Activity 1

Prediction with Back-Propagation and Linear Regression

- ## Git Repository

https://github.com/YoussefEzz/Prediction-BP-and-LR

- ## Part 1 : Selecting and analyzing the datasets

Since we do not want to give a priori more importance to some of the input variables w.r.t. the others, we should scale all of them to the same range of variation.

The scaling of the output variables has an additional requirement: since the output of a sigmoid lays in the range (0.0, 1.0), the desired output values must strictly fall within these limits. For predictions tasks (e.g. A1), where the output variable takes values in a certain [min, max] range, a convenient choice is its linear scaling to a range like [0.1, 0.9]

Preprocess of Dataset 1 and 2 A1-synthetic.txt and A1-turbine.txt

1. read data from "Data\A1-synthetic.txt" and "Data\A1-turbine.txt"

```python
#read the information of A1-synthetic.txt and load it into a dataframe to preprocess it
import pandas as pd
import numpy as np


#read the .txt file
df = pd.read_table('Data/A1-turbine.txt', delimiter = '\t')
df.head()
```

	#height_over_sea_level	fall	net_fall	flow	power_of_hydroelectrical_turbine
0	624.0	89.16	89.765	3.5	2512.85
1	628.0	93.16	93.765	3.5	2583.79
2	602.0	67.84	66.415	6.5	3748.77
3	599.0	64.84	63.415	6.5	3520.65
4	630.0	94.69	93.540	8.0	6673.84

2. separate linear scaling of each input variable v1 to v9 for A1-synthetic - v3 and v8 are already between [0.0, 1.0] – and [height_over_sea_level fall net_fall flow] for A1-turbine from its [min, max] range to [0.0, 1.0] .

```python
#preprocess input 4 columns to scale it's values from 0 to 1
columns = df.shape[1]
inputcolumns = df.columns[0 : 4]
smin = 0
smax = 1
df_normalized = df.copy()
for inp_col in inputcolumns:
    column_values = df[inp_col]
    #print(column_values)
    xmin = min(column_values)
    xmax = max(column_values)
    #print( smin + ((smax - smin) / (xmax - xmin) ) * (df[inp_col] - xmin) )
    df_normalized[inp_col] = np.round(smin + ((smax - smin) / (xmax - xmin) ) * (df[inp_col] - xmin),

print(df_normalized)
```

```
     #height_over_sea_level    fall  net_fall    flow  \
0                    0.8462  0.8212    0.8488  0.0833
1                    0.9487  0.9226    0.9468  0.0833
2                    0.2821  0.2803    0.2764  0.5833
3                    0.2051  0.2042    0.2028  0.5833
4                    1.0000  0.9614    0.9413  0.8333
..                      ...     ...       ...     ...
446                  0.3590  0.3630    0.3777  0.1667
```

3. separate linear scaling of each output variable to [0.1, 0.9] since the output of a sigmoid lies in the range (0.0, 1.0) .

```python
#preprocess output 5th column to scale it's values from 0.1 to 0.9
columns = df.shape[1]
outputcolumn = df.columns[4]
smin = 0.1
smax = 0.9

column_values = df[outputcolumn]
xmin = min(column_values)
xmax = max(column_values)
df_normalized[outputcolumn] = np.round(smin + ((smax - smin) / (xmax - xmin) ) * (df[outputcolumn] -
print(df_normalized)
```

Python

```
    #height_over_sea_level    fall  net_fall    flow  \
0                   0.8462  0.8212    0.8488  0.0833
1                   0.9487  0.9226    0.9468  0.0833
2                   0.2821  0.2803    0.2764  0.5833
3                   0.2051  0.2042    0.2028  0.5833
4                   1.0000  0.9614    0.9413  0.8333
..                     ...     ...       ...     ...
446                 0.3590  0.3630    0.3777  0.1667
447                 0.7692  0.7306    0.7035  1.0000
448                 0.4103  0.3780    0.3775  0.8333
449                 0.5385  0.5086    0.5215  0.5833
450                 0.4872  0.4630    0.4958  0.2500
```

4. write normalized csv data to “Normalized Data\A1-synthetic_normalized.txt” and “Normalized Data\A1-turbine_normalized.txt”

```python
# Write normalized DataFrame to a table-like format (CSV file)
df_normalized.to_csv('Normalized Data/A1-turbine_normalized.txt', index=False, sep='\t')
```

Python

MyNeuralNetwork.py M × | A1-turbine_normalized.txt ×

Normalized Data > A1-turbine_normalized.txt

```
#height_over_sea_level	fall	net_fall	flow	power_of_hydroelectrical_turbine
0.8462	0.8212	0.8488	0.0833	0.22
0.9487	0.9226	0.9468	0.0833	0.2301
0.2821	0.2803	0.2764	0.5833	0.397
0.2051	0.2042	0.2028	0.5833	0.3643
1.0	0.9614	0.9413	0.8333	0.8159
0.7436	0.7128	0.7237	0.5	0.5093
0.2564	0.2562	0.258	0.5	0.3473
0.7949	0.7971	0.8039	0.0	0.1507
0.2051	0.208	0.2212	0.3333	0.2472
0.5128	0.4845	0.5031	0.5	0.4397
0.0	0.0038	0.0191	0.4167	0.219
0.0	0.0051	0.0251	0.3333	0.1855
0.1795	0.1814	0.1907	0.4167	0.2798
0.6923	0.6659	0.6919	0.25	0.3179
0.4615	0.4338	0.4541	0.5	0.4236
0.4359	0.4059	0.4167	0.6667	0.5043
0.4615	0.4673	0.4852	0.0	0.1112
0.5897	0.5644	0.5938	0.25	0.2969
0.9487	0.9132	0.907	0.6667	0.6886
0.4359	0.4097	0.4358	0.4167	0.3661
0.8718	0.8346	0.8187	0.8333	0.765
0.3077	0.3123	0.3287	0.1667	0.1838
0.5128	0.4833	0.497	0.5833	0.4883
0.7436	0.7078	0.6962	0.8333	0.7123
0.9487	0.9145	0.9137	0.5833	0.6293
0.6667	0.6436	0.6772	0.0833	0.1956
0.1538	0.1585	0.1771	0.25	0.1918
```

Preprocess of Dataset 3 real estate price prediction

Source : https://www.kaggle.com/code/mehmetutkubala/real-estate-price-prediction

1. Get the real_estate.csv from above source
2. Change column names to be more readable using Jupiter Notebook

```
#data Preprocessing#  https://www.kaggle.com/code/mehmetutkubala/real-estate-price-prediction
df.rename(columns ={'X2 house age':'house_age'},inplace=True)
df.rename(columns ={'X3 distance to the nearest MRT station':'distance_to_the_nearest_MRT_station'},inplace=True)
df.rename(columns ={'X4 number of convenience stores':'number_of_convenience_stores'},inplace=True)
df.rename(columns ={'X5 latitude':'latitude'},inplace=True)
df.rename(columns ={'X6 longitude':'longitude'},inplace=True)
df.rename(columns ={'Y house price of unit area':'house_price_of_unit_area'},inplace=True)
df.rename(columns ={'X1 transaction date':'transaction_date'},inplace=True)
```

3. Change column transaction date to integer data type

```
df["transaction_date"]=df["transaction_date"].astype("int") #We change the type of data in transaction_date to inte

df.head()
```

4. Linearize to the range [0, 1] using sklearn.preprocessing MinMaxscaler

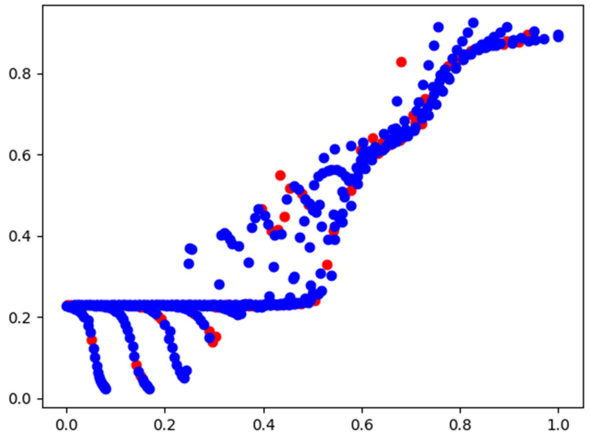
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
x = scaler.fit_transform(x)
y = scaler.fit_transform(y)

print(x)
```
✓ 0.2s

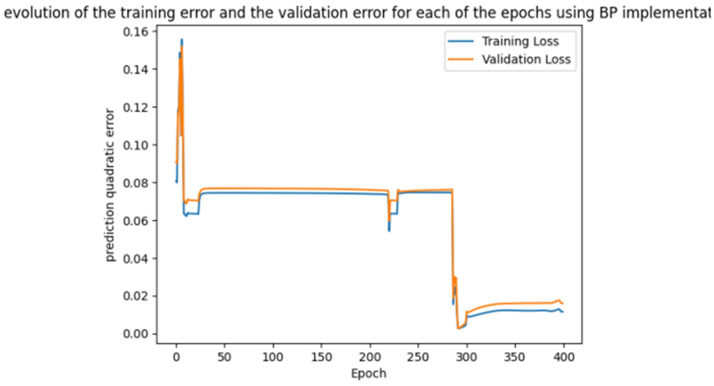
```
[[0.73059361 0.00951267 1.         0.61694135 0.71932284]
 [0.44520548 0.04380939 0.9        0.5849491  0.71145137]
 [0.30365297 0.08331505 0.5        0.67123122 0.75889584]
 ...
 [0.42922374 0.05686115 0.7        0.57149782 0.71522536]
 [0.18493151 0.0125958  0.5        0.42014057 0.72395946]
 [0.14840183 0.0103754  0.9        0.51211827 0.75016174]]
```

- **Part 2: Implementation of BP**

the plot of evolution of training and validation error with epoch obtained by training the model in myNeuralNetwork.py using dataset **A1-turbine_normalized.txt** with following parameters:

1. layers = [4, 9, 5, 1] , 1 input layer + 2 hidden layers + 1 output layer
2. epochs = 400
3. Learning rate = 0.1
4. Momentum = 0.9
5. Activation function = tanh
6. Validation percent = 20%
7. Weights initialized randomly between 0 and 1
8. Thresholds initialized randomly between 0 and 1

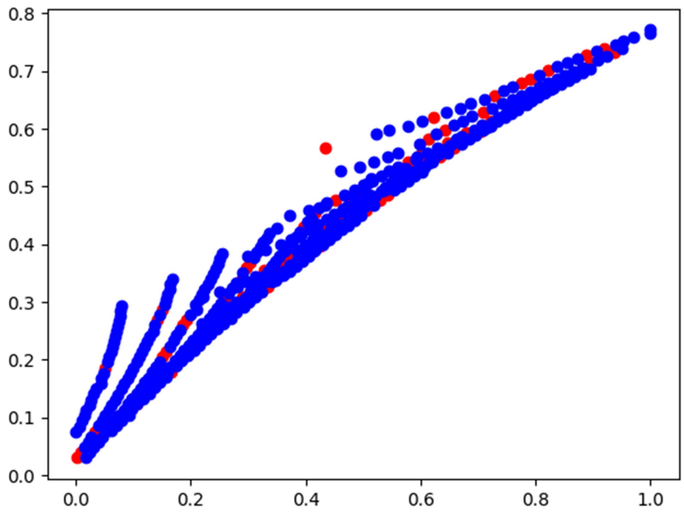


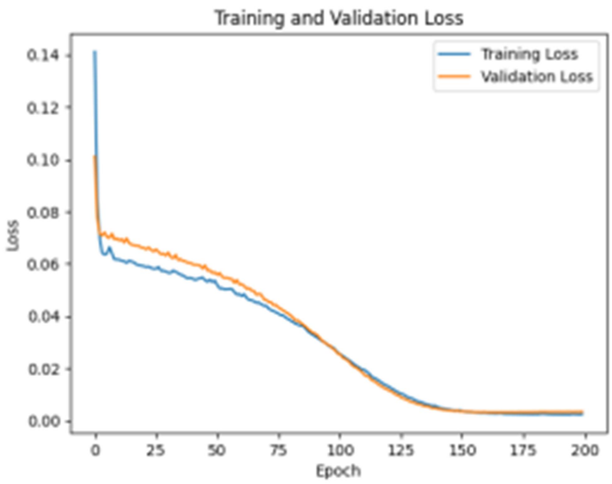


- **Part 3: Obtaining and comparing predictions**

  Using **BP-F**

  1. layers = [4, 9, 5, 1] , 1 input layer + 2 hidden layers + 1 output layer
  2. epochs = 200
  3. Learning rate = 0.01
  4. Momentum = 0.9
  5. Activation function = sigmoid
  6. Validation percent = 20%
  7. Weights initialized randomly between 0 and 1
  8. Thresholds initialized randomly between 0 and 1





Using **MLR-F**