```
def create_model(x_train, y_train, epochs, batch_size, units=256,
cell=LSTM, n_layers=2):
    model = Sequential()
    for _ in range(n_layers):
        model.add(cell(units=units, return_sequences=True,
input_shape=(x_train.shape[1], 1)))
        model.add(Dropout(0.2))

    model.add(cell(units=units))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))
    model.compile(optimizer='adam', loss='mean_squared_error')
    model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size)
    return model
```

function to create the model. Loops through the number of layers till each is added then compiles and returns the model to be tested.

As a base I started with
epochs = 50

batch_size = 25

units = 256

cell = LSTM

layers = 2

these where taken from P1 and seemed like a good starting point.

for each test I only changed 1 parameter to test

my first tests where on cell using LSTM, GRU and SimpleRNN

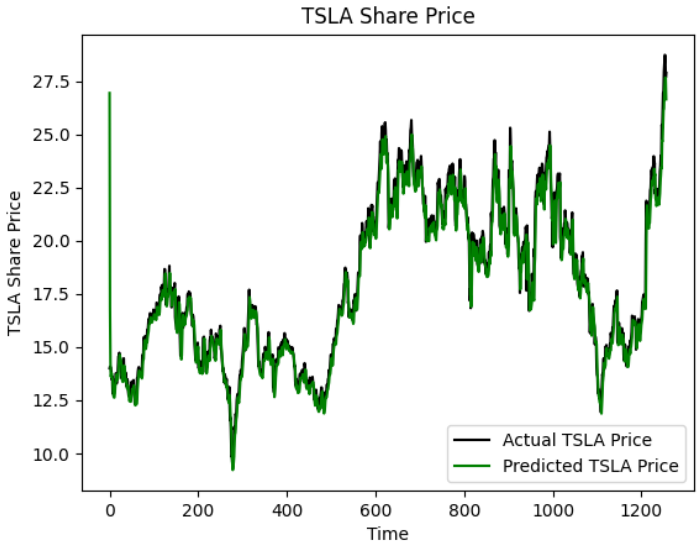

Figure 1 LSTM

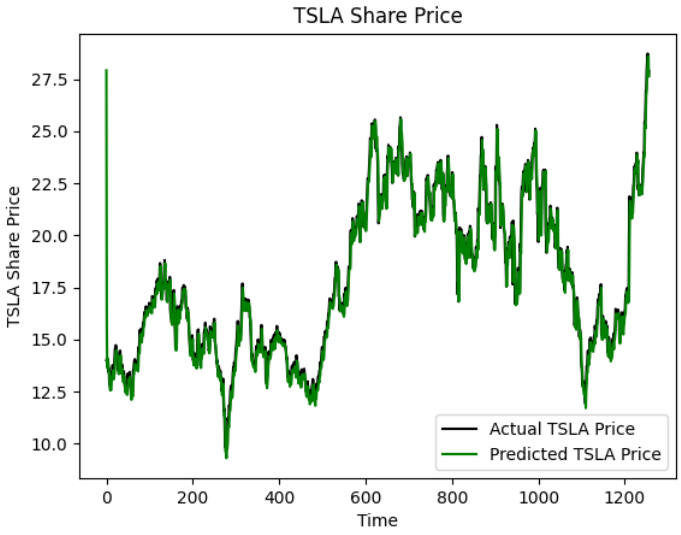

Figure 2 GRU

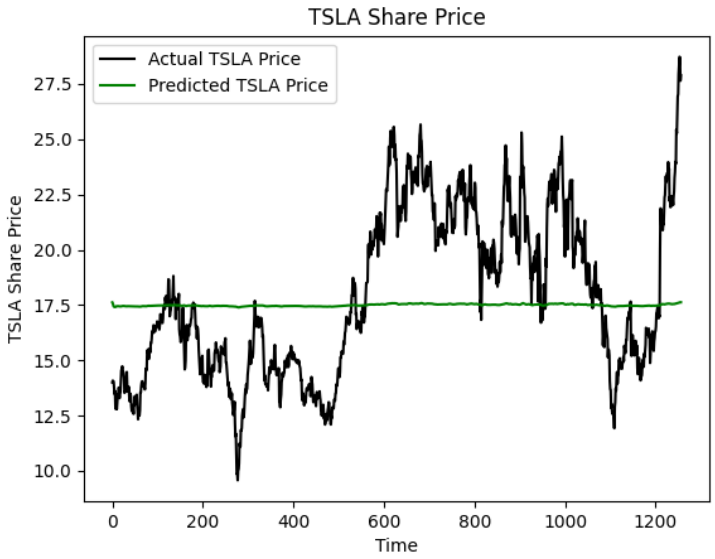

Figure 3 SimpleRNN

From the Cell types with these configurations, GRU seems to be the most accurate. SimpleRNN seems to want different parameters then LSTM and GRU. Because GRU appears to be the closest at the moment, GRU will be used as the cell type for the remaining parameters.

as for the remaining parameters I will double and half the value of each meaning I will run tests for the following values

test epochs = 25/100

batch_size = 12/50 (25/2 isnt perfect rounded down to 12 arbitrarily)

units = 128/512

layers = 1/4

EPOCH, this is the amount of times the model sees the test data, (25 epoch sees the test data 25 diff times, epoch of 5 sees it 5 times)

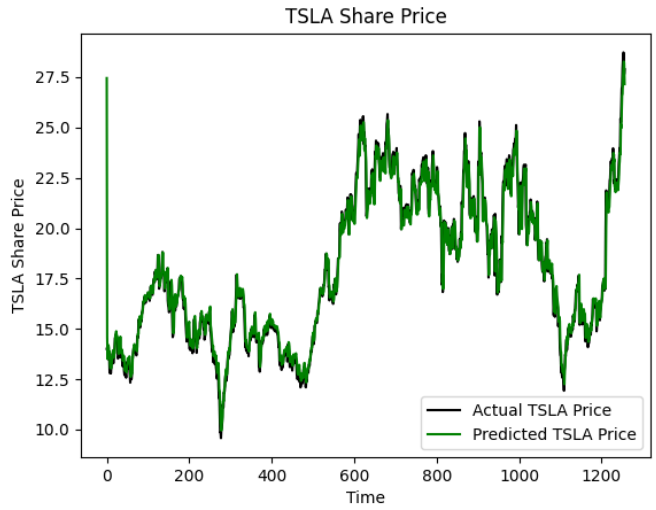

Figure 4 25 epoch

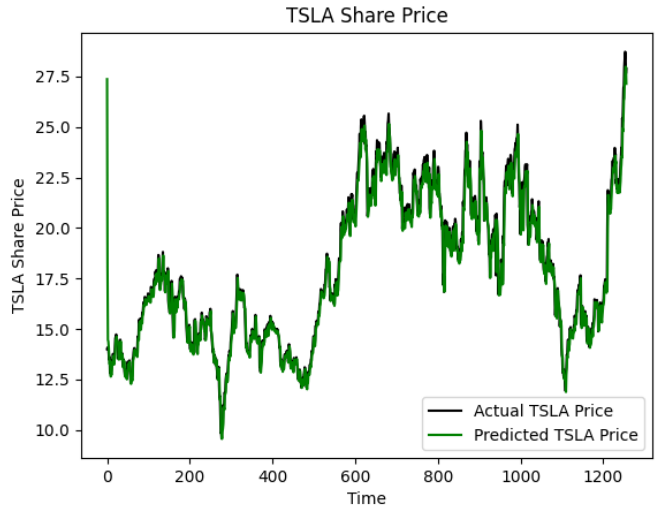

Figure 5 100 epoch

The 2 graphs are pretty similar, both look less accurate then 50 epoch. 25 appears to be slightly over predicting and 100 epoch is under predicting. It is worth noting that the runtime is closely related to the number of epoch so 25 takes half as long as 50 which runs half as long as 100. Due to the amount of training cycles the model goes through.

Batch size, is the amount of data points the model sees at one time.

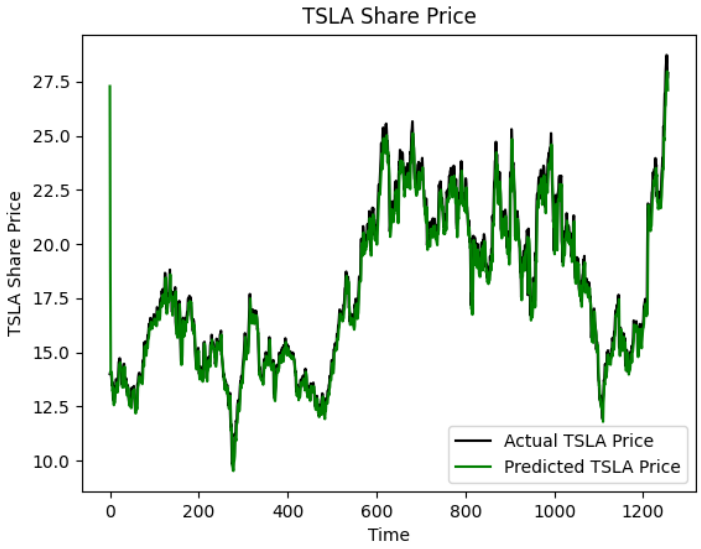

Figure 6 12 batch

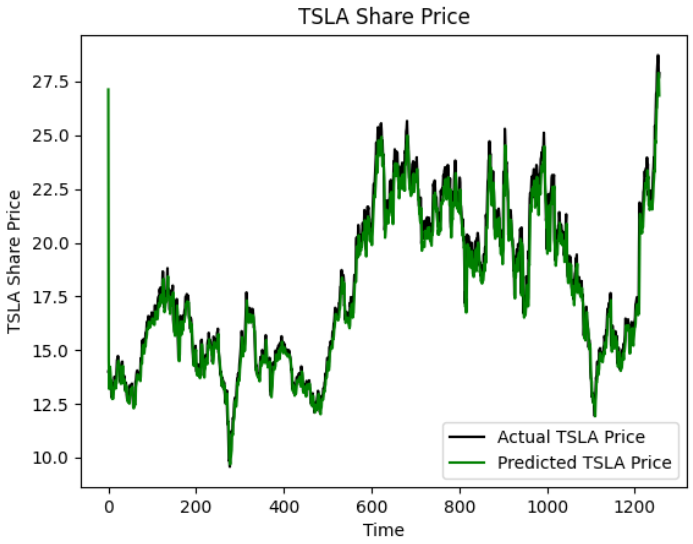

Figure 7 50 batch

Overall less accurate then 25 batch size. Which is interesting as smaller batch sizes should capture more nuance. But too small could over fit the data.

Units, adjust the number of “neurons in each cell”

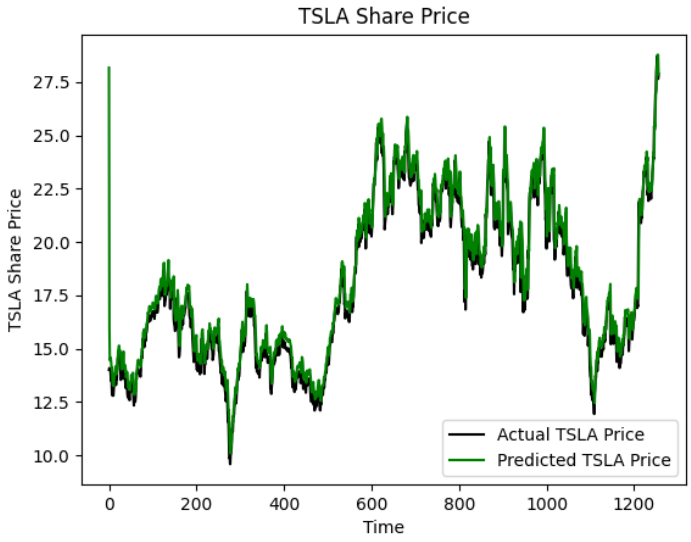

Figure 8 128 units

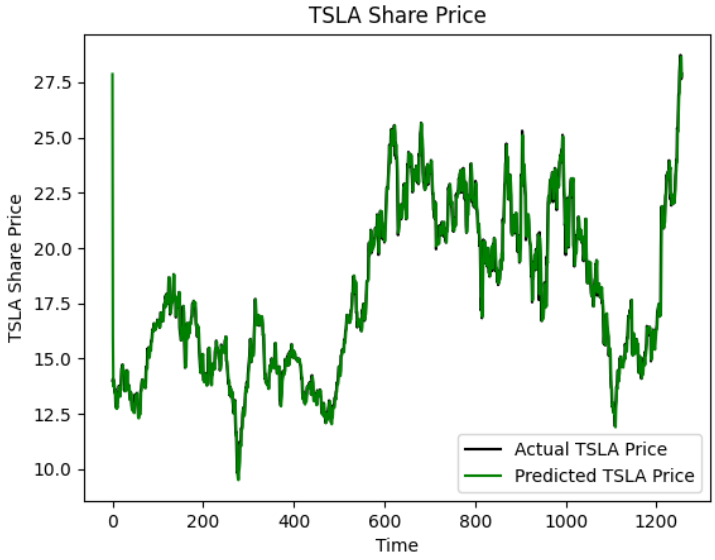

Figure 9 512 units

The accuracy of the model increases as the amount of units increase, likey due to more connections being able to find nuances in the data, could also mean that too many units could cause patterns to be detected that don’t exist. also takes longer to train the more units you have.

Layers determine the architecture of the model and how cells communicate.

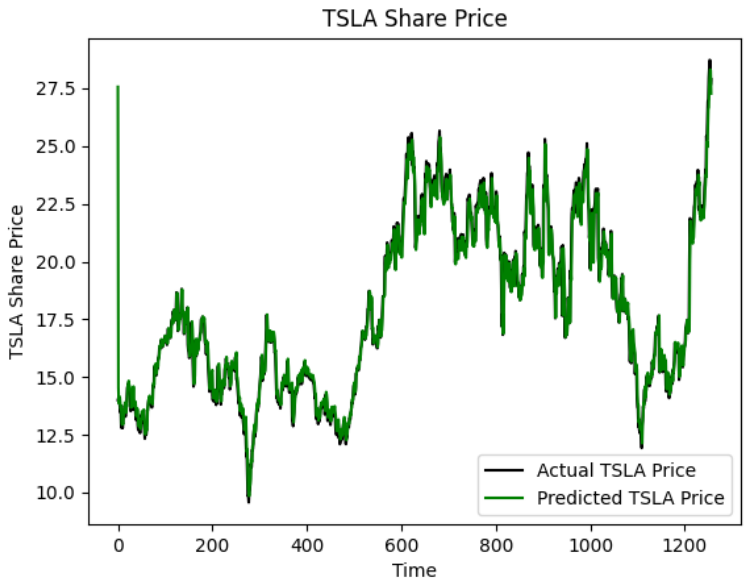

Figure 10 1 layer

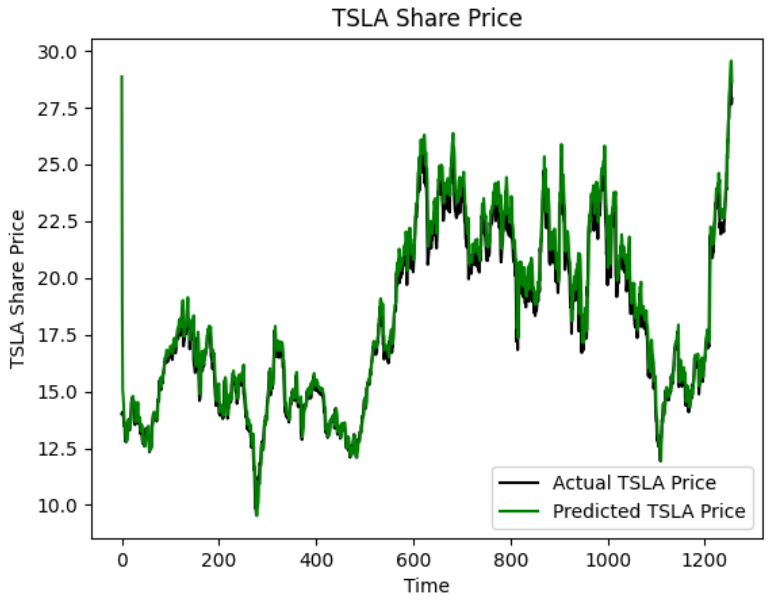

Figure 11 4 layers

1 layer is similar in accuracy to 2 layers, and 4 is noticeably more inaccurate. Most likely due to this model not needing much complexity, or the model becoming messier and harder to train as the number of layers increase.

Overall the default parameters from P1 seem to be pretty accurate. GRU, in my test seemed to be more accurate however. Aside from increasing the units for accuracy, P1s parameters fit a good balance of accuracy and training time. Tuning all these values would probably require using a genetic algorithm to test combinations.