COS30018 Assignment B – Task 4

Aidan Grimmett: 103606838 – Friday 12:30 class

This week I created a function which takes in the desired type of deep learning network (LSTM, GRU, RNN), amount of layers, layer size and dropout amount. It will create a new keras sequential model of n layers and compile it to use for training.

The type of DL network is parsed into the function as a string, so the first thing that the function contains is a dictionary to map a string into the correct keras layer class. If the 'model_type' parsed to the func is not recognised, then the default of LSTM will be used. Then a new sequential model will be created. From there, the function will iterate through a loop 'n_layers' – 1 times, adding a new layer to the model on each iteration. Note that on the first iteration, the layer added includes the input data as 'input_shape'. The subsequent layers do not need this as they receive the data from the previous layers. Once that is done, the final layer is added which has 'return_sequences' set to false as there are no more layers for it to return the data to. Lastly, the function defines loss and optimizer values, the output layer with a single neuron and compiles the model to return to the main program.

See screenshot of code for specifics on the implementation with line by line commenting. Output results for a variety of settings have also been included. LSTM seemed to be the most effective network, but it was sensitive to overtraining and performed worse with an excessive amount of layers.

CreateCustomModel.py 2, U

CreateCustomModel.py > ...

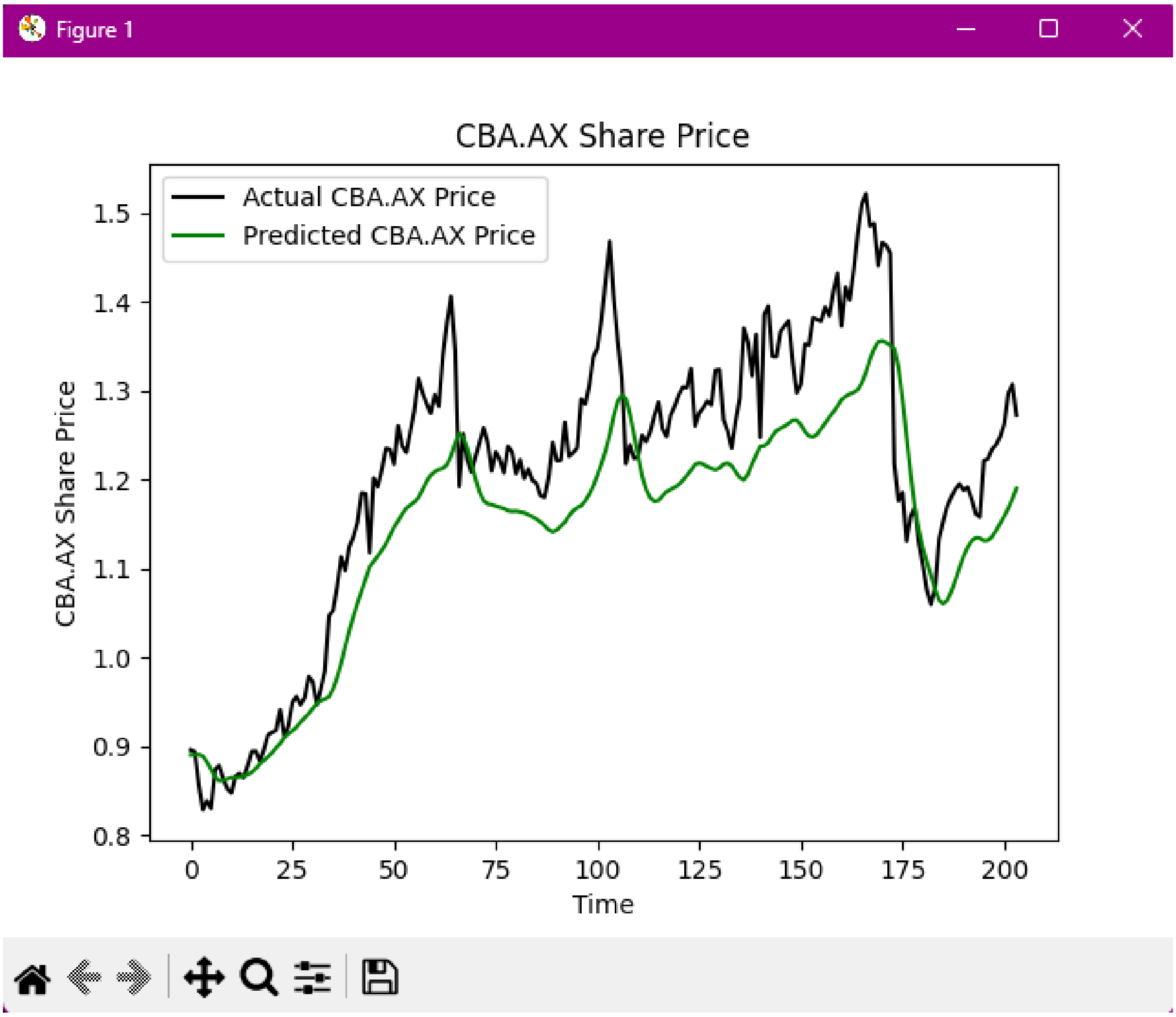
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, SimpleRNN, GRU

def MakeCustomModel(model_type, input_data, n_layers, units, dropout):

    #dictionary to map model_type to the appropriate Keras class
    model_layer_dict = {
        'LSTM': LSTM,
        'RNN': SimpleRNN,
        'GRU': GRU
    }

    #if not valid model type is parsed, select LSTM as default
    if (model_type not in model_layer_dict):
        model_type = LSTM


    #create a blank sequential model that we will add layers to
    model = Sequential()

    for i in range(n_layers - 1):
        if (i == 0):
            #Add first layer, specify input shape which is not needed in future layers
            model.add(model_layer_dict[model_type](units, return_sequences=True, input_shape = input_data))
        else:
            #add the rest of the layers
            model.add(model_layer_dict[model_type](units, return_sequences=True))

        # add dropout on each layer, prevents overfitting by randomly dropping some neurons
        model.add(Dropout(dropout))


    # add final layer, dont need to return sequences as this is the output layer
    model.add(model_layer_dict[model_type](units, return_sequences=False))

    #specify loss function, mean absolute error punishes severe miscalculations
    loss = "mean_absolute_error"
    #optimizer minimizes the loss and updates the weights
    optimizer="rmsprop"
    # add output layer
    model.add(Dense(1, activation="linear"))
    #compile the model and specify the loss, optimizer and metrics to track
    model.compile(loss=loss, metrics = [loss], optimizer = optimizer)
    return model #return complete model
```

```python
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
# We now reshape x_train into a 3D array(p, q, 1); Note that x_train
# is an array of p inputs with each input being a 2D array

input_shape=(x_train.shape[1], 1)

model = MakeCustomModel('LSTM', input_shape, 2, 256, 0.3)
```

LSTM, 3 layers, 256 units, 0.3 dropout, 25 epochs

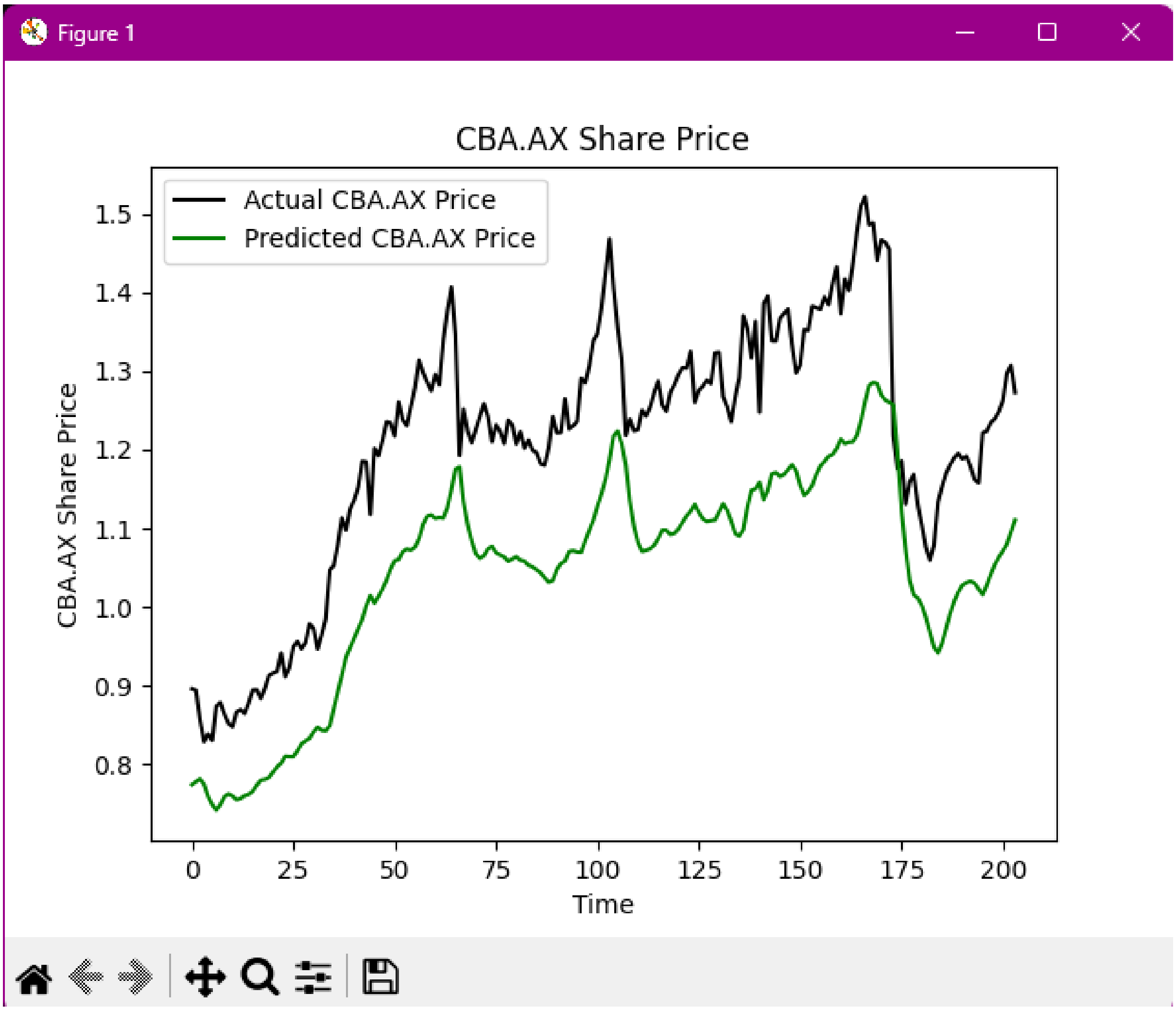
These settings were relatively accurate, but undershooting most of the time.

LSTM, 3 layers, 0.3 dropout, 30 epochs:



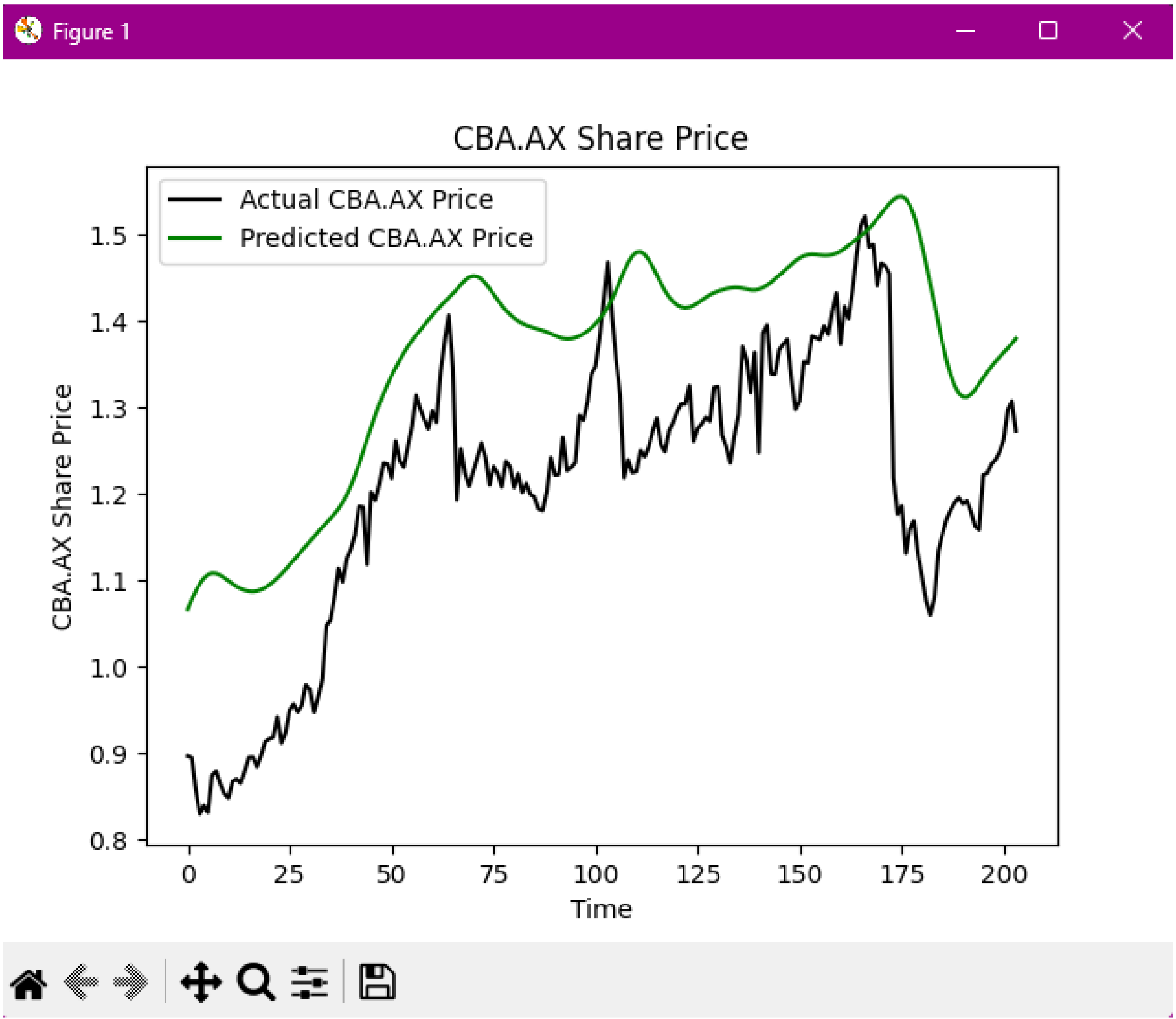
Adding 5 epochs did not have a positive effect.

GRU, 3 layers, 256 units, 0.3 dropout, 25 epochs



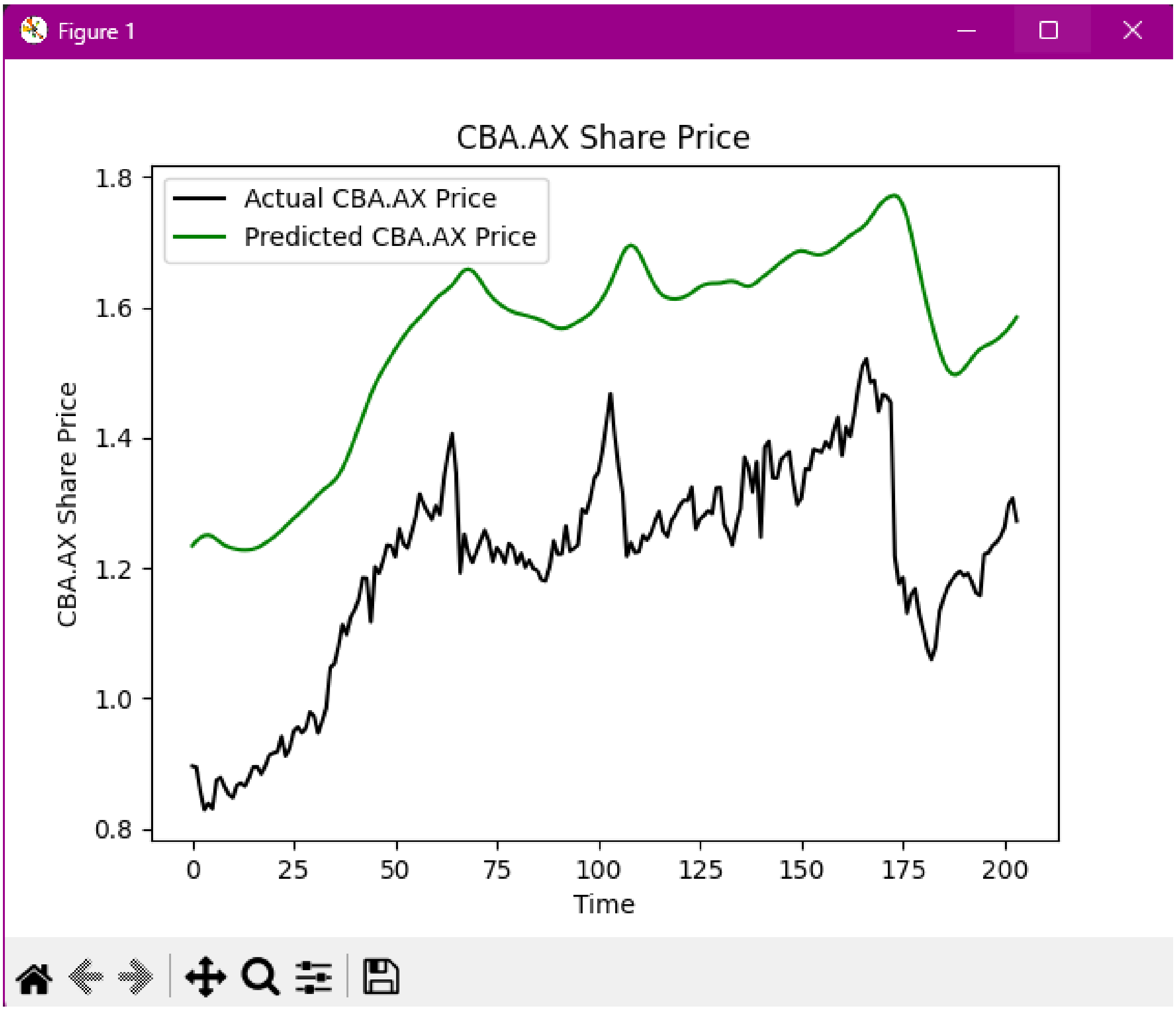
GRU seemed to be less accurate than LSTM.

LSTM, 6 layers, 256 units, 0.3 dropout, 25 epochs



LSTM 6 layers seemed much less accurate than 3 before.

LSTM, 4 layers, 256 units, 0.3 dropout, 25 epochs

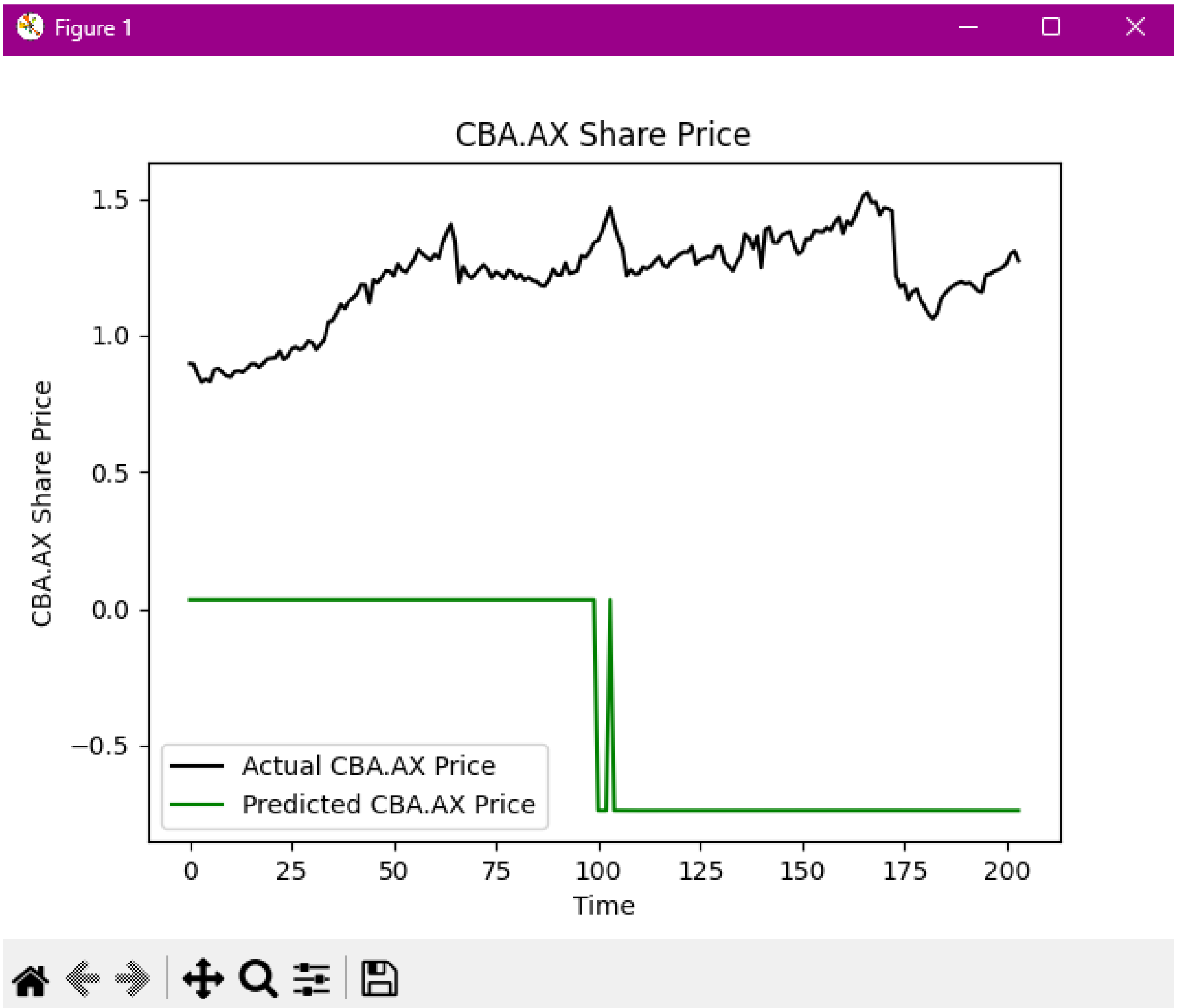


LSTM 4 layers seemed even worse than 6 which is interesting. Not sure why it would get worse after 3 and better at 6.

LSTM, 2 layers, 256 units, 0.3 dropout, 25 epochs

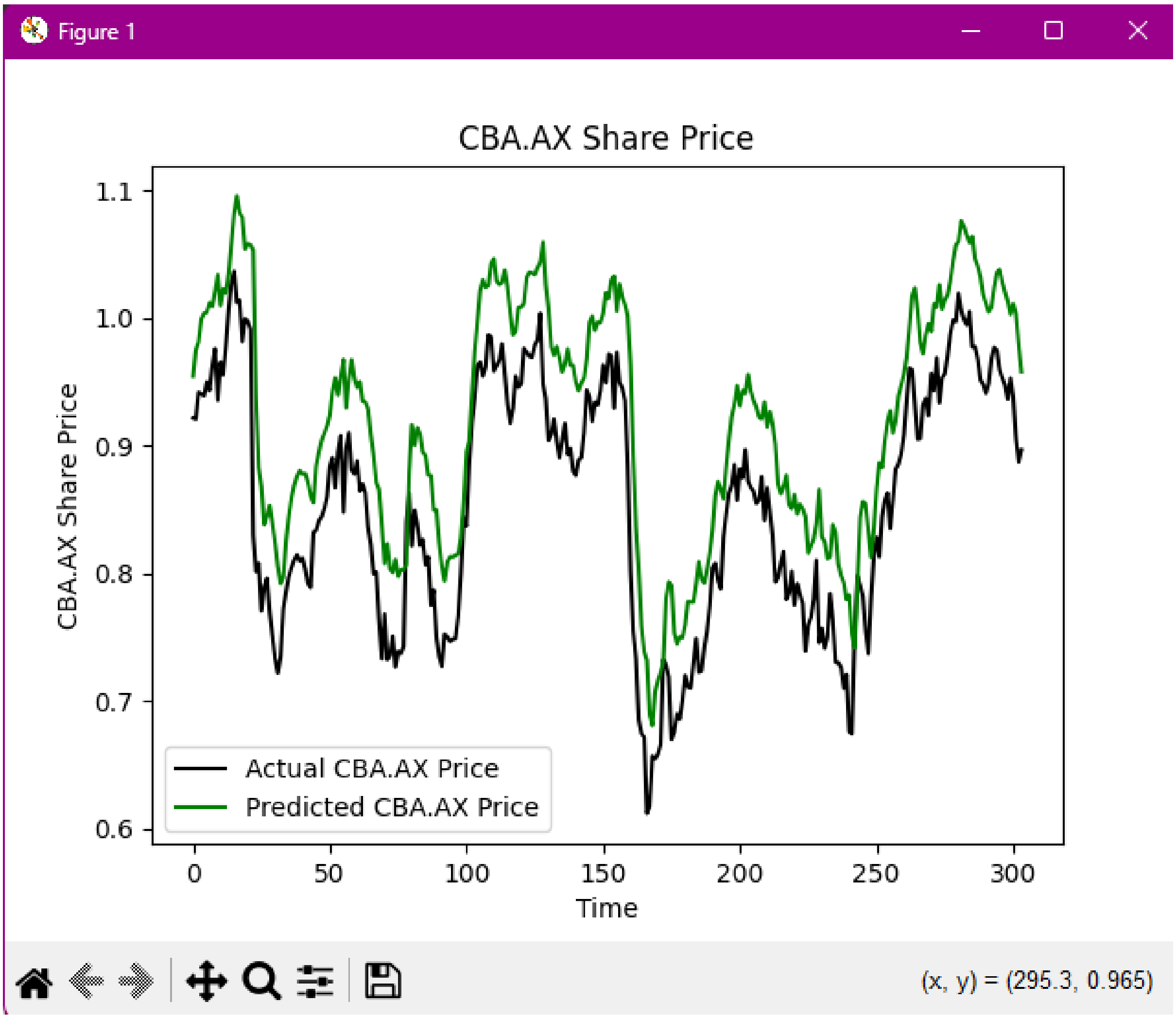
This was by far the best result, with 2 layers and 25 epochs.

RNN, 2 layers, 1024 units, 0.3 dropout, 50 epochs



RNN networks did not seem to be very effective, even with a large layer size and amount of epochs.

LSTM, 2 layers, 1024 units, 0.3 dropout and 50 epochs



LSTM with 2 large layers and 50 epochs was good but not as good as the 2 layer LSTM from before.