Backpropagation

Backpropagation, short for "backward propagation of errors," is an algorithm for supervised learning of artificial neural networks using gradient descent. Given an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural network's weights. It is a generalization of the delta rule for perceptrons to multilayer feedforward neural networks.

What is backpropagation?

- In machine learning, backpropagation is an effective algorithm used to train artificial neural networks, especially in feed-forward neural networks.
- Backpropagation is an iterative algorithm, that helps to minimize the cost function by determining which weights and biases should be adjusted. During every epoch, the model learns by adapting the weights and biases to minimize the loss by moving down toward the gradient of the error. Thus, it involves the two most popular optimization algorithms, such as gradient descent or stochastic gradient descent.
- Computing the gradient in the backpropagation algorithm helps to minimize the cost function and it can be implemented by using the mathematical rule called chain rule from calculus to navigate through complex layers of the neural network.

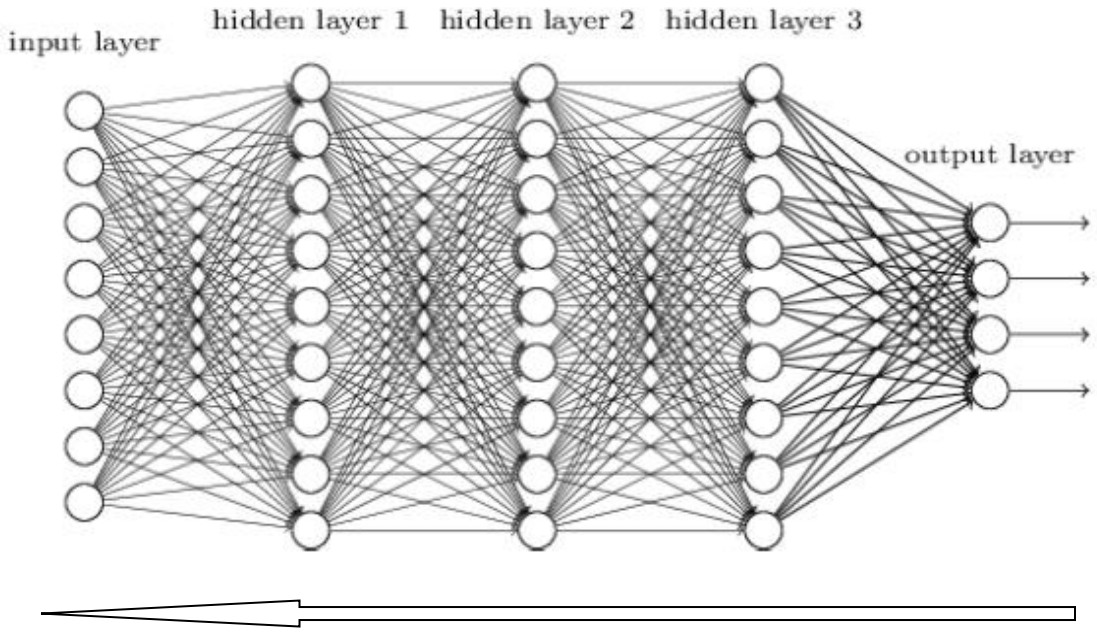
Backpropagation is analogous to calculating the delta rule for a multilayer feedforward network. Thus, like the delta rule, backpropagation requires three things:

1) **Dataset** consisting of input-output pairs $(\vec{x}_i, \vec{y}_i)$, where $\vec{x}_i$ is the input and $\vec{y}_i$ is the desired output of the network on input $\vec{x}_i$. The set of input-output pairs of size N is denoted $X = \left\{(\vec{x}_1, \vec{y}_1), \ldots, (\vec{x}_N, \vec{y}_N)\right\}$.

2) A **feedforward neural network**, as formally defined in the article concerning feedforward neural networks, whose parameters are collectively denoted θ. In backpropagation, the parameters of primary interest are w_{ij}^k, the weight between node j in layer l_k and node i in layer l_{k-1}, and b_i^k, the bias for node i in layer l_k. There are no connections between nodes in the same layer and layers are fully connected.

3) An **error function**, $E(X, \theta)$, which defines the error between the desired output $\vec{y}_i$ and the calculated output $\hat{\vec{y}}_i$ of the neural network on input $\vec{x}_i$ for a set of input-output pairs $(\vec{x}_i, \vec{y}_i) \in X$ and a particular value of the parameters θ.

Backpropagation



Training a neural network with gradient descent requires the calculation of the gradient of the error function $E(X, \theta)$ with respect to the weights w_{ij}^k and biases b_i^k. Then, according to the learning rate α, each iteration of gradient descent updates the weights and biases (collectively denoted θ) according to

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial E(X, \theta^t)}{\partial \theta},$$

where θ^t denotes the parameters of the neural network at iteration t in gradient descent.

Working of Backpropagation Algorithm

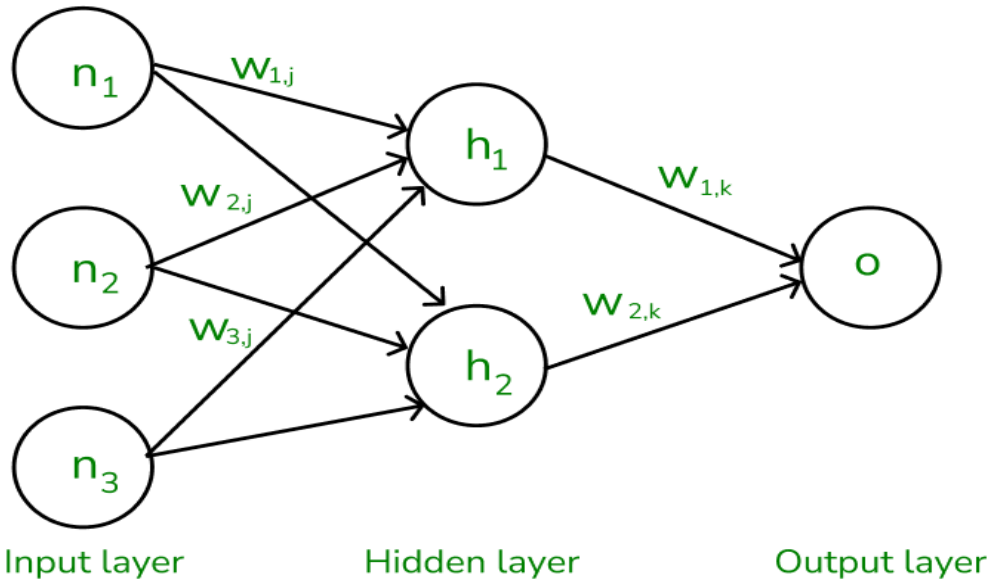
The Backpropagation algorithm works by two different passes, they are:

- Forward pass
- Backward pass

How does Forward pass work?

- In forward pass, initially the input is fed into the input layer. Since the inputs are raw data, they can be used for training our neural network.
- The inputs and their corresponding weights are passed to the hidden layer. The hidden layer performs the computation on the data it receives. If there are two hidden layers in the neural network, for instance, consider the illustration fig(a), h_1 and h_2 are the two hidden layers, and the output of h_1 can be used as an input of h_2. Before applying it to the activation function, the bias is added.
- To the weighted sum of inputs, the activation function is applied in the hidden layer to each of its neurons. One such activation function that is commonly used is ReLU can also be used, which is responsible for

returning the input if it is positive otherwise it returns zero. By doing this so, it introduces the non-linearity to our model, which enables the network to learn the complex relationships in the data. And finally, the weighted outputs from the last hidden layer are fed into the output to compute the final prediction, this layer can also use the activation function called the softmax function which is responsible for converting the weighted outputs into probabilities for each class.



How does backward pass work?

- In the backward pass process shows, the error is transmitted back to the network which helps the network, to improve its performance by learning and adjusting the internal weights.
- To find the error generated through the process of forward pass, we can use one of the most commonly used methods called mean squared error which calculates the difference between the predicted output and desired output. The formula for mean squared error is:
- $Meansquarederror=(predictedoutput–actualoutput)^2$

- Once we have done the calculation at the output layer, we then propagate the error backward through the network, layer by layer.
- The key calculation during the backward pass is determining the gradients for each weight and bias in the network. This gradient is responsible for telling us how much each weight/bias should be adjusted to minimize the error in the next forward pass. The chain rule is used iteratively to calculate this gradient efficiently.

- In addition to gradient calculation, the activation function also plays a crucial role in backpropagation, it works by calculating the gradients with the help of the derivative of the activation function.

Implementing forward propagation:

Step1: Before proceeding to calculating forward propagation, we need to know the two formulae:

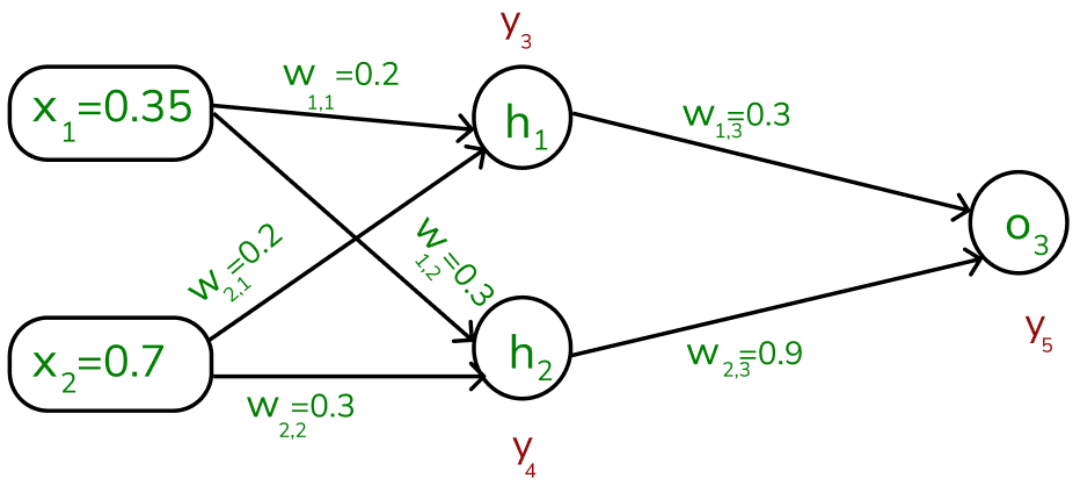
$aj=\sum(wi,j*xi)$

Where,

- aj is the weighted sum of all the inputs and weights at each node,
- wi,j – represents the weights associated with the jth input to the ith neuron,
- xi – represents the value of the jth input,

$yj=F(aj)=1/(1+e^{aj})$ is the output value, F denotes the activation function [sigmoid activation function is used here), which transforms the weighted sum into the output value.

Step 2: To compute the forward pass, we need to compute the output for y3 , y4 , and y5.



We start by calculating the weights and inputs by using the formula: $aj=\sum(wi,j*xi)$ To find y3 , we need to consider its incoming edges along with its weight and input. Here the incoming edges are from X1 and X2.

At h1 node,

$$a_1 = (w_{1,1}x_1) + (w_{2,1}x_2)$$
$$= (0.2 * 0.35) + (0.2 * 0.7)$$
$$= 0.21$$

Once, we calculated the a_1 value, we can now proceed to find the y_3 value:

$$y_j = F(a_j) = \frac{1}{1+e^{-aj}}$$

$$y_3 = F(0.21) = \frac{1}{1+e^{-0.21}}$$

$$y_3 = 0.56$$

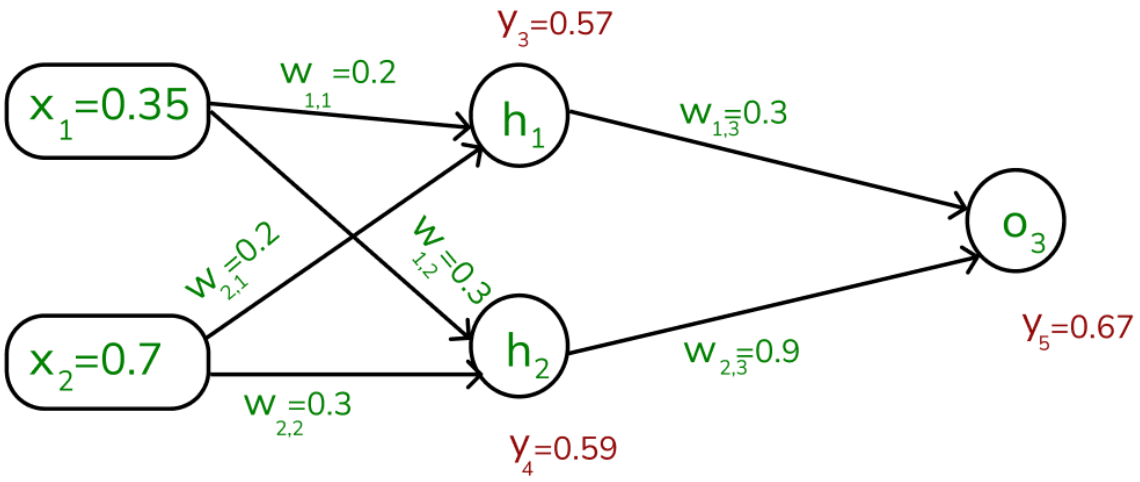
Similarly find the values of y_4 at **h_2** and y_5 at O_3 ,

$$a2 = (w_{1,2} * x_1) + (w_{2,2} * x_2) = (0.3 * 0.35) + (0.3 * 0.7) = 0.315$$

$$y_4 = F(0.315) = \frac{1}{1+e^{-0.315}}$$

$$a3 = (w_{1,3} * y_3) + (w_{2,3} * y_4) = (0.3 * 0.57) + (0.9 * 0.59) = 0.702$$

$$y_5 = F(0.702) = \frac{1}{1+e^{-0.702}} = 0.67$$



Note that, our actual output is 0.5 but we obtained 0.67. To calculate the error, we can use the below formula:
Errorj=ytarget–y5
Error = 0.5 – 0.67
= -0.17
Using this error value, we will be backpropagating.

Implementing Backward Propagation

Each weight in the network is changed by,

$\nabla wij = \eta\ ?j\ Oj$
$?j = Oj\ (1-Oj)(tj - Oj)$ (if j is an output unit)
$?j = Oj\ (1-O)\sum k\ ?k\ wkj$ (if j is a hidden unit)
where ,

η is the constant which is considered as learning rate,
tj is the correct output for unit j
?j is the error measure for unit j

Step 3: To calculate the backpropagation, we need to start from the output unit:

To compute the ?5, we need to use the output of forward pass,
?5 = y5(1-y5) (ytarget -y5)
= 0.67(1-0.67) (-0.17)
= -0.0376

For hidden unit,

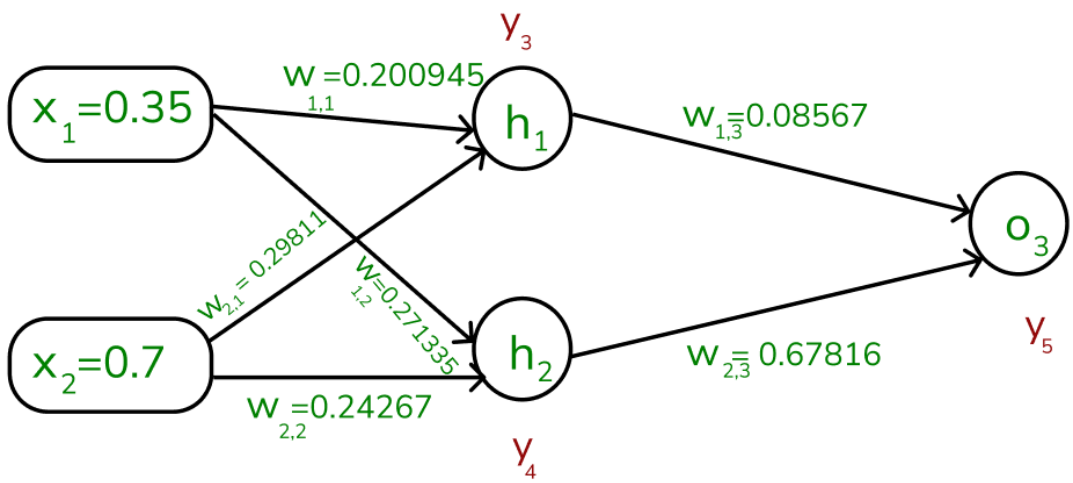
To compute the hidden unit, we will take the value of ?5
?3 = y3(1-y3) (w1,3 * ?5)
=0.56(1-0.56) (0.3*-0.0376)
=-0.0027
?4 = y4 (1-y5) (w2,3 * ?5)
=0.59(1-0.59) (0.9*-0.0376)
=-0.0819

Step 4: We need to update the weights, from output unit to hidden unit,

$\nabla\ wj,i = \eta\ ?j\ Oj$

Note- Here our learning rate is 1

∇ w2,3 = η ?5 O4
= 1 * (-0.376) * 0.59
= -0.22184
We will be updating the weights based on the old weight of the network,
w2,3(new) = ∇ w4,5 + w4,5 (old)
= -0.22184 + 0.9
= 0.67816
From hidden unit to input unit,
For an hidden to input node, we need to do calculations by the following;
∇ w1,1 = η ?3 O4
= 1 * (-0.0027) * 0.35
= 0.000945
Similarly, we need to calculate the new weight value using the old one:
w1,1(new) = ∇ w1,1+ w1,1 (old)
= 0.000945 + 0.2
= 0.200945
Similarly, we update the weights of the other neurons: The new weights are mentioned below
w1,2 (new) = 0.271335
w1,3 (new) = 0.08567
w2,1 (new) = 0.29811
w2,2 (new) = 0.24267
The updated weights are illustrated below,



Through backward pass the weights are updated

Once, the above process is done, we again perform the forward pass to find if we obtain the actual output as 0.5.
While performing the forward pass again, we obtain the following values:
y3 = 0.57
y4 = 0.56
y5 = 0.61
We can clearly see that our y5 value is 0.61 which is not an expected actual output, So again we need to find the error and backpropagate through the network by updating the weights until the actual output is obtained.
Error=ytarget–y5
= 0.5 – 0.61
= -0.11
This is how the backpropagate works, it will be performing the forward pass first to see if we obtain the actual output, if not we will be finding the error rate and then backpropagating backwards through the layers in the network by adjusting the weights according to the error rate. This process is said to be continued until the actual output is gained by the neural network.