2019103555 PRANAVA RAMAN B M S 25/10/2021

WEEK - 08 - MLP

1. Use Pima Indians Diabetes dataset.
Implement MLP. Use the sigmoidal activation function used in the algorithm in your text book. Fix the number of hidden neurons based on experimentation. With the same number of hidden neurons experiment using three different activation functions.

Definition of class MLP:-

```
import numpy as np
def tanh(z):
    return (np.exp(z) - np.exp(-z)) / (np.exp(z) + np.exp(-z))

class mlp:
    """A Multi-Layer Perceptron"""

    def __init__(
        self, inputs, targets, nhidden, beta=1, momentum=0.9, outtype="logistic"
    ):
        """Constructor"""
        self.nin = np.shape(inputs)[1]
        self.nout = np.shape(targets)[1]
        self.ndata = np.shape(inputs)[0]
        self.nhidden = nhidden

        self.beta = beta
        self.momentum = momentum
        self.outtype = outtype

        # Initialise network
        self.weights1 = (
            (np.random.rand(self.nin + 1, self.nhidden) - 0.5) * 2 / np.sqrt(self.nin)
        )
        self.weights2 = (
            (np.random.rand(self.nhidden + 1, self.nout) - 0.5)
            * 2
            / np.sqrt(self.nhidden)
        )

    def earlystopping(
        self, inputs, targets, valid, validtargets, eta, niterations=10000
    ):

        valid = np.concatenate((valid, -np.ones((np.shape(valid)[0], 1))), axis=1)

        old_val_error1 = 100002
        old_val_error2 = 100001
        new_val_error = 100000

        print("No. of neurons in hidden layers = ", self.nhidden)
        while ((old_val_error1 - new_val_error) > 0.001) or (
            (old_val_error2 - old_val_error1) > 0.001
        ):
            self.mlptrain(inputs, targets, eta, niterations)
            old_val_error2 = old_val_error1
            old_val_error1 = new_val_error
            validout = self.mlpfwd(valid)
            new_val_error = 0.5 * np.sum((validtargets - validout) ** 2)
```

```python
        print("Stopped, error = ", new_val_error)
        return new_val_error

    def mlptrain(self, inputs, targets, eta, niterations):
        """Train the neural network"""
        # Add the inputs that match the bias node
        inputs = np.concatenate((inputs, -np.ones((self.ndata, 1))), axis=1)
        change = range(self.ndata)

        updatew1 = np.zeros((np.shape(self.weights1)))
        updatew2 = np.zeros((np.shape(self.weights2)))

        for n in range(niterations):

            self.outputs = self.mlpfwd(inputs)

            error = 0.5 * np.sum((self.outputs - targets) ** 2)

            # Different types of output neurons and their activation functions
            if self.outtype == "linear":
                deltao = (self.outputs - targets) / self.ndata
            elif self.outtype == "logistic":
                deltao = (
                    self.beta
                    * (self.outputs - targets)
                    * self.outputs
                    * (1.0 - self.outputs)
                )
            elif self.outtype == "softmax":
                deltao = (
                    (self.outputs - targets)
                    * (self.outputs * (-self.outputs) + self.outputs)
                    / self.ndata
                )
            elif self.outtype == "tanh":
                deltao = (
                    (self.outputs - targets)
                    * (1.0 - np.power(self.outputs, 2))
                )
            else:
                print("error")

            # hidden network delta
            deltah = (
                self.hidden
                * self.beta
                * (1.0 - self.hidden)
                * (np.dot(deltao, np.transpose(self.weights2)))
            )

            updatew1 = (
                eta * (np.dot(np.transpose(inputs), deltah[:, :-1]))
                + self.momentum * updatew1
            )
            updatew2 = (
                eta * (np.dot(np.transpose(self.hidden), deltao))
                + self.momentum * updatew2
            )
            self.weights1 -= updatew1
            self.weights2 -= updatew2

        return error
```

```
    def mlpfwd(self, inputs):
        """Run the network forward"""

        self.hidden = np.dot(inputs, self.weights1)
        self.hidden = 1.0 / (1.0 + np.exp(-self.beta * self.hidden))
        self.hidden = np.concatenate(
            (self.hidden, -np.ones((np.shape(inputs)[0], 1))), axis=1
        )

        outputs = np.dot(self.hidden, self.weights2)

        # Different types of output neurons
        if self.outtype == "linear":
            return outputs
        elif self.outtype == "logistic":
            return 1.0 / (1.0 + np.exp(-self.beta * outputs))
        elif self.outtype == "softmax":
            normalisers = np.sum(np.exp(outputs), axis=1) * np.ones(
                (1, np.shape(outputs)[0])
            )
            return np.transpose(np.transpose(np.exp(outputs)) / normalisers)
        elif self.outtype == "tanh":
            return tanh(outputs)
        else:
            print("error")

    def confmat(self, inputs, targets):
        """Confusion matrix"""

        # Add the inputs that match the bias node
        inputs = np.concatenate((inputs, -np.ones((np.shape(inputs)[0], 1))), axis=1)
        outputs = self.mlpfwd(inputs)

        nclasses = np.shape(targets)[1]

        if nclasses == 1:
            nclasses = 2
            outputs = np.where(outputs > 0.5, 1, 0)
        else:
            # 1-of-N encoding
            outputs = np.argmax(outputs, 1)
            targets = np.argmax(targets, 1)

        cm = np.zeros((nclasses, nclasses))
        for i in range(nclasses):
            for j in range(nclasses):
                cm[i, j] = np.sum(
                    np.where(outputs == i, 1, 0) * np.where(targets == j, 1, 0)
                )
        output = cm
        print("Percentage Correct: ", np.trace(cm) / np.sum(cm) * 100)
        return output
```

Helper functions:-

To display confusion matrix-

```
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
```

```python
def displayConfusionMatrix(cm, plt):
    out_cm = np.array(cm)
    df_cm = pd.DataFrame(out_cm)
    plt.figure(figsize=(10,7))
    sn.set(font_scale=1)  # for label size
    sn.heatmap(df_cm, annot=True, annot_kws={"size": 14})  # font size
```

To show classification report:-

```python
from sklearn.metrics import classification_report
def printClassificationReport(network, test, testt):
    targets=testt
    inputs = np.concatenate((test, -np.ones((np.shape(test)[0], 1))), axis=1)
    nclasses = np.shape(targets)[1]
    output = network.mlpfwd(inputs)
    if nclasses == 1:
        nclasses = 2
        output = np.where(output > 0.5, 1, 0)
    else:
        # 1-of-N encoding
        output = np.argmax(output, 1)
        targets = np.argmax(targets, 1)


    print(classification_report(targets, output))
```

Including 'pima-indian-diabetes' dataset:-

loading data

```python
#loading data
dataset = np.loadtxt('diabetes.csv',delimiter=',')
no_of_columns = 8
#normalizing the input
dataset[:,:no_of_columns] = dataset[:,:no_of_columns]-dataset[:,:no_of_columns].mean(axis=0)
imax = np.concatenate((dataset.max(axis=0)
        *np.ones((1,no_of_columns+1)),np.abs(dataset.min(axis=0)
        *np.ones((1,no_of_columns+1)))),axis=0).max(axis=0)
dataset[:,:no_of_columns] = dataset[:,:no_of_columns]/imax[:no_of_columns]
```

[150] ✓ 0.2s Python

```python
print(dataset.shape)
```

[120] ✓ 0.3s Python

```
... (768, 9)
```

```python
# Split into training, validation, and test sets
target = np.zeros((np.shape(dataset)[0],2))
indices = np.where(dataset[:,no_of_columns]==0)
target[indices,0] = 1
indices = np.where(dataset[:,no_of_columns]==1)
target[indices,1] = 1

# Randomly order the data
order = np.arange(np.shape(dataset)[0])
```

```python
np.random.shuffle(order)
dataset = dataset[order,:]
target = target[order,:]

train = dataset[::2,0:no_of_columns]
traint = target[::2]
valid = dataset[1::4,0:no_of_columns]
validt = target[1::4]
test = dataset[3::4,0:no_of_columns]
testt = target[3::4]

print (train.max(axis=0), train.min(axis=0))
```

[121] ✓ 0.4s Python

```
[1.         0.64606288 0.64965237 1.         0.78334653 1.
 1.         1.        ] [-0.29228942 -1.         -1.         -0.26173249 -0.1041496  -0.91127677
 -0.20218239 -0.2563047 ]
```

Find the number of neurons

using sigmoid function

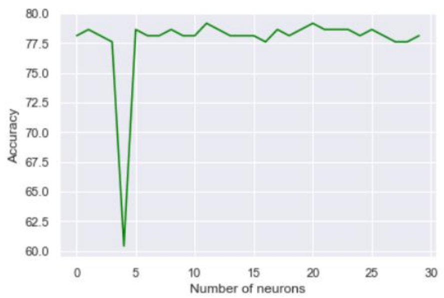
```python
print("Using Sigmoid function")
acc = np.zeros(30)
err = np.zeros(30)
cm = []
net = []
for i in range(30):
    net.append(mlp(train,traint,i+1, outtype='logistic'))
    err[i] = net[i].earlystopping(train,traint,valid,validt,0.001)
    #err = net.mlptrain(train, traint, 0.25, 10000)
    cm.append(net[i].confmat(test,testt))
    acc[i]= np.trace(cm[i]) / np.sum(cm[i]) * 100
```

[122] ✓ 20.6s Python

```
Using Sigmoid function
No. of neurons in hidden layers =  1
Stopped, error =  31.96408393423567
Percentage Correct:  78.125
No. of neurons in hidden layers =  2
Stopped, error =  32.141457877281724
Percentage Correct:  78.64583333333334
No. of neurons in hidden layers =  3
Stopped, error =  32.05194270972359
Percentage Correct:  78.125
No. of neurons in hidden layers =  4
Stopped, error =  32.03907183671007
Percentage Correct:  77.60416666666666
```

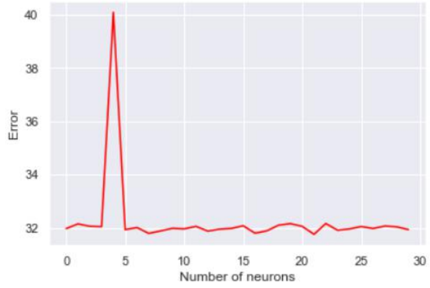
```python
plt.xlabel("Number of neurons")
plt.ylabel("Accuracy")
plt.plot(acc, color='green')
plt.show()
```

[123] ✓ 0.2s Python



```python
plt.xlabel("Number of neurons")
plt.ylabel("Error")
plt.plot(err, color='red')
plt.show()
```

[124] ✓ 0.2s Python




```python
n = np.argmax(acc)
print("Number of neurons for maximum accuracy =", n)
print("Accuracy = ", acc[n])
```

[125] ✓ 0.3s Python

```
Number of neurons for maximum accuracy = 11
Accuracy =  79.16666666666666
```

```python
cm[n]
```

[126] ✓ 0.3s Python

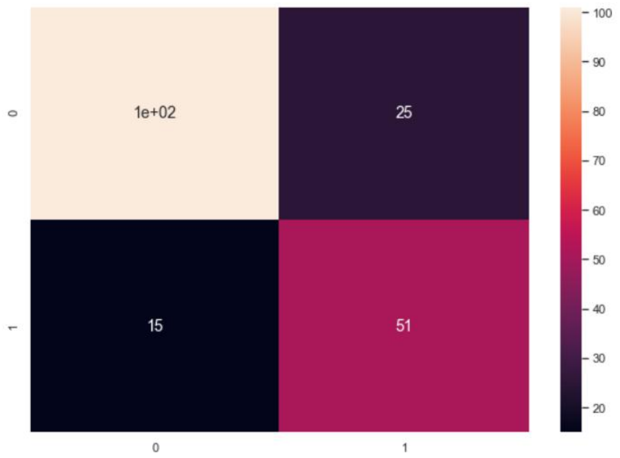
```
array([[101.,  25.],
       [ 15.,  51.]])
```

Using 11 neurons gave the maximum accuracy for sigmoid activation function.

```python
print("Using sigmoid function")
displayConfusionMatrix(cm[n],plt)
plt.show()
```

[127] ✓ 0.2s Python

```
Using sigmoid function
```




```python
print("Classification report using sigmoid activation function")
printClassificationReport(net[n], test, testt)
```

[128] ✓ 0.3s Python

```
Classification report using sigmoid activation function
              precision    recall  f1-score   support

           0       0.80      0.87      0.83       116
           1       0.77      0.67      0.72        76

    accuracy                           0.79       192
   macro avg       0.79      0.77      0.78       192
weighted avg       0.79      0.79      0.79       192
```

Trying out different activation functions:-

I. Softmax activation function

softmax activation function

```
print("Using Softmax activation function")
softnet = mlp(train,traint,n, outtype='softmax')
softnet.earlystopping(train,traint,valid,validt,0.25)
softcm = softnet.confmat(test, testt)
```

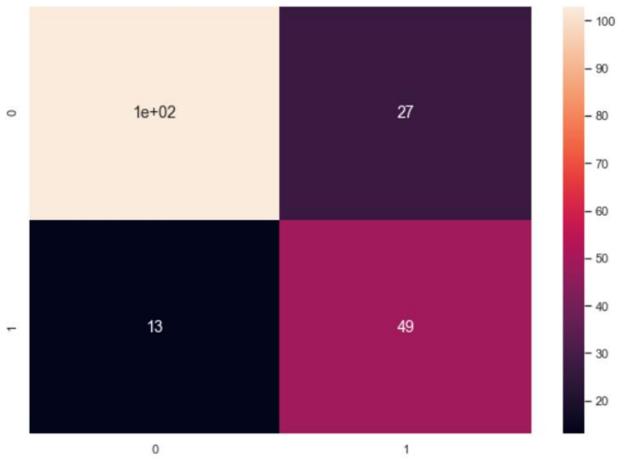
[139] ✓ 1.2s Python

```
Using Softmax activation function
No. of neurons in hidden layers =  11
Stopped, error =  32.36775940519345
Percentage Correct:  79.16666666666666
```

```
print("Using softmax function")
displayConfusionMatrix(softcm,plt)
plt.show()
```

[140] ✓ 0.2s Python

```
Using softmax function
```




```
print("Classification report using Softmax activation function")
printClassificationReport(softnet, test, testt)
```

[141] ✓ 0.3s Python

```
Classification report using Softmax activation function
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       116
           1       0.79      0.64      0.71        76

    accuracy                           0.79       192
   macro avg       0.79      0.77      0.77       192
weighted avg       0.79      0.79      0.79       192
```

II. Linear activation function

Linear activation function

```
print("Using Linear activation function")
signet = mlp(train,traint,n, outtype='linear')
signet.earlystopping(train,traint,valid,validt,0.25)
sigcm = signet.confmat(test, testt)
```

[132] ✓ 0.3s Python

```
Using Linear activation function
No. of neurons in hidden layers =  11
Stopped, error =  32.38248436034704
Percentage Correct:  77.60416666666666
```

```
print("Using linear activation function")
displayConfusionMatrix(sigcm,plt)
plt.show()
```

[133] ✓ 0.2s Python

```
Using linear activation function
```


```
print("Classification report using linear activation function")
printClassificationReport(signet, test, testt)
```

[134] ✓ 0.6s Python

```
Classification report using linear activation function
              precision    recall  f1-score   support

           0       0.79      0.86      0.82       116
           1       0.75      0.64      0.70        76

    accuracy                           0.78       192
   macro avg       0.77      0.75      0.76       192
weighted avg       0.77      0.78      0.77       192
```

III. Tanh activation function

Tanh activation function

```
print("Using tanh activation function")
tanhnet = mlp(train,traint,n, outtype='tanh')
#tanhnet.mlptrain(train, traint, 0.25, 20000)
tanhnet.earlystopping(train,traint,valid,validt,0.0001)
tanhcm = tanhnet.confmat(test, testt)
```

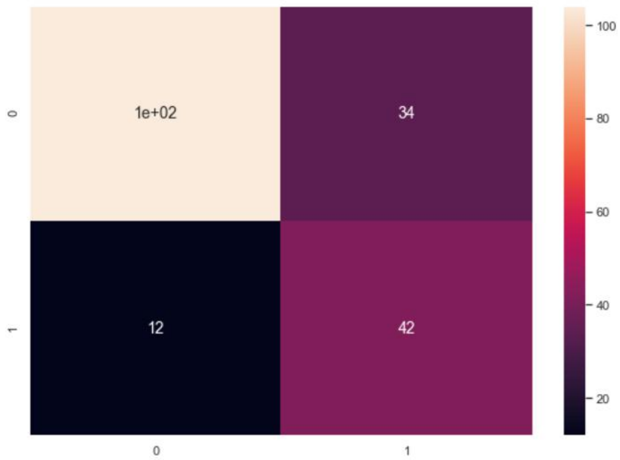
[147] ✓ 1.4s Python

```
Using tanh activation function
No. of neurons in hidden layers =  11
Stopped, error =  33.152478697860715
Percentage Correct:  76.04166666666666
```

```
print("Using tanh activation function")
displayConfusionMatrix(tanhcm,plt)
plt.show()
```

[148] ✓ 0.2s Python

```
Using tanh activation function
```




```python
print("Classification report using tanh activation function")
printClassificationReport(tanhnet, test, testt)
```

```
Classification report using tanh activation function
              precision    recall  f1-score   support

           0       0.75      0.90      0.82       116
           1       0.78      0.55      0.65        76

    accuracy                           0.76       192
   macro avg       0.77      0.72      0.73       192
weighted avg       0.76      0.76      0.75       192
```

Result:-

For the given dataset, both Sigmoid and Softmax give the highest accuracy of about 79% while linear activation gives 77% accuracy and tanh follows very closely by 76% of test accuracy.

2. Choose a dataset suitable for regression and apply regression using MLP

Using real estate dataset,

```python
df1
```

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
0	1	2012.917	32.0	84.87882	10	24.98298	121.54024	37.9
1	2	2012.917	19.5	306.59470	9	24.98034	121.53951	42.2
2	3	2013.583	13.3	561.98450	5	24.98746	121.54391	47.3
3	4	2013.500	13.3	561.98450	5	24.98746	121.54391	54.8
4	5	2012.833	5.0	390.56840	5	24.97937	121.54245	43.1
...	...	...	...	...	...	...	...	...
409	410	2013.000	13.7	4082.01500	0	24.94155	121.50381	15.4
410	411	2012.667	5.6	90.45606	9	24.97433	121.54310	50.0
411	412	2013.250	18.8	390.96960	7	24.97923	121.53986	40.6
412	413	2013.000	8.1	104.81010	5	24.96674	121.54067	52.5
413	414	2013.500	6.5	90.45606	9	24.97433	121.54310	63.9

414 rows × 8 columns

```python
import numpy as np
import pandas as pd

from sklearn.neural_network import MLPRegressor
from sklearn.metrics import r2_score
df1 = pd.read_csv('Real estate.csv')

data1 = df1.drop(['Y house price of unit area'], axis = 1)
target1 = df1['Y house price of unit area']

from sklearn.model_selection import train_test_split
datasets = train_test_split(data1, target1,
                            test_size=0.3)

X_train, X_test, y_train, y_test = datasets
```

Python

```python
reg = MLPRegressor(hidden_layer_sizes=(64,64,64),activation="relu" , max_iter=5000).fit(X_train, y_train)
```

Python

```python
y_pred=reg.predict(X_test)
print("The Score with ",(r2_score(y_pred, y_test)))
```

Python

```
The Score with  0.5128951299414292
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('ggplot')
plt.figure(figsize=(10,10))
sns.regplot(y_test, y_pred, fit_reg=True, scatter_kws={"s": 100})
plt.show()
```

[6] ✓ 2.1s Python