EDU ENGINEERING

PIONEER OF ENGINEERING NOTES

TAMIL NADU'S BEST EDTECH PLATFORM FOR ENGINEERING

CONNECT WITH US

WEBSITE: www.eduengineering.net

TELEGRAM: @eduengineering

INSTAGRAM: @eduengineering

- **Regular Updates for all Semesters**
- **All Department** Notes AVAILABLE
- **Handwritten** Notes AVAILABLE
- **Past Year** Question Papers AVAILABLE
- **Subject wise** Question Banks AVAILABLE
- **Important Questions** for Semesters AVAILABLE
- Various **Author Books** AVAILABLE

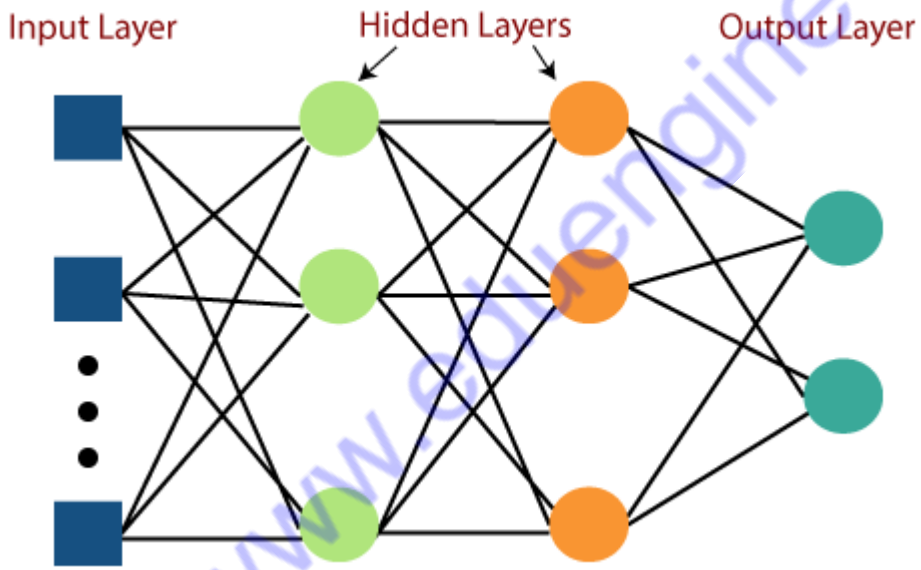
UNIT IV NEURAL NETWORKS 9

Multilayer perceptron, activation functions, network training – gradient descent optimization – stochastic gradient descent, error backpropagation, from shallow networks to deep networks –Unit saturation (aka the vanishing gradient problem) – ReLU, hyper parameter tuning, batch normalization, regularization, dropout

4.1 Multi-layer Perceptron

Multi-Layer perceptron defines the most complex architecture of artificial neural networks. It is substantially formed from multiple layers of the perceptron.

The pictorial representation of multi-layer perceptron learning is as shown below-



MLP networks are used for supervised learning format. A typical learning algorithm for MLP networks is also called **back propagation's algorithm**.

A multilayer perceptron (MLP) is a feed forward artificial neural network that generates a set of outputs from a set of inputs. An MLP is characterized by several layers of input nodes connected as a directed graph between the input nodes connected as a directed graph between the input and output layers. MLP uses backpropagation for training the network. MLP is a deep learning method.

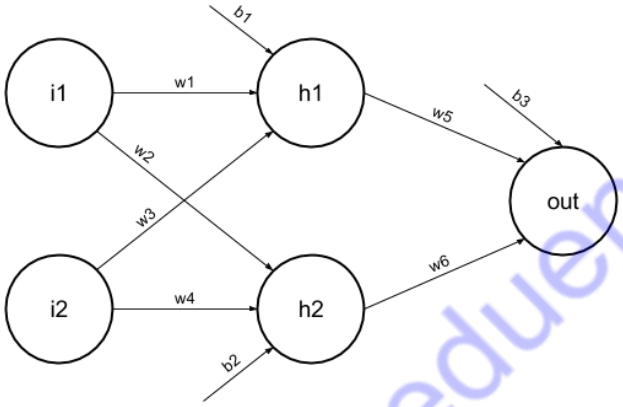
4.2 Activation Functions in Neural Networks

Elements of a Neural Network

Input Layer: This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.
Hidden Layer*:* Nodes of this layer are not exposed to the outer world, they are part of the abstraction provided by any neural network. The hidden layer performs all sorts of computation on the features entered through the input layer and transfers the result to the output layer.
Output Layer: This layer bring up the information learned by the network to the outer world.
What is an activation function and why use them?
The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

Explanation: We know, the neural network has neurons that work in correspondence with *weight, bias,* and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as ***back-propagation***. Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.
Why do we need Non-linear activation function?
A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Mathematical proof
Suppose we have a Neural net like this :-



Elements of the diagram are as follows:
Hidden layer i.e. layer 1:
z(1) = W(1)X + b(1) a(1)

Here,

- *z(1) is the vectorized output of layer 1*
- *W(1) be the vectorized weights assigned to neurons of hidden layer i.e. w1, w2, w3 and w4*
- *X be the vectorized input features i.e. i1 and i2*
- *b is the vectorized bias assigned to neurons in hidden layer i.e. b1 and b2*
- *a(1) is the vectorized form of any linear function.*

*(**Note:** We are not considering activation function here)*

Layer 2 i.e. output layer :-
Note : *Input for layer 2 is output from layer 1*
z(2) = W(2)a(1) + b(2)

a(2) = z(2)

Calculation at Output layer
*z(2) = (W(2) * [W(1)X + b(1)]) + b(2)*

*z(2) = [W(2) * W(1)] * X + [W(2)*b(1) + b(2)]*

Let,

*[W(2) * W(1)] = W*

*[W(2)*b(1) + b(2)] = b*

*Final output : z(2) = W*X + b*

which is again a linear function

This observation results again in a linear function even after applying a hidden layer, hence we can conclude that, doesn't matter how many hidden layer we attach in neural net, all layers will behave same way because ***the composition of two linear function is a linear function itself***. Neuron can not learn with just a linear function attached to it. A non-linear activation function will let it learn as per the difference w.r.t error. **Hence we need an activation function.**

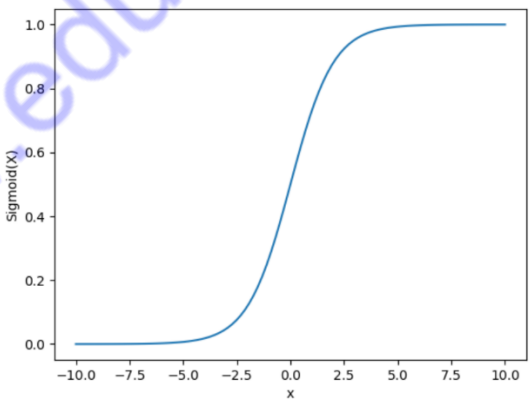
Variants of Activation Function

Linear Function

- **Equation :** Linear function has the equation similar to as of a straight line i.e. **y = x**
- No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
- **Range :** -inf to +inf
- **Uses : Linear activation function** is used at just one place i.e. output layer.
- **Issues :** If we will differentiate linear function to bring non-linearity, result will no more depend on *input "x"* and function will become constant, it won't introduce any ground-breaking behavior to our algorithm.

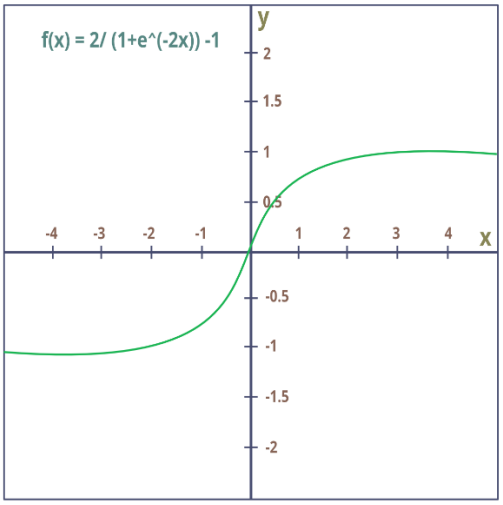
For example : Calculation of price of a house is a regression problem. House price may have any big/small value, so we can apply linear activation at output layer. Even in this case neural net must have any non-linear function at hidden layers.

Sigmoid Function



- It is a function which is plotted as **'S'** shaped graph.
- **Equation :** $A = 1/(1 + e^{-x})$
- **Nature :** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range :** 0 to 1
- **Uses :** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.

Tanh Function



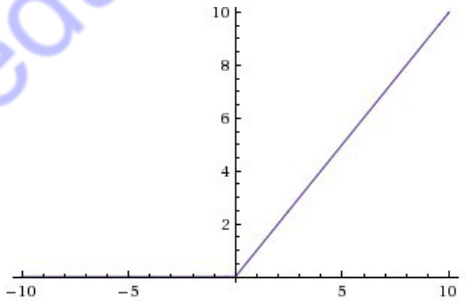
ƏG

- The activation that works almost always better than sigmoid function is Tanh function also known as **Tangent Hyperbolic function**. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
- **Equation :-**

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

- **Value Range :-** -1 to +1
- **Nature :-** non-linear
- **Uses :-** Usually used in hidden layers of a neural network as it's values lies between **-1 to 1** hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.

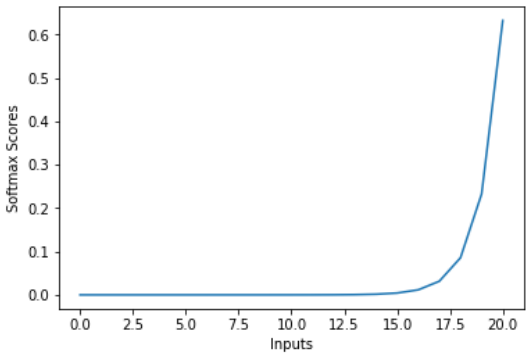
RELU Function



- It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.
- **Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range :-** [0, inf)
- **Nature :-** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- **Uses :-** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

In simple words, RELU learns *much faster* than sigmoid and Tanh function.

Softmax Function



The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi- class classification problems.

- **Nature :-** non-linear
- **Uses :-** Usually used when trying to handle multiple classes. the softmax function was commonly found in the output layer of image classification problems.The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
- **Output:-** The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.
- The basic rule of thumb is if you really don't know what activation function to use, then simply use *RELU* as it is a general activation function in hidden layers and is used in most cases these days.
- If your output is for binary classification then, *sigmoid function* is very natural choice for output layer.
- If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each classes.

4.3. Network Training

- **Training: It is the process in which the network is taught to change its weight and bias.**
- Learning: It is the internal process of training where the artificial neural systemlearns to update/adapt the weights and biases.

Different Training /Learning procedure available in ANN are

- *Supervised learning*
- **Unsupervised learning**
- *Reinforced learning*
- **Hebbian learning**
- *Gradient descent learning*
- **Competitive learning**
- *Stochastic learning*

1.4.1. Requirements of Learning Laws:

- *Learning Law should lead to convergence of weights*
- **Learning or training time should be less for capturing the**

information from the trainingpairs

- *Learning should use the local information*
    - **Learning process should able to capture the complex non linear mapping availablebetween the input & output pairs**
- *Learning should able to capture as many as patterns as possible*
    - **Storage of pattern information's gathered at the time of learning should be high for thegiven network**

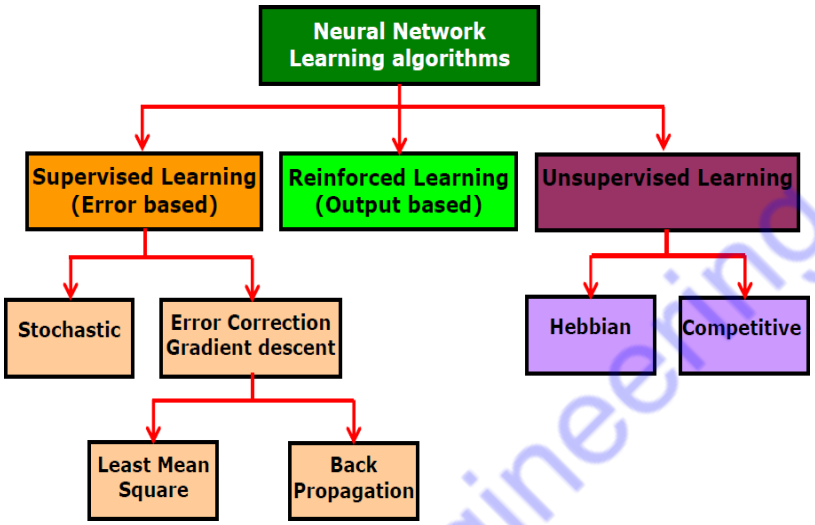


Figure 3: Different Training methods of ANN

Supervised learning :

Every input pattern that is used to train the network is associated with an output pattern which isthe target or the desired pattern.

A teacher is assumed to be present during the training process, when a comparison is made between the network's computed output and the correct expected output, to determine the error.The error can then be used to change network parameters, which result in an improvement in performance.

Unsupervised learning:

In this learning method the target output is not presented to the network.It is as if there is no teacher to present the desired patterns and hence the system learns of its own by discovering and adapting to structural features in the input patterns.

Reinforced learning:

In this method, a teacher though available, doesnot present the expected answer but only indicates if the computed output correct or incorrect.The information provided helps the network in the learning process.

Hebbian learning:

This rule was proposed by Hebb and is based on correlative weight adjustment.This is the

oldestlearning mechanism inspired by biology.In this, the input-output pattern pairs (x_i, y_i) are associated by the weight matrix W, known as the correlation matrix.

It is computed as

$$W = \sum_{i=1}^{n} x_i y_i^T \quad \text{---------- eq(1)}$$

Here y_i^T is the transposeof the associated output vector y_i.Numerous variants of the rule havebeen proposed.

Gradient descent learning:

This is based on the minimization of error E defined in terms of weights and activation function of the network.Also it is required that the activation function employed by the network is differentiable, as the weight update is dependent on the gradient of the error E.

Thus if Δw_{ij} is the weight update of the link connecting the i^{th} and j^{th} neuron of the two neighbouring layers, then Δw_{ij} is defined as,

$$\Delta w_{ij} = \eta \frac{\partial E}{\partial w_{ij}} \quad \text{---------- eq(2)}$$

Where, η is the learning rate parameter and $\frac{\partial E}{\partial w_{ij}}$ is the error gradient with reference to the

weight w_{ij}.

4.4 Gradient Descent:

- Gradient Descent is a popular optimization technique in Machine Learning and Deep Learning of the learning algorithms.
- A gradient is the slope of a function.
- It measures the degree of change of a variable in response to the changes of another variable.
- Mathematically, Gradient Descent is a convex function whose output is the partial derivativeof a set of parameters of its inputs.
- The greater the gradient, the steeper the slope.Starting from an initial value, Gradient Descent is run iteratively to find the optimal values of the parameters to find the minimum possible value of the given cost function.

Types of Gradient Descent:

Typically, there are three types of Gradient Descent:

1. Batch Gradient Descent
2. Stochastic Gradient Descent
3. Mini-batch Gradient Descent

Stochastic Gradient Descent (SGD):

- The word 'stochastic' means a system or a process that is linked with a random probability.
- Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration.
- In Gradient Descent, there is a term called "batch" which denotes the totalnumber of samples from a dataset that is used for calculating the gradient for each iteration.
- In typicalGradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset.
- Although, using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, but the problem arises when our datasets gets big.
- Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima is reached. Hence, it becomes computationally very expensive to perform

4.5 Backpropagation

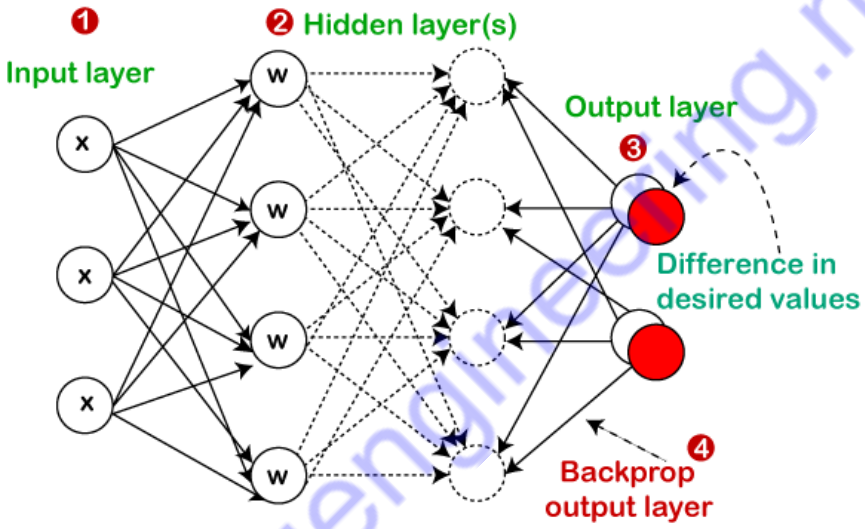
- The backpropagation consists of an input layer of neurons, an output layer, and at least one hidden layer.
- The neurons perform a weighted sum upon the input layer, which is then used by the activation function as an input, especially by the sigmoid activation function.
- It also makes use of supervised learning to teach the network.
- It constantly updates the weights of the network until the desired output is met by the network.

❖ It includes the following factors that are responsible for the training and performance of the network:

- Random (initial) values of weights.
- A number of training cycles.
- A number of hidden neurons.
- The training set.
- Teaching parameter values such as learning rate and momentum.

Working of Backpropagation

Consider the diagram given below.



1. The preconnected paths transfer the inputs **X**.
2. Then the weights **W** are randomly selected, which are used to model the input.
3. After then, the output is calculated for every individual neuron that passes from the input layer to the hidden layer and then to the output layer.
4. Lastly, the errors are evaluated in the outputs. **$Error_B$= Actual Output - Desired Output**
5. The errors are sent back to the hidden layer from the output layer for adjusting the weights to lessen the error.
6. Until the desired result is achieved, keep iterating all of the processes.

Need of Backpropagation

- Since it is fast as well as simple, it is very easy to implement.
- Apart from no of inputs, it does not encompass of any other parameter to perform tuning.

- As it does not necessitate any kind of prior knowledge, so it tends out to be more flexible.
- It is a standard method that results well.

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:

- Static Back-propagation
- Recurrent Backpropagation

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.

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.

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.

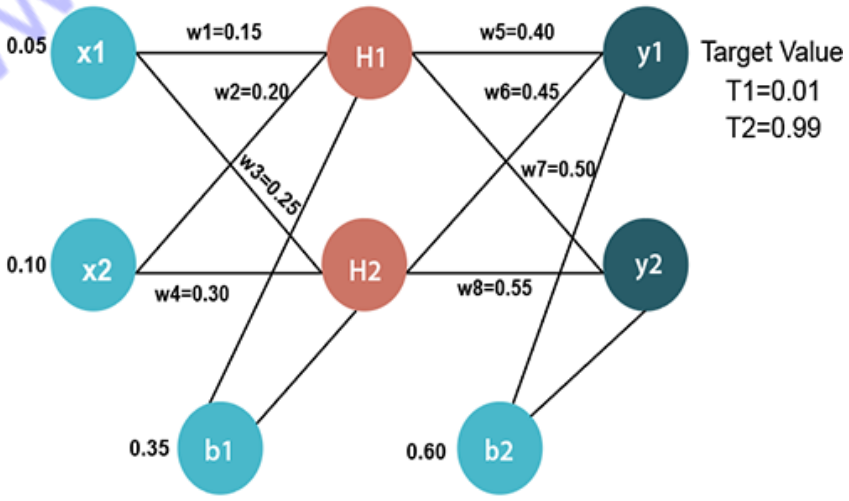
Backpropagation Process in Deep Neural Network

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.



Input values

X1=0.05
X2=0.10

Initial weight

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

$$H1=x1\times w_1+x2\times w_2+b1$$
$$H1=0.05\times 0.15+0.10\times 0.20+0.35$$

H1=0.3775

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

$$H1_{final}=\frac{1}{1+\frac{1}{e^{H1}}}$$

$$H1_{final}=\frac{1}{1+\frac{1}{e^{0.3775}}}$$

$$\mathbf{H1_{final}=0.593269992}$$

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

$$H2=x1\times w_3+x2\times w_4+b1$$
$$H2=0.05\times 0.25+0.10\times 0.30+0.35$$

H2=0.3925

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

$$H2_{final} = \frac{1}{1+\frac{1}{e^{H2}}}$$

$$H2_{final} = \frac{1}{1+\frac{1}{e^{0.3925}}}$$

$$\mathbf{H2_{final} = 0.596884378}$$

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

$$y1=H1\times w_5+H2\times w_6+b2$$
$$y1=0.593269992\times 0.40+0.596884378\times 0.45+0.60$$
$$\mathbf{y1=1.10590597}$$

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

$$y1_{final} = \frac{1}{1+\frac{1}{e^{y1}}}$$

$$y1_{final} = \frac{1}{1+\frac{1}{e^{1.10590597}}}$$

$$\mathbf{y1_{final} = 0.75136507}$$

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

$$y2=H1\times w_7+H2\times w_8+b2$$
$$y2=0.593269992\times 0.50+0.596884378\times 0.55+0.60$$
$$\mathbf{y2=1.2249214}$$

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

$$y2_{final} = \frac{1}{1+\frac{1}{e^{y2}}}$$

$$y2_{final} = \frac{1}{1+\frac{1}{e^{1.2249214}}}$$

$$\mathbf{y2_{final} = 0.772928465}$$

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

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

So, the total error is

$$= \frac{1}{2}(t1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2$$

$$= \frac{1}{2}(0.01 - 0.75136507)^2 + \frac{1}{2}(0.99 - 0.772928465)^2$$

$$= 0.274811084 + 0.0235600257$$

$$\mathbf{E_{total} = 0.29837111}$$

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.

$$Error_w = \frac{\partial E_{total}}{\partial w}$$

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

$$Error_{w5} = \frac{\partial E_{total}}{\partial w5} \dots\dots\dots (1)$$

$$E_{total} = \frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2 \dots\dots\dots (2)$$

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

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial y1_{final}} \times \frac{\partial y1\_final}{\partial y1} \times \frac{\partial y1}{\partial w5} \dots\dots\dots (3)$$

Now, we calculate each term one by one to differentiate E_{total} with respect to w5 as

$$\frac{\partial E_{total}}{\partial y1_{final}} = \frac{\partial(\frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2)}{\partial y1_{final}}$$

$$= 2 \times \frac{1}{2} \times (T1 - y1_{final})^{2-1} \times (-1) + 0$$

$$= -(T1 - y1_{final})$$

$$= -(0.01 - 0.75136507)$$

$$\frac{\partial E_{total}}{\partial y1_{final}} = \mathbf{0.74136507} \ldots\ldots\ldots (4)$$

$$y1_{final} = \frac{1}{1 + e^{-y1}} \ldots\ldots\ldots\ldots\ldots (5)$$

$$\frac{\partial y1_{final}}{\partial y1} = \frac{\partial(\frac{1}{1 + e^{-y1}})}{\partial y1}$$

$$= \frac{e^{-y1}}{(1 + e^{-y1})^2}$$

$$= e^{-y1} \times (y1_{final})^2 \ldots\ldots\ldots\ldots (6)$$

$$y1_{final} = \frac{1}{1 + e^{-y1}}$$

$$e^{-y1} = \frac{1 - y1_{final}}{y1_{final}} \ldots\ldots\ldots\ldots (7)$$

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

$$= \frac{1 - y1_{final}}{y1_{final}} \times (y1_{final})^2$$

$$= y1_{final} \times (1 - y1_{final})$$

$$= 0.75136507 \times (1 - 0.75136507)$$

$$\frac{\partial y1_{final}}{\partial y1} = \mathbf{0.186815602} \ldots\ldots\ldots (8)$$

$$y1 = H1_{final} \times w5 + H2_{final} \times w6 + b2 \ldots\ldots\ldots\ldots\ldots (9)$$

$$\frac{\partial y1}{\partial w5} = \frac{\partial(H1_{final} \times w5 + H2_{final} \times w6 + b2)}{\partial w5}$$

$$= H1_{final}$$

$$\frac{\partial y1}{\partial w5} = \mathbf{0.596884378} \ldots\ldots\ldots (10)$$

So, we put the values of $\frac{\partial E_{total}}{\partial y1_{final}}$, $\frac{\partial y1_{final}}{\partial y1}$, and $\frac{\partial y1}{\partial w5}$ in equation no (3) to find the final result.

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial y1_{final}} \times \frac{\partial y1_{final}}{\partial y1} \times \frac{\partial y1}{\partial w5}$$

$$= 0.74136507 \times 0.186815602 \times 0.593269992$$

$$\mathbf{Error_{w5} = \frac{\partial E_{total}}{\partial w5} = 0.0821670407 \ldots\ldots\ldots (11)}$$

Now, we will calculate the updated weight $w5_{new}$ with the help of the following formula

$$w5_{new} = w5 - \eta \times \frac{\partial E_{total}}{\partial w5} \text{ Here, } \eta = \text{learning rate} = 0.5$$

$$= 0.4 - 0.5 \times 0.0821670407$$

$$\mathbf{w5_{new} = 0.35891648 \ldots\ldots\ldots (12)}$$

In the same way, we calculate $w6_{new}$,$w7_{new}$, and $w8_{new}$ and this will give us the following values

$w5_{new}$=0.35891648
$w6_{new}$=408666186
$w7_{new}$=0.511301270

$w8_{new}$=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

$$Error_{w1} = \frac{\partial E_{total}}{\partial w1}$$

$$E_{total} = \frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2$$

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

$$\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_{total}}{\partial H1_{final}} \times \frac{\partial H1\_final}{\partial H1} \times \frac{\partial H1}{\partial w1} \ldots\ldots\ldots\ldots (13)$$

Now, we calculate each term one by one to differentiate E_{total} with respect to w1 as

$$\frac{\partial E_{total}}{\partial H1_{final}} = \frac{\partial(\frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2)}{\partial H1} \ldots\ldots\ldots\ldots (14)$$

We again split this because there is no any $H1^{final}$ term in E^{toatal} as

$$\frac{\partial E_{total}}{\partial H1_{final}} = \frac{\partial E_1}{\partial H1_{final}} + \frac{\partial E_2}{\partial H1_{final}} \dots\dots\dots (15)$$

$\frac{\partial E_1}{\partial H1_{final}}$ and $\frac{\partial E_2}{\partial H1_{final}}$ will again split because in E1 and E2 there is no H1 term. Splitting is done as

$$\frac{\partial E_1}{\partial H1_{final}} = \frac{\partial E_1}{\partial y1} \times \frac{\partial y1}{\partial H1_{final}} \dots\dots\dots (16)$$

$$\frac{\partial E_2}{\partial H1_{final}} = \frac{\partial E_2}{\partial y2} \times \frac{\partial y2}{\partial H1\_final} \dots\dots\dots (17)$$

We again Split both $\frac{\partial E_1}{\partial y1}$ and $\frac{\partial E_2}{\partial y2}$ because there is no any y1 and y2 term in E1 and E2. We split it as

$$\frac{\partial E_1}{\partial y1} = \frac{\partial E_1}{\partial y1_{fianl}} \times \frac{\partial y1_{final}}{\partial y1} \dots\dots\dots (18)$$

$$\frac{\partial E_2}{\partial y2} = \frac{\partial E_2}{\partial y2_{fianl}} \times \frac{\partial y2_{final}}{\partial y2} \dots\dots\dots (19)$$

Now, we find the value of $\frac{\partial E_1}{\partial y1}$ and $\frac{\partial E_2}{\partial y2}$ by putting values in equation (18) and (19) as

From equation (18)

$$\frac{\partial E_1}{\partial y1} = \frac{\partial E_1}{\partial y1_{fianl}} \times \frac{\partial y1_{final}}{\partial y1}$$

$$= \frac{\partial(\frac{1}{2}(T1 - y1_{final})^2)}{\partial y1_{final}} \times \frac{\partial y1_{final}}{\partial y1}$$

$$= 2 \times \frac{1}{2}(T1 - y1_{final}) \times (-1) \times \frac{\partial y1_{final}}{\partial y1}$$

From equation (8)

$$= 2 \times \frac{1}{2}(0.01 - 0.75136507) \times (-1) \times 0.186815602$$

$$\mathbf{\frac{\partial E_1}{\partial y1} = 0.138498562} \dots\dots\dots (20)$$

From equation (19)

$$\frac{\partial E_2}{\partial y2} = \frac{\partial E_2}{\partial y2_{fianl}} \times \frac{\partial y2_{final}}{\partial y2}$$

$$= \frac{\partial(\frac{1}{2}(T2 - y2_{final})^2)}{\partial y2_{final}} \times \frac{\partial y2_{final}}{\partial y2}$$

$$= 2 \times \frac{1}{2}(T2 - y2_{final}) \times (-1) \times \frac{\partial y2_{final}}{\partial y2} \dots\dots\dots (21)$$

$$y2_{final} = \frac{1}{1 + e^{-y2}} \dots\dots\dots\dots\dots\dots (22)$$

$$\frac{\partial y2_{fianl}}{\partial y2} = \frac{\partial(\frac{1}{1 + e^{-y2}})}{\partial y2}$$

$$= \frac{e^{-y2}}{(1 + e^{-y2})^2}$$

$$= e^{-y2} \times (y2_{final})^2 \dots\dots\dots\dots\dots (23)$$

$$y2_{final} = \frac{1}{1 + e^{-y2}}$$

$$e^{-y2} = \frac{1 - y2_{final}}{y2_{final}} \dots\dots\dots\dots\dots (24)$$

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

$$= \frac{1 - y2_{final}}{y2_{final}} \times (y2_{final})^2$$

$$= y2_{final} \times (1 - y2_{final})$$

$$= 0.772928465 \times (1 - 0.772928465)$$

$$\mathbf{\frac{\partial y2_{fianl}}{\partial y2} = 0.175510053 \dots\dots\dots (25)}$$

From equation (21)

$$= 2 \times \frac{1}{2}(0.99 - 0.772928465) \times (-1) \times 0.175510053$$

$$\mathbf{\frac{\partial E_1}{\partial y1} = -0.0380982366126414 \dots\dots\dots (26)}$$

Now from equation (16) and (17)

$$\frac{\partial E_1}{\partial H1_{final}} = \frac{\partial E_1}{\partial y1} \times \frac{\partial y1}{\partial H1_{final}}$$

$$= 0.138498562 \times \frac{\partial(H1_{final} \times w_5 + H2_{final} \times w_6 + b2)}{\partial H1_{final}}$$

$$= 0.138498562 \times \frac{\partial(H1_{final} \times w_5 + H2_{final} \times w_6 + b2)}{\partial H1_{final}}$$

$$= 0.138498562 \times w5$$

$$= 0.138498562 \times 0.40$$

$$\mathbf{\frac{\partial E_1}{\partial H1_{final}} = 0.0553994248 \ldots\ldots\ldots (27)}$$

$$\frac{\partial E_2}{\partial H1_{final}} = \frac{\partial E_2}{\partial y2} \times \frac{\partial y2}{\partial H1\_final}$$

$$= -0.0380982366126414 \times \frac{\partial(H1_{final} \times w_7 + H2_{final} \times w_8 + b2)}{\partial H1_{final}}$$

$$= -0.0380982366126414 \times w7$$

$$= -0.0380982366126414 \times 0.50$$

$$\mathbf{\frac{\partial E_2}{\partial H1_{final}} = -0.0190491183063207 \ldots\ldots\ldots (28)}$$

Put the value of $\frac{\partial E_1}{\partial H1_{final}}$ and $\frac{\partial E_2}{\partial H1_{final}}$ in equation (15) as

$$\frac{\partial E_{total}}{\partial H1_{final}} = \frac{\partial E_1}{\partial H1_{final}} + \frac{\partial E_2}{\partial H1_{final}}$$

$$= 0.0553994248 + (-0.0190491183063207)$$

$$\mathbf{\frac{\partial E_{total}}{\partial H1_{final}} = 0.0364908241736793 \ldots\ldots\ldots (29)}$$

We have $\frac{\partial E_{total}}{\partial H1_{final}}$, we need to figure out $\frac{\partial H1\_final}{\partial H1}$, $\frac{\partial H1}{\partial w1}$ as

$$\frac{\partial H1_{final}}{\partial H1} = \frac{\partial(\frac{1}{1+e^{-H1}})}{\partial H1}$$

$$= \frac{e^{-H1}}{(1+e^{-H1})^2}$$

$$e^{-H1} \times (H1_{final})^2 \dots\dots\dots (30)$$

$$H1_{final} = \frac{1}{1+e^{-H1}}$$

$$e^{-H1} = \frac{1 - H1_{final}}{H1_{final}} \dots\dots\dots\dots (31)$$

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

$$= \frac{1 - H1_{final}}{H1_{final}} \times (H1_{final})^2$$

$$= H1_{final} \times (1 - H1_{final})$$

$$= 0.593269992 \times (1 - 0.593269992)$$

$$\mathbf{\frac{\partial H1_{final}}{\partial H1} = 0.2413007085923199}$$

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:

$$H1 = H1_{final} \times w5 + H2_{final} \times w6 + b2 \dots\dots\dots\dots\dots (32)$$

$$\frac{\partial y1}{\partial w1} = \frac{\partial(x1 \times w1 + x2 \times w3 + b1 \times 1)}{\partial w1}$$

$$= x1$$

$$\mathbf{\frac{\partial H1}{\partial w1} = 0.05 \dots\dots\dots (33)}$$

So, we put the values of $\frac{\partial E_{total}}{\partial H1_{final}}$, $\frac{\partial H1_{final}}{\partial H1}$, and $\frac{\partial H1}{\partial w1}$ in equation (13) to find the final result.

$$\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_{total}}{\partial H1_{final}} \times \frac{\partial H1_{final}}{\partial H1} \times \frac{\partial H1}{\partial w1}$$

$$= 0.0364908241736793 \times 0.2413007085923199 \times 0.05$$

$$\mathbf{Error_{w1} = \frac{\partial E_{total}}{\partial w1} = 0.000438568 \dots\dots\dots (34)}$$

Now, we will calculate the updated weight $w1_{new}$ with the help of the following formula

$$w1_{new} = w1 - \eta \times \frac{\partial E_{total}}{\partial w1} \text{ Here } \eta = \text{learning rate} = 0.5$$

$$= 0.15 - 0.5 \times 0.000438568$$

$$\mathbf{w1_{new} = 0.149780716 \ldots\ldots\ldots (35)}$$

In the same way, we calculate $w2_{new}$,$w3_{new}$, and w4 and this will give us the following values

$w1_{new}$=0.149780716
$w2_{new}$=0.19956143
$w3_{new}$=0.24975114

$w4_{new}$=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.

2.5.1 Difference Between a Shallow Net & Deep Learning Net:

Sl.No	Shallow Net's	Deep Learning Net's
1	One Hidden layer(or very less no. ofHidden Layers)	Deep Net's has many layers of Hiddenlayers with more no. of neurons in each layers
2	Takes input only as VECTORS	DL can have raw data like image, textas inputs
3	Shallow net's needs more parametersto have better fit	DL can fit functions better with lessparameters than a shallow network
4	Shallow networks with one Hidden layer (same no of neurons as DL) cannot place complex functions overthe input space	DL can compactly express highly complex functions over input space
5	The number of units in a shallow network grows exponentially withtask complexity.	DL don't need to increase it size(neurons) for complex problems

6	Shallow network is more difficult to train with our current algorithms (e.g.it has issues of local minima etc)	Training in DL is easy and no issue oflocal minima in DL

4.6 The Vanishing Gradient Problem

The Problem, Its Causes, Its Significance, and Its Solutions

The problem:

As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train.

Why:

Certain activation functions, like the sigmoid function, squishes a large input space into a small input space between 0 and 1. Therefore, a large change in the input of the sigmoid function will cause a small change in the output. Hence, the derivative becomes small.

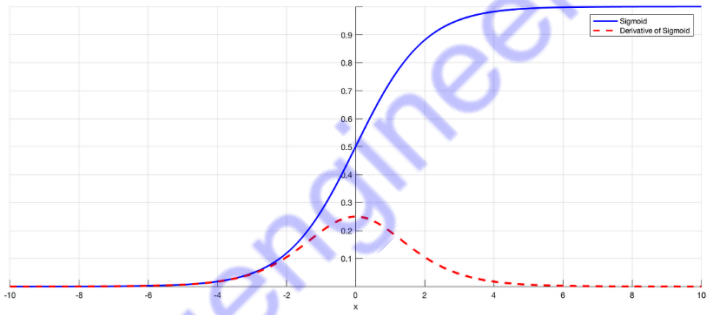


Image 1: The sigmoid function and its derivative

As an example, Image 1 is the sigmoid function and its derivative. Note how when the inputs of the sigmoid function becomes larger or smaller (when |x| becomes bigger), the derivative becomes close to zero.

Why it's significant:

For shallow network with only a few layers that use these activations, this isn't a big problem. However, when more layers are used, it can cause the gradient to be too small for training to work effectively.

Gradients of neural networks are found using backpropagation. Simply put, backpropagation finds the derivatives of the network by moving layer by layer from the final layer to the initial one. By the chain rule, the derivatives of each layer are multiplied down the network (from the final layer to the initial) to compute the derivatives of the initial layers.

However, when *n* hidden layers use an activation like the sigmoid function, *n* small derivatives are multiplied together. Thus, the gradient decreases exponentially as we propagate down to the initial layers.

A small gradient means that the weights and biases of the initial layers will not be updated effectively with each training session. Since these initial layers are often crucial to recognizing the core elements of the input data, it can lead to overall inaccuracy of the whole network.

Solutions:

The simplest solution is to use other activation functions, such as ReLU, which doesn't cause a small derivative.

Residual networks are another solution, as they provide residual connections straight to earlier layers. As seen in Image 2, the residual connection directly adds the value at the beginning of the block, **x**, to the end of the block (F(x)+x). This residual connection doesn't go through activation functions that "squashes" the derivatives, resulting in a higher overall derivative of the block.

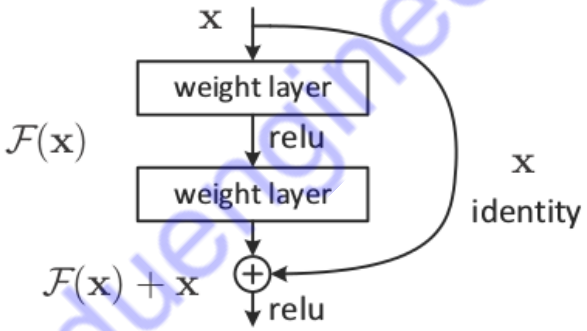


Image 2: A residual block

Finally, batch normalization layers can also resolve the issue. As stated before, the problem arises when a large input space is mapped to a small one, causing the derivatives to disappear. In Image 1, this is most clearly seen at when |x| is big. Batch normalization reduces this problem by simply normalizing the input so |x| doesn't reach the outer edges of the sigmoid function. As seen in Image 3, it normalizes the input so that most of it falls in the green region, where the derivative isn't too small.

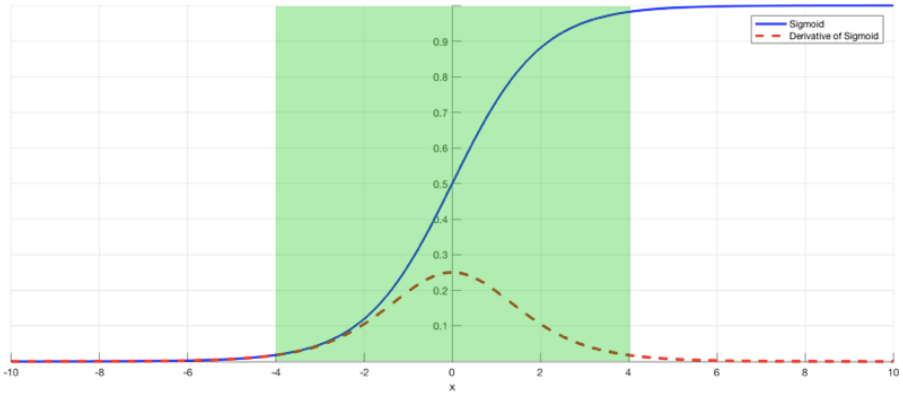


Image 3: Sigmoid function with restricted inputs

4.7 Hyperparameters in Machine Learning

Hyperparameters in Machine learning are those parameters that are explicitly defined by the user to control the learning process. These hyperparameters are used to improve the learning of the model, and their values are set before starting the learning process of the model.

- Here the prefix "hyper" suggests that the parameters are top-level parameters that are used in controlling the learning process.
- The value of the Hyperparameter is selected and set by the machine learning engineer before the learning algorithm begins training the model.
- **Hence, these are external to the model, and their values cannot be changed during the training process.**

Some examples of Hyperparameters in Machine Learning

- The k in kNN or K-Nearest Neighbour algorithm
- Learning rate for training a neural network
- Train-test split ratio
- Batch Size
- Number of Epochs
- Branches in Decision Tree
- Number of clusters in Clustering Algorithm

Model Parameters:

Model parameters are configuration variables that are internal to the model, and a model learns them on its own. For example**, W Weights or Coefficients of independent variables in the Linear regression model**. or **Weights or Coefficients of independent variables in SVM, weight, and biases of a neural network, cluster centroid in clustering.** Some key points for model parameters are as follows:

- They are used by the model for making predictions.
- They are learned by the model from the data itself
- These are usually not set manually.
- These are the part of the model and key to a machine learning Algorithm.

Model Hyperparameters:

Hyperparameters are those parameters that are explicitly defined by the user to control the learning process. Some key points for model parameters are as follows:

- These are usually defined manually by the machine learning engineer.
- One cannot know the exact best value for hyperparameters for the given problem. The best value can be determined either by the rule of thumb or by trial and error.
- Some examples of Hyperparameters are **the learning rate for training a neural network, K in the KNN algorithm,**

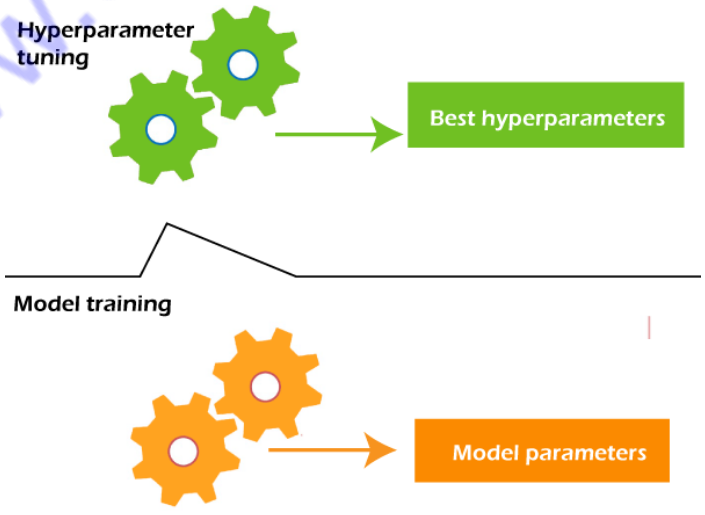
Categories of Hyperparameters

Broadly hyperparameters can be divided into two categories, which are given below:

1. **Hyperparameter for Optimization**
2. **Hyperparameter for Specific Models**

Hyperparameter for Optimization

The process of selecting the best hyperparameters to use is known as hyperparameter tuning, and the tuning process is also known as hyperparameter optimization. Optimization parameters are used for optimizing the model.



Some of the popular optimization parameters are given below:

- **Learning Rate:** The learning rate is the hyperparameter in optimization algorithms that controls how much the model needs to change in response to the estimated error for each time when the model's weights are updated. It is one of the crucial parameters while building a neural network, and also it determines the frequency of cross-checking with model parameters. Selecting the optimized learning rate is a challenging task because if the learning rate is very less, then it may slow down the training process. On the other hand, if the learning rate is too large, then it may not optimize the model properly.

- **Batch Size:** To enhance the speed of the learning process, the training set is divided into different subsets, which are known as a batch. **Number of Epochs:** An epoch can be defined as the complete cycle for training the machine learning model. Epoch represents an iterative learning process. The number of epochs varies from model to model, and various models are created with more than one epoch. To determine the right number of epochs, a validation error is taken into account. The number of epochs is increased until there is a reduction in a validation error. If there is no improvement in reduction error for the consecutive epochs, then it indicates to stop increasing the number of epochs.

Hyperparameter for Specific Models

Hyperparameters that are involved in the structure of the model are known as hyperparameters for specific models. These are given below:

- **A number of Hidden Units:** Hidden units are part of neural networks, which refer to the components comprising the layers of processors between input and output units in a neural network.

It is important to specify the number of hidden units hyperparameter for the neural network. It should be between the size of the input layer and the size of the output layer. More specifically, the number of hidden units should be 2/3 of the size of the input layer, plus the size of the output layer.

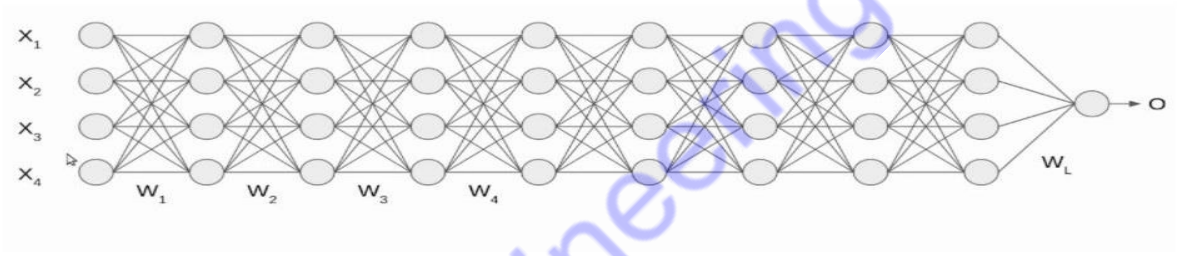
For complex functions, it is necessary to specify the number of hidden units, but it should not overfit the model.

- **Number of Layers:** A neural network is made up of vertically arranged components, which are called layers. There are mainly **input layers, hidden layers, and output layers**. A 3-layered neural network gives a better performance than a 2-layered network. For a Convolutional Neural network, a greater number of layers make a better model.

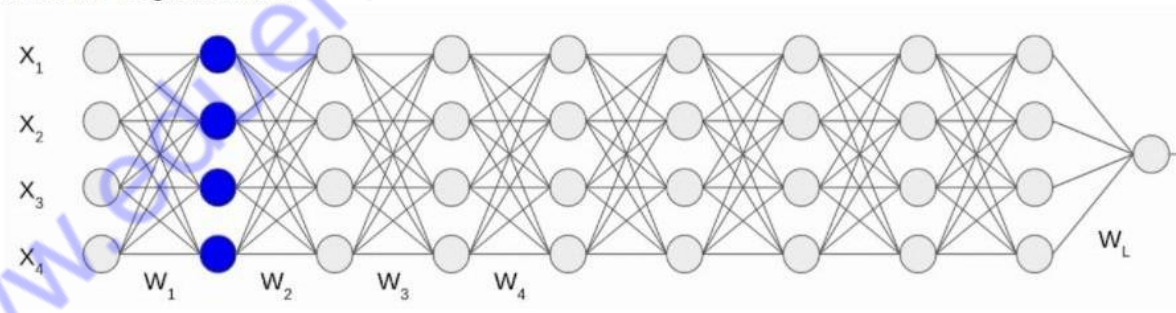
4.8 Batch Normalization:

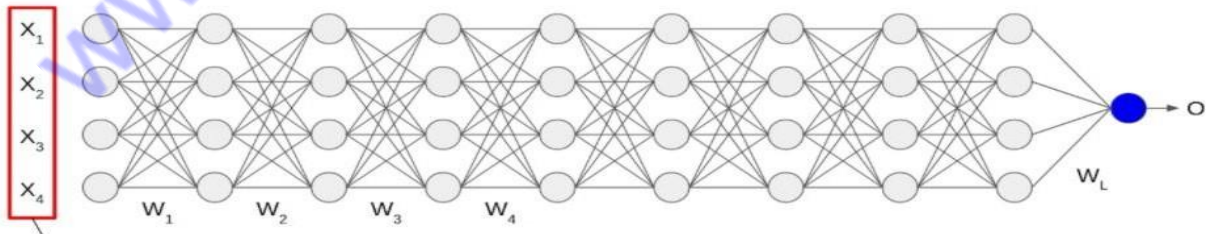
- ❖ It is a method of adaptive reparameterization, motivated by the difficulty of training very deep models.In Deep networks, the weights are updated for each layer.
- ❖ So the output will no longer be on the same scale as the input (even though input is normalized).
- ❖ Normalization - is a data pre-processing tool used to bring the numerical data toa common scale without distorting its shape.
- ❖ when we input the data to a machine or deep learning algorithm we tend to change the values to a balanced scale because, we ensure thatour model can generalize appropriately.(Normalization is used to bring the input into a balanced scale/ Range).

Let's understand this through an example, we have a deep neural network as shown in the following image.



Initially, our inputs X1, X2, X3, X4 are in normalized form as they are coming from the pre-processing stage. the input passes through the first layer, it transforms, as a sigmoid function applied over the dot product of X and the weight matrix W.





Normalize the inputs

$h_1 = \sigma(W_1 X)$

$h_2 = \sigma(W_2 h_1) = \sigma(W_2 \sigma(W_1 X))$

$O = \sigma(W_L h_{L-1})$

Image Source: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/

- Even though the input X was normalized but the output is no longer on the same scale.
- The data passes through multiple layers of network with multiple times(sigmoidal) activation functions are applied, which leads to an internal co-variate shift in the data.
- This motivates us to move towards Batch Normalization
- Normalization is the process of altering the input data to have mean as zero and standard deviationvalue as one.

Procedure to do Batch Normalization:

(1) Consider the batch input from layer h, for this layer we need to calculate the mean of this hiddenactivation.After calculating the mean the next step is to calculate the standard deviation of the hidden activations.

(2) Now we normalize the hidden activations using these Mean & Standard Deviation values. To dothis, we subtract the mean from each input and divide the whole value with the sum of standard deviation and the smoothing term (ε).

(3) As the final stage, the re-scaling and offsetting of the input is performed. Here two components of the BN algorithm is used, γ(gamma) and β (beta). These parameters are used for re-scaling (γ) andshifting(β) the vector contains values from the previous operations.

These two parameters are learnable parameters, Hence during the training of neural network,the optimal values of γ and β are obtained and used. Hence we get the accurate normalization of eachbatch.

4.9 Regularization

Definition: - "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error."

- In the context of deep learning, most regularization strategies are based onregularizing estimators.
- Regularization of an estimator works by trading increased bias for reducedvariance.An effective regularizer is one that makes a profitable trade, reducing variancesignificantly while not overly increasing the bias.
- Many regularization approaches are based on limiting the capacity of models, such as neural networks, linear regression, or logistic regression, by adding a parameter norm penalty Ω(θ) to the objective function J. We denote the regularized objective function by J˜

  $\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$

where α ∈ [0, ∞) is a hyperparameter that weights the relative contribution of the normpenalty term, Ω, relative to the standard objective function J. Setting α to 0 results in no regularization. Larger values of α correspond to more regularization.

The parameter norm penalty Ω that penalizes only the weights of the affine transformation at each layer and leaves the biases unregularized.

L2 Regularization

One of the simplest and most common kind of parameter norm penalty is L2 parameter & it's also called commonly as weight decay. This regularization strategy drives the weights closerto the origin by adding a regularization term

$\Omega(\theta) = \frac{1}{2} ||w||_2^2$.

L2regularization is also known as ridge regression or Tikhonov regularization. To simplify, weassume no bias parameter, so θ is just w. Such a model has the following total objective function.

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2} \boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

with the corresponding parameter gradient

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

To take a single gradient step to update the weights, we perform this update

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon (\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})).$$

Written another way, the update is

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

We can see that the addition of the weight decay term has modified the learning rule to multiplicatively shrink the weight vector by a constant factor on each step, just before performing the usual gradient update. This describes what happens in a single step.The approximation ^J is Given by

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*),$$

Where H is the Hessian matrix of J with respect to w evaluated at w∗.

The minimum of ˆJ occurs where its gradient ∇wˆJ(w) = H(w − w∗) is equal to '0'To study the eff ect of weight decay,

$$\alpha\tilde{\boldsymbol{w}} + \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0$$
$$(\boldsymbol{H} + \alpha\boldsymbol{I})\tilde{\boldsymbol{w}} = \boldsymbol{H}\boldsymbol{w}^*$$
$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^*$$

As α approaches 0, the regularized solution ˜w approaches w*. But what happens as α grows? Because H is real and symmetric, we can decompose it into a diagonal matrix Λ and an orthonormal basis of eigenvectors, Q, such that H = QΛQ^T. Applying Decomposition to theabove equation, We Obtain

$$\begin{aligned}\tilde{\boldsymbol{w}} &= (\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^* \\ &= \left[\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})\boldsymbol{Q}^\top\right]^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^* \\ &= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^*.\end{aligned}$$

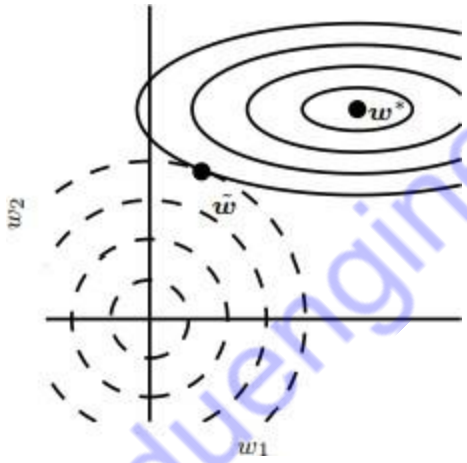


Figure 2: Weight updation effect

The solid ellipses represent contours of equal value of the unregularized objective. The dotted circles represent contours of equal value of the L 2 regularizer. At the point w˜, these competing objectives reach an equilibrium. In the first dimension, the eigenvalue of the Hessian of J is small. The objective function does not increase much when moving horizontally away from w∗ . Because the objective function does not express a strong preference along this direction, the regularizer has astrong effect on this axis. The regularizer pulls w1 close to zero. In the second dimension, the objective function is very sensitive to movements away from w∗ . The corresponding eigenvalue is large, indicating high curvature. As a result, weight decay affects the position of w2 relatively little.

L1 Regularization

While L2 weight decay is the most common form of weight decay, there are other ways to penalize the size of the model parameters. Another option is to use L1 regularization.

- L1 regularization on the model parameter w is defined as the sum of absolute values of theindividual parameters.

$$\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i|,$$

L1 weight decay controls the strength of the regularization by scaling the penalty Ω using a positive hyperparameter α. Thus, the regularized objective function J˜(w; X, y) is given by

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha||\boldsymbol{w}||_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

with the corresponding gradient as

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{w}), \longrightarrow \text{Eq-1}$$

By inspecting equation 1, we can see immediately that the effect of L 1 regularization is quite different from that of L 2 regularization. Specifically, we can see that the regularization contribution to the gradient no longer scales linearly with each wi ; instead it is a constant factorwith a sign equal to sign(wi).

Quadratic approximation of the L 1 regularized objective function decomposes into a sum over the parameters

$$\hat{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y}) + \sum_i \left[\frac{1}{2} H_{i,i}(\boldsymbol{w}_i - \boldsymbol{w}_i^*)^2 + \alpha|w_i|\right].$$

The problem of minimizing this approximate cost function has an analytical solution with the following form:

$$w_i = \text{sign}(w_i^*) \max\left\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\right\}.$$

Consider the situation where w ∗ i > 0 for all i. There are two possible outcomes:

1. The case where $w_i^* \leq \frac{\alpha}{H_{i,i}}$. Here the optimal value of w_i under the regularized objective is simply $w_i = 0$. This occurs because the contribution of $J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ to the regularized objective $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is overwhelmed—in direction i—by the L^1 regularization, which pushes the value of w_i to zero.

2. The case where $w_i^* > \frac{\alpha}{H_{i,i}}$. In this case, the regularization does not move the optimal value of w_i to zero but instead just shifts it in that direction by a distance equal to $\frac{\alpha}{H_{i,i}}$.

Difference between L1 & L2 Parameter Regularization

S.No	L1 Regularization	L2 Regularization
1	Panelizes the sum of absolute value of weights.	penalizes the sum of square weights.
2	It has a sparse solution.	It has a non-sparse solution.
3	It gives multiple solutions.	It has only one solution.
4	Constructed in feature selection.	No feature selection.
5	Robust to outliers.	Not robust to outliers.
6	It generates simple and interpretable models.	It gives more accurate predictions when the output variable is the function of whole input variables.
7	Unable to learn complex data patterns.	Able to learn complex data patterns.
8	Computationally inefficient over non-sparse conditions.	Computationally efficient because of having analytical solutions.

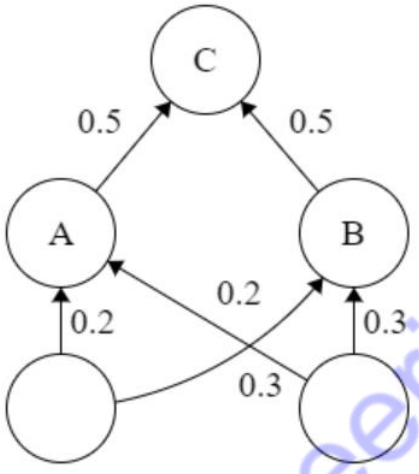
Difference between Normalization and Standardization

Normalization	Standardization
This technique uses minimum and max values for scaling of model.	This technique uses mean and standard deviation for scaling of model.
It is helpful when features are of different scales.	It is helpful when the mean of a variable is set to 0 and the standard deviation is set to 1.
Scales values ranges between [0, 1] or [-1, 1].	Scale values are not restricted to a specific range.
It got affected by outliers.	It is comparatively less affected by outliers.
Scikit-Learn provides a transformer called MinMaxScaler for Normalization.	Scikit-Learn provides a transformer called StandardScaler for Normalization.
It is also called Scaling normalization.	It is known as Z-score normalization.
It is useful when feature distribution is unknown.	It is useful when feature distribution is normal.

4.10 Dropout in Neural Networks

A Neural Network (NN) is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Since such a network is created artificially in machines, we refer to that as Artificial Neural Networks (ANN).

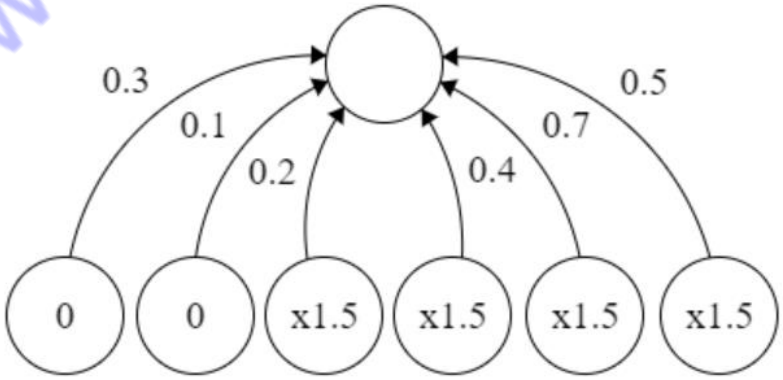
Problem: When a fully-connected layer has a large number of neurons, co-adaptation is more likely to happen. Co-adaptation refers to when multiple neurons in a layer extract the same, or very similar, hidden features from the input data. This can happen when the connection weights for two different neurons are nearly identical.

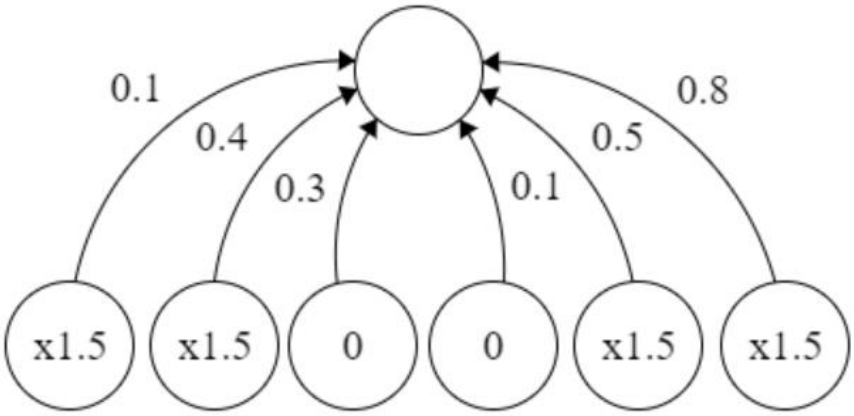


This poses two different problems to our model:

- Wastage of machine's resources when computing the same output.
- If many neurons are extracting the same features, it adds more significance to those features for our model. This leads to overfitting if the duplicate extracted features are specific to only the training set.

Solution to the problem: As the title suggests, we use dropout while training the NN to minimize co-adaptation. In dropout, we randomly shut down some fraction of a layer's neurons at each training step by zeroing out the neuron values. The fraction of neurons to be zeroed out is known as the dropout rate, . The remaining neurons have their values multiplied by so that the overall sum of the neuron values remains the same.





The two images represent dropout applied to a layer of 6 units, shown at multiple training steps. The dropout rate is 1/3, and the remaining 4 neurons at each training step have their value scaled by x1.5. Thereby, we are choosing a random sample of neurons rather than training the whole network at once. This ensures that the co-adaptation is solved and they learn the hidden features better.

Why dropout works?

- By using dropout, in every iteration, you will work on a smaller neural network than the previous one and therefore, it approaches regularization.
- Dropout helps in shrinking the squared norm of the weights and this tends to a reduction in overfitting.

EDU ENGINEERING

PIONEER OF ENGINEERING NOTES

TAMIL NADU'S BEST EDTECH PLATFORM FOR ENGINEERING

CONNECT WITH US

WEBSITE: www.eduengineering.net

TELEGRAM: @eduengineering

INSTAGRAM: @eduengineering

- **Regular Updates for all Semesters**
- **All Department** Notes AVAILABLE
- **Handwritten** Notes AVAILABLE
- **Past Year** Question Papers AVAILABLE
- **Subject wise** Question Banks AVAILABLE
- **Important Questions** for Semesters AVAILABLE
- Various **Author Books** AVAILABLE