Project B

Task 4 – Machine Learning 1

Write a function that takes as input several parameters including the number of layers, the size of each layer, the layer name and return a Deep Learning model.

Before creating the Deep Learning model creation function, I had to set the variables that would be used in it.

```
N_LAYERS = 2
# LSTM cell
CELL = LSTM
# 256 LSTM neurons
UNITS = 256
# 40% dropout
DROPOUT = 0.4
# whether to use bidirectional RNNs
BIDIRECTIONAL = True

LOSS = "mae"
OPTIMIZER = "adam"
BATCH_SIZE = 64
EPOCHS = 500

LOOKUP_STEP = 15
N_STEPS = 50
```

Then I took inspiration from P1 and implemented a create_model function:

```
def create_model(sequence_length, n_features, units=256, cell=LSTM, n_layers=2,
dropout=0.3,loss="mean_absolute_error", optimizer="rmsprop", bidirectional=False):
    model = Sequential()

    # Add Input layer
    model.add(Input(shape=(sequence_length, n_features)))

    for i in range(n_layers):
        if i == 0:
            # first layer
            if bidirectional:
                model.add(Bidirectional(cell(units, return_sequences=True)))
            else:
                model.add(cell(units, return_sequences=True))
        elif i == n_layers - 1:
            # last layer
            if bidirectional:
                model.add(Bidirectional(cell(units, return_sequences=False)))
```

```
            else:
                model.add(cell(units, return_sequences=False))
        else:
            # hidden layers
            if bidirectional:
                model.add(Bidirectional(cell(units, return_sequences=True)))
            else:
                model.add(cell(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


model = create_model(N_STEPS, len(FEATURE_COLUMN), loss=LOSS, units=UNITS,
cell=CELL, n_layers=N_LAYERS, dropout=DROPOUT, optimizer=OPTIMIZER,
bidirectional=BIDIRECTIONAL)
```

This function works as so:
Creates an initial input layer that defines the sequence length and number of features using parameters. The sequence length is set by N_STEPS, this defines how many days the model uses to make its next prediction. E.g. if N_STEPS is 30, then the the model will use the last 30 days of data to make a prediction. N_features is the number of feature columns used. This needs to be set to define the shape of the input, the data that this model trains on has to match this same input shape.

After setting the Input layer, the next step is add the rest of the model layers. The only difference between setting the first, hidden, and last layers is the state of return_sequences. If it is the last layer then the return_sequence is false, this means that it only returns just the final ouput. For the first and middle, it returns the whole sequence, this is because it needs to pass everything to the next layer.

The next section of the create_model function adds a dropout after each layer, this is to prevent overfitting.

Then after every layer is created, a final output layer is added. In this instance, the output function is linear, which means that the activation function does not modify the outputs. Other activation functions include, ReLU and Sigmoid.

The model is then compiled.

After creating the model, it is then trained on the training dataset.

```
model_name = f"{DATE_NOW}_{COMPANY_TICKER}-{START_DATE}-{END_DATE}-{TEST_SIZE}-
{NAN_HANDLER}\
{LOSS}-{OPTIMIZER}-{CELL.__name__}-seq-layers-{N_LAYERS}-units-{UNITS}"
if BIDIRECTIONAL:
    model_name += "-b"

# some tensorflow callbacks
checkpointer = ModelCheckpoint(os.path.join("results", model_name + ".weights.h5"),
save_weights_only=True, save_best_only=True, verbose=1)
tensorboard = TensorBoard(log_dir=os.path.join("logs", model_name))
# train the model and save the weights whenever we see
# a new optimal model using ModelCheckpoint

history = model.fit(X_train, y_train,
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS,
                    validation_data=(X_test, y_test),
                    callbacks=[checkpointer, tensorboard],
                    verbose=1)
```

This function is identical to the one from P1. The first section is just related to the naming of the model. The second section is where the model is trained. The variables checkpointer is the location where the most successful model is saved as a checkpoint, the model is only ever saved if it is a statistically improved version. The tensorboard variable defiens a TensorBoard object that allows viewing of the training whilst it is happening.

The model is then fit to the dataset with assigned epochs and batch size. Batch size is the size of chunks that the overall dataset is split into. For example, if the dataset as a whole has 128 data points and the batch size is 64, then the dataset would be split into two batches of 64.

Epochs is how many times the model will iterate over the whole dataset; 1 epoch is when the model has seen the entire dataset once. This is a very important hyperparameter, as too few epochs could lead to underfitting where the model hasn't learned enough from the data, but too many epochs can cause overfitting, where the model has learned to much from the data and cannot generalise on new data.

Use the above function to experiment with different DL network (LSTM, RNN, GRU) and with different hyperparameter configuration (number of layers, layer size, epochs, batch size.)

The hyperparameters are as follows:

```
N_LAYERS = 2
# type of recurrent unit
CELL = LSTM
# Amount of neurons per layer
UNITS = 256
# 40% dropout
DROPOUT = 0.4
# whether to use bidirectional RNNs
BIDIRECTIONAL = False

LOSS = "mae"
OPTIMIZER = "adam"
BATCH_SIZE = 64
EPOCHS = 100

N_STEPS = 120 # How many days of past data the model uses to make a prediction
LOOKUP_STEP = 120 # How far into the future the model is predicting
```

N_Layers: How many layers the model has. Does not include the input and output layers.

CELL: The type of recurrent unit. Can be LSTM, GRU, SimpleRNN, or others.

UNITS: The number of neurons in each layer.

DROPOUT: The fraction of neurons that are randomly dropped during training to prevent overfitting.

BIDIRECTIONAL: Whether the layers should be bidirectional. E.g., if True, each layer processes the sequence both forward and backward. If False, it only processes the sequence in the forward direction.

LOSS: The loss function to evaluate the model's performance.

OPTIMIZER: The algorithm to adjust the model's weight during framing.

BATCH_SIZE: The number of samples processed before the model updates its weights.

EPOCHS: The number of complete passes through the training dataset.

N_STEPS: The number of past days the model uses to make its prediction.

LOOKUP_STEP: How far into the future the model is predicting.

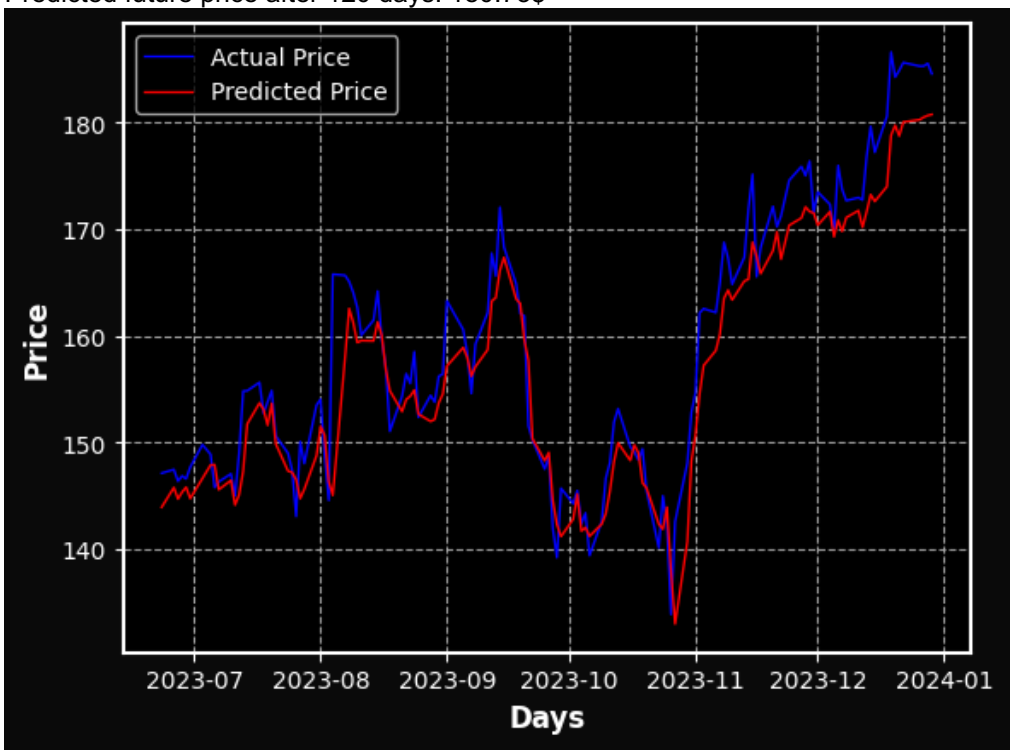
Below are some trained models and their statistics with different hyperparameters.

```
N_LAYERS = 2
# LSTM cell
CELL = LSTM
# 256 LSTM neurons
UNITS = 256
# 40% dropout
DROPOUT = 0.4
# whether to use bidirectional RNNs
BIDIRECTIONAL = False

LOSS = "mae"
OPTIMIZER = "adam"
BATCH_SIZE = 64
EPOCHS = 100

N_STEPS = 120 # How many days of past data the model uses to make a prediction
LOOKUP_STEP = 120 # How far into the future the model is predicting
```

Loss: 0.027265217155218124
Mean Absolute Error: 0.027265217155218124
Accuracy Score: 0.7424242424242424
Total Buy Profit: 87.16813332772256
Total Sell Profit: 135.94992437197772
Total Profit: 223.11805769970027
Profit Per Trade: 1.6902883159068203
Predicted future price after 120 days: 180.73$

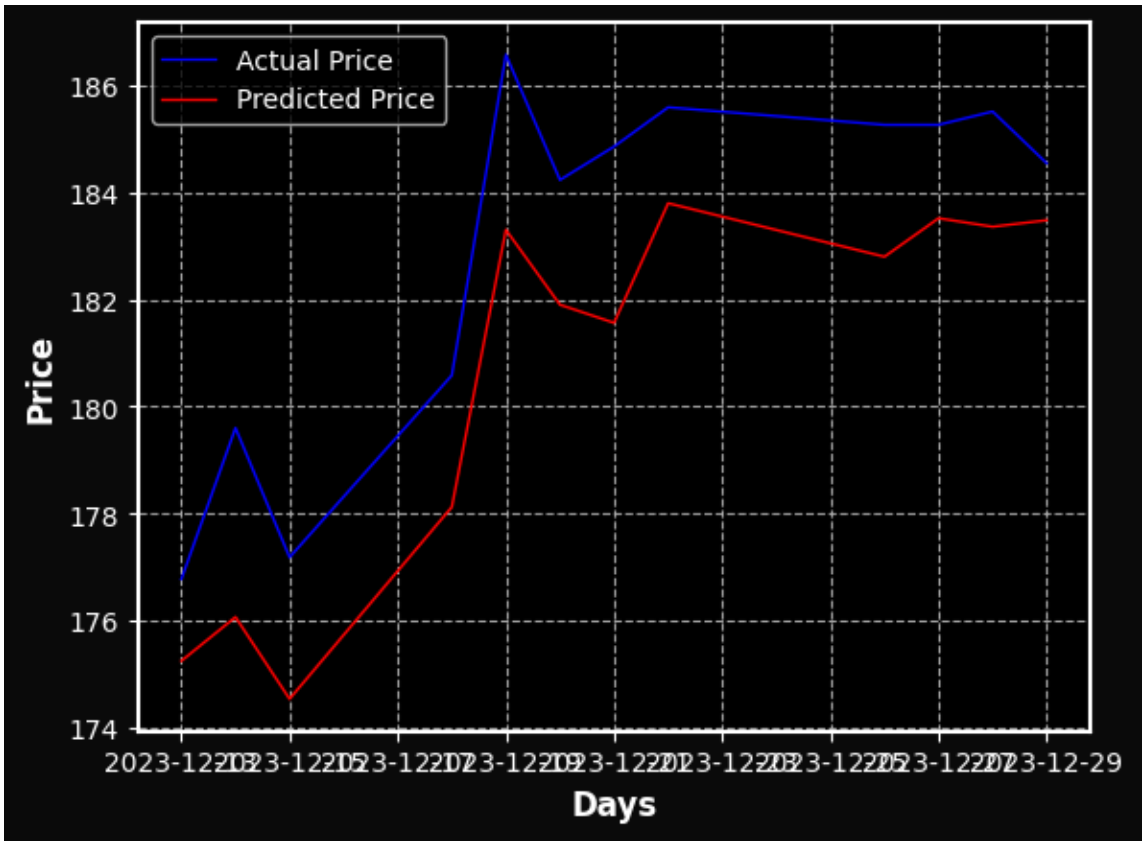


```
N_LAYERS = 2
# LSTM cell
CELL = LSTM
# 256 LSTM neurons
UNITS = 256
# 40% dropout
DROPOUT = 0.4
# whether to use bidirectional RNNs
BIDIRECTIONAL = False

LOSS = "mae"
OPTIMIZER = "adam"
BATCH_SIZE = 64
EPOCHS = 200

N_STEPS = 240 # How many days of past data the model uses to make a prediction
LOOKUP_STEP = 240 # How far into the future the model is predicting
```

```
Loss: 0.02112419903278351
Mean Absolute Error: 0.02112419903278351
Accuracy Score: 0.5833333333333334
Total Buy Profit: 1.3216303633145117
Total Sell Profit: 7.5745588474331385
Total Profit: 8.89618921074765
Profit Per Trade: 0.7413491008956375
Predicted future price after 240 days: 183.48$
```

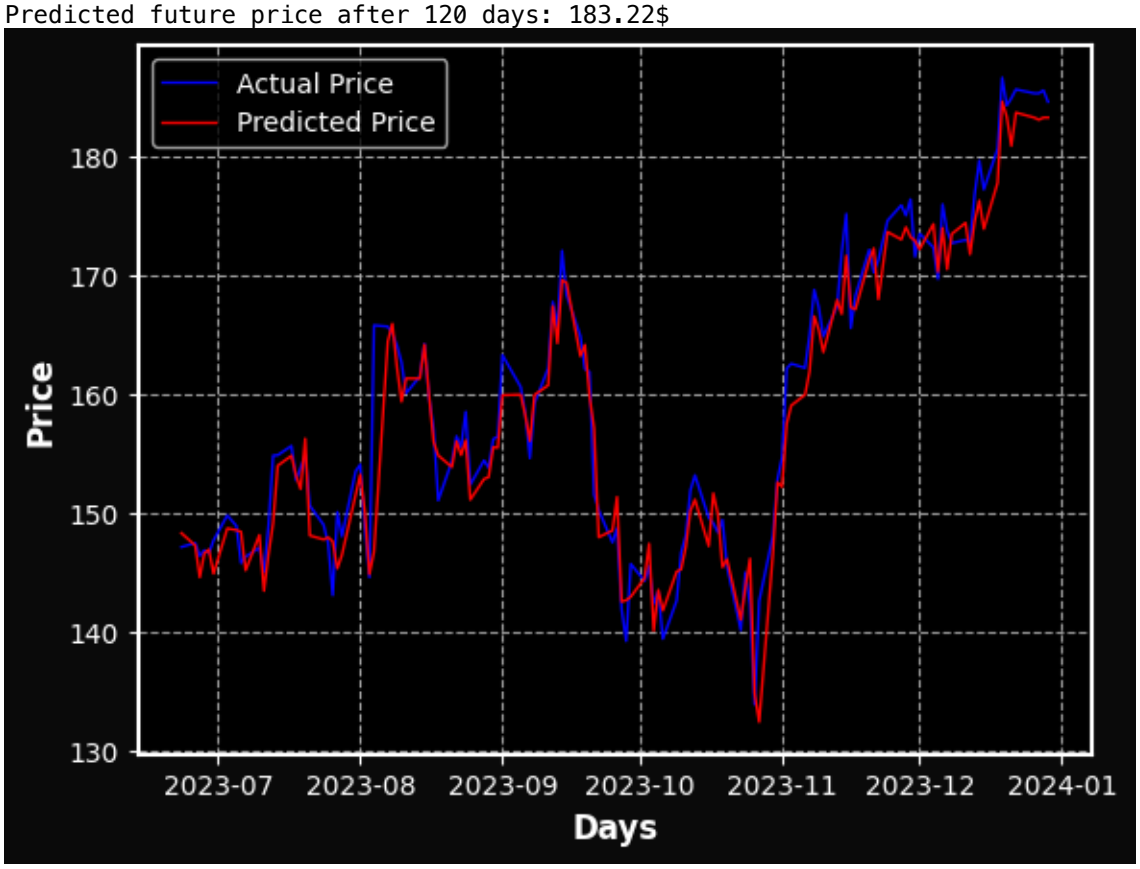


```
N_LAYERS = 2
# LSTM cell
CELL = GRU
# 256 LSTM neurons
UNITS = 256
# 40% dropout
DROPOUT = 0.4
# whether to use bidirectional RNNs
BIDIRECTIONAL = False

LOSS = "mae"
OPTIMIZER = "adam"
BATCH_SIZE = 64
EPOCHS = 200

N_STEPS = 120 # How many days of past data the model uses to make a prediction
LOOKUP_STEP = 120 # How far into the future the model is predicting
```

```
Loss: 0.017783349379897118
Mean Absolute Error: 0.017783349379897118
Accuracy Score: 0.7651515151515151
Total Buy Profit: 94.57725399578561
Total Sell Profit: 143.35904504004077
Total Profit: 237.93629903582638
Profit Per Trade: 1.8025477199683817
Predicted future price after 120 days: 183.22$
```



```python
N_LAYERS = 2
# LSTM cell
CELL = SimpleRNN
# 256 LSTM neurons
UNITS = 256
# 40% dropout
DROPOUT = 0.4
# whether to use bidirectional RNNs
BIDIRECTIONAL = False

LOSS = "mae"
OPTIMIZER = "adam"
BATCH_SIZE = 64
EPOCHS = 200

N_STEPS = 120 # How many days of past data the model uses to make a prediction
LOOKUP_STEP = 120 # How far into the future the model is predicting
```

```
Loss: 0.0910729169845581
Mean Absolute Error: 0.0910729169845581
Accuracy Score: 0.5303030303030303
Total Buy Profit: -7.97847766635428
Total Sell Profit: 40.80331337790088
Total Profit: 32.8248357115466
Profit Per Trade: 0.24867299781474694
Predicted future price after 120 days: 174.80$
```