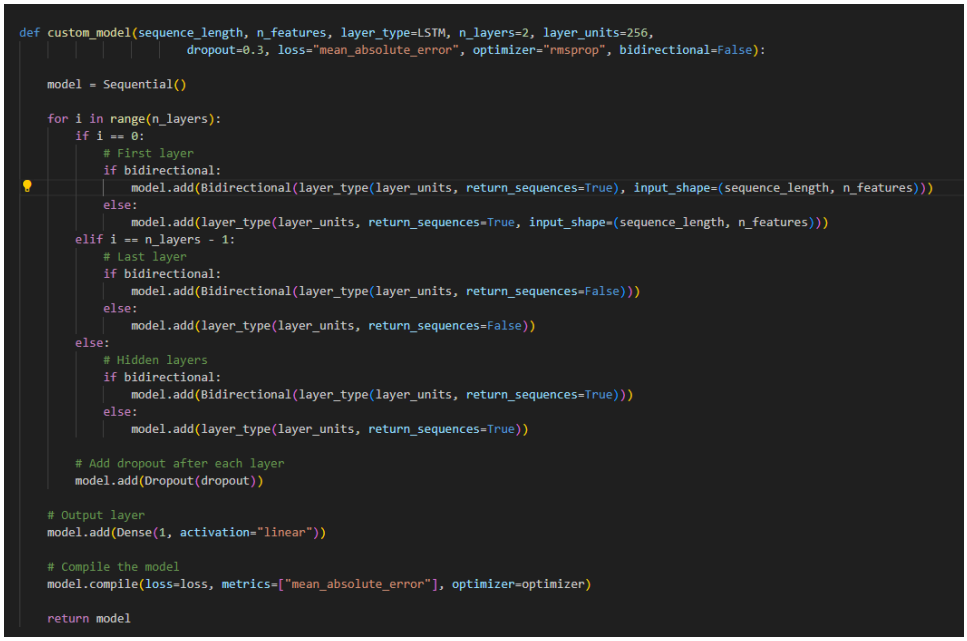
COS30018 - Option B - Task 4: Machine Learning 1

Jiin Wen Tan 102846565

The goal for this task is to explore better alternatives to construct our deep learning model instead of constructing it manually and explore different deep learning networks such as LSTM, RNN and GRU.

Function for custom models



```python
def custom_model(sequence_length, n_features, layer_type=LSTM, n_layers=2, layer_units=256,
                 dropout=0.3, loss="mean_absolute_error", optimizer="rmsprop", bidirectional=False):

    model = Sequential()

    for i in range(n_layers):
        if i == 0:
            # First layer
            if bidirectional:
                model.add(Bidirectional(layer_type(layer_units, return_sequences=True), input_shape=(sequence_length, n_features)))
            else:
                model.add(layer_type(layer_units, return_sequences=True, input_shape=(sequence_length, n_features)))
        elif i == n_layers - 1:
            # Last layer
            if bidirectional:
                model.add(Bidirectional(layer_type(layer_units, return_sequences=False)))
            else:
                model.add(layer_type(layer_units, return_sequences=False))
        else:
            # Hidden layers
            if bidirectional:
                model.add(Bidirectional(layer_type(layer_units, return_sequences=True)))
            else:
                model.add(layer_type(layer_units, return_sequences=True))

        # Add dropout after each layer
        model.add(Dropout(dropout))

    # Output layer
    model.add(Dense(1, activation="linear"))

    # Compile the model
    model.compile(loss=loss, metrics=["mean_absolute_error"], optimizer=optimizer)

    return model
```

Resource: https://github.com/x4nth055/pythoncode-tutorials/blob/master/machine-learning/stock-prediction/stock_prediction.py

Function above takes in several parameters to return a deep learning model:
sequence length: length of the input sequences

N_features: number of input features

Layer_type: type of recurrent layer (in this case by default LSTM)

N_layers: number of recurrent layers

Layer_units: number of neurons in each recurrent layers

Dropout: dropout rate between layers

Loss: loss function for model compilation

Optimizer: optimizer for model compilation

Bidirectional: boolean

The concept is to create a sequential model first then enter a for loop to iterate through n_layers to add recurrent layers to the model. After each recurrent layer a dropout layer will be added to prevent

overfitting. Then an output layer with a single unit and linear activation function is added for regression tasks. Finally, the model is then compiled with specified loss function, metrics (mean_absolute_error) and optimizer.

Why bidirectional? What is bidirectional?

https://en.wikipedia.org/wiki/Bidirectional_recurrent_neural_networks#:~:text=Bidirectional%20recurrent%20neural%20networks%20(BRNN,future%20(forward)%20states%20simultaneously.

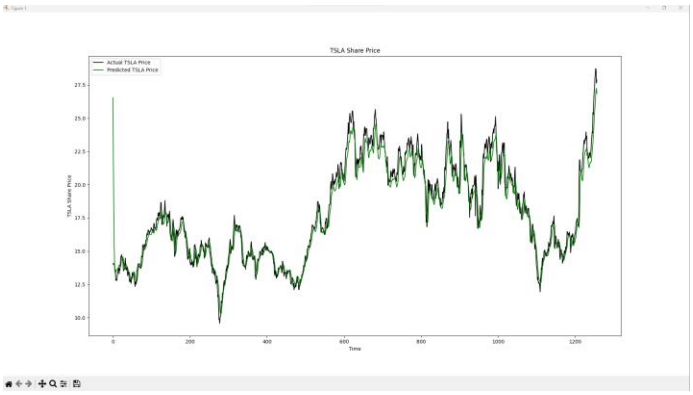
https://d2l.ai/chapter_recurrent-modern/bi-rnn.html

Bidirectional: functioning in two directions, connecting two hidden layers of opposite directions to the same output. With this form of generative deep learning, the output layer can get information from past(backwards) and future (forward) states simultaneously.

Experiment with different DL networks with different parameters

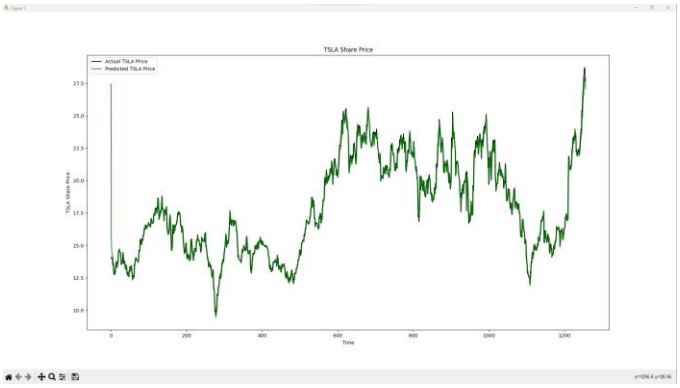
LSTM

Layer units = 50, epoch 25



As since from the result above this model with the given parameters predicted prices that were lower than the actual price, this is due to data underfitting. Underfitting occurred because of the low number of epochs, which determines how many times the model learns from the data and possibly limits the model's ability to learn complex patterns in the data. More importantly, LSTM model doesn't have enough opportunities to capture the nuances of stock price fluctuations. Also, with moderate numbers of layer units might not provide enough capacity to capture the complexities of the data which causes underfitting. LSTM models tend to perform better when exposed to more complex patterns and both epoch and layer units are not sufficient to generate accurate predictions.

Layer units = 250, epoch = 50

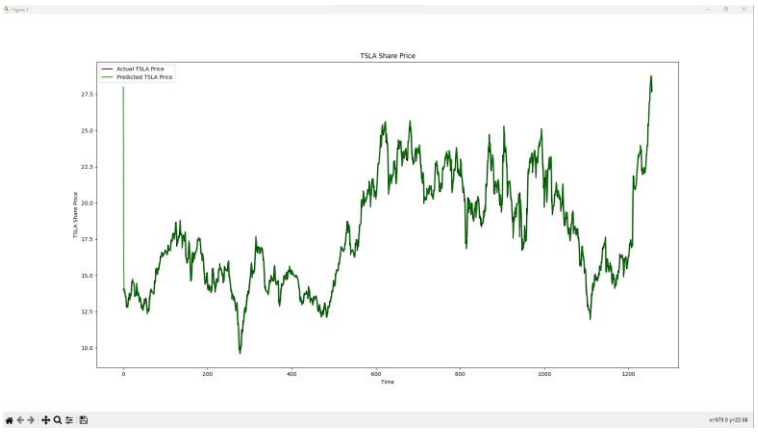


In this scenario, by increasing both epoch to 50 and layer units to 250 we can clearly tell that the predicted prices were so much closer to the actual price compared to the previous scenario for LSTM model. The higher number of epochs allowed the model more opportunity to learn from the data which helps capture more complex patterns. The increased-on layer units provided the model with capacity to

capture more complex patterns in the data that ultimately contributes to the improvement of the predictions.

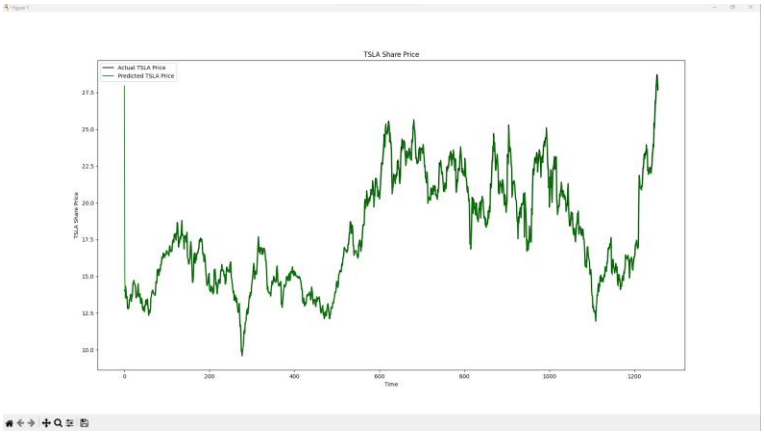
GRU

layer units = 50, epoch = 25



As per the result it is extremely close to the actual price with the GRU model. This was evidence of GRU's well-known ability in identifying temporary dependencies in data. With predictions that closely matched the actual prices, the GRU model displayed high accuracy. This is due to GRUs' ability to detect short-term dependencies, an important aspect of stock price forecasting. Additionally, the model was able to effectively learn from the data and create precise assumptions because of the combination of a moderate number of layer units and a suitable number of epochs.

Layer units = 250, epoch =50

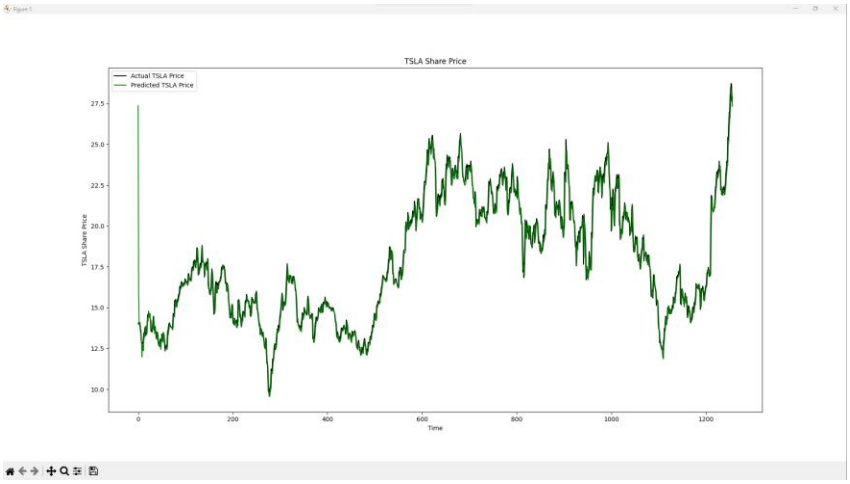


However, after increasing the layer units and epoch the prediction does not meet my expectation of better accuracy. Seems to me that the model has already reached a point of diminishing returns in terms of model complexity, meaning that it is not benefiting from the increased number of epochs and layer units. Although with the changes of epochs and layer units, the prediction remains extremely close to the actual prices showing that it might have already learned the essentials patterns in the data during

the previous scenario (epoch = 25, layer units = 50). Therefore, the increase of epochs and layer units will not lead to improvements in prediction accuracy.

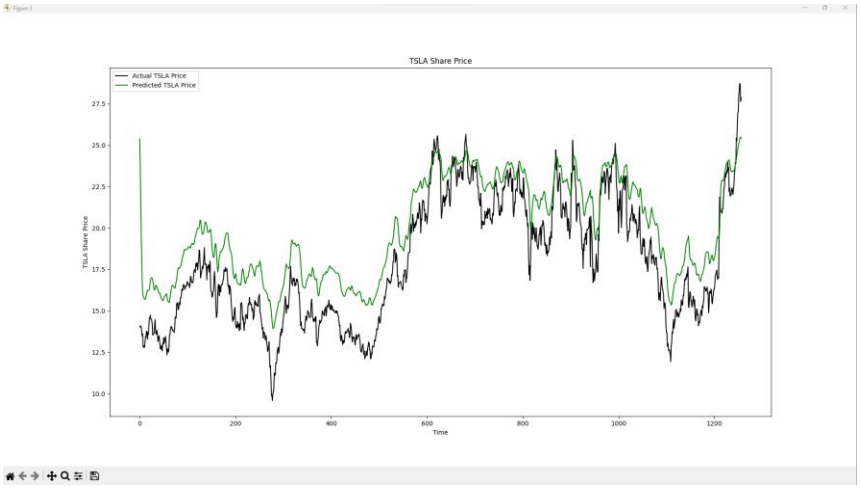
RNN

layer units = 50, epoch = 25



RNN predicted a promising result which is pretty accurate with such low layer units and epoch counts. This result signified the model adeptness in capturing the patterns of the data without becoming overly complicated that will lead to overfitting. The low count of epoch and layer units prevents the model from memorizing the training data and instead focuses on identifying fundamental patterns which resulted in a really close and accurate prediction.

Layer units = 250, epoch = 50



Right after the increase of layer units and epoch the predictions are way above the actual price. This is due to overfitting. Overfitting means that when a model overly specialized in the training data and instead of generating they start to memorize the data sets. The extra layer units and epoch allowed the model to capture noise and randomness that resulted in overfitting. This is to show that not always

having high number/large amount of layer units and epoch will necessarily result in better prediction accuracy.

For more information about the top 10 deep learning algorithms:

https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm