Name:Subham Beura

Branch : CE

ID : B521060

LAB 5

Build an Artificial Neural Network by implementing the Back-propagation algorithm and test

the same using appropriate data sets.

Note-Use basic feed forward neural network with one hidden layer and Evaluate the model.

Dataset provided on Lab1

```
import numpy as np
import pandas as pd

df = pd.read_csv('lab1_dataset.csv')
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	targe
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	

```
X = df.drop(columns='target')
y = df['target']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state = 42)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)

X_train_scaled.shape, X_test_scaled.shape
```

```
((242, 13), (61, 13))
```

```
print(X_train_scaled)
```

```
[[-1.35679832  0.72250438  0.00809909 ...  0.95390513 -0.68970073
 -0.50904773]
 [ 0.38508599  0.72250438 -0.97189094 ...  0.95390513 -0.68970073
 1.17848036]
 [-0.92132724  0.72250438  0.98808912 ... -0.69498803 -0.68970073
 -0.50904773]
 ...
 [ 1.58263146  0.72250438  1.96807914 ... -0.69498803  0.32186034
 -0.50904773]
 [-0.92132724  0.72250438 -0.97189094 ...  0.95390513 -0.68970073
 1.17848036]
 [ 0.92942484 -1.38407465  0.00809909 ...  0.95390513  1.33342142
  -0.50904773]]
```

Explanation : Initialize the hyper_parameters value randomly and accordingly compute on different activation functions to reduce loss function, setting the hyper_parameters for optimal solution.

```
input_size = X_train_scaled.shape[1]
hidden_size = 15
output_size = 1
learning_rate = 0.1
epochs = 1200

np.random.seed(42)
W1 = np.random.randn(input_size, hidden_size) * 0.01
b1 = np.zeros((1, hidden_size))
W2 = np.random.randn(hidden_size, output_size) * 0.01
b2 = np.zeros((1, output_size))

def sigmoid(z):
 return 1 / (1 + np.exp(-z))

def sigmoid_derivative(z):
 return z * (1 - z)

def tanh(z):
 return np.tanh(z)

def tanh_derivative(z):
 return (1 - np.tanh(z)**2)

def relu(z):
 return np.maximum(0, z)

def relu_derivative(z):
 return np.where(z > 0, 1, 0)
```

```
def forward_propagation(X):
 Z1 = np.dot(X, W1) + b1
 A1 = sigmoid(Z1)
 Z2 = np.dot(A1, W2) + b2
 A2 = sigmoid(Z2)
    return Z1, A1, Z2, A2

def backward_propagation(X, y, Z1, A1, Z2, A2):
    m = X.shape[0]
    dZ2 = A2 - y.reshape(m, 1)
    dW2 = np.dot(A1.T, dZ2) / m
    db2 = np.sum(dZ2, axis=0, keepdims=True) / m
    dA1 = np.dot(dZ2, W2.T)
    dZ1 = dA1 * sigmoid_derivative(A1)
    dW1 = np.dot(X.T, dZ1) / m
    db1 = np.sum(dZ1, axis=0, keepdims=True) / m
    return dW1, db1, dW2, db2

y_train_reshaped  = y_train.values.reshape(-1, 1)
loss_values = []
epoch_values = []
m = X_train_scaled.shape[0]
for epoch in range(epochs):
    Z1, A1, Z2, A2 = forward_propagation(X_train_scaled)

    dW1, db1, dW2, db2 = backward_propagation(X_train_scaled,
y_train_reshaped, Z1, A1, Z2, A2)

    W1 -= learning_rate * dW1
    b1 -= learning_rate * db1
    W2 -= learning_rate * dW2
    b2 -= learning_rate * db2

    if epoch % 10 == 0:
        loss = -np.mean(y_train_reshaped*np.log(A2) + (1 -
y_train_reshaped)*np.log(1 - A2))
        print(f"Epoch {epoch}, Loss: {loss:.4f}")
        loss_values.append(loss)
        epoch_values.append(epoch)

print("Training complete!")
```

Explanation : Create forward and backward propagation functions and define the mathematical equations for both the functions, since only one hidden layer and one output layer is used hence out of all the activation function sigmoid function can be used to categorize the output either towards 0 or 1 value, Hence a threshold is set to correctly assign the output values.

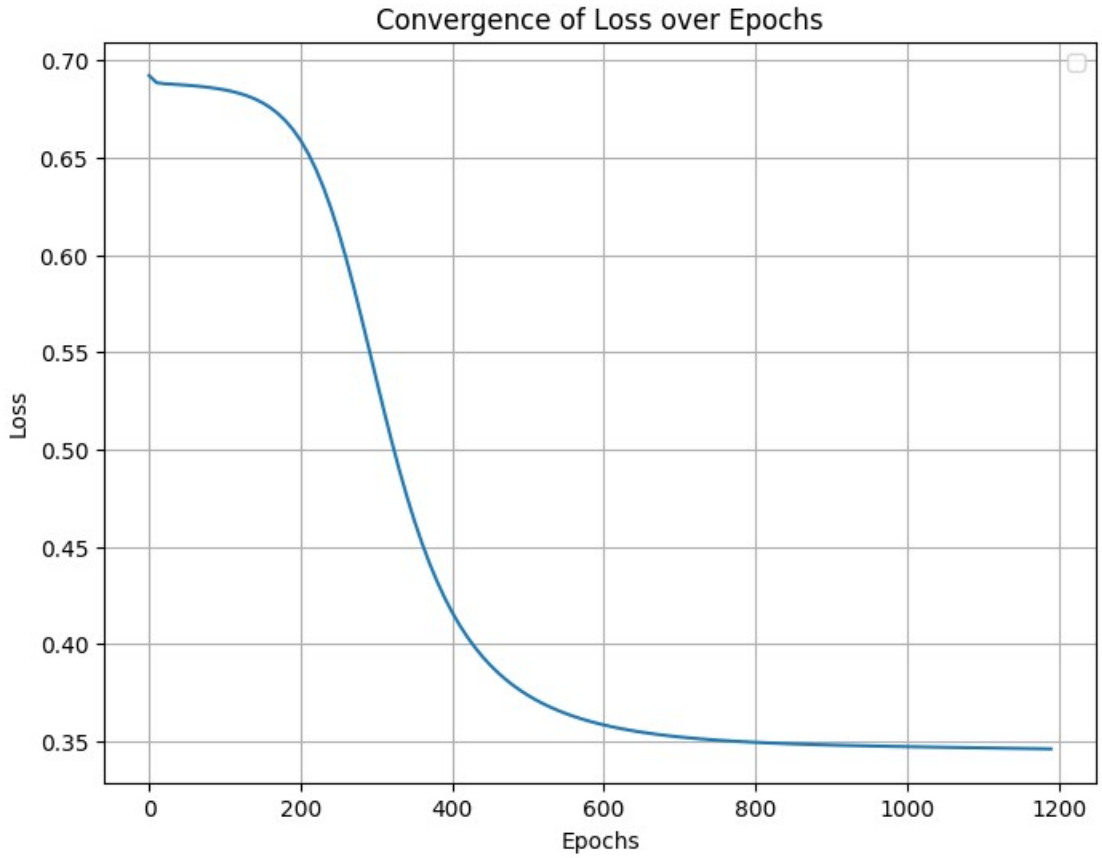
```
Epoch 0, Loss: 0.6921
Epoch 10, Loss: 0.6884
Epoch 20, Loss: 0.6879
Epoch 30, Loss: 0.6877
Epoch 40, Loss: 0.6874
Epoch 50, Loss: 0.6871
Epoch 60, Loss: 0.6868
Epoch 70, Loss: 0.6864
Epoch 80, Loss: 0.6860
Epoch 90, Loss: 0.6854
Epoch 100, Loss: 0.6847
Epoch 110, Loss: 0.6839
Epoch 120, Loss: 0.6829
Epoch 130, Loss: 0.6816
Epoch 140, Loss: 0.6800
Epoch 150, Loss: 0.6780
Epoch 160, Loss: 0.6756
Epoch 170, Loss: 0.6726
Epoch 180, Loss: 0.6688
Epoch 190, Loss: 0.6643
Epoch 200, Loss: 0.6587
Epoch 210, Loss: 0.6520
Epoch 220, Loss: 0.6440
Epoch 230, Loss: 0.6347
Epoch 240, Loss: 0.6239
Epoch 250, Loss: 0.6116
Epoch 260, Loss: 0.5981
Epoch 270, Loss: 0.5834
Epoch 280, Loss: 0.5679
Epoch 290, Loss: 0.5518
Epoch 300, Loss: 0.5356
Epoch 310, Loss: 0.5197
Epoch 320, Loss: 0.5042
Epoch 330, Loss: 0.4896
Epoch 340, Loss: 0.4759
Epoch 350, Loss: 0.4633
Epoch 360, Loss: 0.4518
Epoch 370, Loss: 0.4414
Epoch 380, Loss: 0.4320
Epoch 390, Loss: 0.4237
Epoch 400, Loss: 0.4162
Epoch 410, Loss: 0.4095
Epoch 420, Loss: 0.4035
Epoch 430, Loss: 0.3982
Epoch 440, Loss: 0.3934
Epoch 450, Loss: 0.3891
Epoch 460, Loss: 0.3853
Epoch 470, Loss: 0.3819
Epoch 480, Loss: 0.3788
Epoch 490, Loss: 0.3760
```

```
Epoch 500, Loss: 0.3735
Epoch 510, Loss: 0.3713
Epoch 520, Loss: 0.3692
Epoch 530, Loss: 0.3674
Epoch 540, Loss: 0.3657
Epoch 550, Loss: 0.3642
Epoch 560, Loss: 0.3628
Epoch 570, Loss: 0.3615
Epoch 580, Loss: 0.3603
Epoch 590, Loss: 0.3593
Epoch 600, Loss: 0.3583
Epoch 610, Loss: 0.3574
Epoch 620, Loss: 0.3566
Epoch 630, Loss: 0.3559
Epoch 640, Loss: 0.3552
Epoch 650, Loss: 0.3546
Epoch 660, Loss: 0.3540
Epoch 670, Loss: 0.3535
Epoch 680, Loss: 0.3530
Epoch 690, Loss: 0.3526
Epoch 700, Loss: 0.3521
Epoch 710, Loss: 0.3518
Epoch 720, Loss: 0.3514
Epoch 730, Loss: 0.3511
Epoch 740, Loss: 0.3508
Epoch 750, Loss: 0.3505
Epoch 760, Loss: 0.3502
Epoch 770, Loss: 0.3500
Epoch 780, Loss: 0.3498
Epoch 790, Loss: 0.3496
Epoch 800, Loss: 0.3494
Epoch 810, Loss: 0.3492
Epoch 820, Loss: 0.3490
Epoch 830, Loss: 0.3489
Epoch 840, Loss: 0.3487
Epoch 850, Loss: 0.3486
Epoch 860, Loss: 0.3484
Epoch 870, Loss: 0.3483
Epoch 880, Loss: 0.3482
Epoch 890, Loss: 0.3481
Epoch 900, Loss: 0.3480
Epoch 910, Loss: 0.3479
Epoch 920, Loss: 0.3478
Epoch 930, Loss: 0.3477
Epoch 940, Loss: 0.3476
Epoch 950, Loss: 0.3475
Epoch 960, Loss: 0.3475
Epoch 970, Loss: 0.3474
Epoch 980, Loss: 0.3473
Epoch 990, Loss: 0.3472
```

```
Epoch 1000, Loss: 0.3472
Epoch 1010, Loss: 0.3471
Epoch 1020, Loss: 0.3470
Epoch 1030, Loss: 0.3470
Epoch 1040, Loss: 0.3469
Epoch 1050, Loss: 0.3468
Epoch 1060, Loss: 0.3468
Epoch 1070, Loss: 0.3467
Epoch 1080, Loss: 0.3467
Epoch 1090, Loss: 0.3466
Epoch 1100, Loss: 0.3465
Epoch 1110, Loss: 0.3465
Epoch 1120, Loss: 0.3464
Epoch 1130, Loss: 0.3464
Epoch 1140, Loss: 0.3463
Epoch 1150, Loss: 0.3462
Epoch 1160, Loss: 0.3462
Epoch 1170, Loss: 0.3461
Epoch 1180, Loss: 0.3461
Epoch 1190, Loss: 0.3460
Training complete!
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.plot(epoch_values, loss_values)
plt.title('Convergence of Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in
legend.  Note that artists whose label start with an underscore are
ignored when legend() is called with no argument.
```



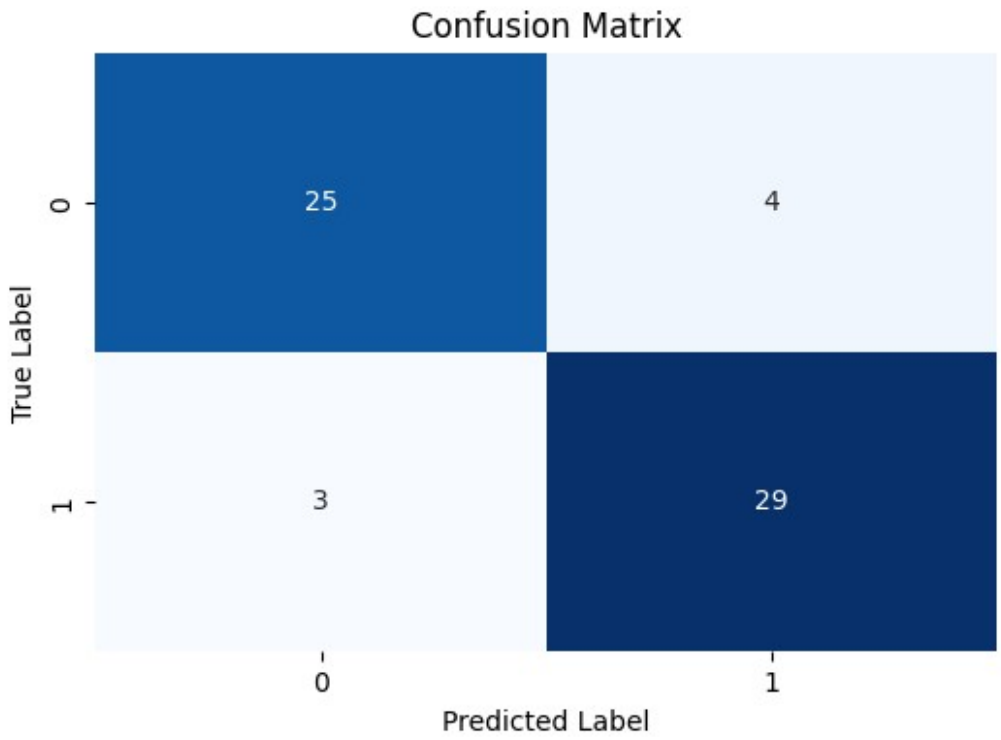

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

def predict(X):
    Z1 = np.dot(X, W1) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(A1, W2) + b2
    A2 = sigmoid(Z2)
    return A2


y_pred_proba = predict(X_test_scaled)
print(y_pred_proba.reshape(-1))

y_pred = (y_pred_proba >= 0.2).astype(int)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(" \nScores : \n")
print(f"Accuracy: {accuracy:.4f}")
```

```
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

```
[0.07334519 0.67066967 0.8475168  0.02794635 0.95854798 0.92170618
 0.66103709 0.01069371 0.01560589 0.40443715 0.80638491 0.05081896
 0.92660582 0.04218059 0.98452349 0.95275905 0.98318778 0.06899469
 0.017313   0.021017   0.62142364 0.02069186 0.1600701  0.77807716
 0.95613507 0.67867444 0.90991865 0.5676288  0.01767621 0.95144128
 0.03758224 0.03290468 0.01682934 0.08204604 0.79475799 0.05238676
 0.71481608 0.89676919 0.83911517 0.87068146 0.38941294 0.80253712
 0.87012304 0.74687617 0.87265741 0.01711923 0.83995765 0.96094906
 0.07568444 0.02553741 0.05808168 0.01883186 0.91657739 0.97765059
 0.12705828 0.01074194 0.03522049 0.96671793 0.02359578 0.01289406
 0.0403482 ]

Scores :

Accuracy: 0.8852
Precision: 0.8788
Recall: 0.9062
F1-Score: 0.8923
```

Explanation : Create a function for the test dataset and call the neural network for sigmoid activation function to find out the output, produce the scores for the test result and compare to the actual values to understand the relation and optimize the problem.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
cbar=False, xticklabels=[0, 1], yticklabels=[0, 1])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```


Confusion Matrix
25
4
3
29
0
1
True Label
0
1
Predicted Label