X_train, y_train = [], []
    X_test, y_test = [], []

    for i in range(time_steps, len(train_data)):
        X_train.append(train_data[i-time_steps:i, 0])
        y_train.append(train_data[i, 0])

    for i in range(time_steps, len(test_data)):
        X_test.append(test_data[i-time_steps:i, 0])
        y_test.append(test_data[i, 0])

    X_train, y_train = np.array(X_train), np.array(y_train)
    X_test, y_test = np.array(X_test), np.array(y_test)

    X_train = np.reshape(X_train, (X_train.shape[0], time_steps, 1))
    X_test = np.reshape(X_test, (X_test.shape[0], time_steps, 1))

    for config in layers_configs:
        model = create_dl_model(input_shape, config['layers'])
        history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_test, y_test), verbose=2)
```

Figure 6: Code to experiment with model (1).

```python
    X_test = np.reshape(X_test, (X_test.shape[0], time_steps, 1))

    for config in layers_configs:
        model = create_dl_model(input_shape, config['layers'])
        history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_test, y_test), verbose=2)

        # Predict and inverse transform
        predicted_stock_price = model.predict(X_test)
        predicted_stock_price = scaler.inverse_transform(predicted_stock_price)
        real_stock_price = scaler.inverse_transform(y_test.reshape(-1, 1))

        # Plot the results
        plt.figure(figsize=(14, 5))
        plt.plot(real_stock_price, color='red', label='Real Stock Price')
        plt.plot(predicted_stock_price, color='blue', label='Predicted Stock Price')

        # Create title from config without duplicates
        unique_layers = []
        for layer in config['layers']:
            layer_desc = f"{layer['type']} (units={layer.get('units', 50)})"
            if layer_desc not in unique_layers:
                unique_layers.append(layer_desc)
        model_description = ' - '.join(unique_layers)

        plt.title(f'Stock Price Prediction: {model_description}')

        plt.xlabel('Time')
        plt.ylabel('Stock Price')
        plt.legend()
        plt.show()

        results.append({
            "config": config,
            "history": history,
            "predicted_stock_price": predicted_stock_price,
            "real_stock_price": real_stock_price
        })

    return results
```

Figure 7: Code to experiment with model (2).

- We define "time_steps" as 60, which means each input sequence will contain 60 previous data points.
- We construct sliding windows of 60 data points for each time step in the training and testing data ("X_train" and "X_test"), with the next data point serving as the desired output ("y_train" and "y_test").
- The input data is reshaped to fit the required input shape for the RNN models.
- We use "create_dl_model" to make a model for each configuration in "layers_configs" and then train it with the training set. To train the model, we define the batch size and number of epochs using the "fit" method.
- Using test data, we apply the model to forecast stock prices once it has been trained. The scaler is then used to inversely translate these predictions back to their original scale.
- We plot the real and predicted stock prices to visually evaluate the model's performance. The unique layer configurations are used as the title for each plot to differentiate between the models.
- A list including the model configuration, training history, and predictions is created as a result and kept for additional examination.

VI.   Main script run:

```python
# Example usage
if __name__ == "__main__":
    ticker = 'AMZN'
    start_date = '2016-01-01'
    end_date = '2024-03-20'
    local_file = 'amzn_data.csv'

    # Specify columns to scale
    columns_to_scale = ["Volume"]

    # Load and process data
    train_data, test_data, scalers, original_data, scaled_data = load_and_process_data(
        ticker=ticker,
        start_date=start_date,
        end_date=end_date,
        local_file=local_file,
        split_ratio=0.7,
        columns_to_scale=columns_to_scale
    )

    # Display the original and scaled data tables
    display_custom_table(scaled_data, num_rows=5)

    # Define layer configurations to test
    layers_configs = [
        {"layers": [{"type": "LSTM", "units": 50, "return_sequences": True}, {"type": "LSTM", "units": 50}]},
        {"layers": [{"type": "GRU", "units": 50, "return_sequences": True}, {"type": "GRU", "units": 50}]},
        {"layers": [{"type": "RNN", "units": 50, "return_sequences": True}, {"type": "RNN", "units": 50}]}
    ]

    # Experiment with models
    results = experiment_with_models(train_data, test_data, scalers["Close"], layers_configs, epochs=50, batch_size=32)
```

Figure 8: The script to run the code and the prediction.

- LSTM Configuration: Two LSTM layers with 50 units each, the first one returning sequences.
- GRU Configuration: Two GRU layers with 50 units each, the first one returning sequences.
- RNN Configuration: Two SimpleRNN layers with 50 units each, the first one returning sequences.
- The line calls the "experiment_with_models" function to train and evaluate the defined DL models on the training and testing data.

Stock Price Prediction: LSTM (units=50)
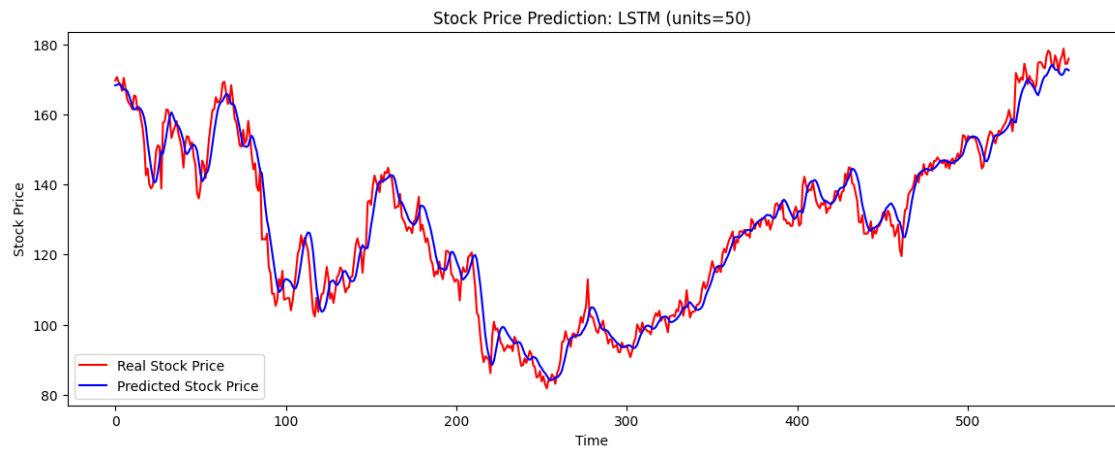
Figure 9: The predicted chart if we use LTSM layer type.
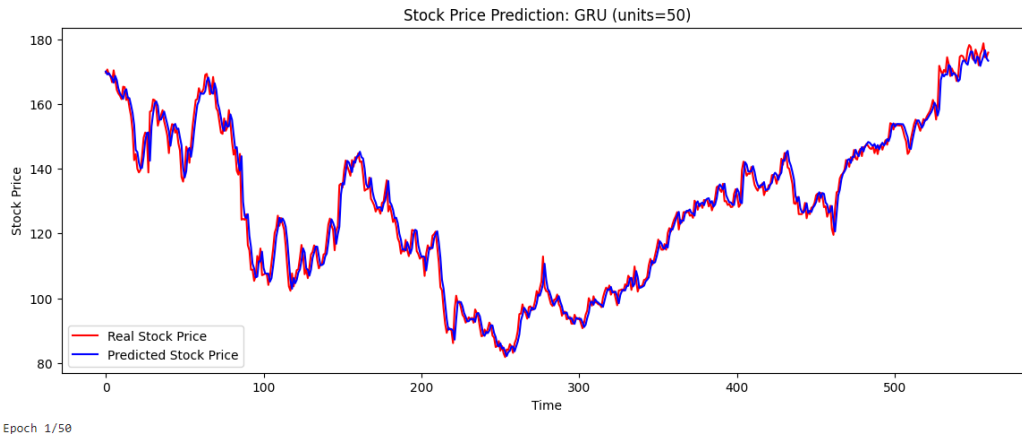
Stock Price Prediction: GRU (units=50)

Figure 10: The predicted chart if we use GRU layer type.

Epoch 49/50
44/44 - 1s - loss: 1.8816e-04 - val_loss: 4.1321e-04 - 1s/epoch - 26ms/step
Epoch 50/50
44/44 - 1s - loss: 1.7835e-04 - val_loss: 4.3049e-04 - 1s/epoch - 26ms/step
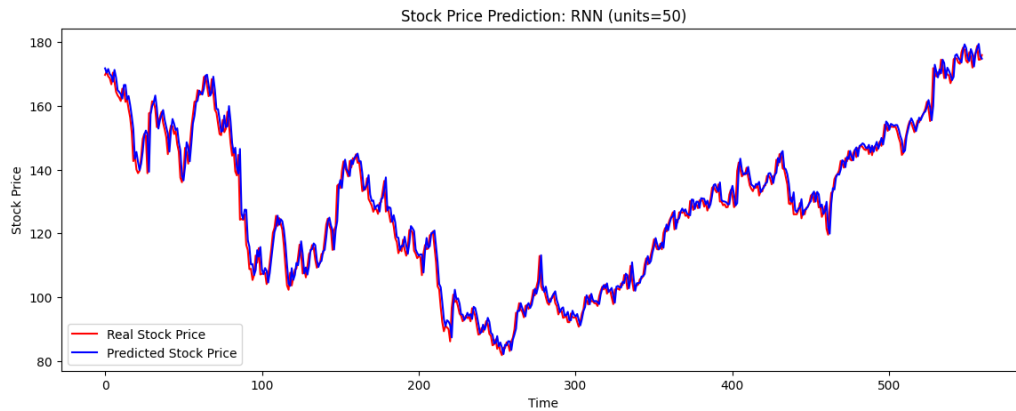18/18 [==============================] - 0s 9ms/step

Figure 11: The predicted chart if we use RNN layer type.
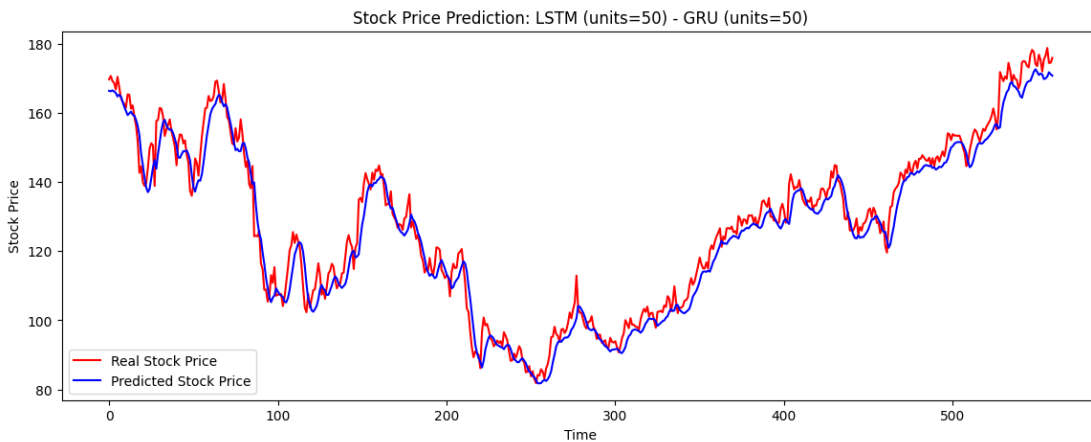

Figure 12: The predicted chart if we use LTSM and GRU layer types.

VII.   References:

SAYAH,   F.   (2023).   *Stock   Market   Analysis   +   Prediction   using   LSTM*.
https://www.kaggle.com/code/faressayah/stock-market-analysis-prediction-using-lstm

OZTURK,   O.   (2020).   *Stock   Price   prediction   by   simple   RNN   and   LSTM*.
https://www.kaggle.com/code/ozkanozturk/stock-price-prediction-by-simple-rnn-and-lstm