به نام خدا

پروژه دوم هوش مصنوعی

فرناز خوش دوست آزاد
99521253

سوال 1

بعد از انتخاب کردن 5000 نمونه ی تصادفی از دیتای مورد نظر و جای گذاری None values با مقادیری دیگر و Normalize کردن دیتاها حال شبکه های عصبی مختلف را با مقادیر مختلف می سازیم.

```
[58] model = keras.Sequential()
     model.add(layers.Dense(64, activation='relu'))
     model.add(layers.Dropout(0.15))  # Dropout layer to prevent overfitting
     model.add(layers.Dense(128, activation='relu'))
     model.add(layers.Dropout(0.5))
     model.add(layers.Dense(32, activation='relu'))
     model.add(layers.Dense(1, activation='softmax'))
```

```
[59] model.compile(loss = 'binary_crossentropy',
                   optimizer=keras.optimizers.Adam(learning_rate=0.001),
                   metrics=['accuracy'])
```

```
model.fit(X_train, y_train, epochs=40, batch_size=32,
          verbose=1, validation_split=0.2)
```

```
94/94 [==============================] - 0s 4ms/step - loss: 0.6727 - accuracy: 0.4200 - val_loss: 0.6708 - val_accuracy: 0.4067
Epoch 13/40
94/94 [==============================] - 2s 16ms/step - loss: 0.6718 - accuracy: 0.4200 - val_loss: 0.6699 - val_accuracy: 0.4067
Epoch 14/40
94/94 [==============================] - 0s 4ms/step - loss: 0.6659 - accuracy: 0.4200 - val_loss: 0.6682 - val_accuracy: 0.4067
Epoch 15/40
94/94 [==============================] - 0s 4ms/step - loss: 0.6643 - accuracy: 0.4200 - val_loss: 0.6658 - val_accuracy: 0.4067
Epoch 16/40
94/94 [==============================] - 2s 17ms/step - loss: 0.6603 - accuracy: 0.4200 - val_loss: 0.6651 - val_accuracy: 0.4067
Epoch 17/40
94/94 [==============================] - 0s 4ms/step - loss: 0.6607 - accuracy: 0.4200 - val_loss: 0.6646 - val_accuracy: 0.4067
Epoch 18/40
94/94 [==============================] - 0s 4ms/step - loss: 0.6594 - accuracy: 0.4200 - val_loss: 0.6625 - val_accuracy: 0.4067
Epoch 19/40
94/94 [==============================] - 0s 4ms/step - loss: 0.6583 - accuracy: 0.4200 - val_loss: 0.6627 - val_accuracy: 0.4067
Epoch 20/40
```

با عوض کردن activation آخرین لایه ی عصبی به sigmoid، accuracy بالاتری داریم.

```
[67] model.add(layers.Dense(128, activation='relu'))
     model.add(layers.Dropout(0.15))  # Dropout layer to prevent overfitting
     model.add(layers.Dense(128, activation='relu'))
     model.add(layers.Dropout(0.25))
     model.add(layers.Dense(256, activation='relu'))
     model.add(layers.Dense(32, activation='relu'))
     model.add(layers.Dense(1, activation='sigmoid'))
```

```
[68] model.compile(loss = 'binary_crossentropy',
                   optimizer=keras.optimizers.Adam(learning_rate=0.001),
                   metrics=['accuracy'])
```

```
model.fit(X_train, y_train, epochs=40, batch_size=32,
          verbose=1, validation_split=0.2)
```

```
94/94 [==============================] - 1s 7ms/step - loss: 0.6551 - accuracy: 0.6230 - val_loss: 0.6667 - val_accuracy: 0.6093
Epoch 13/40
94/94 [==============================] - 1s 8ms/step - loss: 0.6566 - accuracy: 0.6223 - val_loss: 0.6572 - val_accuracy: 0.6347
Epoch 14/40
94/94 [==============================] - 1s 8ms/step - loss: 0.6547 - accuracy: 0.6277 - val_loss: 0.6585 - val_accuracy: 0.6480
Epoch 15/40
94/94 [==============================] - 1s 8ms/step - loss: 0.6542 - accuracy: 0.6290 - val_loss: 0.6560 - val_accuracy: 0.6280
Epoch 16/40
94/94 [==============================] - 1s 6ms/step - loss: 0.6553 - accuracy: 0.6243 - val_loss: 0.6564 - val_accuracy: 0.6267
Epoch 17/40
94/94 [==============================] - 0s 5ms/step - loss: 0.6527 - accuracy: 0.6317 - val_loss: 0.6556 - val_accuracy: 0.6307
Epoch 18/40
94/94 [==============================] - 0s 5ms/step - loss: 0.6528 - accuracy: 0.6340 - val_loss: 0.6583 - val_accuracy: 0.6227
Epoch 19/40
94/94 [==============================] - 0s 5ms/step - loss: 0.6511 - accuracy: 0.6397 - val_loss: 0.6559 - val_accuracy: 0.6493
Epoch 20/40
94/94 [==============================] - 0s 5ms/step - loss: 0.6527 - accuracy: 0.6323 - val_loss: 0.6584 - val_accuracy: 0.6227
Epoch 21/40
```

با تغییر تعداد لایه های از 4 به 6 و 7، accuracy کمتر شد و با تغییر learning_rate و activation های مختلف و درصد های مختلف برای دراپ کردن، accuracy تغغیر فاحشی نکرد و من به درصد بالاتری از 64% دست پیدا نکردم.

و در آخر تعداد داده های خود را بیشتر کرده و این بالاترین درصدی است که به آن دست پیدا کردم.

```
[33] 144/144 [==============================] - 0s 2ms/step - loss: 0.6609 - accuracy: 0.6253 - val_loss: 0.6570 - val_accuracy: 0.6236
Epoch 43/50
144/144 [==============================] - 0s 3ms/step - loss: 0.6627 - accuracy: 0.6170 - val_loss: 0.6540 - val_accuracy: 0.6333
Epoch 44/50
144/144 [==============================] - 0s 2ms/step - loss: 0.6615 - accuracy: 0.6203 - val_loss: 0.6544 - val_accuracy: 0.6340
Epoch 45/50
144/144 [==============================] - 0s 2ms/step - loss: 0.6609 - accuracy: 0.6205 - val_loss: 0.6547 - val_accuracy: 0.6236
Epoch 46/50
144/144 [==============================] - 0s 2ms/step - loss: 0.6631 - accuracy: 0.6148 - val_loss: 0.6534 - val_accuracy: 0.6292
Epoch 47/50
144/144 [==============================] - 0s 2ms/step - loss: 0.6595 - accuracy: 0.6253 - val_loss: 0.6541 - val_accuracy: 0.6333
Epoch 48/50
144/144 [==============================] - 0s 2ms/step - loss: 0.6619 - accuracy: 0.6194 - val_loss: 0.6546 - val_accuracy: 0.6326
Epoch 49/50
144/144 [==============================] - 0s 2ms/step - loss: 0.6594 - accuracy: 0.6217 - val_loss: 0.6552 - val_accuracy: 0.6299
Epoch 50/50
144/144 [==============================] - 1s 4ms/step - loss: 0.6593 - accuracy: 0.6198 - val_loss: 0.6523 - val_accuracy: 0.6313
<keras.src.callbacks.History at 0x7d736de2ec80>
```

```
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
57/57 [==============================] - 0s 1ms/step
Accuracy: 64.94%
```

سوال 2

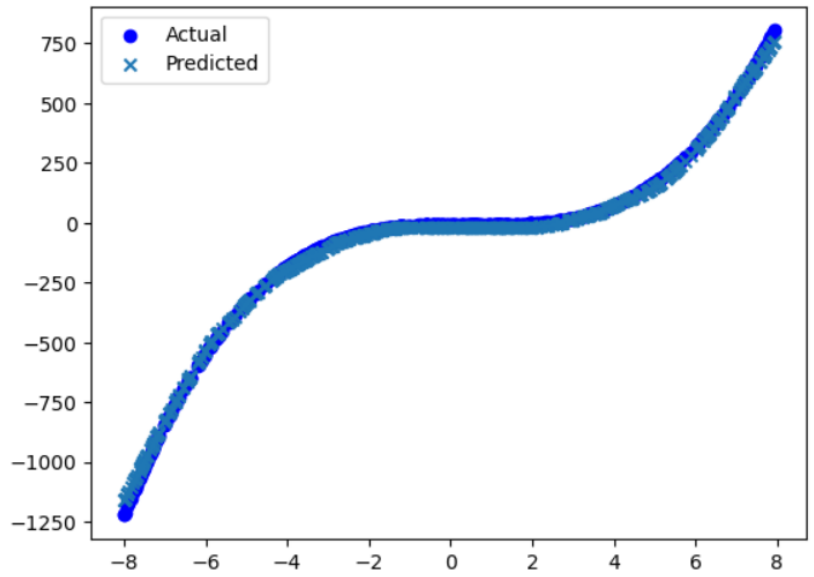
Complicated :

تابعی که برای قسمت comlicated سوال در نظر گرفته ام،

f(x) = 2* x**3 - 3*x**2 - 7 می باشد که با کد زیر به پاسخ های زیر رسیده ام. برای این سوال در ابتدا از sequential استفاده کردم برای محاسبه درصد خطای آن نیز از evaluate method استفاده کردم که تصاویر آن در زیر موجود است.

```python
def function(x):
    return 2* x**3 - 3*x**2 - 7
np.random.seed(0)
X_train = np.random.uniform(-8, 8, 2000)
y_train = function(X_train)
model = Sequential()
model.add(Dense(128, activation='relu' , input_dim=1))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dense(units=1, activation='linear'))

opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='mean_squared_error')

model.fit(X_train, y_train, epochs=400, batch_size=40, validation_split=0.2)

X_test = np.random.uniform(-8, 8, 500)
y_test = function(X_test)
predictions = model.predict(X_test)
```

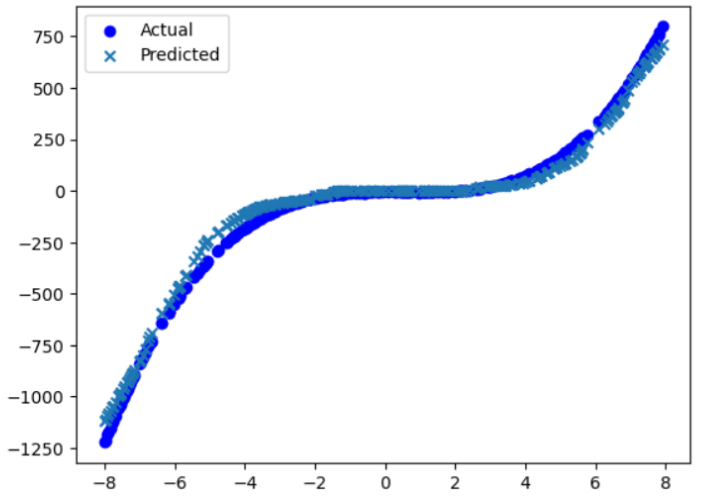
```
=======] - 0s 4ms/step - loss: 2185.6018 - val_loss: 280.0128

=======] - 0s 4ms/step - loss: 2316.1797 - val_loss: 71.3840

=======] - 0s 4ms/step - loss: 2360.9863 - val_loss: 173.9925

40/40 [==============================] - 0s 4ms/step - loss: 2107.7419 - val_loss: 211.8940
16/16 [==============================] - 0s 2ms/step
Mean Squared Error: 223.21
```



پس از آن یک لایه به شبکه اضافه کردم که در آخر به MSE بیشتری رسیدم.

```python
def function(x):
    return 2* x**3 - 3*x**2 - 7
np.random.seed(0)
X_train = np.random.uniform(-8, 8, 2000)
y_train = function(X_train)
model = Sequential()
model.add(Dense(128, activation='relu' , input_dim=1))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(units=1, activation='linear'))

opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='mean_squared_error')

model.fit(X_train, y_train, epochs=400, batch_size=40, validation_split=0.2)

X_test = np.random.uniform(-8, 8, 500)
y_test = function(X_test)
predictions = model.predict(X_test)
```

```
40/40 [==============================] - 0s 4ms/step - loss: 1711.8425 - val_loss: 1485.5648
Epoch 397/400
40/40 [==============================] - 0s 4ms/step - loss: 1647.6528 - val_loss: 1300.5302
Epoch 398/400
40/40 [==============================] - 0s 4ms/step - loss: 1695.9742 - val_loss: 1668.3285
Epoch 399/400
40/40 [==============================] - 0s 4ms/step - loss: 1501.6327 - val_loss: 1112.1582
Epoch 400/400
40/40 [==============================] - 0s 4ms/step - loss: 1713.5327 - val_loss: 2139.3484
10/10 [==============================] - 0s 5ms/step
Mean Squared Error: 1754.28
```

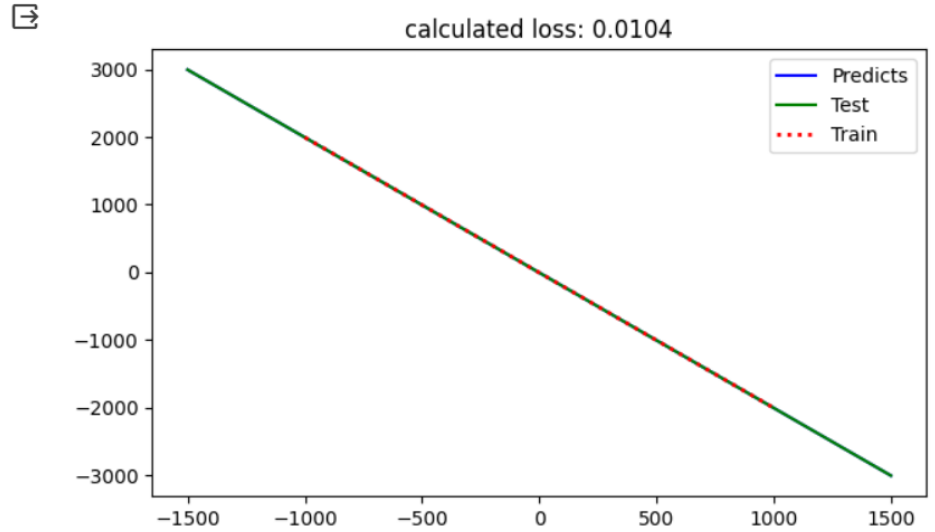


سوال 2 : Simple

$F(x) = 2 * x - 5$

This is the best that I could found.

```
Run((30,20,10),300,(5000,-1000,1000),(1000,-1500,1500))
```



Train Dataset:5000 points range from -1000 to 1000
Test Dataset :1000 points range from -1500 to 1500
Number of Hidden layers : 3
Epochs : 300

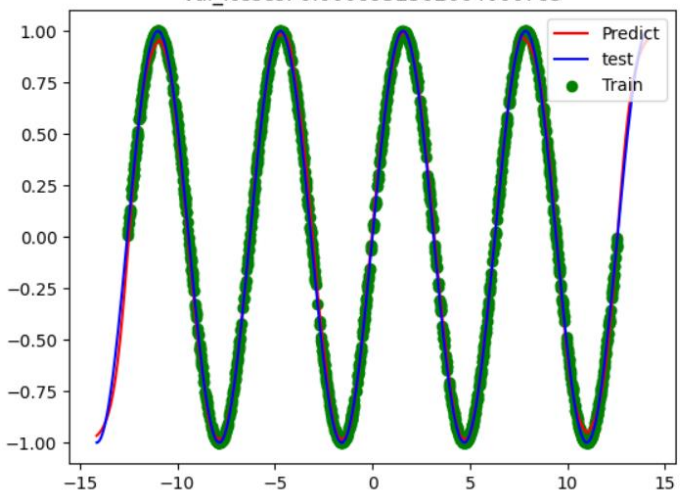
```
def Run(hidden_layer, num_of_iteration, train_info, test_info):
    x_train, y_train = correct_input_output(
        train_info[0], train_info[1], train_info[2])
    x_test, y_test = correct_input_output(
        test_info[0], test_info[1], test_info[2])
    x_input, y_input = correct_input_output(
        num_of_points=train_info[0], min=train_info[1], max=train_info[2])
    train_network = MLPRegressor(hidden_layer_sizes=hidden_layer,
                                 max_iter=num_of_iteration,
                                 random_state=0, shuffle=True).fit(x_input, y_in

    predictions = train_network.predict(x_test)

    fig, ax = plt.subplots()
    test_plt,  = plt.plot(x_test, predictions, color='b', label='Predicts'
```

سوال 2 : Simple

$F(x) = \sin(x)$

```python
def Run(hidden_layer, num_of_iteration, train_info, test_info):
    x_input, y_input = correct_input_output(
        train_info[0], train_info[1], train_info[2])
    x_test, y_test = generate_test(
        test_info[0], test_info[1], test_info[2])
    model = keras.Sequential()
    for h in hidden_layer:
        model.add(layers.Dense(units=h, input_dim=1, activation='relu'))

    model.add(layers.Dense(units=1, activation='tanh'))
    model.compile(optimizer='adam', loss='mean_squared_error')

    kfold = KFold(n_splits=4, shuffle=True, random_state=0)

    # Perform k-fold cross-validation
    for train_index, test_index in kfold.split(x_input):
        X_train, X_val = x_input[train_index], x_input[test_index]
        y_train, y_val = y_input[train_index], y_input[test_index]
        history = model.fit(X_train, y_train, epochs=num_of_iteration,
                  batch_size=32, verbose=1, validation_data=(X_val, y_val))
    predictions = model.predict(x_test)
    train_losses = np.mean(history.history['loss'])
    val losses = np.mean(history.history['val loss'])
```

```
47/47 [==============================] - 0s 4ms/step - loss: 0.0013 - val_loss: 8.8431e-04
Epoch 500/500
47/47 [==============================] - 0s 4ms/step - loss: 0.0010 - val_loss: 9.1294e-04
16/16 [==============================] - 0s 2ms/step
train_losses: 0.0008965688768075779
val_losses: 0.0008952582064666785
```



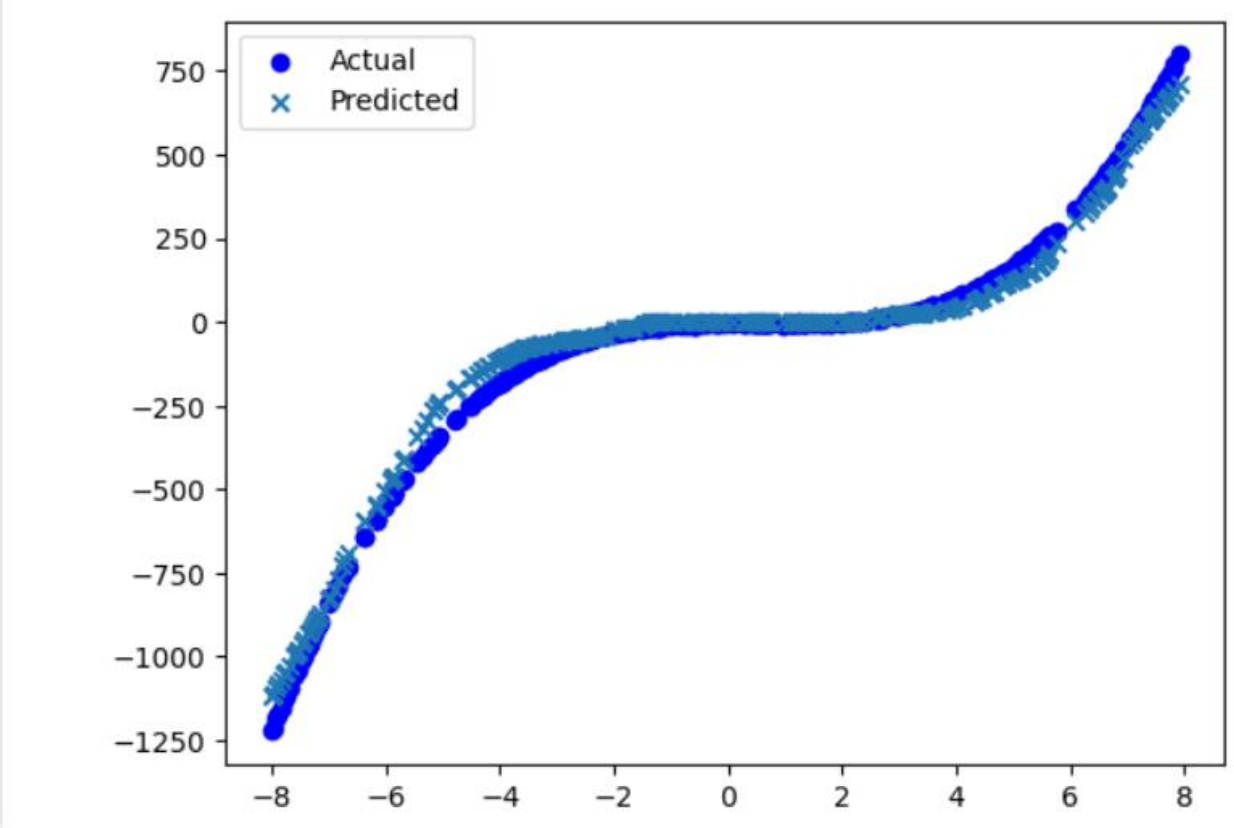

```
Train Dataset:2000 points range from -12.566370614359172 to 12.5663706143
Test  Dataset :500 points range from -14.137166941154069 to 14.1371669411
Number of Hidden layers : 2
Epochs : 500
```

S

سوال3:

با توجه به توضیحات در سوال دوم برای ایجاد نویز در بخش های مختلف از تابع random استفاده کردم و نتیج آن در عکس ها و نمودار مشخص است.

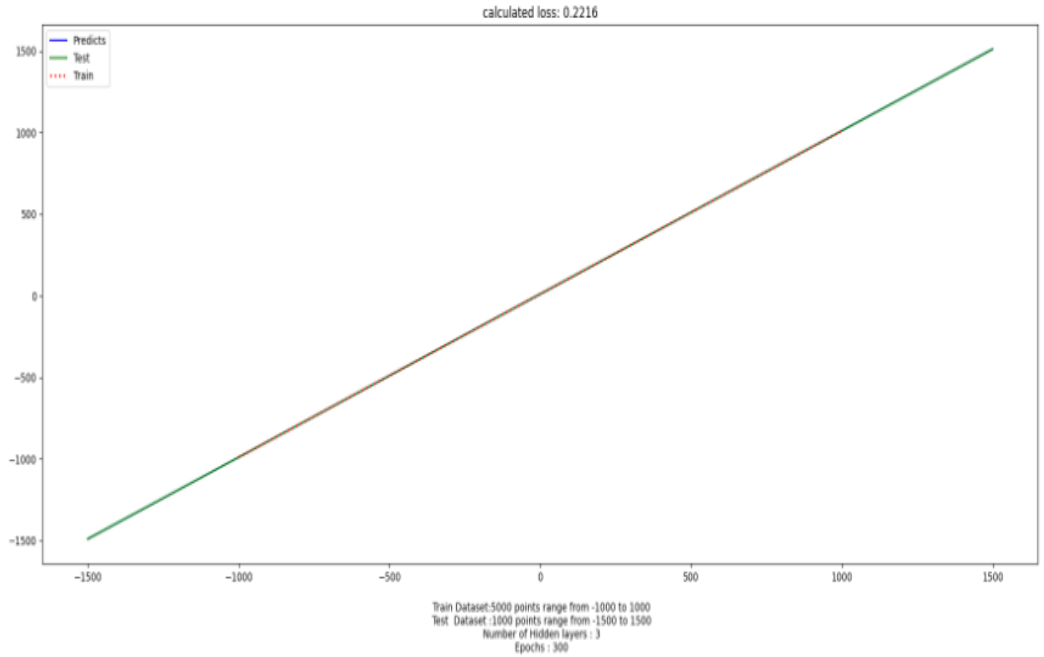
: Complicated



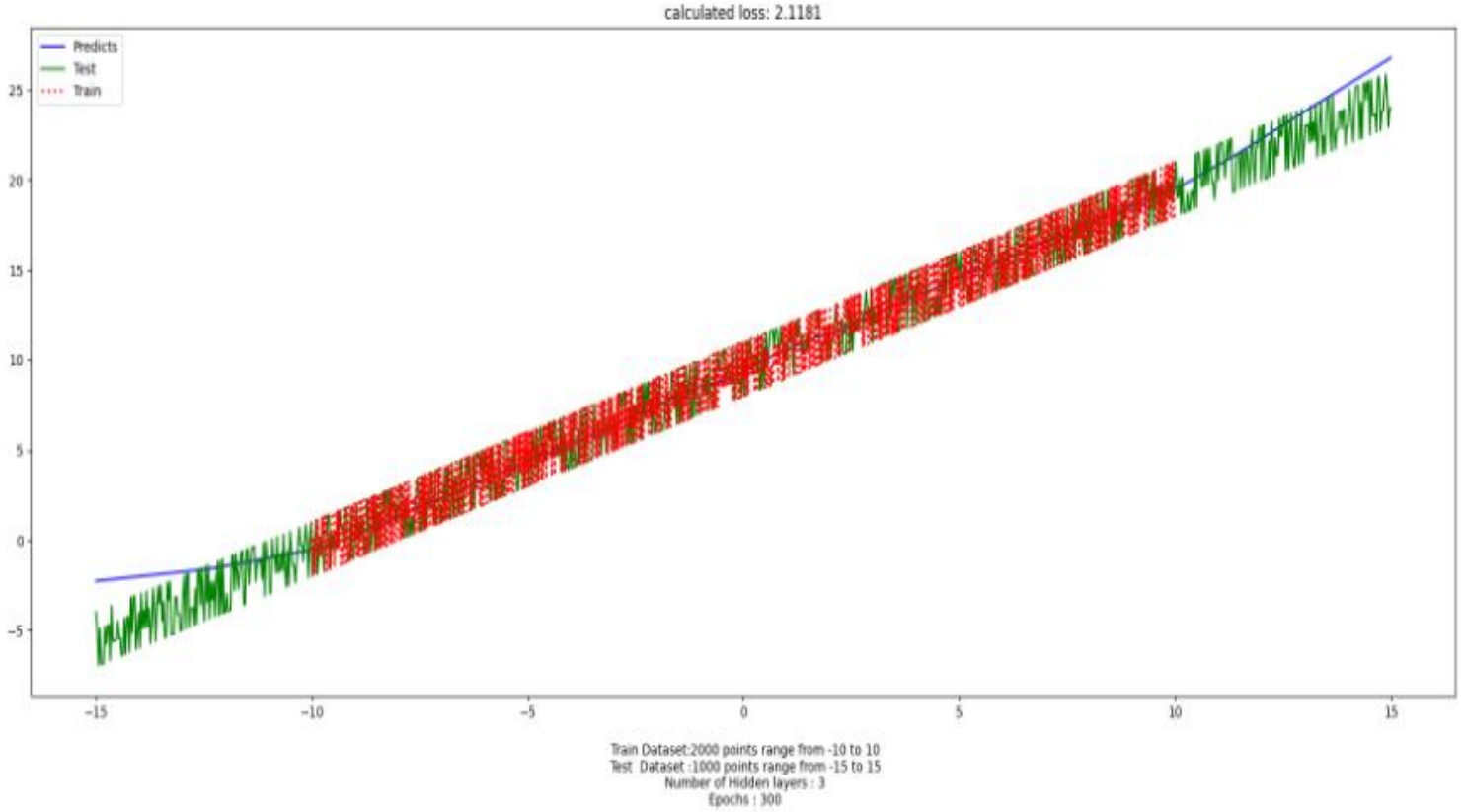
سوال 3 : Simple

F(x) = x + 10

This is the best that I could found.

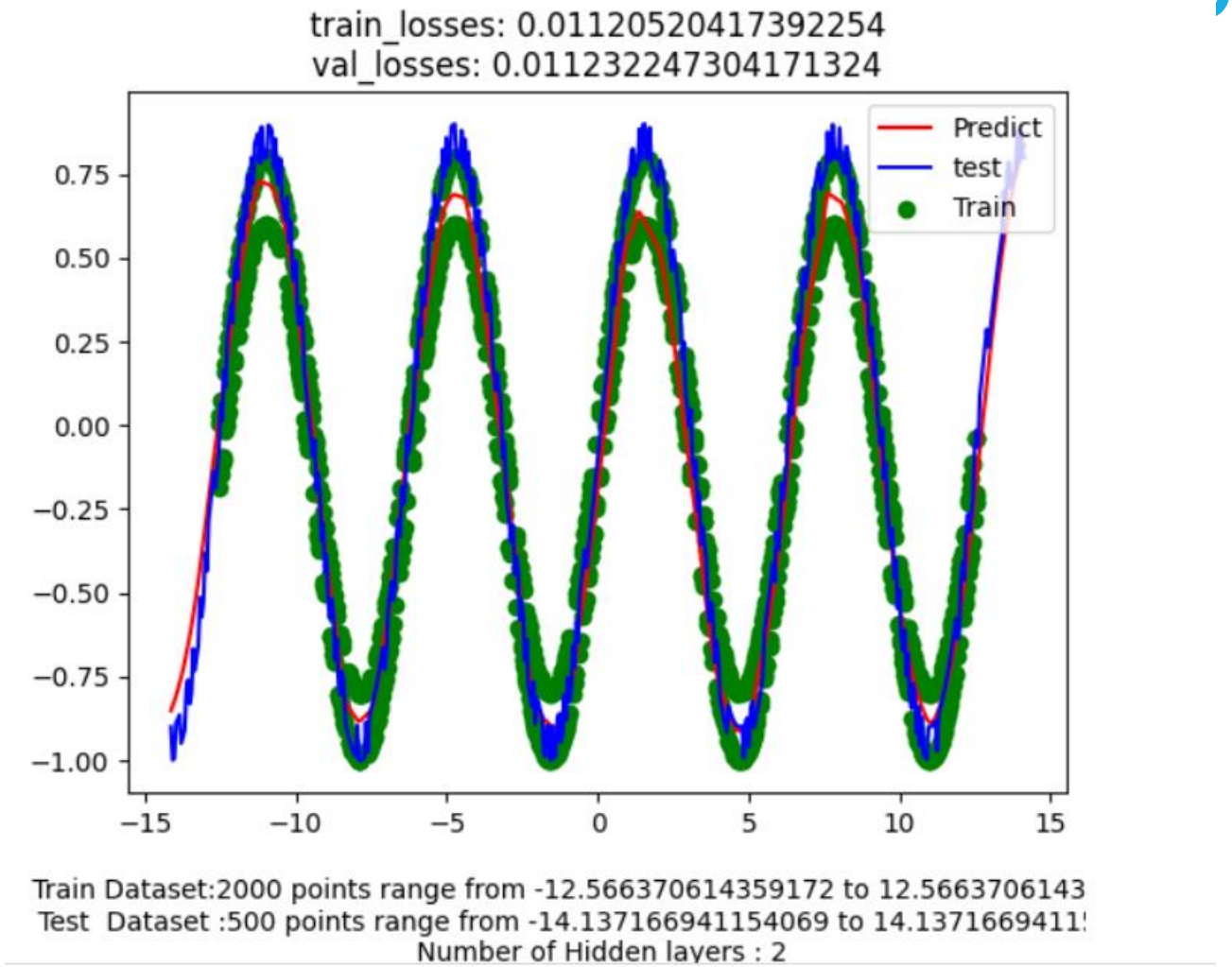


با نویز و شبیه سازی بهتر و بالاتر:



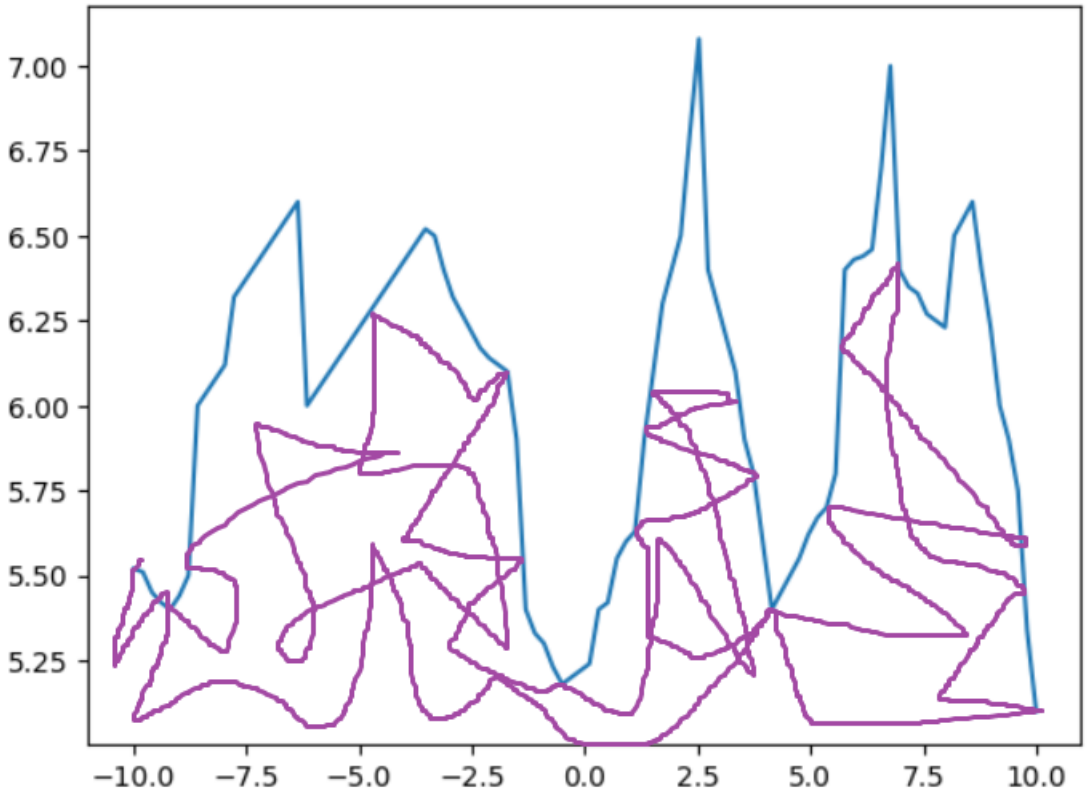
سوال 3 : Simple

F(x) = sin(x)

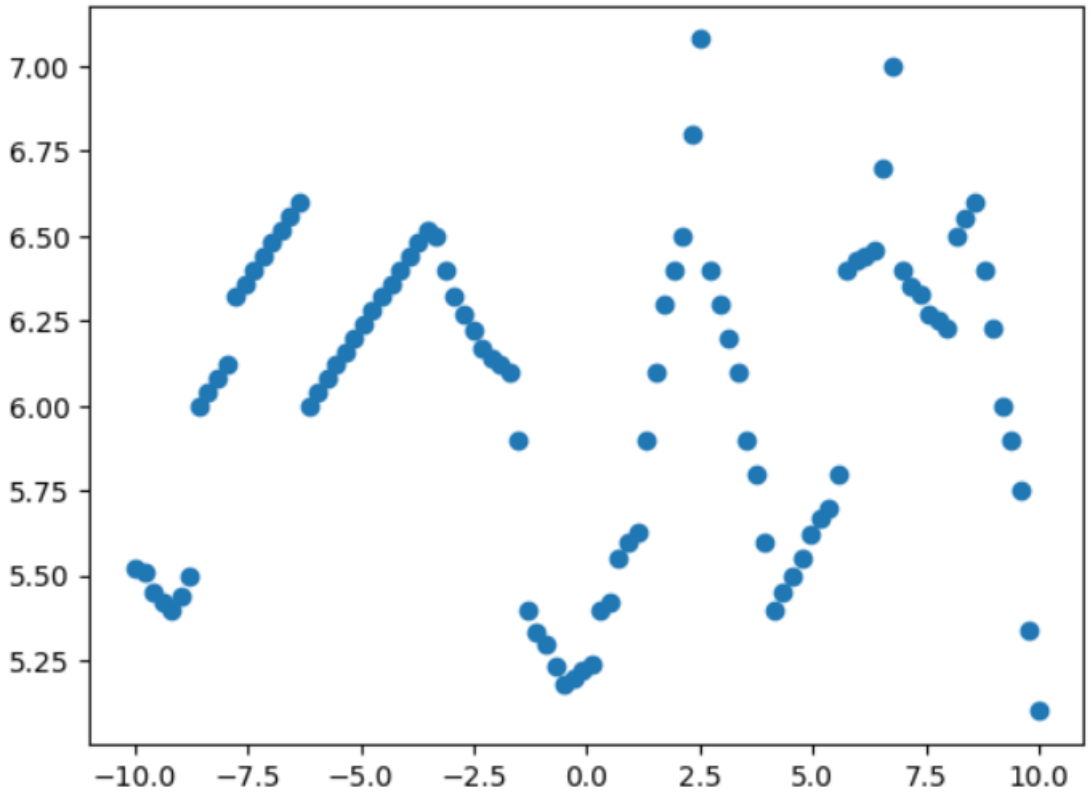


سوال 4

این اعصاب منه:

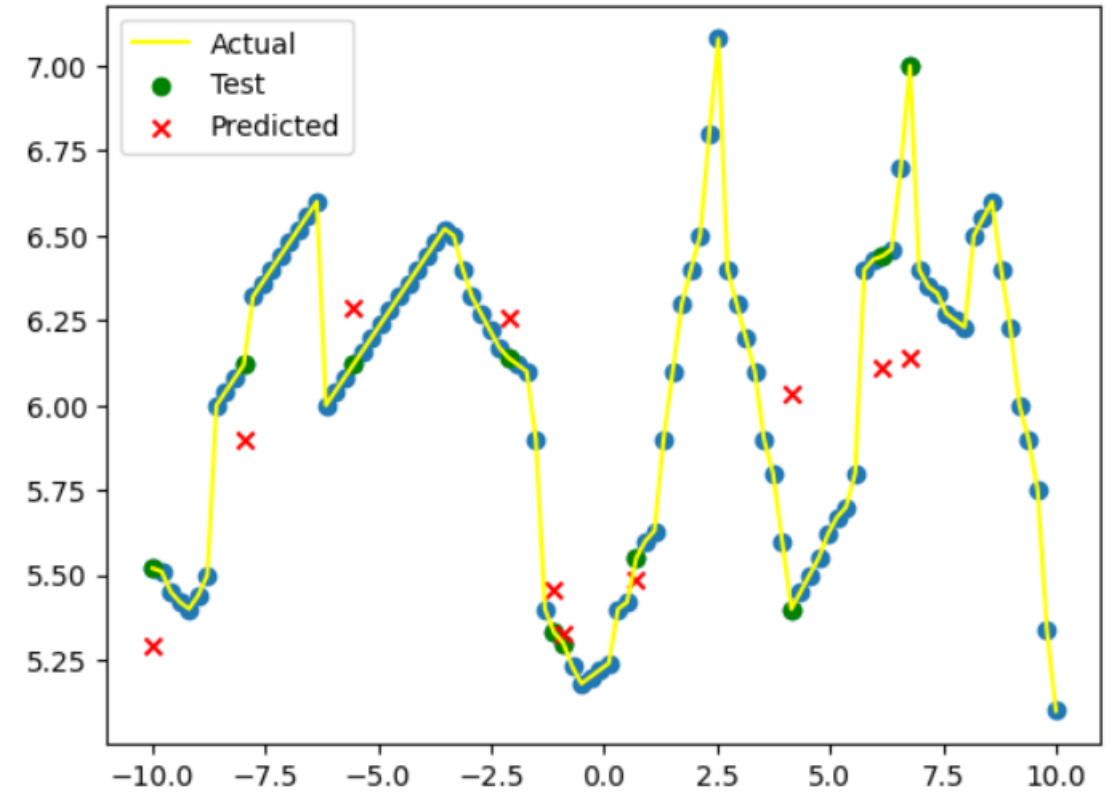


این هم نقاط انتخاب شده بالای اعصاب من می باشد.



با توجه به کدی که زده ام به ارور نسبتا خوبی رسیده ام و تابع به دست آمده تا حد زیادی درست می باشد.

```
Mean Squared Error (MSE): 0.14150102137083545
Mean Absolute Error (MAE): 0.27750905418395994
```



سوال 6

- در این سوال هرچه میزان نویز داده شده در محیط سیاه سفید را بیشتر کنیم بازیابی ان سخت تر خواهد بود