Multi-layer Perceptron

1. Components: Input layer, Hidden layer(s) and Output layer.
2. Fully connected

Steps:

For every epoch,

1. Training data is propagated to the MLP through input layers. It passes through the hidden layers, if any forwarding outputs of activation functions to the next layer. Finally the output is generated at the output layer by applying activation functions.
2. The predicted output will be compared with actual output and hence error will be calculated.
3. If error>0, apply backpropagation methodology to modify weights starting from output layer moving towards input layer.
4. Check accuracy score. If satisfied, stop. Else, go to step 1.

Bank Notes dataset

Download from [https://www.kaggle.com/aariyan101/bank-notes (https://www.kaggle.com/aariyan101/bank-notes)](https://www.kaggle.com/aariyan101/bank-notes)

Attributes

1. Variance of image
2. Skewness of image
3. Curtosis of image
4. Entropy
5. Class

Import dataset

In [1]:

```
import pandas as pd
import numpy as np

bnotes = pd.read_csv("C:\\Users\\kgnan\\Downloads\\bank_note_data.csv")
print(bnotes.head())
print(bnotes['class'].unique())
```

```
   variance  skewness  curtosis  entropy  class
0   3.62160    8.6661   -2.8073 -0.44699      0
1   4.54590    8.1674   -2.4586 -1.46210      0
2   3.86600   -2.6383    1.9242  0.10645      0
3   3.45660    9.5228   -4.0112 -3.59440      0
4   0.32924   -4.4552    4.5718 -0.98880      0
[0 1]
```

In [ ]:

```

```

In [2]:

```
bnotes.shape
```

Out[2]:

```
(1372, 5)
```

In [3]:

```
bnotes.describe(include = 'all')
```

Out[3]:

	variance	skewness	curtosis	entropy	class
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	0.000000
50%	0.496180	2.319650	0.616630	-0.586650	0.000000
75%	2.821475	6.814625	3.179250	0.394810	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

In [4]:

```
X = bnotes.drop('class', axis=1)
y = bnotes['class']
print(X.head(2))
print(y.head(2))
```

```
   variance  skewness  curtosis  entropy
0    3.6216    8.6661   -2.8073 -0.44699
1    4.5459    8.1674   -2.4586 -1.46210
0    0
1    0
Name: class, dtype: int64
```

Splitting to training and testing

In [5]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3)
print(X_train.shape)
print(y_test.shape)
```

```
(960, 4)
(412,)
```

Normalized input X train

Train the model

Import the MLP classifier model from sklearn

In [7]:

```
from sklearn.neural_network import MLPClassifier
```

In [16]:

```
mlp = MLPClassifier(max_iter=500, activation='relu')
mlp
```

Out[16]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=
0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=500,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

About parameters

1. hidden_layer_sizes : tuple, length = n_layers - 2, default (100,)

The ith element represents the number of neurons in the ith hidden layer.

2. activation : {'identity', 'logistic', 'tanh', 'relu'}, default 'relu'

Activation function for the hidden layer.

'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)). 'tanh', the hyperbolic tan function, returns f(x) = tanh(x). 'relu', the rectified linear unit function, returns f(x) = max(0, x)

3. learning_rate : {'constant', 'invscaling', 'adaptive'}, default 'constant'

4. learning_rate_init : double, optional, default 0.001
5. max_iter : int, optional, default 200

Maximum number of iterations. The solver iterates until convergence (determined by ‘tol’) or this number of iterations. For stochastic solvers (‘sgd’, ‘adam’), note that this determines the number of epochs (how many times each data point will be used), not the number of gradient steps.

6. shuffle : bool, optional, default True

Whether to shuffle samples in each iteration. Only used when solver=’sgd’ or ‘adam’.

7. momentum : float, default 0.9

Momentum for gradient descent update. Should be between 0 and 1. Only used when solver=’sgd’.

8. early_stopping : bool, default False

Whether to use early stopping to terminate training when validation score is not improving. If set to true, it will automatically keep 10% of training data as validation and terminate training when validation score is not improving by at least tol for two consecutive epochs. Only effective when solver=’sgd’ or ‘adam’

Training

In [17]:

```
mlp.fit(X_train,y_train)
```

Out[17]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=
0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=500,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

Testing

In [18]:

```
pred = mlp.predict(X_test)
pred
```

Out[18]:

```
array([1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0], dtype=int64)
```

Evaluation metrics- Confusion matrix and F2 score

In [19]:

```
from sklearn.metrics import classification_report,confusion_matrix

confusion_matrix(y_test,pred)
```

Out[19]:

```
array([[211,   0],
       [  0, 201]], dtype=int64)
```

In [20]:

```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       211
           1       1.00      1.00      1.00       201

    accuracy                           1.00       412
   macro avg       1.00      1.00      1.00       412
weighted avg       1.00      1.00      1.00       412
```

TOTAL CODE :

In [2]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report,confusion_matrix
bnotes = pd.read_csv("C:\\Users\\kgnan\\Downloads\\bank_note_data.csv")
X = bnotes.drop('Class', axis=1)
y = bnotes['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3)

mlp = MLPClassifier(hidden_layer_sizes=(64,60,50),max_iter=500, activation='relu')
mlp.fit(X_train,y_train)
pred = mlp.predict(X_test)
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[221   0]
 [  0 191]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       221
           1       1.00      1.00      1.00       191

    accuracy                           1.00       412
   macro avg       1.00      1.00      1.00       412
weighted avg       1.00      1.00      1.00       412
```

In [ ]:

```

```