HOMEWORK 1

MULTILAYER PERCEPTRONS AND ARIMA MODEL FOR TIME SERIES FORECASTING

BMEN 4470 - Deep Learning for Biomedical Signal Processing

Fall 2021

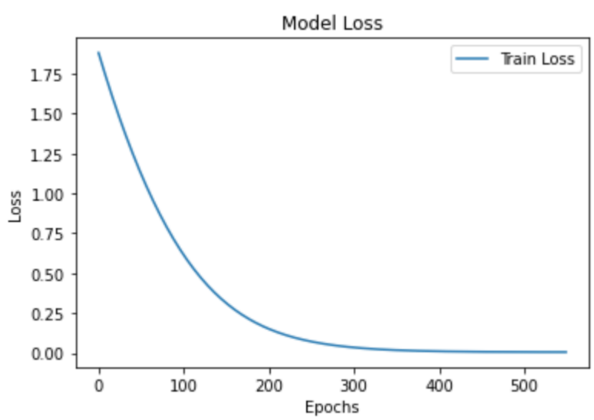
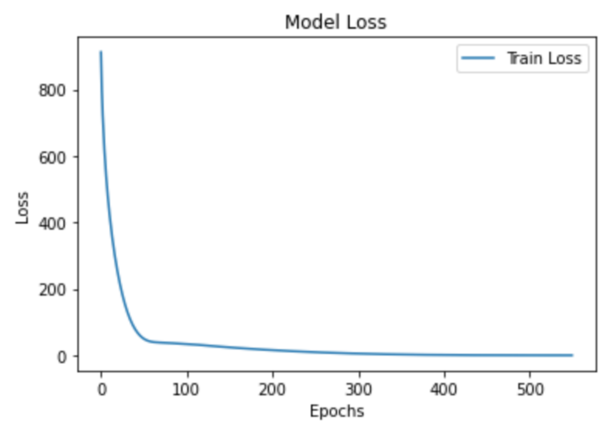
Michelle Campoli

mec2308

Problem 1a.

Accuracy of a perceptron is dependent on the weights selected and whether the model can be linearly or logistically fit. Increasing the number of epochs increases the accuracy of the prediction.

In the table below, a perceptron and a multi-layer perceptron are compared.

	Perceptron	Add 1 hidden layer
Model	Model: "sequential_23" Layer (type) Output Shape Param # === dense_32 (Dense) (None, 1) 4 === Total params: 4 Trainable params: 4 Non-trainable params: 0	Model: "sequential" Layer (type) Output Shape Param # === dense (Dense) (None, 8) 32 dense_1 (Dense) (None, 1) 9 === Total params: 41 Trainable params: 41 Non-trainable params: 0
Loss		

Prediction	#Demonstrate Prediction using model.predict
	x_input = np.array([70, 80, 90]) # predict the output
	x_input = x_input.reshape((1, n_steps))
	y_output = model.predict(x_input)
	print("X=%s, Predicted=%s" % (x_input, y_output))
	
	X=[[70 80 90]], Predicted=[[100.150566]]
		x_input = np.array([70, 80, 90]) # predict the out
		x_input = x_input.reshape((1, n_steps))
		y_output = model.predict(x_input)
		print("X=%s, Predicted=%s" % (x_input, y_output))
		
		X=[[70 80 90]], Predicted=[[101.54166]]

By adding hidden layers, the number of epochs required to minimize the loss decreases.

Problem 1b.

The model from 1a was used to predict the test sequence [1^2, 2^2, 3^2].

```
x_input = np.array([1, 4, 9]) # predict the output
x_input = x_input.reshape((1, n_steps))

y_output = model.predict(x_input)
print("X=%s, Predicted=%s" % (x_input, y_output))
```

```
X=[[1 4 9]], Predicted=[[13.181818]]
```

The model from 1a. does not accurately predict the test sequence [1^2, 2^2, 3^2] 4^2, as shown above. The prediction is 13 and the actual value is 16. This is because the sequence is non-linear, and the model has been trained on data that is linearly separable. The model was also built for a linearly separable dataset as the weights implemented at each layer are the same ones used to optimize linear regression models.

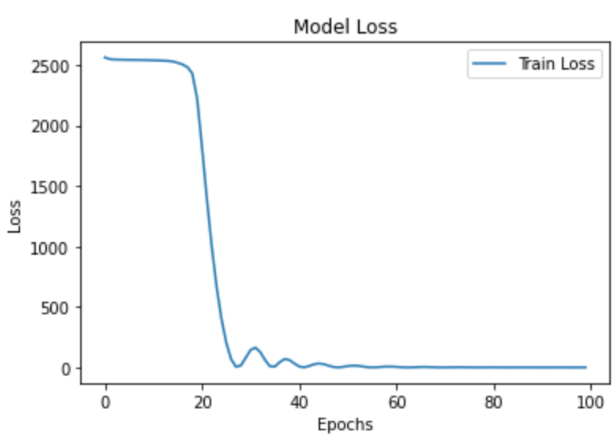
Problem 2a.

Using Keras Functional API, a model was built to predict exponential growth. Three hidden layers were added to reduce the number of epochs necessary for an accurate prediction. Weights were also implemented at each layer. Given the non-linear nature of the dataset, the weights were similar to those used in logistic regression.

Model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 3)]               0
_________________________________________________________________
dense (Dense)                (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 129
=================================================================
Total params: 50,561
Trainable params: 50,561
Non-trainable params: 0
_________________________________________________________________
```

Loss



Prediction

```
x_input = np.array([49, 64, 81]) # predict the ooutput for this input
x_input = x_input.reshape((1, n_steps))
y_output = model.predict(x_input)
print("X=%s, Predicted=%s" % (x_input, y_output))
```

```
X=[[49 64 81]], Predicted=[[102.64433]]
```

The Functional API model was able to better predict the next exponential outcome with a 2% error (102.6 v. 100).

Problem 2b.

The model above has 3 inputs, 3 hidden layers with 256, 128, and 128 neurons, and 1 output. There are a total of 512 neurons. 4 weights were used to simulate a logistic regression, one weight per layer. There are 50,561 trainable parameters in this model.

Problem 3.

look_back	RMSE
7	Train RMSE = 1204 Test RMSE = 1774
14	Train RMSE = 1029 Test RMSE = 1893
30	Train RMSE = 898 Test RMSE = 1857
90	Train RMSE = 1333 Test RMSE = 2353

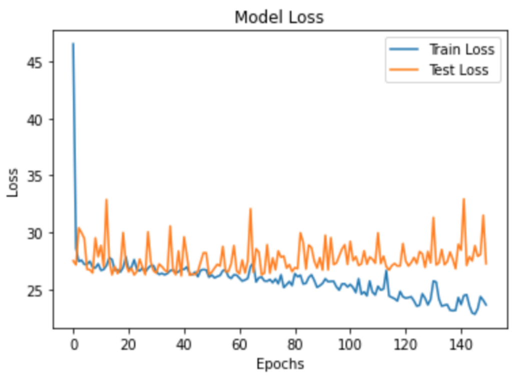
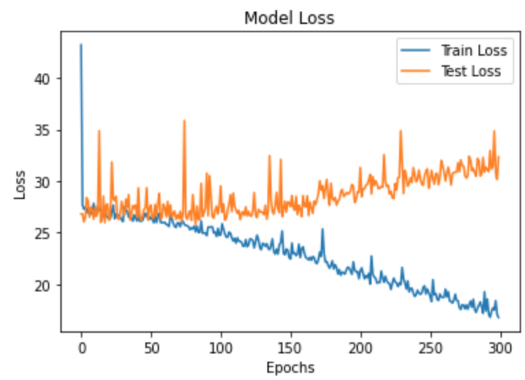
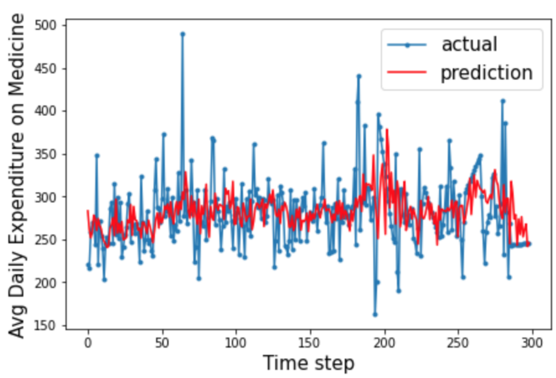
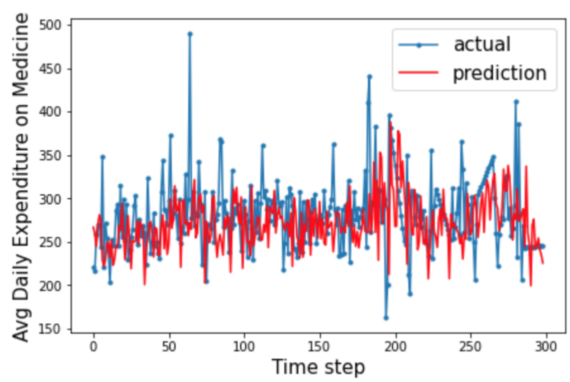
Different look-back time-steps were tested to achieve the lowest root mean square error (RMSE). The given look-back was 7 days, correlating to weekly medical expenditures. I also tested 14 days (biweekly), 30 days (monthly), and 90 days (quarterly) to see which was best to predict the trend in medical expenditures. I chose a model from Problem 2 and ran each look_back value listed. The RMSE for train and validation (test) datasets are shown. There was no clear best look-back time-step. Look_back = 14 was chosen because the train and test RMSEs were lowest without the model overfitting the data.

Experimental Models

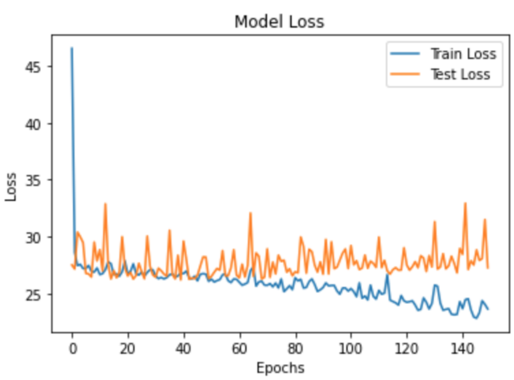
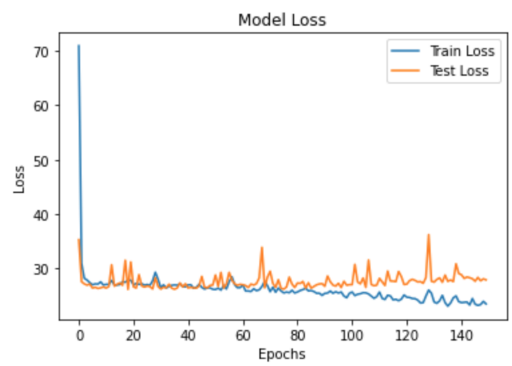
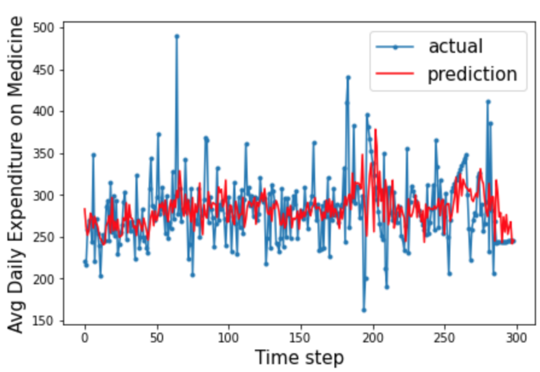
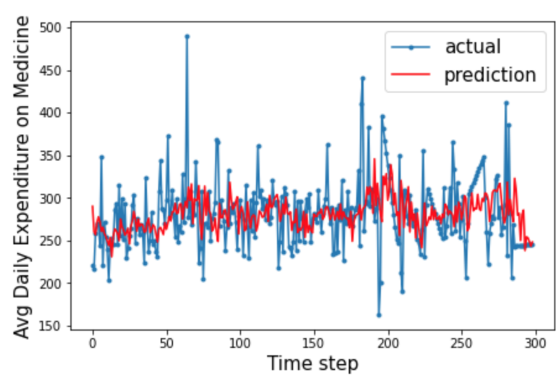
A model was designed that achieved the lowest RMSE given what was learned in Problems 1 and 2. The following three parameters will be changed:

1. Number of epochs
2. Number of hidden layers
3. Activation Function

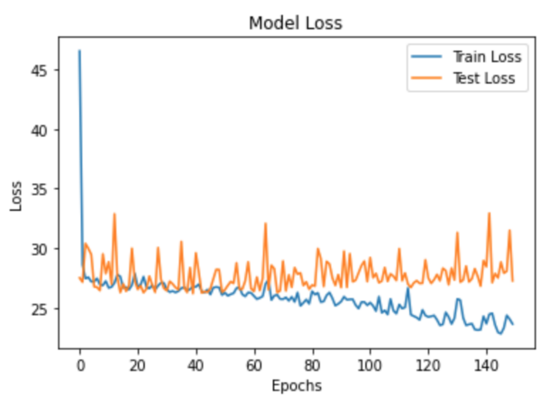
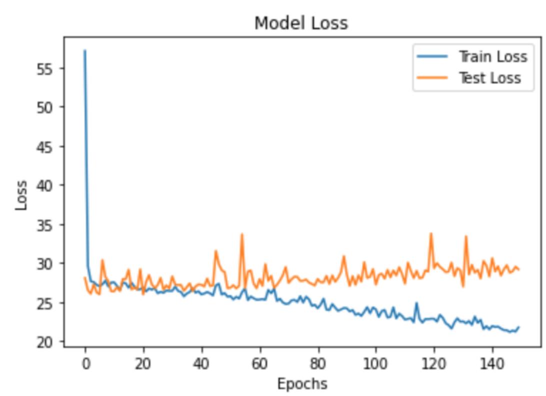
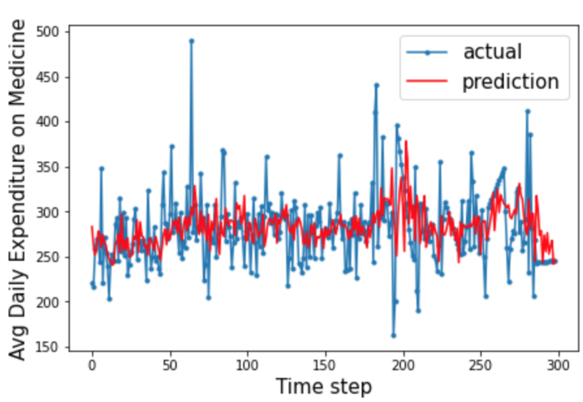
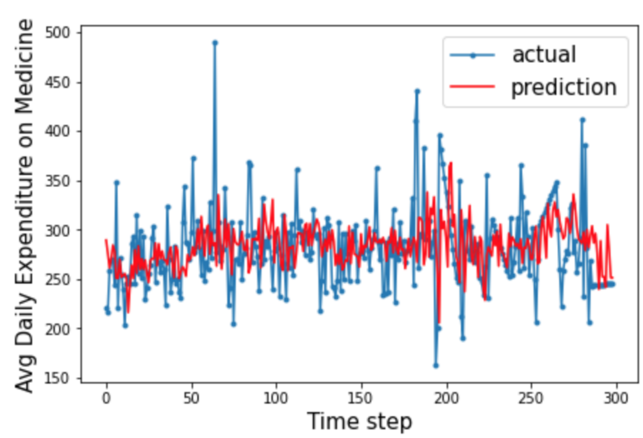
to see their effects on the model's accuracy (RMSE) and loss. The results will be analyzed in the following three tables.

<table>
<tr><th></th><th>Initial Model, Epoch = 150</th><th>Change Epoch = 300</th></tr>
<tr><td>Model</td><td>Layer (type) Output Shape Param #
input_2 (InputLayer) [(None, 14)] 0
dense_4 (Dense) (None, 128) 1920
dense_5 (Dense) (None, 64) 8256
dense_6 (Dense) (None, 64) 4160
dense_7 (Dense) (None, 1) 65
Total params: 14,401
Trainable params: 14,401
Non-trainable params: 0</td><td>Layer (type) Output Shape Param #
input_2 (InputLayer) [(None, 14)] 0
dense_4 (Dense) (None, 128) 1920
dense_5 (Dense) (None, 64) 8256
dense_6 (Dense) (None, 64) 4160
dense_7 (Dense) (None, 1) 65
Total params: 14,401
Trainable params: 14,401
Non-trainable params: 0</td></tr>
<tr><td>Training</td><td>loss='mean_absolute_error'
optimizer="adam"
epochs=150, validation_split=0.2</td><td>loss='mean_absolute_error'
optimizer="adam"
epochs=300, validation_split=0.2</td></tr>
<tr><td>Loss</td><td></td><td></td></tr>
<tr><td>RMSE</td><td>Train RMSE: 23.317861557006836
Test RMSE: 32.44575119018555</td><td>Train RMSE: 19.65681266784668
Test RMSE: 37.33937454223633</td></tr>
<tr><td>Prediction v. Actual</td><td></td><td></td></tr>
</table>

Increasing epoch number decreases the RMSE and increases accuracy of predicted values. However, the model is overfitting, as seen by the test loss increasing relative to the train loss.

	Initial Model (3 Layers)	Add 2 hidden layers (5 Layers)
Model	Layer (type) Output Shape Param # input_2 (InputLayer) [(None, 14)] 0 dense_4 (Dense) (None, 128) 1920 dense_5 (Dense) (None, 64) 8256 dense_6 (Dense) (None, 64) 4160 dense_7 (Dense) (None, 1) 65 Total params: 14,401 Trainable params: 14,401 Non-trainable params: 0	Layer (type) Output Shape Param # input_1 (InputLayer) [(None, 14)] 0 dense (Dense) (None, 128) 1920 dense_1 (Dense) (None, 64) 8256 dense_2 (Dense) (None, 64) 4160 dense_3 (Dense) (None, 64) 4160 dense_4 (Dense) (None, 64) 4160 dense_5 (Dense) (None, 1) 65 Total params: 22,721 Trainable params: 22,721 Non-trainable params: 0
Training	loss='mean_absolute_error' optimizer="adam" epochs=150, validation_split=0.2	loss='mean_absolute_error' optimizer="adam" epochs=150, validation_split=0.2
Loss		
RMSE	Train RMSE: 23.317861557006836 Test RMSE: 32.44575119018555	Train RMSE: 23.906476974487305 Test RMSE: 31.657373428344727
Prediction v. Actual		

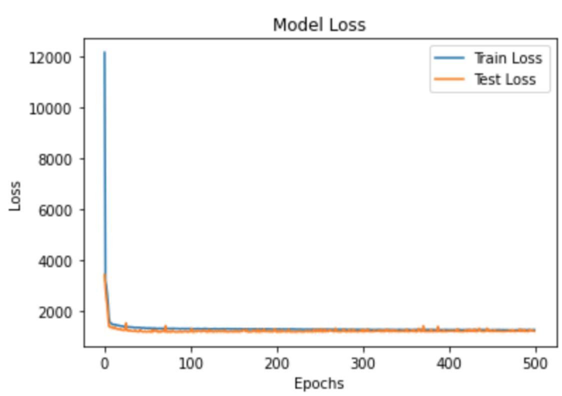
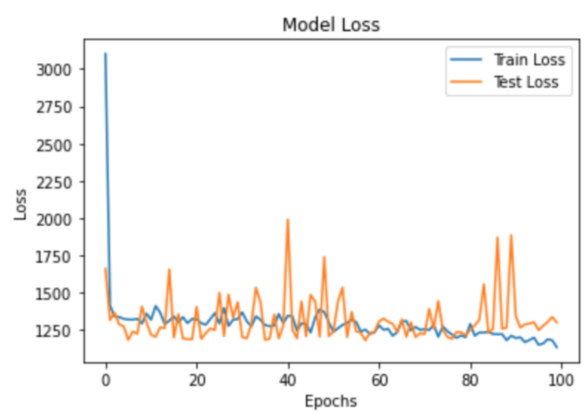
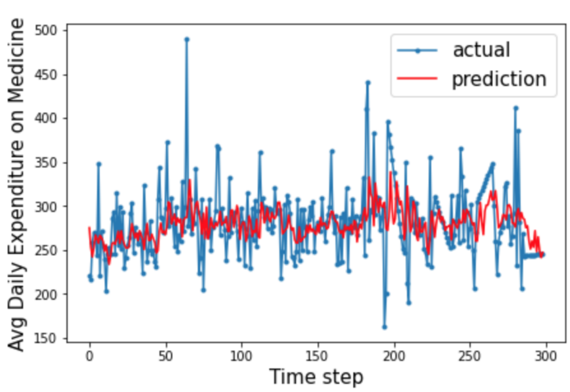
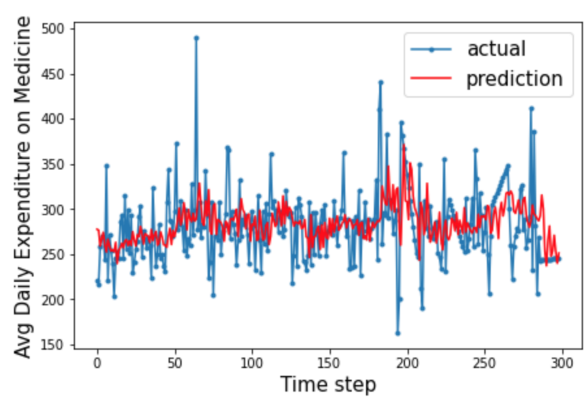
The RMSE does not change with increasing hidden layers. The train and test loss are similar, indicating that the model is neither over nor under fit. Adding hidden layers decreases the overfitting of the model, making it just right.

	Initial Model (Activation function= ReLU)	Change Activation Function to Swish
Model	Layer (type) Output Shape Param # input_2 (InputLayer) [(None, 14)] 0 dense_4 (Dense) (None, 128) 1920 dense_5 (Dense) (None, 64) 8256 dense_6 (Dense) (None, 64) 4160 dense_7 (Dense) (None, 1) 65 Total params: 14,401 Trainable params: 14,401 Non-trainable params: 0	Layer (type) Output Shape Param # input_1 (InputLayer) [(None, 14)] 0 dense (Dense) (None, 128) 1920 dense_1 (Dense) (None, 64) 8256 dense_2 (Dense) (None, 64) 4160 dense_3 (Dense) (None, 1) 65 Total params: 14,401 Trainable params: 14,401 Non-trainable params: 0
Training	loss='mean_absolute_error' optimizer="adam" epochs=150, validation_split=0.2	loss='mean_absolute_error' optimizer="adam" epochs=150, validation_split=0.2
Loss		
RMSE	Train RMSE: 23.317861557006836 Test RMSE: 32.44575119018555	Train RMSE: 22.17965316772461 Test RMSE: 33.41248321533203
Prediction v. Actual		

Activation function was originally set to "relu" for each hidden layer. For this experimental test they were all changed to "swish". The loss function has less variance after the change, but there is a minimal difference in RMSE. It also appears that there is overfitting using swish, as the train and test loss seem to drift further apart.

Use Models from Problem 1 & 2:

The MLP models from Problems 1 and 2 were trained with the Medical Expenditures dataset. The following table shows their results.

	Model from Problem 1 (Sequential)	Model from Problem 2 (Functional API)
Model	Model: "sequential_8" Layer (type) Output Shape Param # dense_154 (Dense) (None, 8) 120 dense_155 (Dense) (None, 1) 9 Total params: 129 Trainable params: 129 Non-trainable params: 0	Model: "model_38" Layer (type) Output Shape Param # input_39 (InputLayer) [(None, 14)] 0 dense_156 (Dense) (None, 256) 3840 dense_157 (Dense) (None, 128) 32896 dense_158 (Dense) (None, 128) 16512 dense_159 (Dense) (None, 1) 129 Total params: 53,377 Trainable params: 53,377 Non-trainable params: 0
Loss	 	
RMSE	Train RMSE: 1235.6380615234375 Test RMSE: 1710.6234130859375	Train RMSE: 1129.9097900390625 Test RMSE: 1859.9356689453125
Prediction v. Actual	 	

Models from problems 1 and 2 return high RMSE values whose loss does not improve with increasing epochs. The graphs also show that the predicted outputs do not align with the actual data. These are not good models for predicting the Medical Expenditures dataset.

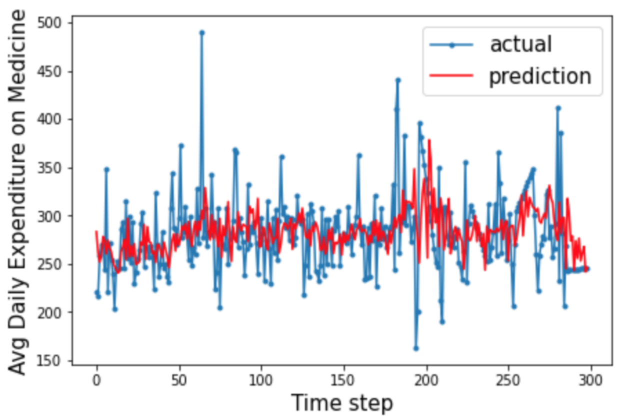
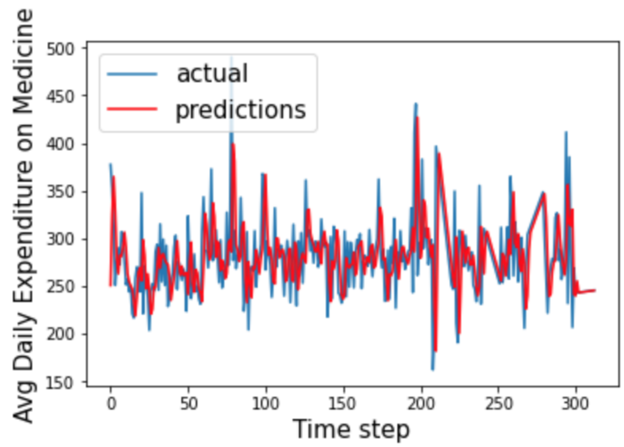
Problem 4.

A non-Deep Learning, ARIMA model was trained on the Medical Expenditures dataset for comparison purposes. ARIMA is an acronym for: AutoRegressive Integrated Moving Average. These models are specifically used to forecast a time series using the series past values and are characterized by three terms: lag order (p), degree of differencing (d), and order of the moving average (q). In the table below, these three factors were modified and the mean square error (MSE) was recorded for each.

p	d	q	Test MSE
1	0	0	2513
0	1	0	2565
0	0	1	4413
1	1	1	1608
1	1	0	2116
2	1	1	1605
3	1	1	1600
7	1	1	1613
3	1	2	1606
5	2	1	Matrix non-invertible
1	0	7	Matrix non-invertible
1	0	12	Matrix non-invertible
0	0	7	2379
2	0	1	Did not converge

It would be assumed that increasing the moving average (q) component would result in a lower MSE, but that is not indicated from the data collected above. Instead, increasing the autoregressive (p) component resulted in less error. But increasing this factor too much raised the error. This particular model also had difficulties with certain p, d, q combinations and would result in non-convergence or a 'matrix non-invertible' error. The best trial run based on lowest MSE was 3, 1, 1 for p, d, q, respectively.

In the following table the ARIMA model prediction and MSE is compared to the Deep Learning model prediction and RMSE.

	Deep Learning (DL) Model	ARIMA Model
RMSE/ MSE	`Train RMSE: 23.317861557006836` `Test RMSE: 32.44575119018555`	`Test MSE: 1600.211`
Prediction		

Although the RMSE is much lower for the DL model, the ARIMA model better predicts the actual dataset as shown in the prediction graphs. The DL model has a smoother curve and does not account for the variance of the actual dataset. I would say the ARIMA model, for the reason of better prediction accuracy to the actual dataset, is a better model. However, both models do not have a very low MSE or RMSE and are still not "good enough".