In [1]:
```
import numpy as np
import pandas as pd
```

Loading and Procession of dataset

In [2]:
```
data = pd.read_csv("C:\\Users\\ASUS\\titanic_processed.csv")
x = data.drop("Survived",axis=1).values
y = data["Survived"].values
```

In [3]:
```
hidden_layers = [5,3]
lr = 30
n_epochs = 20
batch_size = 25
n = len(hidden_layers)+1
```

In [4]:
```
s = int(0.8*x.shape[0])
x_train = x[:s].T
y_train = y[:s]
outputs = [None]*4
delta = [None]*3
x_test = x[s:]
y_test = y[s:]
n_input = x_train.shape[0]
z = y_train.reshape(-1,1)
classes = np.unique(y_train)
target = (z==classes).T
n_output = target.shape[0]
m = x_train.shape[1]
```

In [5]:
```
np.random.seed(0)
weights = [np.random.standard_normal((o,i+1))*0.01
           for i,o in zip([n_input]+hidden_layers,hidden_layers+[n_output])]
```

In [6]:
```
def act(x):
    return 1/(1+np.exp(-x))
```

In [7]:
```
def dact(x):
    return x*(1-x)
```

In [8]:
```
def pad(x):
    pad_w = [(1,0),(0,0)]
    return np.pad(x,pad_w,constant_values=1)
```

In [9]:
```
def forward_propagate(n,input):
    outputs[-1] = input
    for i in range(n):
```

```
        outputs[i] = act(weights[i]@pad(outputs[i-1]))
    return outputs[n-1]
```

In [10]:

```
def error(actual,predicted):
    error = actual - predicted
    return np.sum(error*error)
```

In [11]:

```
def back_propagate(n,target):
    for i in range(n-1,-1,-1):
        if i == n-1:
            errors = outputs[i]-target
        else:
            errors = weights[i+1][:,1:].T@delta[i+1]
        delta[i] = errors*dact(outputs[i])
```

In [12]:

```
def update_weight(n,lr,batch_size):
    for i in range(n):
        weights[i] -= lr*delta[i]@pad(outputs[i-1]).T/batch_size
```

In [13]:

```
def predict(n,inputs):
    pred = forward_propagate(n,inputs.T).T.argmax(axis=-1)
    return pred
```

In [14]:

```
def accuracy(x_test,y_test,n):
    return (predict(n,x_test)==y_test).mean()
```

Implementing BPNN

In [15]:

```
verbose = True
if verbose:
    print("Initital Weights:")
    print(*weights,sep="\n")
    print("Training:")

for epoch in range(n_epochs):
    sum_error = 0
    fs = range(m+batch_size)
    for f,t in zip(fs,fs[1:]):
        forwd_out = forward_propagate(n,x_train[:,f:t])
        sum_error += error(target[:,f:t],forwd_out)
        back_propagate(n,target[:,f:t])
        update_weight(n,lr,batch_size)
    if verbose:
        print(f'> epoch={epoch+1}, lrate={lr:.1f}, error={sum_error/m:.5f}')


if verbose:
    print("Trained Weights:")
    print(*weights,sep="\n")
    print("Evaluation:")
    print(f"Accuracy of the classifer:{accuracy(x_test,y_test,n)*100:.2f}%")
```

```
Initital Weights:
[[ 0.01764052  0.00400157  0.00978738  0.02240893  0.01867558 -0.00977278
   0.00950088 -0.00151357 -0.00103219  0.00410599]
 [ 0.00144044  0.01454274  0.00761038  0.00121675  0.00443863  0.00333674
   0.01494079 -0.00205158  0.00313068 -0.00854096]
 [-0.0255299   0.00653619  0.00864436 -0.00742165  0.02269755 -0.01454366
   0.00045759 -0.00187184  0.01532779  0.01469359]
 [ 0.00154947  0.00378163 -0.00887786 -0.01980796 -0.00347912  0.00156349
   0.01230291  0.0120238  -0.00387327 -0.00302303]
 [-0.01048553 -0.01420018 -0.0170627   0.01950775 -0.00509652 -0.00438074
  -0.01252795  0.0077749  -0.01613898 -0.0021274 ]]
[[-0.00895467  0.00386902 -0.00510805 -0.01180632 -0.00028182  0.00428332]
 [ 0.00066517  0.00302472 -0.00634322 -0.00362741 -0.0067246  -0.00359553]
 [-0.00813146 -0.01726283  0.00177426 -0.00401781 -0.01630198  0.00462782]]
[[-0.00907298  0.00051945  0.00729091  0.00128983]
 [ 0.01139401 -0.01234826  0.00402342 -0.0068481 ]]
Training:
> epoch=1, lrate=30.0, error=0.49509
> epoch=2, lrate=30.0, error=0.49015
> epoch=3, lrate=30.0, error=0.42328
> epoch=4, lrate=30.0, error=0.32519
> epoch=5, lrate=30.0, error=0.31955
> epoch=6, lrate=30.0, error=0.31736
> epoch=7, lrate=30.0, error=0.31585
> epoch=8, lrate=30.0, error=0.31467
> epoch=9, lrate=30.0, error=0.31368
> epoch=10, lrate=30.0, error=0.31279
> epoch=11, lrate=30.0, error=0.31195
> epoch=12, lrate=30.0, error=0.31113
> epoch=13, lrate=30.0, error=0.31029
> epoch=14, lrate=30.0, error=0.30946
> epoch=15, lrate=30.0, error=0.30865
> epoch=16, lrate=30.0, error=0.30789
> epoch=17, lrate=30.0, error=0.30718
> epoch=18, lrate=30.0, error=0.30652
> epoch=19, lrate=30.0, error=0.30591
> epoch=20, lrate=30.0, error=0.30533
Trained Weights:
[[ 2.59377565 -3.30489323 -1.36582057  0.55108638  0.54484425 -3.2269304
   0.40831845 -0.37603816 -0.49050768 -3.62382812]
 [ 2.55310358 -3.23245764 -1.35879432  0.50578107  0.51561022 -3.1856521
   0.4094337  -0.39589713 -0.53693558 -3.54442153]
 [ 2.4621099  -3.18672024 -1.30013933  0.49196919  0.55699735 -3.27845112
   0.4317014  -0.42330159 -0.44407793 -3.51849201]
 [ 2.56214525 -3.26541624 -1.36150036  0.51779539  0.50778396 -3.20546279
   0.42429237 -0.38049015 -0.51475034 -3.5775894 ]
 [ 2.62232846 -3.30355771 -1.44113874  0.54280011  0.47086607 -2.99803274
   0.35224981 -0.37702196 -0.64849008 -3.48690878]]
[[ 0.41666756 -2.29532261 -2.27306919 -2.26138852 -2.27356199 -2.2601175 ]
 [ 2.14367882 -2.05891705 -2.00511851 -2.01911792 -2.02489772 -1.93553795]
 [ 1.5681226  -2.13860288 -2.06685704 -2.07847008 -2.09969811 -2.01787684]]
[[-2.63598113  1.55565258  1.8048985   1.630234  ]
 [ 2.63438194 -1.56389923 -1.79391282 -1.63487603]]
Evaluation:
Accuracy of the classifer:85.96%
```