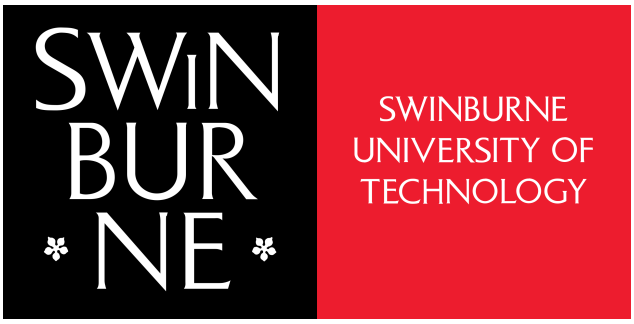


# Task B4: Machine Learning 1

Conducted by:

## Group 1


| *Team Members* | *Email* |
|---|---|
| Thanh Tam Vo | 103487596@student.swin.edu.au |
| Tai Minh Huy Nguyen | 104220352@student.swin.edu.au |


March 28, 2024

# Table of Contents

# 1 Introduction

In this **Task B4: Machine Learning 1**, we will now investigate more effective methods for building our Deep Learning model. Rather than requiring the manual construction of the Deep Learning (DL) network by directly adding the layers, we can design a function that accepts as inputs the number of layers, the size of each layer, and the name of the layer.

This report cover how did we construct 3 deeplearning models (RNNs, LSTM, GRU) and the compare prediction result on Apple share price from 3 mentioned models.

# 2 Importing Dependencies

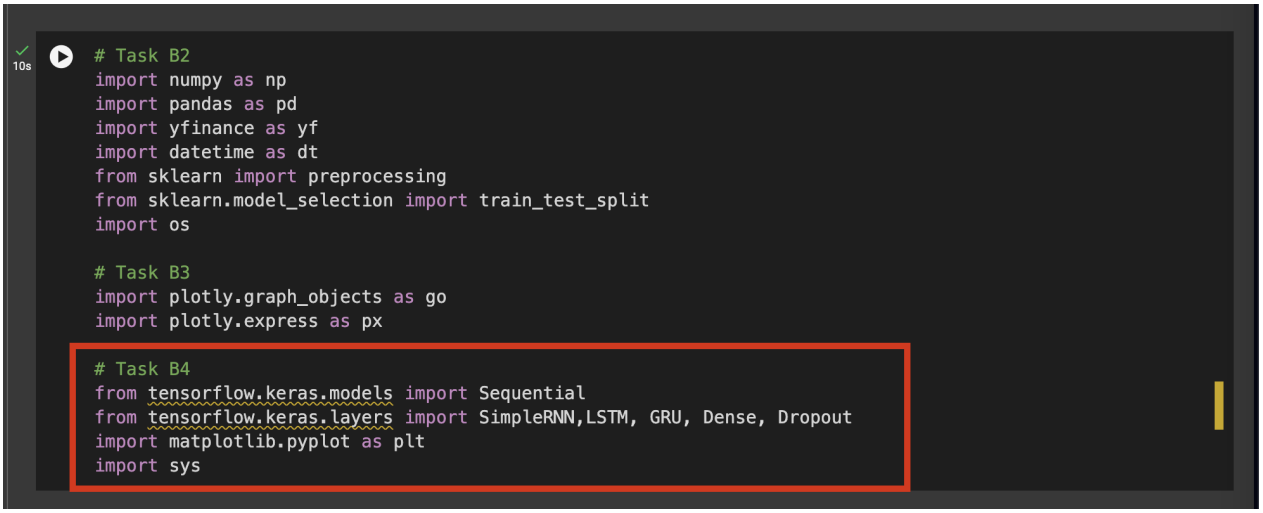


Figure 1: Importing Python Packages for Task B4

There are some new several packages that are need to be imported (Figure 1):

- **`Sequential`** is built-in sequence model provided by Keras
- **`SimpleRNN, LSTM, GRU, Dropout, Dense`** deeplearning layers by Keras
- **`Mathplotlib`** to visuallize prediction result.

# 3 Hyperparameters

```
# Task B2
TICKER = "AAPL"
START_DATE = "2010-01-01"
END_DATE = "2023-12-31"
LOOK_UP_DAYS = 60
TRAINING_RATIO = 0.8    # 0.7 == 70%
SCALE_DATA = True
SCALING_METHOD = "MinMax"       # MinMax, Standard

# Task B3
TRADING_PERIOD = 60
CONSECUTIVE_DAYS = 300

# Task B4
NUMBER_OF_LAYER = 2
NUMBER_OF_HIDDEN_UNITS = 80
MODEL_NAME = "RNN"      ## "RNN". "LSTM", "GRU"
DROP_OUT_RATE = 0     ## dropout rate in [0,1]
NUMBER_OF_EPOCHS = 25
BATCH_SIZE = 12
FEATURE_PREDICT = "Close"    ## "Open", "High", "Close", "Low"
LOSS_FUNCTION = "mean_squared_error" ## "mean_squared_error", "mean_absolute_error", "huber_loss"
OPTIMIZER = "adam"  ## "adam", "RMSprop", "SGD"
```

Figure 2: New Hyperparameters for Task B4

In Figure 2, there are new some new added constant:

- **NUMBER_OF_LAYER**: the number of hidden layer in Sequential Model
- **NUMBER_OF_HIDDEN_UNITS**: the number of units in a hidden layer
- **MODEL_NAME**: model name (RNN, LSTM, GRU)
- **DROP_OUT_RATE**: the dropout rate to reduce the overfitting situation
- **NUMBER_OF_EPOCHS**: the number of epochs for training
- **BATCH_SIZE**: the number of training data point in a batch
- **FEATURE_PREDICT**: the stock feature to predict (*Close* by default)
- **LOSS_FUNCTION**: the loss function to estimate the predict result
- **OPTIMIZER**: the optimized algorithm to find the perfect parameters

# 4 Dataset Preparation

In this task B4, the goal is looking back the *Close Price* of the the last `LOOK_UP_DAYS` to predict the *Close Price* of the current day.

Let:

- $d$ denotes the number of `LOOK_UP_DAYS`
- $s$ denotes the start date in the training set
- $e$ denotes the last date in the training set
- $x_i$ denotes the *Close Price* of the $i^{th}$ date.

Hence, our training data a Matrix as below:

$$\mathbf{X} = \begin{bmatrix} \begin{bmatrix} x_s & x_{s+1} & \cdots & x_{s+d} \end{bmatrix} \\ \begin{bmatrix} x_{s+1} & x_{s+2} & \cdots & x_{s+d+1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} x_{e-d} & x_{e-d+1} & \cdots & x_e \end{bmatrix} \end{bmatrix}, \mathbf{Y} = \begin{bmatrix} x_{s+d+1} \\ x_{s+d+2} \\ \vdots \\ x_{e+1} \end{bmatrix}$$

In Figure 2, we have:

- Our stock data begins from 2010 to the end of 2023
- The training ratio equals 0.8
- The look up days is 60

With the above configuration, the training dataset is a 2D-Array with the shape of $(2000, 60)$ (supposed that 80% of the total records equals to 2000)

# 5 Recurrent Neural Networks

We will start with the very first Deep Learning model that deal with sequential data: Recurrent Neural Networks.

```python
def RecurrentNeuralNetworks(layerNums=NUMBER_OF_LAYER,hidden_units=NUMBER_OF_HIDDEN_UNITS,
                            loss_type=LOSS_FUNCTION, optimizerType=OPTIMIZER, dense_unit=1,
                            activation=["tanh", "linear"], dropoutRate = DROP_OUT_RATE):
    model = Sequential()

    for i in range(layerNums):
        if i == (layerNums - 1):
            model.add(SimpleRNN(hidden_units, activation=activation[0]))
            model.add(Dropout(dropoutRate))
        else:
            model.add(SimpleRNN(hidden_units, activation=activation[0], return_sequences=True))
            model.add(Dropout(dropoutRate))

    model.add(Dense(units=dense_unit, activation=activation[1]))
    model.compile(loss=loss_type, optimizer=optimizerType)
    return model
```

Python

Figure 3: Implementation for RNN model using Keras

Let's take a look at the parameters:

- **`layerNums`** is the number of hidden layers in the model
- **`hidden_units`** is the number of nodes in a hidden layer
- **`loss_type`** is the loss function to evaluate the predicted result
- **`optimizerType`** is the optimized algorithm to train the model
- **`dense_unit`** is the number of units in the output layer
- **`activation`** is the array storing all activation functions
- **`dropoutRate`** is the dropout rate to reduce the overfitting situation

First, we initializes a Sequential model from the Keras library. The Sequential model allows us to stack layers one after another in a linear manner. Then, we iterate **`layerNums`** times, adding RNN layers to the model. If it's the last layer (`i == (layerNums-1)`), a single SimpleRNN layer without returning sequences is added, followed by a Dropout layer to prevent overfitting. If it's not the last layer, a SimpleRNN layer with returning sequences is added to ensure that the layer outputs sequences, which is necessary for subsequent layers in the network to receive sequential data. After that, a Dense (fully connected) layer is added to the model. This layer is responsible for producing the final output of the neural network. It has `dense_unit` units and uses the activation function specified by `activation[1]`. Finally, the model is compiled using the specified loss function `loss_type` and optimizer `optimizerType`. This prepares the model for training by configuring the learning process.

# 6 Long Short-term Memory and Gated Recurrent Unit

The implementation to build the Long Short-term Memory (LSTM) and Gated Recurrent Unit (GRU) is the same as building RNNs model. But instead of adding a **`SimpleRNN()`** layer to the hidden layer, there are a small difference:

- adding **`LSTM()`** layer if the model is LSTM
- adding **`GRU()`** layer if the model is GRU

```python
def LongShortTermMemory(layerNums=NUMBER_OF_LAYER,hidden_units=NUMBER_OF_HIDDEN_UNITS,
loss_type=LOSS_FUNCTION, optimizerType=OPTIMIZER, dense_unit=1, activation=["tanh", "linear"],
dropoutRate = DROP_OUT_RATE):
    model = Sequential()

    for i in range(layerNums):
        if i == (layerNums - 1):
            model.add(LSTM(hidden_units, activation=activation[0]))
            model.add(Dropout(dropoutRate))
        else:
            model.add(LSTM(hidden_units, activation=activation[1], return_sequences=True))
            model.add(Dropout(dropoutRate))

    model.add(Dense(units=dense_unit, activation=activation[1]))
    model.compile(loss=loss_type, optimizer=optimizerType)
    return model
```

Python

Figure 4: Implementation for LSTM model using Keras

```python
def GatedRucurrentUnit(layerNums=NUMBER_OF_LAYER,hidden_units=NUMBER_OF_HIDDEN_UNITS,
loss_type=LOSS_FUNCTION, optimizerType=OPTIMIZER, dense_unit=1, activation=["tanh", "linear"],
dropoutRate = DROP_OUT_RATE):
    model = Sequential()

    for i in range(layerNums):
        if i == (layerNums - 1):
            model.add(GRU(hidden_units, activation=activation[0]))
            model.add(Dropout(dropoutRate))
        else:
            model.add(GRU(hidden_units, activation=activation[0], return_sequences=True))
            model.add(Dropout(dropoutRate))

    model.add(Dense(units=dense_unit, activation=activation[1]))
    model.compile(loss=loss_type, optimizer=optimizerType)
    return model
```

Python

Figure 5: Implementation for GRU model using Keras

# 7 Results

## 7.1 Dataset

The dataset is provided by Yahoo Finance: a CSV File including the stock price of Apple Inc. as below:

Table 1: Stock Table of Apple Inc.

| Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| 2010-01-04 | 7.622 | 7.660 | 7.585 | 7.643 | 6.470 | 493729600 |
| 2010-01-05 | 7.664 | 7.699 | 7.616 | 7.656 | 6.481 | 601904800 |
| 2010-01-06 | 7.656 | 7.686 | 7.526 | 7.534 | 6.378 | 552160000 |

## 7.2 Hyperparameters

In order to compare the result from 3 model: RNNs, LSTM, GRU; they must be evaluated with the same configuration:

- Two hidden layers
- 80 units per layer
- Fully connected (No dropout)
- Optimizer: Adam Optimizer Algorithm
- Number of epochs: 30
- All records are scaled using MinMax Scaler
- In each epoch, 20% of the training set will be used for evaluation set

## 7.3 Results

### 7.3.1 Recurrent Neural Networks

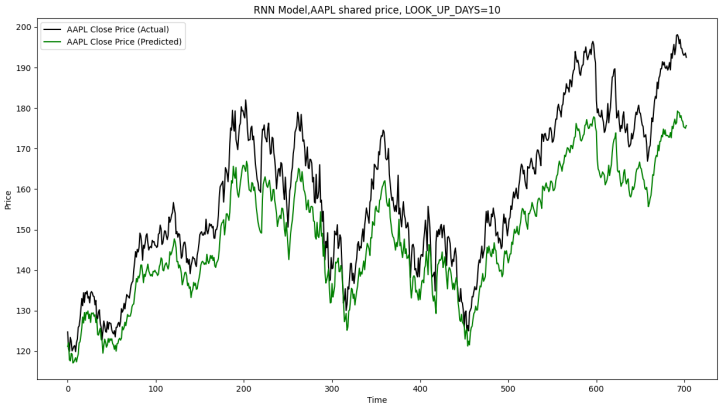


Figure 6: Predicted Result from RNNs Model

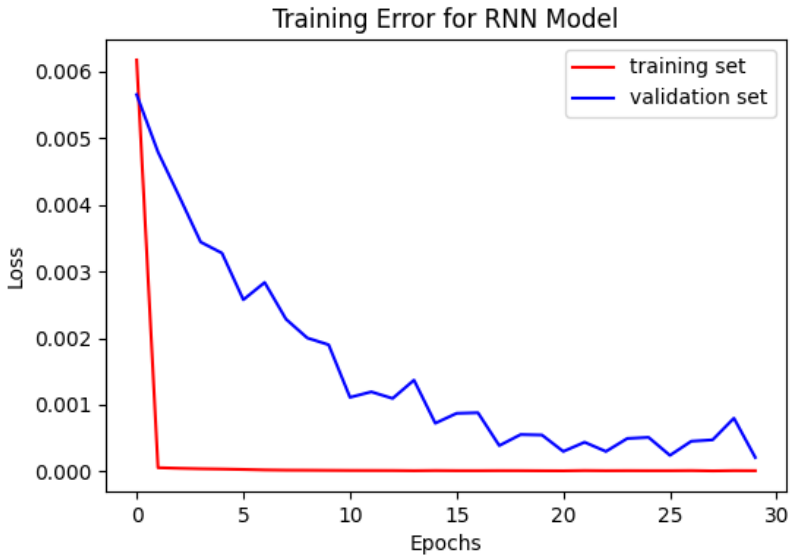


Figure 7: Error Plotting of RNNs Model

### 7.3.2 Long Short-term Memory

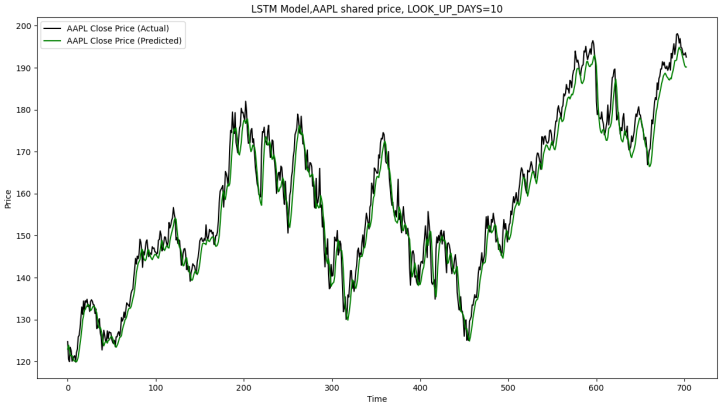


Figure 8: Predicted Result from LSTM Model


Figure 9: Error Plotting of LSTM Model

### 7.3.3 Gated Recurrent Unit

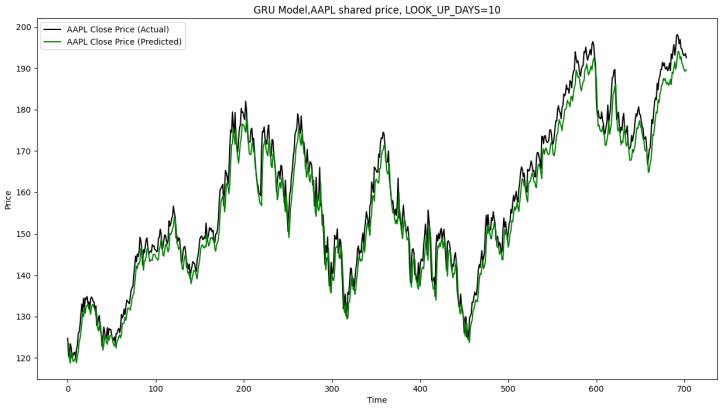


Figure 10: Predicted Result from GRU Model

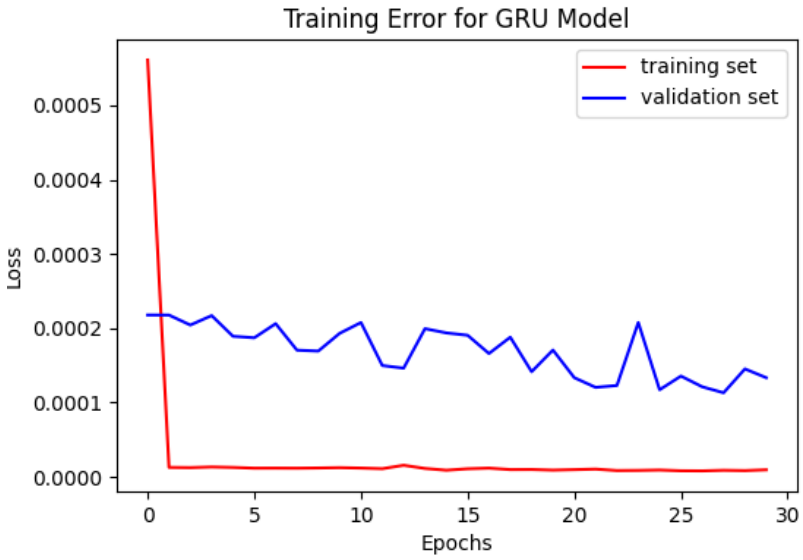


Figure 11: Error Plotting of GRU Model

### 7.3.4 Evaluation Comments

By observing Figure 6, 8, and 10, we can conclude that the LSTM model has the best output result compared to those from RNN and GRU. In the next section, we will modify the configuration: **Increasing the look up days** to answer the question: " *Can we get a good predicted result as Figure 8?*"

## 7.4 Other Evaluations

In Section 7.3.4, we know that the LSTM model performs the best reuslt among other models. In this section, the `LOOK_UP_DAYS` is set to 30. The model will look at the *Close Price* of the last 30 days to perform the prediction of the next day. Figure below is the result.

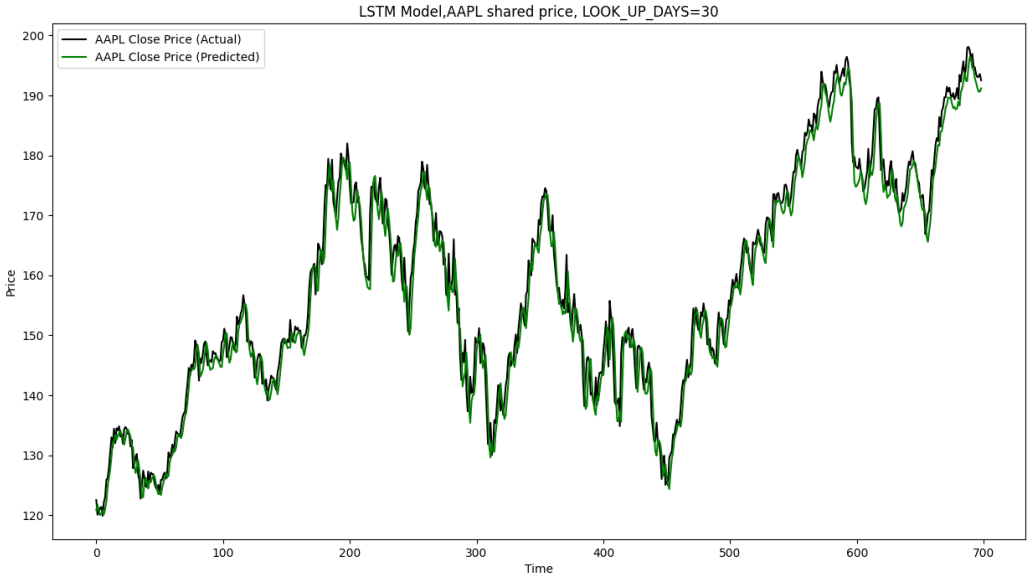


Figure 12: Other Predicted Result from LSTM

# 8 Conclusion

We can see that the LSTM performs perfectly. With a new architecture in hidden layer compared to the basic RNN architecture, LSTM can solve the problem of gradient vanishing if our sequence is longer. In the section 7.4, the `LOOK_UP_DAYS` increases making our sequence data longer, but the LSTM model can handle it and returns a good result for us.