AML Lab Report - Internals 1

Team Members

USN: 01FB15ECS278 Name: Shashank Saran

USN: 01FB15ECS286 Name: Shreyas V Patil

USN: 01FB15ECS289 Name: Siddhanth Vinay

USN: 01FB15ECS290 Name: Siddharth Ganesan

General approach taken for both the target outputs:

1. The dataset was loaded, preprocessed and split into train and test.
2. A train_input function and a val_input function was designed to process the data for input and validation.
3. A custom DNN Regressor estimator, the function my_dnn_regression_fn() was designed to build a DNN regressor and the function takes the features, labels, mode and params parameters. The mode specifies whether the function is used for training, predicting or evaluating the model and the params parameter is used to specify the hyperparameters for building the model. The function builds a DNN regression model according to the specified hyperparameters and computes total loss and Root Mean Squared Error (RMSE) for evaluating the model.
4. A class myDNNRegressorEstimator was built which was initialized with an attribute for the estimator model, and a function get_weights(self) was designed to return the weights and biases for the hidden layer and the output layer. The function was hardcoded in this particular case.
5. A function predict(self, val1, val2) was designed which took two inputs val1 and val2, applied the weights, biases and activations to them (the forward propagation was applied to the inputs val1 and val2), hardcoded for a single hidden layer and the corresponding output was computed and returned.
6. The models were then trained, evaluated on test data and tested with custom sample inputs using the predict function to try and determine the underlying functions that represented the dataset.

Steps followed:

With column 3 is the target output column:

Hidden layers used: 1

Number of units in hidden layer: 100000

Number of steps for training: 500

Learning Rate: 0.1

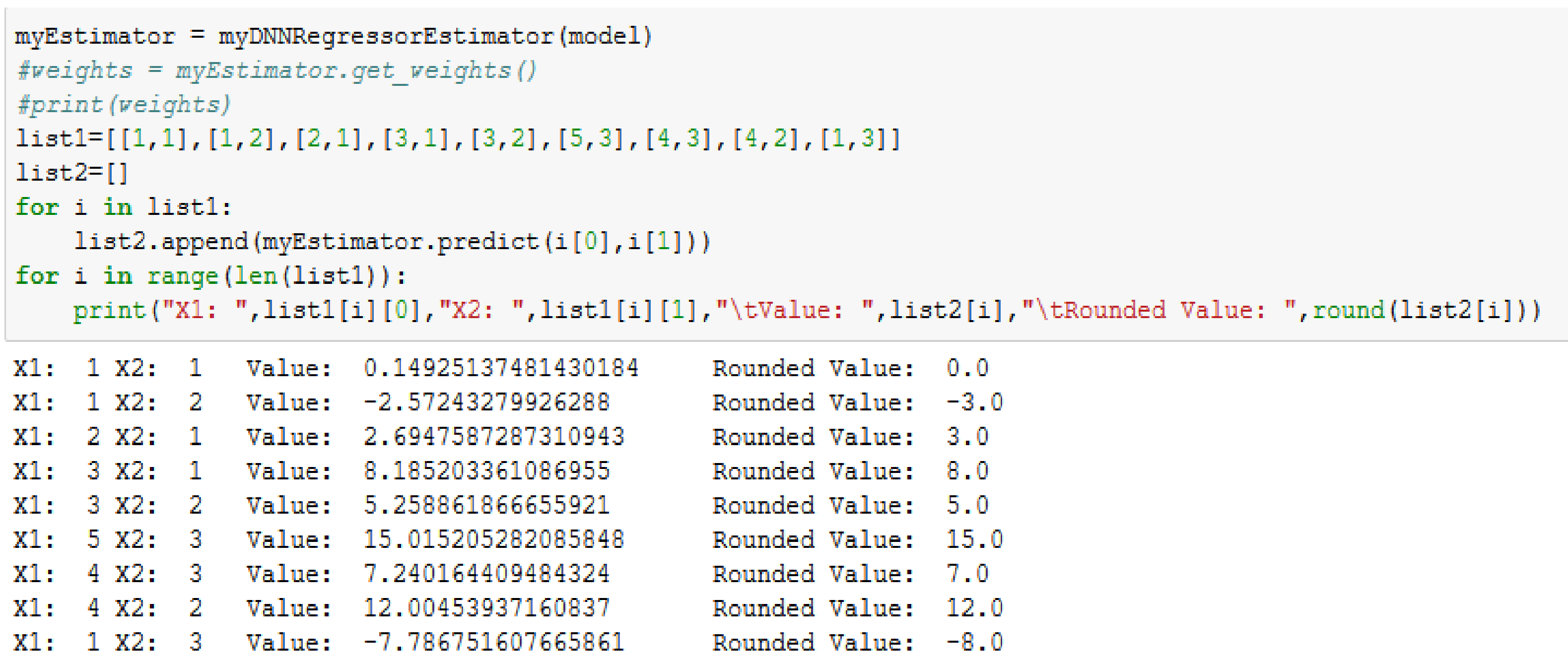
The above parameters were chosen as the model was not able to sufficiently learn the output function with smaller number of hidden units for the hidden layer. Only a single hidden layer with a large number of hidden units was chosen over multiple layers as it made the computation of the weight activations simpler.

Output:

Total loss: 10.957685

RMSE: 0.58595294

Outputs for the custom inputs using the predict function:



```
myEstimator = myDNNRegressorEstimator(model)
#weights = myEstimator.get_weights()
#print(weights)
list1=[[1,1],[1,2],[2,1],[3,1],[3,2],[5,3],[4,3],[4,2],[1,3]]
list2=[]
for i in list1:
    list2.append(myEstimator.predict(i[0],i[1]))
for i in range(len(list1)):
    print("X1: ",list1[i][0],"X2: ",list1[i][1],"\tValue: ",list2[i],"\tRounded Value: ",round(list2[i]))
```

```
X1:  1 X2:  1   Value:  0.14925137481430184     Rounded Value:  0.0
X1:  1 X2:  2   Value:  -2.57243279926288       Rounded Value:  -3.0
X1:  2 X2:  1   Value:  2.6947587287310943      Rounded Value:  3.0
X1:  3 X2:  1   Value:  8.185203361086955       Rounded Value:  8.0
X1:  3 X2:  2   Value:  5.258861866655921       Rounded Value:  5.0
X1:  5 X2:  3   Value:  15.015205282085848      Rounded Value:  15.0
X1:  4 X2:  3   Value:  7.240164409484324       Rounded Value:  7.0
X1:  4 X2:  2   Value:  12.00453937160837       Rounded Value:  12.0
X1:  1 X2:  3   Value:  -7.786751607665861      Rounded Value:  -8.0
```

From the above outputs corresponding to the sample test cases, it is possible to determine that the underlying function is F(X1, X2) = $X1^2 – X2^2$.

With column 4 as the target output column:

Hidden layers used: 1

Number of units in hidden layer: 10000

Number of steps for training: 5000

Learning Rate: 0.01

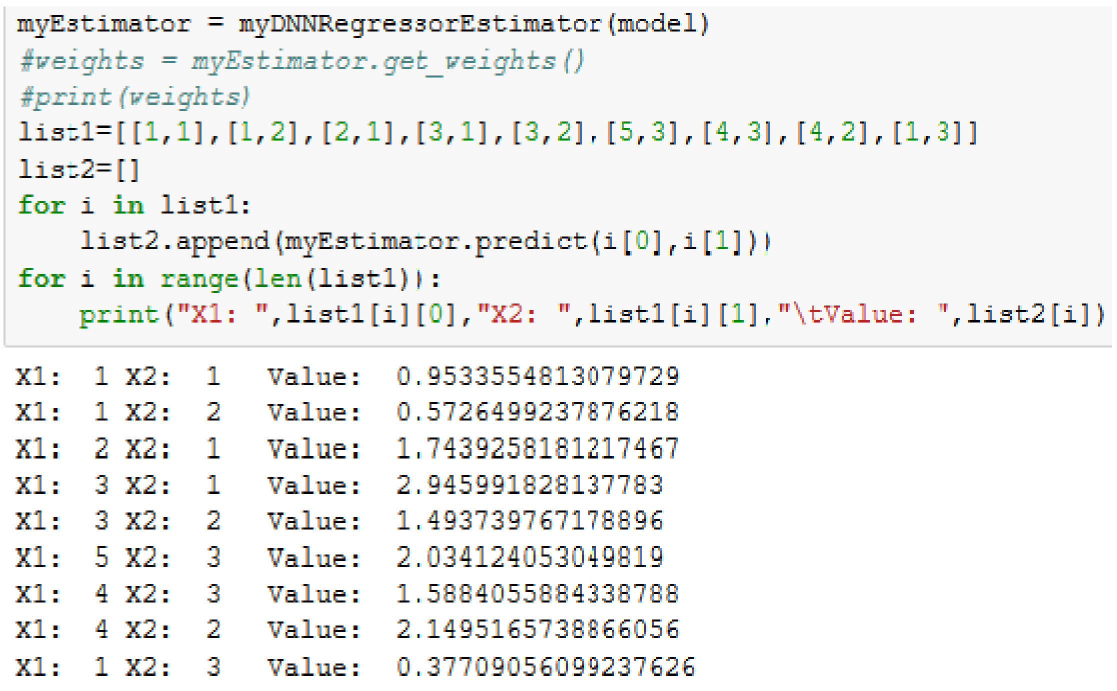
The reasons for choosing high number of units in the hidden layer and only a single hidden layer is the same as for the previous case. The hyperparameters were chosen after multiple trial and error by tuning each of the hyperparameters until a satisfactory model was obtained.

Output:

Total loss: 6435.7285

RMSE: 14.200452

Outputs for the custom inputs using the predict function:



```
myEstimator = myDNNRegressorEstimator(model)
#weights = myEstimator.get_weights()
#print(weights)
list1=[[1,1],[1,2],[2,1],[3,1],[3,2],[5,3],[4,3],[4,2],[1,3]]
list2=[]
for i in list1:
    list2.append(myEstimator.predict(i[0],i[1]))
for i in range(len(list1)):
    print("X1: ",list1[i][0],"X2: ",list1[i][1],"\tValue: ",list2[i])
```

```
X1:  1 X2:  1   Value:  0.9533554813079729
X1:  1 X2:  2   Value:  0.5726499237876218
X1:  2 X2:  1   Value:  1.7439258181217467
X1:  3 X2:  1   Value:  2.945991828137783
X1:  3 X2:  2   Value:  1.493739767178896
X1:  5 X2:  3   Value:  2.034124053049819
X1:  4 X2:  3   Value:  1.5884055884338788
X1:  4 X2:  2   Value:  2.1495165738866056
X1:  1 X2:  3   Value:  0.37709056099237626
```

From the above outputs corresponding to the sample test cases, it is possible to determine that the underlying function is F(X1, X2) = X1 / X2.