WEEK-8

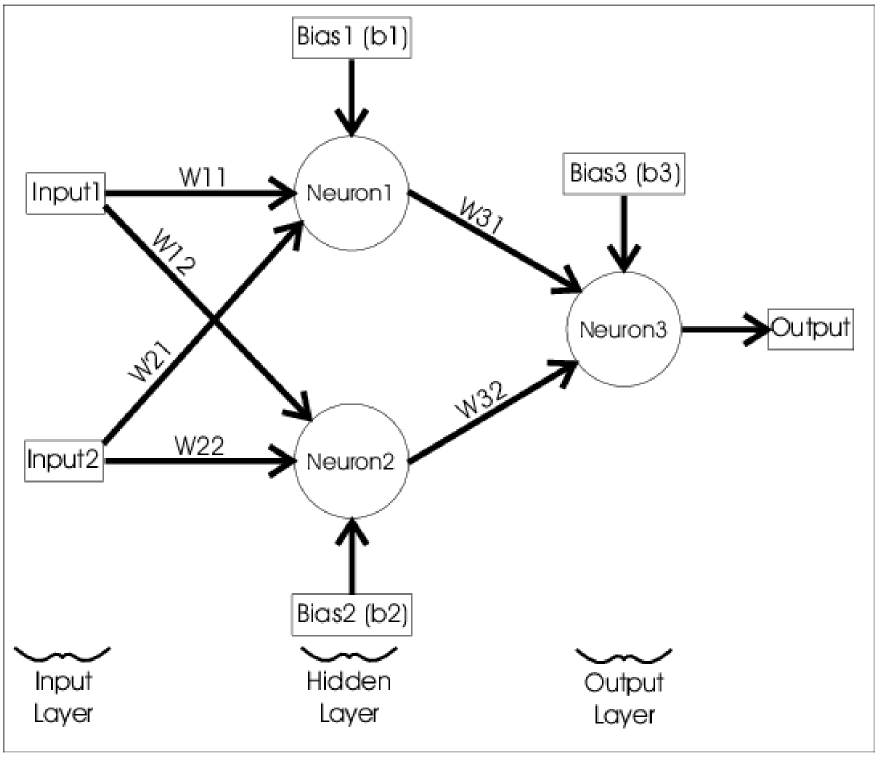
Aim: Write a program to implement logic gates using backpropagation

Description:

Back-propagation is the essence of neural net training. It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration). Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.

THE NEURAL NETWORK MODEL

As mentioned before, the neural network needs to produce two different decision planes to linearly separate the input data based on the output patterns. This is achieved by using the concept of *hidden layers*. The neural network will consist of one input layer with two nodes (X1,X2); one hidden layer with two nodes (since two decision planes are needed); and one output layer with one node (Y). Hence, the neural network looks like this:



THE LEARNING ALGORITHM

The information of a neural network is stored in the interconnections between the neurons i.e. the weights. A neural network learns by updating its weights according to a learning algorithm that helps it converge to the expected output. The learning algorithm is a principled way of changing the weights and biases based on the loss function.

1. Initialize the weights and biases randomly.

2. Iterate over the data
   i. Compute the predicted output
   ii. Compute the loss using the square error loss function
   iii. W(new) = W(old) — α ΔW
   iv. B(new) = B(old) — α ΔB

3. Repeat until the error is minimal

This is a fairly simple learning algorithm consisting of only arithmetic operations to update the weights and biases. The algorithm can be divided into two parts: the *forward pass* and the *backward pass* also known as *"backpropagation."*

Code:

```
# importing libraies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


or_arr = np.array([[-1, -1, -1],
                   [-1, 1, 1],
                   [1, -1, 1],
                   [1, 1, 1]])
and_arr = np.array([[-1, -1, -1],
```

```
                [-1, 1, -1],
                [1, -1, -1],
                [1, 1, 1]])
xor_arr = np.array([[-1, -1, -1],
                [-1, 1, 1],
                [1, -1, 1],
                [1, 1, -1]])

df_or = pd.DataFrame(or_arr, columns = ['X1', 'X2', 'label'])
df_or.head()
```

	X1	X2	label
0	-1	-1	-1
1	-1	1	1
2	1	-1	1
3	1	1	1

```
df_and = pd.DataFrame(and_arr, columns = ['X1', 'X2', 'label'])
df_and.head()
```

	X1	X2	label
0	-1	-1	-1
1	-1	1	-1
2	1	-1	-1
3	1	1	1

```
df_xor = pd.DataFrame(xor_arr, columns = ['X1', 'X2', 'label'])
df_xor.head()
```

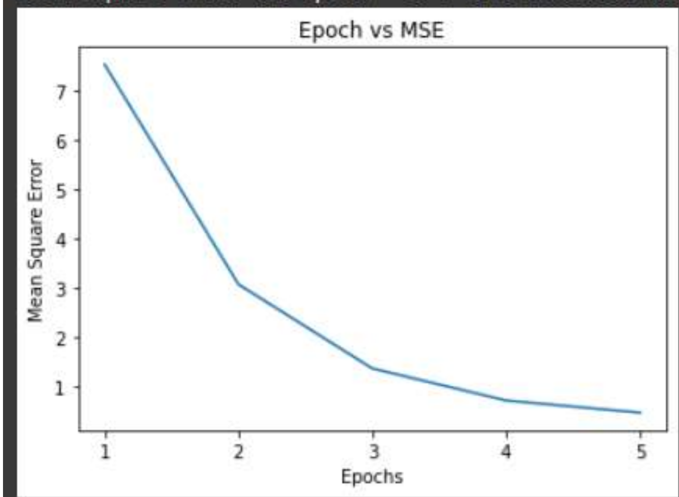
	X1	X2	label
0	-1	-1	-1
1	-1	1	1
2	1	-1	1
3	1	1	-1

```
def train_data(data):
    X = data.drop(['label'], axis=1)
    y = data['label']
    return X, y

def train(X, y):
    # Equation of the model or classifier
    # W1.X1 + W2.X2 + b = Y

    # initialising the Guassian random weights to model
    w1, w2, b = tuple(np.random.normal(size=3))
    print("Initial Weights + bias : ", (w1, w2, b))
    # w1 = w2 = b = 0.1

    # Number of epochs and learning rate
    epochs = 5
    alpha = 0.1
```

```
# delta learning
MSE = []
for i in range(epochs):
    error = []
    for i in range(len(X)):
        # Predicting target label with current weights
        y_pred = w1*X.iloc[i, 0] + w2*X.iloc[i, 1] + b
        # actual target label
        t = y[i]
        # difference between actual and predicted
        diff = t - y_pred
        # error
        err = diff**2
        error.append(err)
        # --- Updation ---
        # gradients
        delta_w1 = diff*X.iloc[i, 0]
        delta_w2 = diff*X.iloc[i, 1]
        delta_b = diff
        # New weights
        w1 = w1 + alpha*delta_w1
        w2 = w2 + alpha*delta_w2
        b = b + alpha*delta_b

    # Mean square error
```

```
        MSE.append(np.array(error).mean())
    for i in range(epochs):
        print("Mean Square error at epoch -", str(i+1)+" :", MSE[i])

    plt.plot(np.array(MSE))
    plt.xticks(np.arange(epochs), [str(i+1) for i in range(epochs)])
    plt.xlabel('Epochs')
    plt.ylabel("Mean Square Error")
    plt.title("Epoch vs MSE")
    plt.show()

# OR GATE
X, y = train_data(df_or)
train(X, y)
```

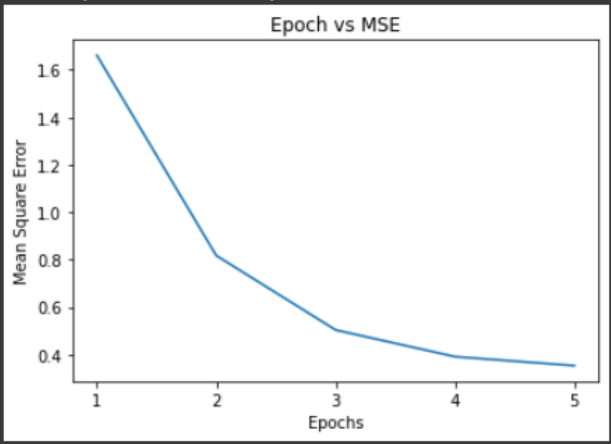
```
Initial Weights + bias :  (1.6420482754172157, 0.7455676747205521, 3.0637679858015936)
Mean Square error at epoch - 1 : 7.516734588192324
Mean Square error at epoch - 2 : 3.0757531417274517
Mean Square error at epoch - 3 : 1.3767089667698544
Mean Square error at epoch - 4 : 0.7294510675901527
Mean Square error at epoch - 5 : 0.484773579074112
```



AND Gate

```
X, y = train_data(df_and)
train(X, y)
```

```
Initial Weights + bias :  (0.25691809058916365, -0.6900600202805931, -0.6798385048658415)
Mean Square error at epoch - 1 : 1.6607913819503057
Mean Square error at epoch - 2 : 0.8170630690327348
Mean Square error at epoch - 3 : 0.5034081465075086
Mean Square error at epoch - 4 : 0.39084174020422563
Mean Square error at epoch - 5 : 0.35310385722168347
```



XOR Gate

```
[90] X, y = train_data(df_xor)
     train(X, y)
```

```
Initial Weights + bias :  (0.4059298954321152, -0.675334019887111, 0.2657674667413705)
Mean Square error at epoch - 1 : 2.012127930147021
Mean Square error at epoch - 2 : 1.645842035276136
Mean Square error at epoch - 3 : 1.4970321247080083
Mean Square error at epoch - 4 : 1.434978920561691
Mean Square error at epoch - 5 : 1.4081815116483924
```