**What are Artificial Neural Networks?**

A neural network is a group of connected I/O units where each connection has a weight associated with its computer programs. It helps you to build predictive models from large databases. This model builds upon the human nervous system. It helps you to conduct image understanding, human learning, computer speech, etc.

**What is Backpropagation?**

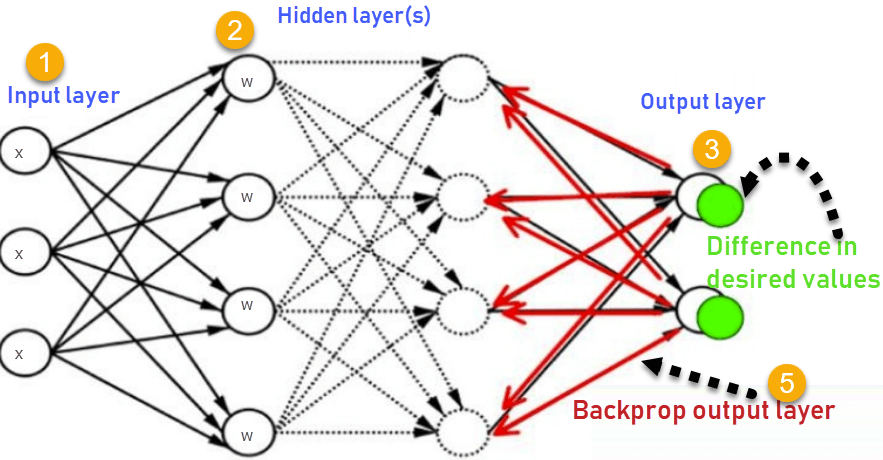
**Backpropagation** is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization.

Backpropagation in neural network is a short form for “backward propagation of errors.” It is a standard method of training artificial neural networks. This method helps calculate the gradient of a loss function with respect to all the weights in the network.

**How Backpropagation Algorithm Works**

The Back propagation algorithm in neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient, but it does not define how the gradient is used. It generalizes the computation in the delta rule.

Consider the following Back propagation neural network example diagram to understand:



1. Inputs X, arrive through the preconnected path
2. Input is modelled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs

ErrorB= Actual Output – Desired Output

Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

Keep repeating the process until the desired output is achieved

**Why We Need Backpropagation?**

Most prominent advantages of Backpropagation are:

* Backpropagation is fast, simple and easy to program
* It has no parameters to tune apart from the numbers of input
* It is a flexible method as it does not require prior knowledge about the network
* It is a standard method that generally works well
* It does not need any special mention of the features of the function to be learned.

**What is a Feed Forward Network?**

A feedforward neural network is an artificial neural network where the nodes never form a cycle. This kind of neural network has an input layer, hidden layers, and an output layer. It is the first and simplest type of artificial neural network.

**Types of Backpropagation Networks**

Two Types of Backpropagation Networks are:

1. Static Back-propagation
2. Recurrent Backpropagation

1. **Static back-propagation:**

It is one kind of backpropagation network which produces a mapping of a static input for static output. It is useful to solve static classification issues like optical character recognition.

2**. Recurrent Backpropagation:**

Recurrent Back propagation in data mining is fed forward until a fixed value is achieved. After that, the error is computed and propagated backward.

The main difference between both of these methods is: that the mapping is rapid in static back-propagation while it is nonstatic in recurrent backpropagation.

**History of Backpropagation**

* In 1961, the basics concept of continuous backpropagation were derived in the context of control theory by J. Kelly, Henry Arthur, and E. Bryson.
* In 1969, Bryson and Ho gave a multi-stage dynamic system optimization method.
* In 1974, Werbos stated the possibility of applying this principle in an artificial neural network.
* In 1982, Hopfield brought his idea of a neural network.
* In 1986, by the effort of David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, backpropagation gained recognition.
* In 1993, Wan was the first person to win an international pattern recognition contest with the help of the backpropagation method.

**Backpropagation Key Points**

* Simplifies the network structure by elements weighted links that have the least effect on the trained network
* You need to study a group of input and activation values to develop the relationship between the input and hidden unit layers.
* It helps to assess the impact that a given input variable has on a network output. The knowledge gained from this analysis should be represented in rules.
* Backpropagation is especially useful for deep neural networks working on error-prone projects, such as image or speech recognition.
* Backpropagation takes advantage of the chain and power rules allows backpropagation to function with any number of outputs.

**Best practice Backpropagation**

Backpropagation in neural network can be explained with the help of “Shoe Lace” analogy

* Too little tension

Not enough constraining and very loose

* Too much tension
* Too much constraint (overtraining)
* Taking too much time (relatively slow process)

Higher likelihood of breaking

* Pulling one lace more than other
* Discomfort (bias)

**Disadvantages of using Backpropagation**

* The actual performance of backpropagation on a specific problem is dependent on the input data.
* Back propagation algorithm in data mining can be quite sensitive to noisy data
* You need to use the matrix-based approach for backpropagation instead of mini-batch.

**Summary**

* A neural network is a group of connected it I/O units where each connection has a weight associated with its computer programs.
* Backpropagation is a short form for “backward propagation of errors.” It is a standard method of training artificial neural networks
* Back propagation algorithm in machine learning is fast, simple and easy to program
* A feedforward BPN network is an artificial neural network.
* Two Types of Backpropagation Networks are 1)Static Back-propagation 2) Recurrent Backpropagation
* In 1961, the basics concept of continuous backpropagation were derived in the context of control theory by J. Kelly, Henry Arthur, and E. Bryson.
* Back propagation in data mining simplifies the network structure by removing weighted links that have a minimal effect on the trained network.
* It is especially useful for deep neural networks working on error-prone projects, such as image or speech recognition.
* The biggest drawback of the Backpropagation is that it can be sensitive for noisy data.

**Backpropagation Process in Deep Neural Network**

**Backpropagation** is one of the important concepts of a neural network. Our task is to classify our data best. For this, we have to update the weights of parameter and bias, but how can we do that in a deep neural network? In the linear regression model, we use gradient descent to optimize the parameter. Similarly, here we also use gradient descent algorithm using Backpropagation.

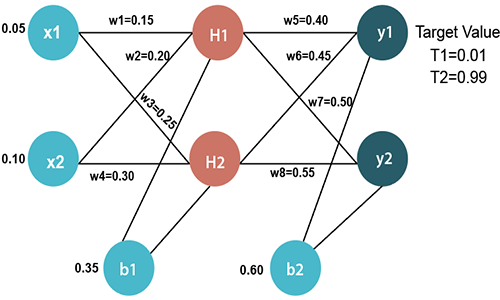
For a single training example, **Backpropagation** algorithm calculates the gradient of the **error function**. Backpropagation can be written as a function of the neural network. Backpropagation algorithms are a set of methods used to efficiently train artificial neural networks following a gradient descent approach which exploits the chain rule.

The main features of Backpropagation are the iterative, recursive and efficient method through which it calculates the updated weight to improve the network until it is not able to perform the task for which it is being trained. Derivatives of the activation function to be known at network design time is required to Backpropagation.

Now, how error function is used in Backpropagation and how Backpropagation works? Let start with an example and do it mathematically to understand how exactly updates the weight using Backpropagation.

66.7M1.4K

Features of Java - Javatpoint



**Input values**

X1=0.05  
X2=0.10

**Initial weight**

W1=0.15     w5=0.40  
W2=0.20     w6=0.45  
W3=0.25     w7=0.50  
W4=0.30     w8=0.55

**Bias Values**

b1=0.35     b2=0.60

**Target Values**

T1=0.01  
T2=0.99

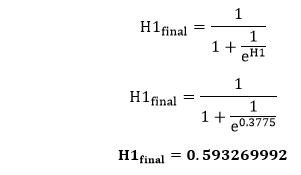
Now, we first calculate the values of H1 and H2 by a forward pass.

**Forward Pass**

To find the value of H1 we first multiply the input value from the weights as

                              H1=x1×w1+x2×w2+b1  
                        H1=0.05×0.15+0.10×0.20+0.35  
                                    **H1=0.3775**

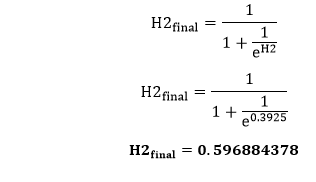
To calculate the final result of H1, we performed the sigmoid function as



We will calculate the value of H2 in the same way as H1

                              H2=x1×w3+x2×w4+b1  
                        H2=0.05×0.25+0.10×0.30+0.35  
                                    **H2=0.3925**

To calculate the final result of H1, we performed the sigmoid function as

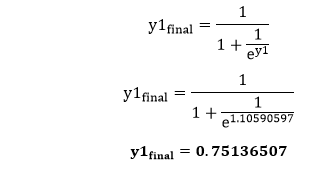


Now, we calculate the values of y1 and y2 in the same way as we calculate the H1 and H2.

To find the value of y1, we first multiply the input value i.e., the outcome of H1 and H2 from the weights as

                              y1=H1×w5+H2×w6+b2  
                        y1=0.593269992×0.40+0.596884378×0.45+0.60  
                                    **y1=1.10590597**

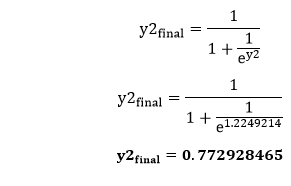
To calculate the final result of y1 we performed the sigmoid function as



We will calculate the value of y2 in the same way as y1

                              y2=H1×w7+H2×w8+b2  
                        y2=0.593269992×0.50+0.596884378×0.55+0.60  
                                    **y2=1.2249214**

To calculate the final result of H1, we performed the sigmoid function as

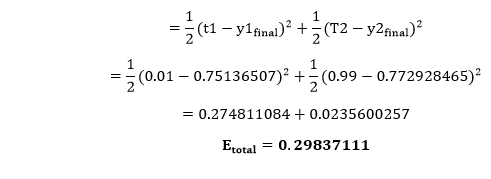


Our target values are 0.01 and 0.99. Our y1 and y2 value is not matched with our target values T1 and T2.

Now, we will find the **total error**, which is simply the difference between the outputs from the target outputs. The total error is calculated as

Backpropagation Process in Deep Neural Network

So, the total error is



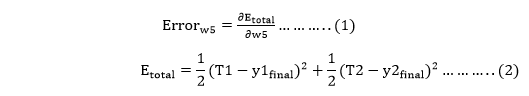
Now, we will backpropagate this error to update the weights using a backward pass.

Backward pass at the output layer

To update the weight, we calculate the error correspond to each weight with the help of a total error. The error on weight w is calculated by differentiating total error with respect to w.

Backpropagation Process in Deep Neural Network

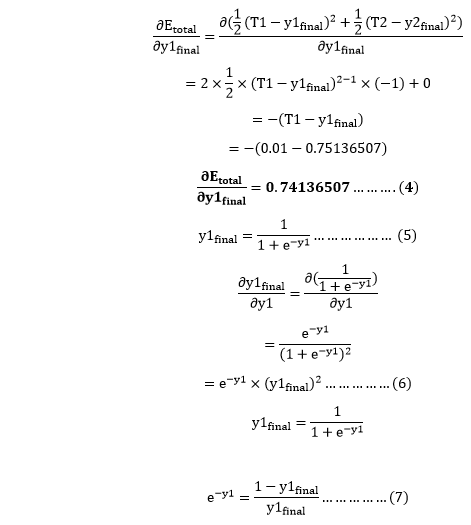
We perform backward process so first consider the last weight w5 as



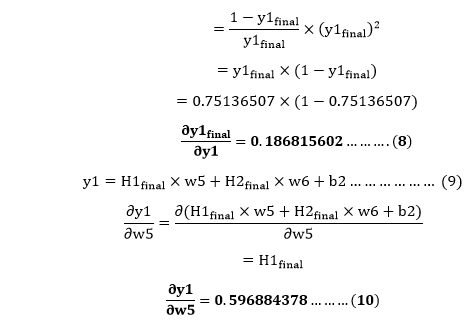
From equation two, it is clear that we cannot partially differentiate it with respect to w5 because there is no any w5. We split equation one into multiple terms so that we can easily differentiate it with respect to w5 as

Backpropagation Process in Deep Neural Network

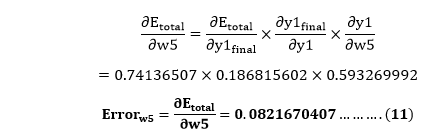
Now, we calculate each term one by one to differentiate Etotal with respect to w5 as



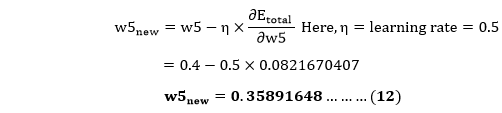
Putting the value of e-y in equation (5)



So, we put the values of Backpropagation Process in Deep Neural Network in equation no (3) to find the final result.



Now, we will calculate the updated weight w5new with the help of the following formula



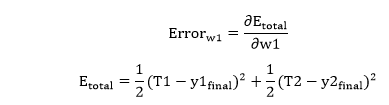
In the same way, we calculate w6new, w7new, and w8new and this will give us the following values

**w5new=0.35891648**  
                        **w6new=408666186**  
                        **w7new=0.511301270**  
                        **w8new=0.561370121**

**Backward pass at Hidden layer**

Now, we will backpropagate to our hidden layer and update the weight w1, w2, w3, and w4 as we have done with w5, w6, w7, and w8 weights.

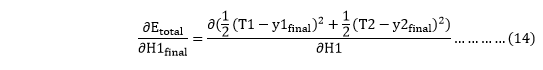
We will calculate the error at w1 as



From equation (2), it is clear that we cannot partially differentiate it with respect to w1 because there is no any w1. We split equation (1) into multiple terms so that we can easily differentiate it with respect to w1 as

Backpropagation Process in Deep Neural Network

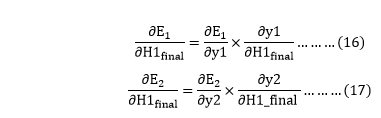
Now, we calculate each term one by one to differentiate Etotal with respect to w1 as



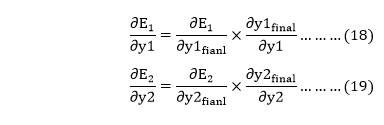
We again split this because there is no any H1final term in Etotal as

Backpropagation Process in Deep Neural Network

Backpropagation Process in Deep Neural Network will again split because in E1 and E2 there is no H1 term. Splitting is done as

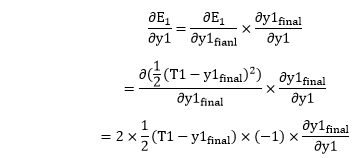


We again Split bothBackpropagation Process in Deep Neural Network because there is no any y1 and y2 term in E1 and E2. We split it as

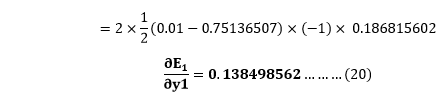


Now, we find the value of Backpropagation Process in Deep Neural Network by putting values in equation (18) and (19) as

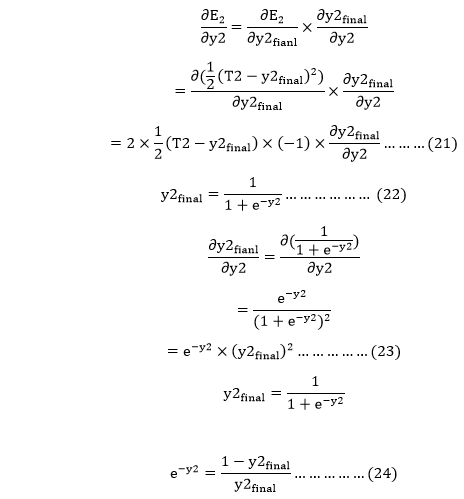
From equation (18)



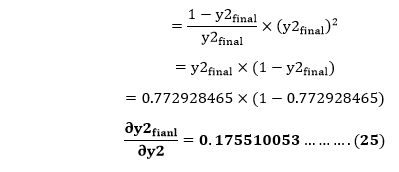
From equation (8)



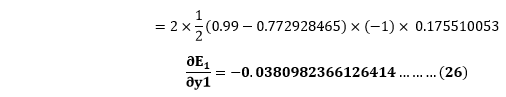
From equation (19)



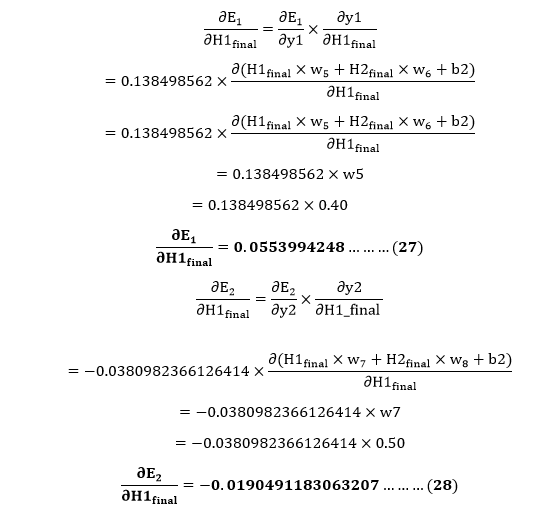
Putting the value of e-y2 in equation (23)



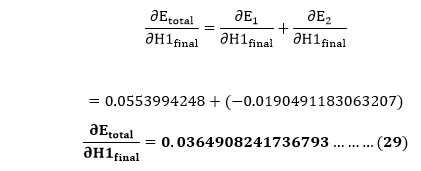
From equation (21)



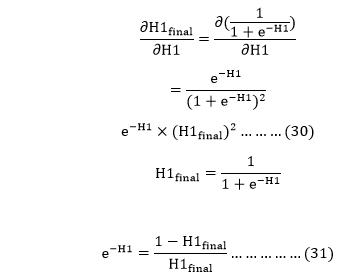
Now from equation (16) and (17)



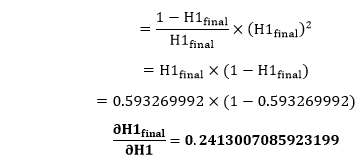
Put the value of Backpropagation Process in Deep Neural Network in equation (15) as



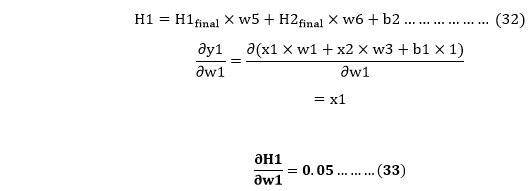
We haveBackpropagation Process in Deep Neural Networkwe need to figure outBackpropagation Process in Deep Neural Networkas



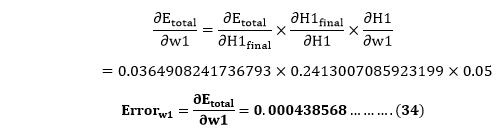
Putting the value of e-H1 in equation (30)



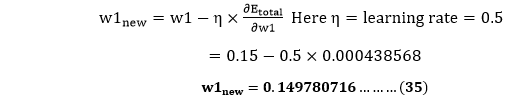
We calculate the partial derivative of the total net input to H1 with respect to w1 the same as we did for the output neuron:



So, we put the values of Backpropagation Process in Deep Neural Network in equation (13) to find the final result.



Now, we will calculate the updated weight w1new with the help of the following formula



In the same way, we calculate w2new, w3new, and w4 and this will give us the following values

**w1new=0.149780716**  
                        **w2new=0.19956143**  
                        **w3new=0.24975114**  
                        **w4new=0.29950229**

We have updated all the weights. We found the error 0.298371109 on the network when we fed forward the 0.05 and 0.1 inputs. In the first round of Backpropagation, the total error is down to 0.291027924. After repeating this process 10,000, the total error is down to 0.0000351085. At this point, the outputs neurons generate 0.159121960 and 0.984065734 i.e., nearby our target value when we feed forward the 0.05 and 0.1.

**Case Study**

Let us perform a case study using backpropagation. For that, we will be using Iris data which contains features such as length and width of sepals and petals. With the help of those, we need to identify the species of a plant.

For this, we will build a multilayered neural network and will use the sigmoid function as it is a classification problem.

**Let us read the libraries required and read the data.**

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

**To ignore warnings, we will import another library called warnings.**

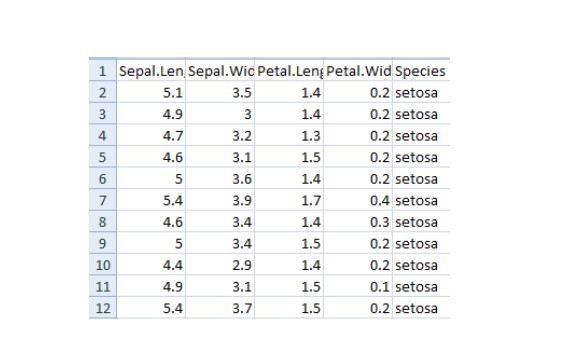
import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)

**Let us now read the data**

iris = pd.read\_csv("iris.csv")

iris.head()



**Now we will put labels to the class as 0,1 and 2.**

iris['Species']. replace (['setosa', 'virginica', 'versicolor'], [0, 1, 2], inplace=True)

**We will now define functions which will do the following.**

Perform one hot encoding to the output.

Perform sigmoid function

Normalize the features.

**For one hot encoding, we define the following function.**

def to\_one\_hot(Y):

n\_col = np.amax(Y) + 1

binarized = np.zeros((len(Y), n\_col))

for i in range(len(Y)):

binarized [i, Y[i]] = 1.

return binarized

**Let us now define a sigmoid function**

def sigmoid\_func(x):

return 1/(1+np.exp(-x))

def sigmoid\_derivative(x):

return sigmoid\_func(x)\*(1 – sigmoid\_func(x))

**Now we will define a function for normalization**

def normalize (X, axis=-1, order=2):

l2 = np. atleast\_1d (np.linalg.norm(X, order, axis))

l2[l2 == 0] = 1

return X / np.expand\_dims(l2, axis)

**Now we will apply normalization to the features and one hot encoding to the output**

columns = ['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width']

x = pd.DataFrame(iris, columns=columns)

x = normalize(x.as\_matrix())

columns = ['Species']

y = pd.DataFrame(iris, columns=columns)

y = y.as\_matrix()

y = y.flatten()

y = to\_one\_hot(y)

**Now it’s time to apply back propagation.** To do that, we need to define weights and a learning rate. Let us do that. But before that we need to split the data for training and testing.

#Split data to training and validation data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.33)

#Weights

w0 = 2\*np.random.random((4, 5)) - 1 #for input - 4 inputs, 3 outputs

w1 = 2\*np.random.random((5, 3)) - 1 #for layer 1 - 5 inputs, 3 outputs

#learning rate

n = 0.1

**We will set a list for errors and see how the change in training decreases the error via visualization.**

errors = []

Let us perform the feed forward and back propagation network. For backpropagation, we will use gradient descent.

for i in range (100000):

**Feed forward network**

layer0 = X\_train

layer1 = sigmoid\_func(np.dot(layer0, w0))

layer2 = sigmoid\_func(np.dot(layer1, w1))

**Back propagation using gradient descent**

layer2\_error = y\_train - layer2

layer2\_delta = layer2\_error \* sigmoid\_derivative(layer2)

layer1\_error = layer2\_delta.dot (w1.T)

layer1\_delta = layer1\_error \* sigmoid\_derivative(layer1)

w1 += layer1.T.dot(layer2\_delta) \* n

w0 += layer0.T.dot(layer1\_delta) \* n

error = np.mean(np.abs(layer2\_error))

errors.append(error)

Accuracy will be gathered and visualized by subtracting the error from the training data

accuracy\_training = (1 - error) \* 100

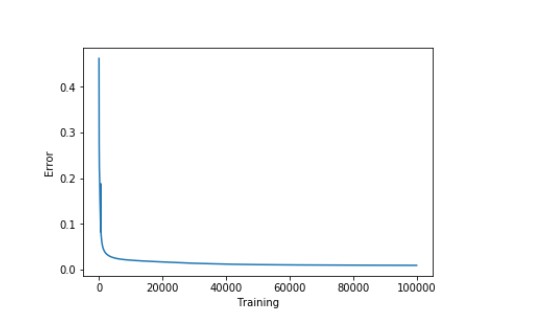
Now let us visualize how accuracy increases by decreasing the error

plt.plot(errors)

plt.xlabel('Training')

plt.ylabel('Error')

plt.show()



**Let us look at the accuracy now**

print ("Training Accuracy of the model " + str (round(accuracy\_training,2)) + "%")

Output: Training Accuracy of the model 99.04%

Our training model is performing really well. Now let us see the validation accuracy.

#Validate

layer0 = X\_test

layer1 = sigmoid\_func(np.dot(layer0, w0))

layer2 = sigmoid\_func(np.dot(layer1, w1))

layer2\_error = y\_test - layer2

error = np.mean(np.abs(layer2\_error))

accuracy\_validation = (1 - error) \* 100

print ("Validation Accuracy of the model “+ str(round(accuracy\_validation,2)) + "%")

Output: Validation Accuracy 92.86%

The performance was as expected.

**Best practices to follow**

Some of the ways to get a good model are discussed below-

* If there is very less constraint, the system may not be effective
* Too much constraint with over training will lead to a slow process
* Focusing on few aspects will lead to bias

**Disadvantages of backpropagation**

* Input data holds the key to the overall performance
* Noisy data can lead to inaccurate results
* Matrix based approach is preferred over a mini-batch

In conclusion, Neural network is a collection of connected units with input and output mechanism, each of the connections has an associated weight. Backpropagation is the “backward propagation of errors” and is useful to train neural networks. It is fast, easy to implement and simple. Backpropagation is very beneficial for deep neural networks working over error prone projects like speech or image recognition.