Experiment :

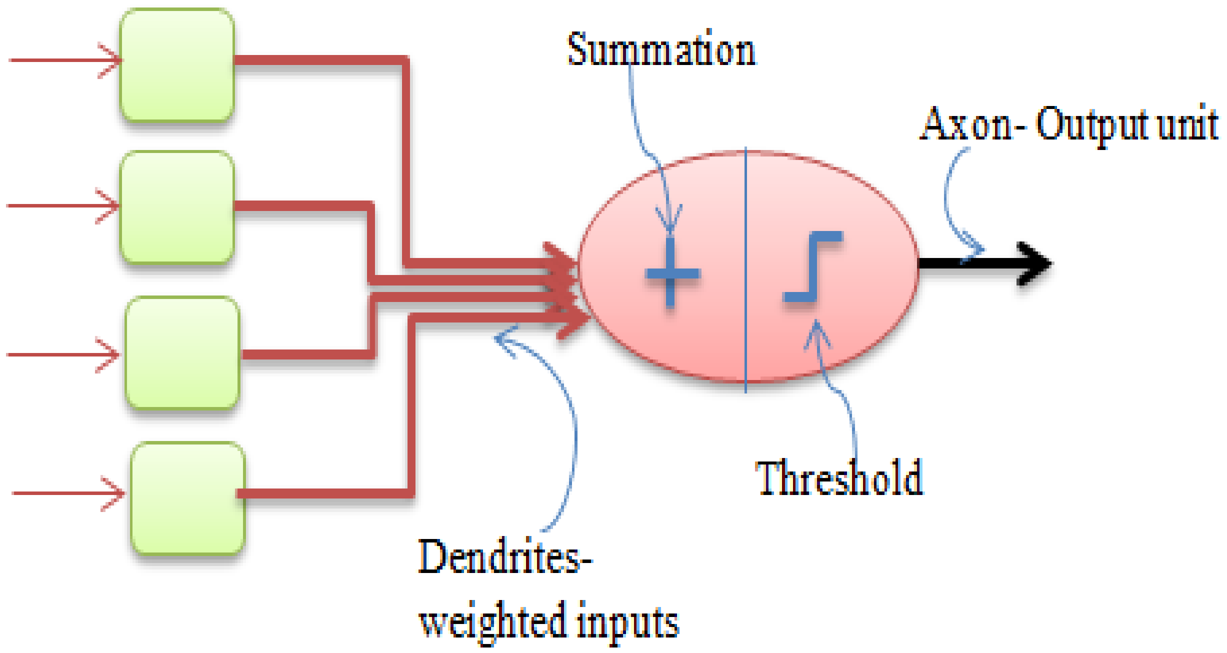
AIM : Write a program to implement back propagation algorithm

THEORY :

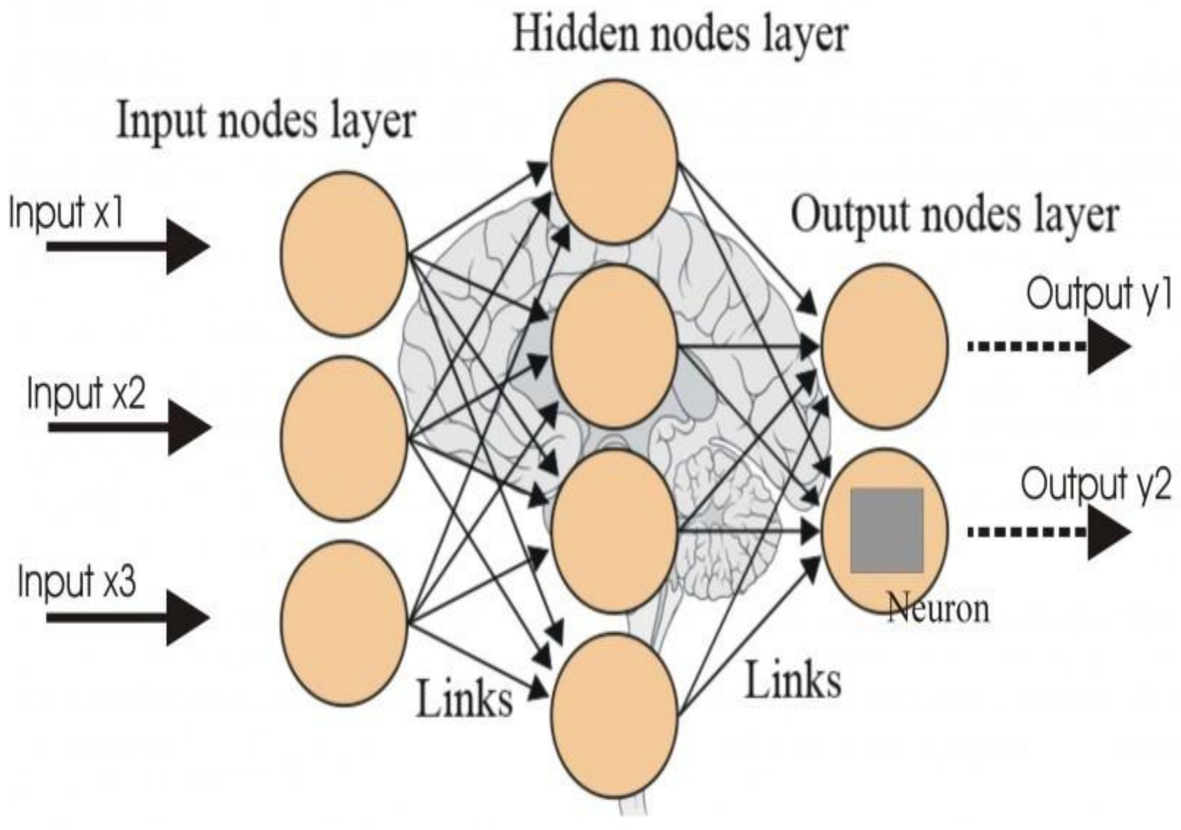
Artificial Neural Network

The brain is the fundamental part in the human body. It is the biological neural network which receives the inputs in the form of signals and processes it and send out the output signals. The fundamental unit of the brain is the Neuron. The AI Expert Maureen Caudill defines ANN as “a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.”

Brain consists of 200 billion of neurons and neuron is formed from 4 basic parts as Dendrites, Soma, Axon, and Synapses. The neuron collect signals from Dendrites, and the Soma cells sums up all the signals collected, and when the summation reaches the threshold the signal pass through the axon to the other neurons. The Synapses indicated the strength of the interconnection in between the neurons.



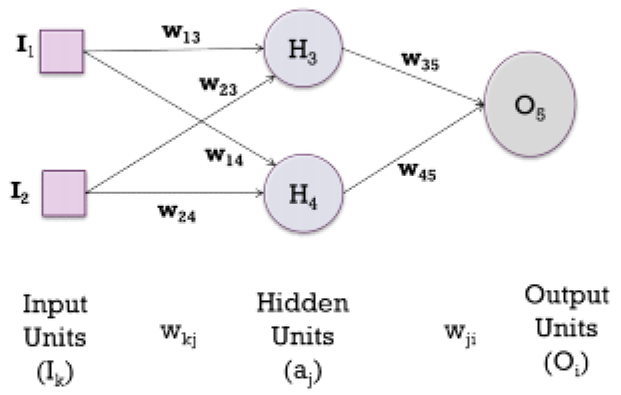
Similar to the brain, the Artificial Neural Network, imitates this biological Neural Network of human body. The first ANN was created so many years before by the neurophysiologists Warren McCulloch and the logician Walter Pits in 1943. They created a computational model for Neural Networks for the threshold logic which is the logic based on the mathematics and algorithms. But due to the inefficient technologies available at that time, this idea didn't come up as a success one. The ANN formed from the artificial neurons made up of Silicon and wires which imitates the neurons and the interconnection which are formed from coefficients (weights). The knowledge of an ANN is stored within inter-neuron connection strengths known as synaptic weights. The ANN is strongly interconnected with each other to make solutions for specific problems and complications and it is highly useful in the sectors of Pattern recognition, data classification, clustering etc. They derive data and information from complicated sources which are much difficult for the machines and human to get the data and they also used to find the complex patterns find in the complex sources of data.



ALGORITHM :

- The back propagation algorithm consists of two steps: the forward pass and the backward pass.
- Forward pass calculates the outputs corresponding to the inputs.
- Backward pass propagates the error backwards through the network starting at the output units.

FEEDFORWARD STEP



$$a_j = f(\sum_k w_{k,j} * I_k) \qquad O_i = f(\sum_j w_{j,i} * a_j)$$

Both the hidden and output units also have bias.

BACKPROPAGATION TRAINING RULE

- Each weight changed by:

$$\Delta w_{ji} = \eta \delta_j o_i$$

$$\delta_j = o_j(1-o_j)(t_j - o_j) \qquad \text{if } j \text{ is an output unit}$$

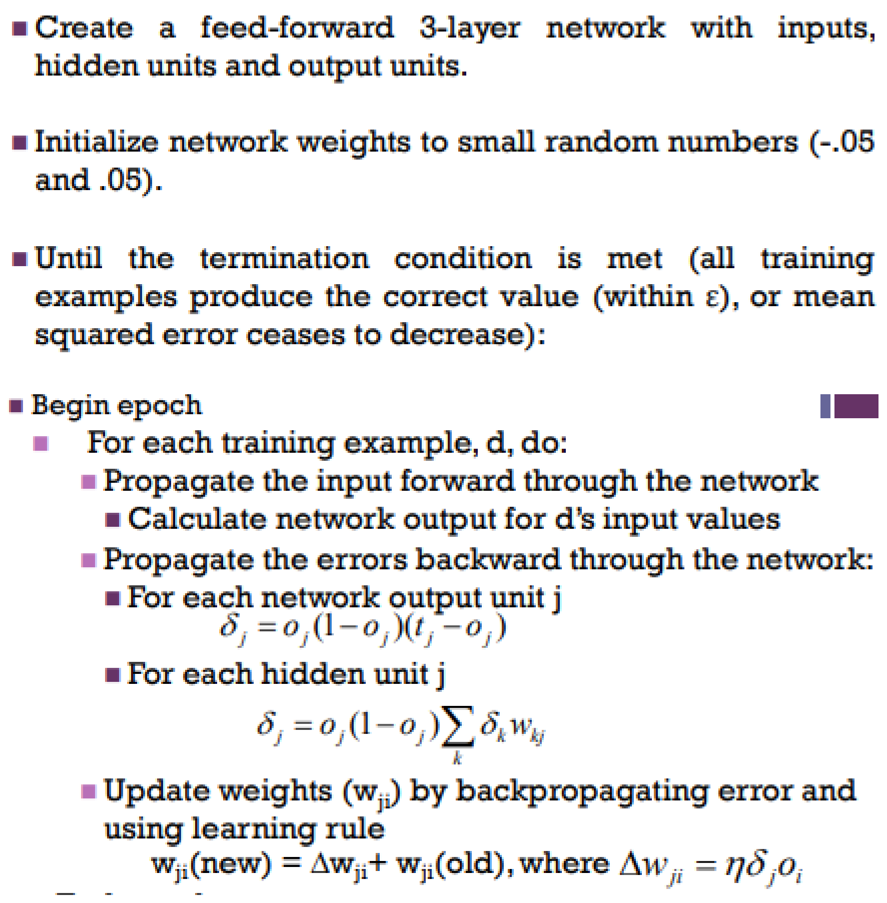
$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj} \qquad \text{if } j \text{ is a hidden unit}$$

where η is a constant called the learning rate

t_j is the correct teacher output for unit *j*

δ_j is the error measure for unit *j*

BACKPROPAGATION TRAINING ALGORITHM



- Create a feed-forward 3-layer network with inputs, hidden units and output units.
- Initialize network weights to small random numbers (-.05 and .05).
- Until the termination condition is met (all training examples produce the correct value (within ε), or mean squared error ceases to decrease):
- Begin epoch
  - For each training example, d, do:
    - Propagate the input forward through the network
      - Calculate network output for d's input values
    - Propagate the errors backward through the network:
      - For each network output unit j

        $$\delta_j = o_j(1-o_j)(t_j - o_j)$$

      - For each hidden unit j

        $$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj}$$

    - Update weights (w_{ji}) by backpropagating error and using learning rule

      w_{ji}(new) = Δw_{ji}+ w_{ji}(old), where $\Delta w_{ji} = \eta \delta_j o_i$

CODE :

```
In [2]: import pandas as pd

        import numpy as np

         from matplotlib import pyplot as plt


In [3]: x = pd.read_csv("./fashion-mnist_test.csv")

            X_ = np.array(x)

            X = X_[:,1:]

            X = X/255.0

            y = X_[:,0]


In [19]: x.head()


In [5]: X_train = X[:8000,:]

            X_val = X[8000:,:]

            y_train = y[:8000]

            y_val = y[8000:]
```

```
In [6]: IMG_SIZE = 28*28

        H1_SIZE = 256

        H2_SIZE = 64

        OUT_SIZE = 10

        BATCH_SIZE = 256

        EPOCH = 50

        ALPHA = 0.0003
```

```
In [7]: def accuracy(pred,y):

            return ( 100.0* np.sum(pred==y) / y.shape[0])
```

```
In [8]: def initial_weights():

            np.random.seed(0)

            model = {}

            model['W1'] = np.random.randn(IMG_SIZE,H1_SIZE)/ np.sqrt(IMG_SIZE)

            model['B1'] = np.zeros((1,H1_SIZE))

            model['W2'] = np.random.randn(H1_SIZE,H2_SIZE)/ np.sqrt(H1_SIZE)

            model['B2'] = np.zeros((1,H2_SIZE))

            model['W3'] = np.random.randn(H2_SIZE,OUT_SIZE)/ np.sqrt(H2_SIZE)

            model['B3'] = np.zeros((1,OUT_SIZE))

            return model
```

In [9]: def forward_prop(model,x):

```
            z1 = x.dot(model['W1']) + model['B1']

            a1 = np.tanh(z1)

            z2 = a1.dot(model['W2']) + model['B2']

            a2 = np.tanh(z2)

            z3 = a2.dot(model['W3']) + model['B3']

            h_x = np.exp(z3)

            y_out = h_x/ np.sum(h_x, axis=1, keepdims=True)

            return a1, a2, y_out
```

In [10]: def back_prop(model, x ,a1 , a2, y, y_out):

```
             delta4 = y_out

             delta4[range(y.shape[0]), y] -= 1

             dw3 = (a2.T).dot(delta4)

             db3 = np.sum(delta4, axis = 0)

             delta3 = (1 - np.square(a2))*delta4.dot(model['W3'].T)

             dw2 = (a1.T).dot(delta3)

             db2 = np.sum(delta3, axis = 0)

             delta2 = (1 - np.square(a1))*delta3.dot(model['W2'].T)

             dw1 = (x.T).dot(delta2)

             db1 = np.sum(delta2, axis = 0)
```

```
            model['W1'] += -ALPHA*dw1

            model['B1'] += -ALPHA*db1

            model['W2'] += -ALPHA*dw2

            model['B2'] += -ALPHA*db2

            model['W3'] += -ALPHA*dw3


            model['B3'] += -ALPHA*db3


            return model
```

In [11]:
```
def loss(model, p, y):

            correct_logprobs = -np.log(p[range(y.shape[0]),y])

            l = np.sum(correct_logprobs)


            return (1.0/y.shape[0]) * l
```

In [12]:
```
def predict(y_out):

            return np.argmax(y_out, axis = 1)
```

In [13]:
```
def main():

            training_loss = []

            val_loss = []

            val_acc = []
```

```
model = initial_weights()

for ix in range(EPOCH):

    print ("\nEpoch : %d" %(ix+1))

    count = 0

    while (count+BATCH_SIZE) < y_train.shape[0]:

        batch_data = X_train[count:(count+BATCH_SIZE),:]

        batch_labels = y_train[count:(count+BATCH_SIZE),]

        count += BATCH_SIZE


        a1, a2 , p = forward_prop(model, batch_data)

        model = back_prop(model,batch_data,a1,a2,batch_labels,p)


    _,_, p = forward_prop(model, X_train)

    training_loss.append(loss(model,p,y_train))

    print ('training_loss : % .3f' % (loss(model,p,y_train)))

    _,_,p = forward_prop(model, X_val)

    pred = predict(p)

    val_loss.append(loss(model,p,y_val))

    val_acc.append(accuracy(pred,y_val))

    print ('val_accuracy : % .3f' % (accuracy(pred,y_val)))

    print ('val_loss : % .3f' % loss(model,p,y_val))
```

```
        print("*************Completed***********")

        return training_loss,val_loss,val_acc
```

```
In [14]: training_loss,val_loss,val_acc = main()
```

```
In [18]: plt.figure(0)

    plt.title("Error vs Epochs")

    plt.xlabel("Epochs")

    plt.ylabel("Error")

    plt.plot(training_loss,color='yellow')

    plt.plot(val_loss,color='brown')

    plt.legend(["Training Error","Testing Error"],loc=0)

    plt.show()
```

```
In [16]: plt.figure(1)

    plt.title("Accuracy vs Epochs")

    plt.xlabel("Epochs")

    plt.ylabel("Accuracy")

    plt.plot(val_acc,color='orange')

    plt.show()
```

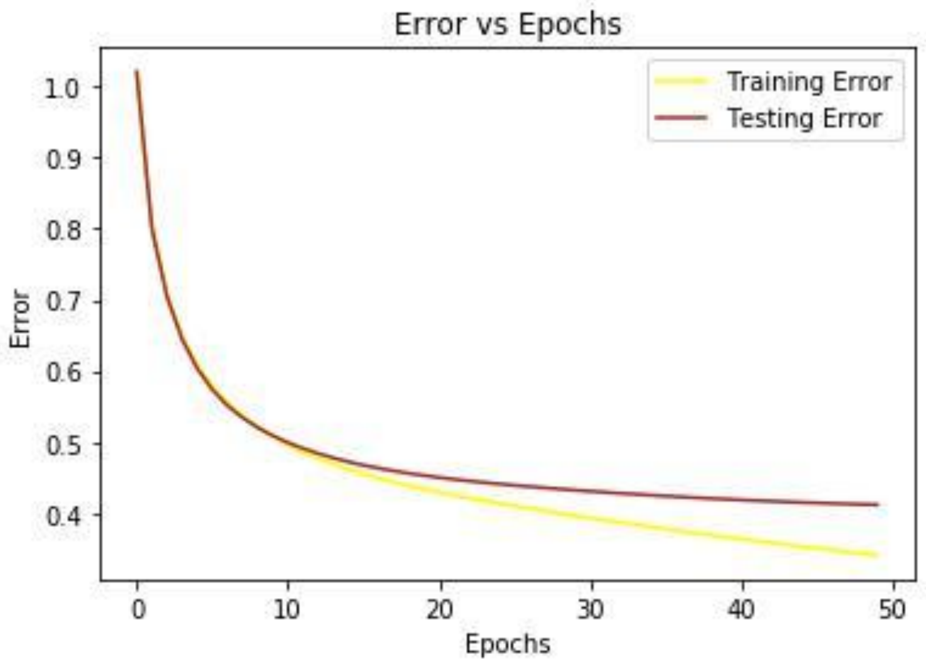
OUTPUT :

DATASET

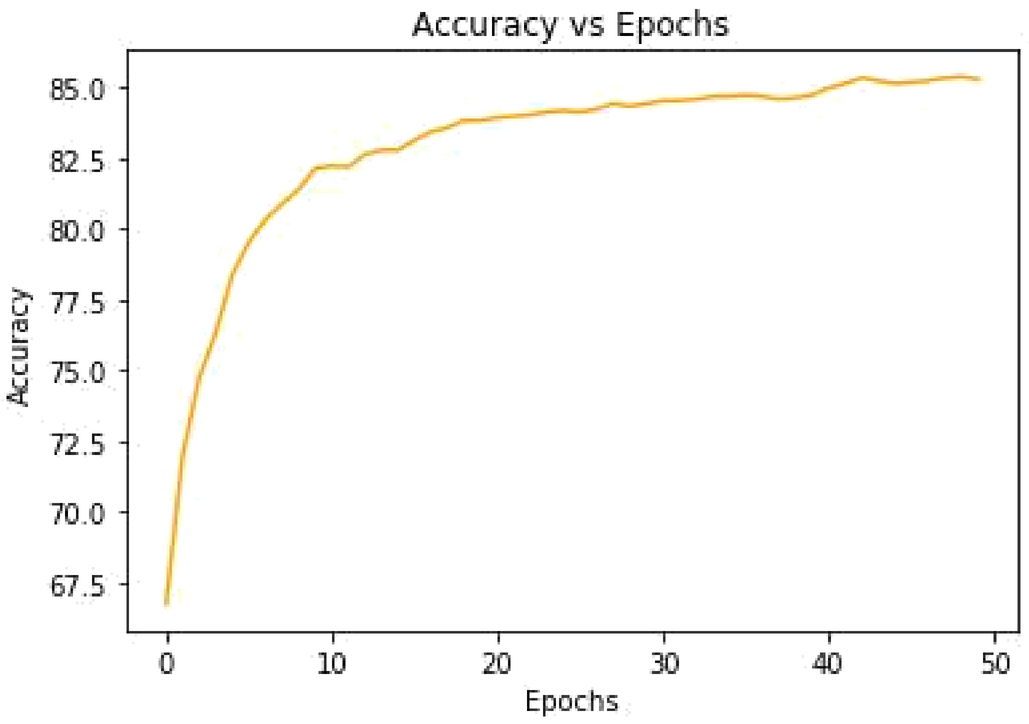
```
   label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0      0       0       0       0       0       0       0       0       9
1      1       0       0       0       0       0       0       0       0
2      2       0       0       0       0       0       0      14      53
3      2       0       0       0       0       0       0       0       0
4      3       0       0       0       0       0       0       0       0

   pixel9  ...  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       8  ...       103        87        56         0         0
1       0  ...        34         0         0         0         0
2      99  ...         0         0         0         0        63
3       0  ...       137       126       140         0       133
4       0  ...         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783  pixel784
0         0         0         0         0         0
1         0         0         0         0         0
2        53        31         0         0         0
3       224       222        56         0         0
4         0         0         0         0         0
```



Above graph shows, error decreases as the number of epochs increases and also that training error is less than the testing error.



Above graph shows that accuracy of artificial neural networks increases as more epochs are performed.

DISCUSSION :

Advantages of Artificial Neural Networks (ANN)

► **Storing information on the entire network** : Information such as in **traditional programming** is stored on the entire network, not on a database. The disappearance of a few pieces of information in one place does not prevent the network from functioning.

► **Ability to work with** incomplete knowledge **:** After ANN training, the data may produce output even with incomplete information. The loss of performance here depends on the importance of the missing information.

► **Having fault tolerance:** Corruption of one or more cells of ANN does not prevent it from generating output. This feature makes the networks fault tolerant.

► **Parallel processing capability:** Artificial neural networks have numerical strength that can perform more than one job at the same time.

Disadvantages of Artificial Neural Networks (ANN)

► **Hardware dependence:** Artificial neural networks require processors with parallel processing power, in accordance with their structure. For this reason, the realization of the equipment is dependent.

► **Unexplained behavior of the network:** This is the most important problem of ANN. When ANN produces a probing solution, it does not give a clue as to why and how. This reduces trust in the network.

► **Determination of proper network structure:** There is no specific rule for determining the structure of artificial neural networks. Appropriate network structure is achieved through experience and trial and error.

FINDING AND LEARNING :

In above experiment I applied backpropagation algorithm on fashion mnist dataset which contains labeled image data for 10 classes and got accuracy of around 85%.