Garrich Farrell Hardjojuwono 102764065

Machine Learning 1 Report

```python
def buildModel(x_train_data, y_train_data, no_steps, no_features, units=256, cell=LSTM, no_layers=2, dropout=0.3, loss='mean_absolute_error',
               optimizer="rmsprop", no_epochs=25, size_batch=32):
    # Use the sequential model type
    model = Sequential()
    print(cell)
    # a for loop to add layers based on the specified number of layers
    for i in range(no_layers):
        if i == 0:
            # The first layer, add a layer with return_sequences=True
            # batch_input_shape helps determine the input shape as in the number of inputs
            # the format of batch_input_shape is supposed to be
            # batch_input_shape(batch_size, number of steps and number of features)
            model.add(cell(units, return_sequences=True, batch_input_shape=(None, no_steps, no_features)))
        elif i == no_layers - 1:
            # The last layer, add a recurrent layer with return_sequences=False. This if false because we want
            # one single final output instead of multiple, return_sequences can be true depending on the use
            # case.
            model.add(cell(units, return_sequences=False))
        else:
            # The layers in between first and last, add a layer with return_sequences=True
            model.add(cell(units, return_sequences=True))

        # a dropout layer is introduced, the function of this is to prevent overfitting
        # if a dropout rate of 0.2 that means that 20% of the data in this layer will be
        # dropped out or set to 0.
        model.add(Dropout(dropout))

    # Add a Dense layer with one output unit and linear activation for regression,
    # linear activation, produces an output that is directly proportional to the input.
    model.add(Dense(1, activation="linear"))

    # Compile the model with specified loss, metrics, and optimizer
    model.compile(loss=loss, metrics=['mean_absolute_error'], optimizer=optimizer)

    model.fit(x_train_data, y_train_data, epochs=no_epochs, batch_size=size_batch)

    # Return the constructed model
    return model
```

The screenshot above contains the code which has been return to function with my code. The code has been referenced from the example project in P1. I have understood the steps that was taken in P1 and used it in this development of the stock prediction v0.3. I have not included the need for the Bidirectional step in the model as it was not a part of the requirements for this task. Some research was done to figure out what the return_sequences and batch_input_shape parameter does.

```python
model.add(cell(units, return_sequences=True, batch_input_shape=(None, no_steps, no_features)))
```

To summarize what the two parameters does is, when return_sequences is true it allows for the returning of more than one output into the next layer whereas if it is set to false, there is only one single output variable. Therefore, for the use case of this project the system only needs to output one value which is the value of the stock price at closing which is why the final layer has return_sequences set to false. An example of a use case where return_sequences is set to true is ChatGPT. ChatGPT outputs a sequence of characters which form sentences this would require the model to return a sequence rather than an individual variable or value.

As for the batch_input_shape, it helps determine the input shape or dimension meaning number of inputs.

There is a dropout parameter which is also used which in short determines the percentage of the overall data to be dropped from the dataset during that layer. In the context of the code above, after the final layer, 20% of the data is dropped by default unless set otherwise by the user within the parameters.py file.

```
175        model.add(Dropout(dropout))
```

In the parameters.py file the other parameters which the user can configure has also been added.

```
# ================================================================================

# Building The Model
unitSize = 256
numberOfLayers = 4
Optimizer = 'adam'
Loss = 'mean_squared_error'
# Deep Learning Type
DLType = LSTM
# Dropout Rate
DORate = 0.2
numberOfSteps = 100

# ================================================================================

# ================================================================================

# do you want to predict?
predict = True
Epochs = 50
BatchSize = 32

# ================================================================================
```

After setting the code up some experiments were conducted.

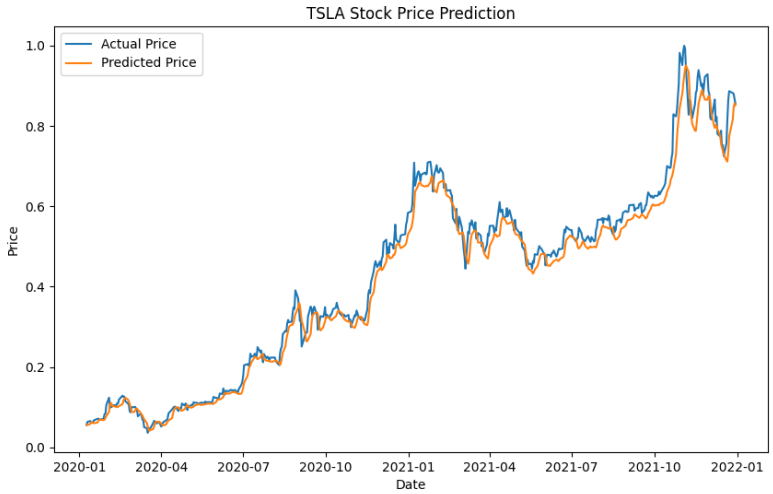
Garrich Farrell Hardjojuwono 102764065

In terms of the experiments, I have conducted using this model. I decided to alter the Deep Learning Type and the number of layers. In the experiment I used two different models: LSTM and GRU with Epochs Set to 50 on all of the tests. The number of layers used are 2, 4, and 6.

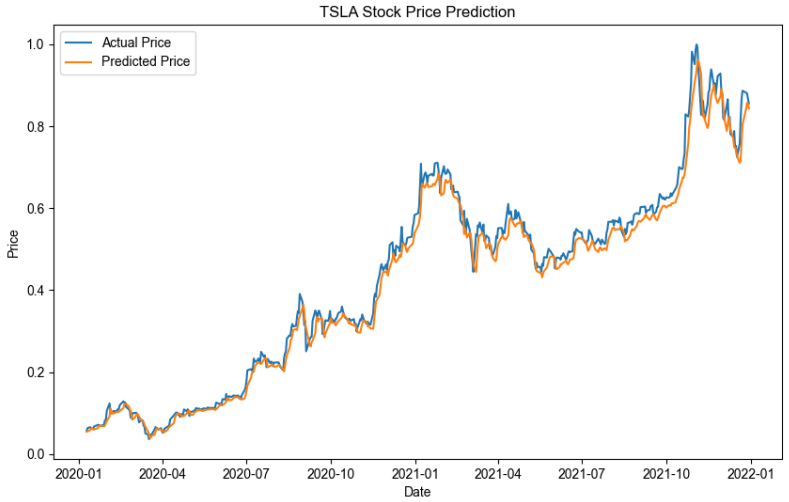
Below I have attached screenshots of the results.

2 Layers

LSTM



GRU



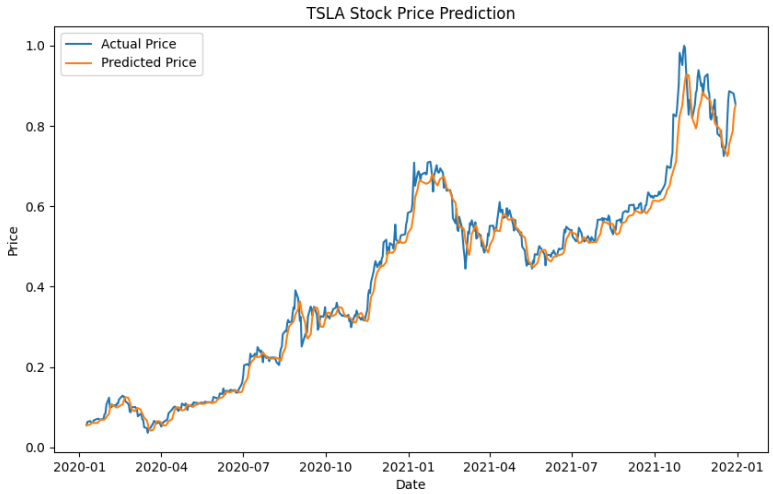
Observation

From this two-layer model, the graph is already functioning quite decently, with the predicted price closely following the actual price. When comparing the models, it seems that the GRU model performs better than the LSTM model as the line from the model when compared to the actual price follows more closely than the LSTM model, it seems that the GRU model reacts better to the fluctuation and changes in price whereas LSTM seems to smoothen the graph more.

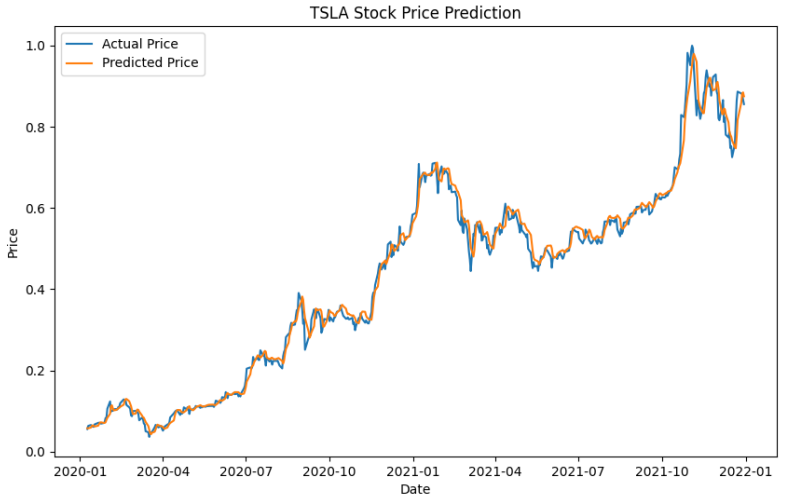
Garrich Farrell Hardjojuwono 102764065

4 Layers

LSTM



GRU



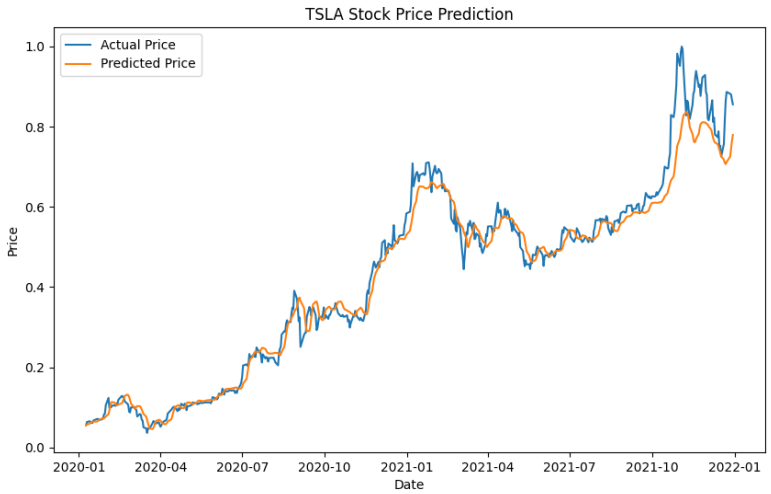
Observation

Now with an increase of two layers in the model, creating a four-layer model. This model has by far created the most accurate line when compared to the actual prices. With the GRU model again outperforming the LSTM model in terms of following the actual price line. The line of the predicted graph from the GRU model, as can be seen on the graph, the predicted line is basically on the same line as the actual price line.

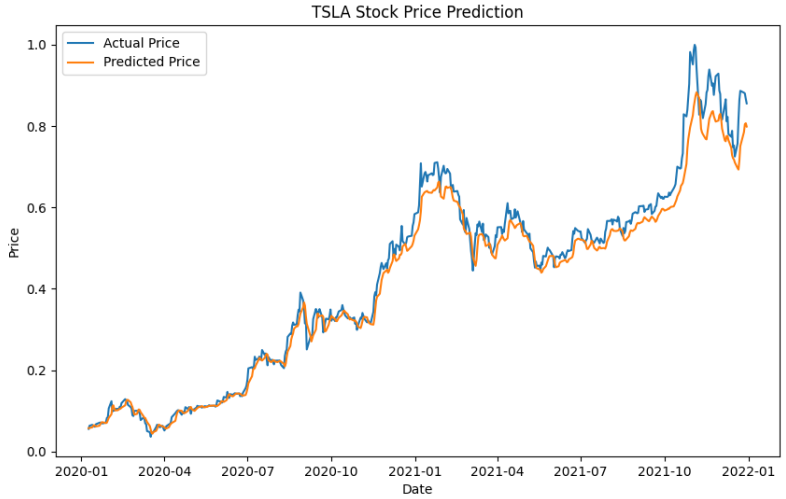
Garrich Farrell Hardjojuwono 102764065

6 Layers

LSTM



GRU



Observation

As can be seen in this graph, once an additional two layers is added for some reason the predicted price is unable to match the actual price as accurately as the models which was made in the 4 Layers. The model seems to lose the accuracy it has.

Garrich Farrell Hardjojuwono 102764065

References

Cloud, S. (2023, August 25). Understanding the batch_input_shape tuple in Keras LSTM for data scientists. Saturn Cloud Blog. https://saturncloud.io/blog/understanding-the-batchinputshape-tuple-in-keras-lstm-for-data-scientists/