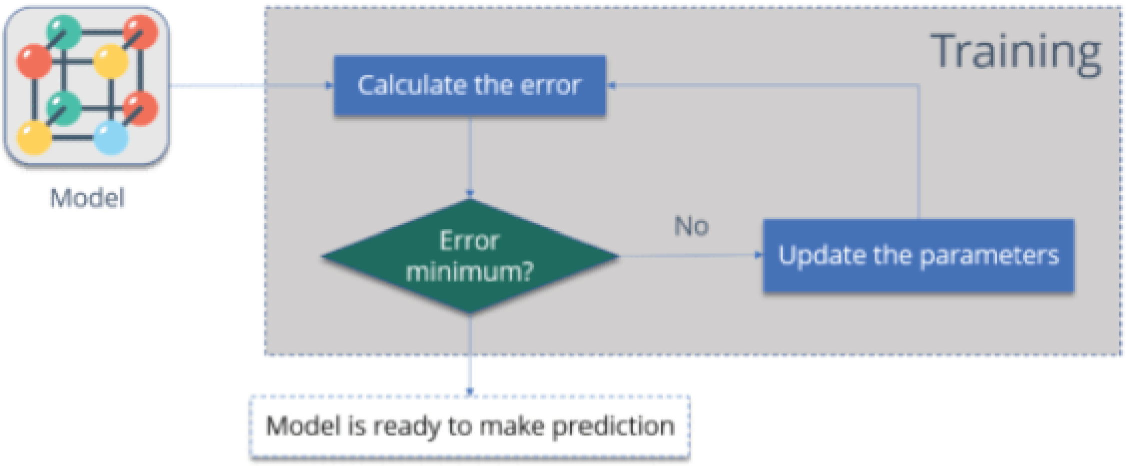
Implementing Back propagation algorithm and test the same using appropriate data sets

Backpropagation is supervised learning algorithm , for training Neural Networks. Every node in Neural Network represent a Neuron, so we can say that Neural Network is a circuit of neurons, Neural Network consist an Input layer, an output layer and a hidden layer.

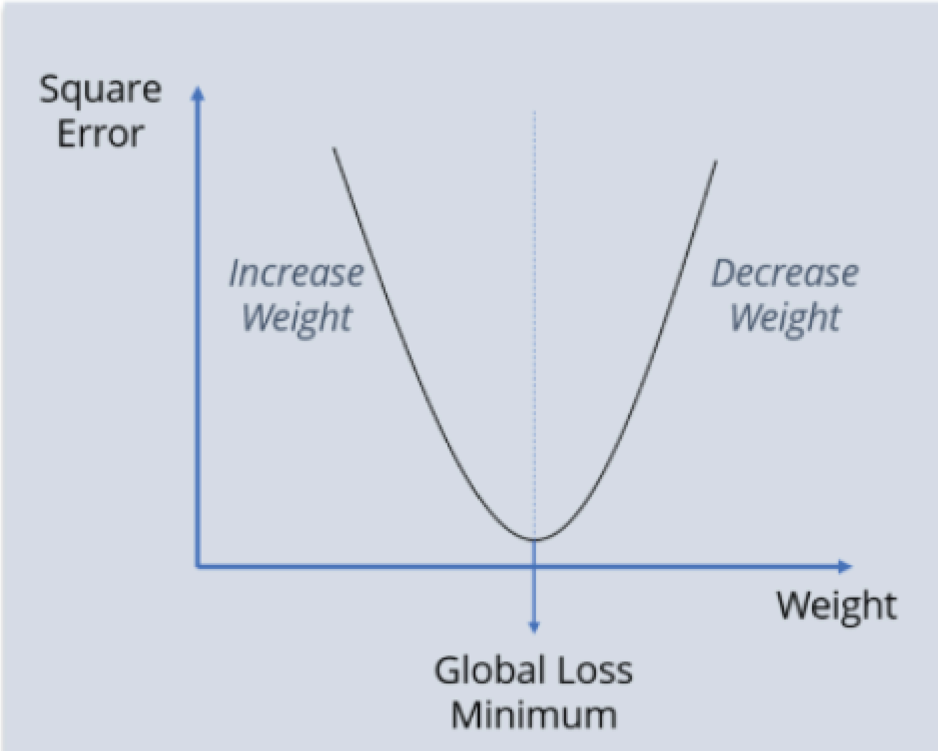
What is the Role of Backpropagation

1. First of all,if I want to create a neural network, then I have to initialize some weights.
2. Now, whatever values i have selected for weights i do not know how much they are correct.
3. To check that the weight values that I have selected are correct or incorrect I have to calculate the error of the model.
4. Suppose my model error occurred too much
5. Meaning my predicated output is very different from the actual output, so what shall I do? I will try to minimize the error.



Gradient Descent

1. We have number of optimizer but here we are using Gradient descent optimizer.
2. Gradient descent work as a optimizer, for finding minimum of a function.
3. In our case we update the weights using gradient descent and try to minimize error function.



How does back propagation algorithm work?

Suppose we have a neural network that has an input layer, a hidden layer and an output layer

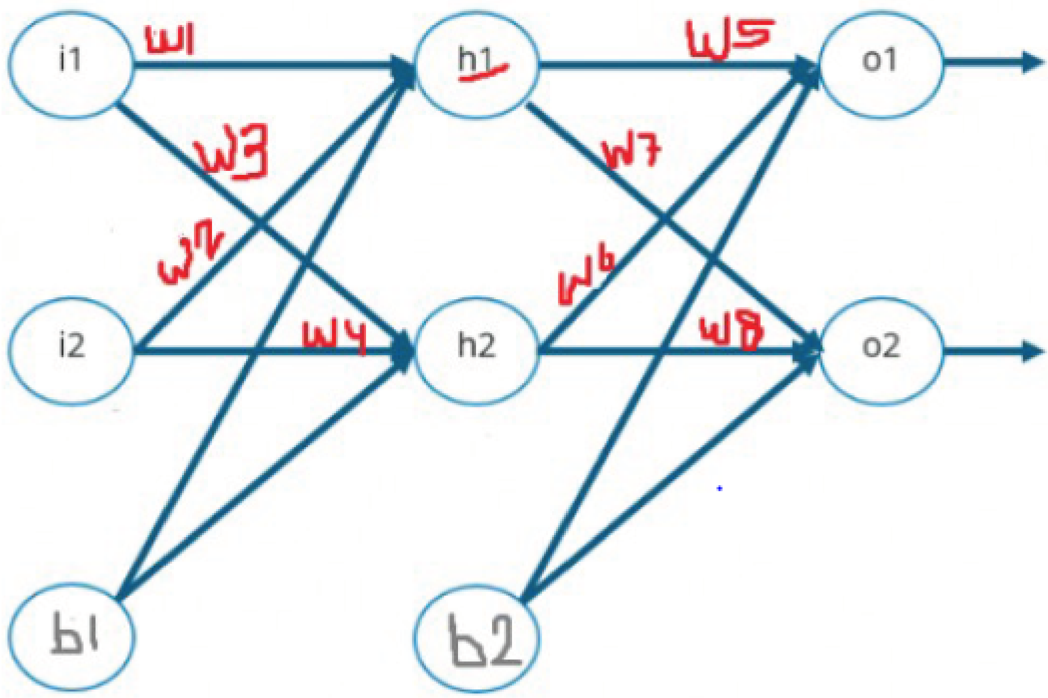
step1: First, we give random weights to the model.

step2: Forward propagation (normal neural network calculation)

step3: Calculate total error.

step4: Backward propagation (gradient descent), updating parameters (weights and bias)

step5: Until the error is minimized (Predicted output to be approximately equal to original output)



Forward Propagation

Back Propagation

Example	**Sleep**	**Study**	**Expected % in Exams**
1	2	9	92
2	1	5	86
3	3	6	89

```
import numpy as np
x=np.array(([2,9],[1,5],[3,6]),dtype=float)
print (x)
y=np.array(([92],[86],[89]),dtype=float)

#Normalization of dataset
x=x/np.max(x,axis=0)
y=y/100
print(x)
print(y)
```

```
[[2. 9.]
 [1. 5.]
 [3. 6.]]
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
[[0.92]
 [0.86]
 [0.89]]
```

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

def derivatives_sigmoid(x):
  return x*(1-x)

epoch=1000

lr=0.01

input_layer_neurons=2
hidden_layer_neurons=2
output_neurons=1

wh=np.random.uniform(size=(input_layer_neurons,hidden_layer_neurons))
bh=np.random.uniform(size=(1,hidden_layer_neurons))
wout=np.random.uniform(size=(hidden_layer_neurons,output_neurons))
```

```
bout=np.random.uniform(size=(1,output_neurons))


for i in range(epoch):
  hinp1=np.dot(x,wh)
  hinp=hinp1+bh
  hlayer_act=sigmoid(hinp)
  outinp1=np.dot(hlayer_act,wout)
  outinp=outinp1+bout
  output=sigmoid(outinp)
  EO=(y-output)



  # back Propagation
  outgrad=derivatives_sigmoid(output)
  d_output=EO*outgrad
  EH=d_output.dot(wout.T)
  hiddengrad=derivatives_sigmoid(hlayer_act)
  d_hiddenlayer=EH*hiddengrad

  #change of weight at each layer
  wout+=hlayer_act.T.dot(d_output)*lr
  bout += np.sum(d_output,axis=0,keepdims=True) *lr
  wh+=x.T.dot(d_hiddenlayer)*lr
  bh +=np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
  #output after each epoch
  print ("-----------Epoch-", i+1, "Starts----------")
  print("Input: \n" + str(x))
  print("Actual Output: \n" + str(y))
  print("Predicted Output: \n" ,output)
  print("Error:\n"+str(EO))
  print ("-----------Epoch-", i+1, "Ends----------\n")

print("Actual ouput"+str(y))
print("Predicted Output"+str(output))
print("Error"+str(EO))
```

Streaming output truncated to the last 5000 lines.

```
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.86622196]
 [0.85761803]
 [0.8696118 ]]
Error:
[[0.05377804]
 [0.00238197]
 [0.0203882 ]]
-----------Epoch- 738 Ends----------

-----------Epoch- 739 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
```

```
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.86624682]
 [0.85764273]
 [0.86963631]]
Error:
[[0.05375318]
 [0.00235727]
 [0.02036369]]
-----------Epoch- 739 Ends----------

-----------Epoch- 740 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.86627165]
 [0.8576674 ]
 [0.86966079]]
Error:
[[0.05372835]
 [0.0023326 ]
 [0.02033921]]
-----------Epoch- 740 Ends----------

-----------Epoch- 741 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
```