PET ENGINEERING COLLEGE

An ISO 9001:2015 Certified Institution

Accredited by NAAC, Approved by AICTE, Recognized by Government of Tamil Nadu and Affiliated to Anna University

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT – V

NEURAL NETWORKS

CLASS : S6 ECE

SUBJECT CODE :CS3491

SUBJECT NAME : ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

REGULATION : 2021

UNIT V

NEURAL NETWORKS

Perceptron- 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, hyperparameter tuning, batch normalization, regularization, dropout.

5.1 Perceptron in Machine Learning

Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, ***Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence***.

Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., **input values, weights and Bias, net sum, and an activation function.**

Basic Components of Perceptron

Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components. These are as follows:
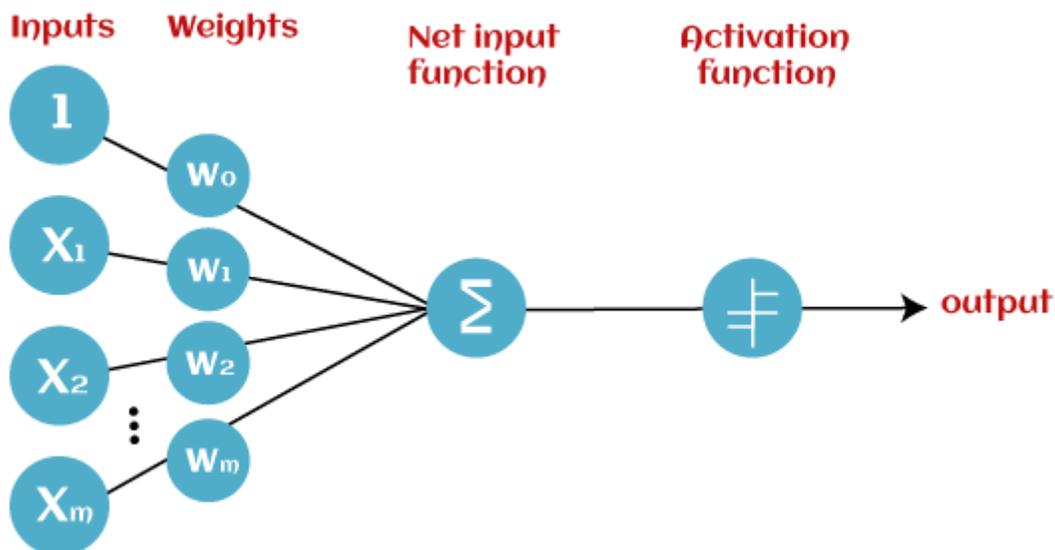

Fig: 5.1

 o **Input Nodes or Input Layer:**

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

 o **Wight and Bias:**

Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

 o **Activation Function:**

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Types of Activation functions:

 o Sign function
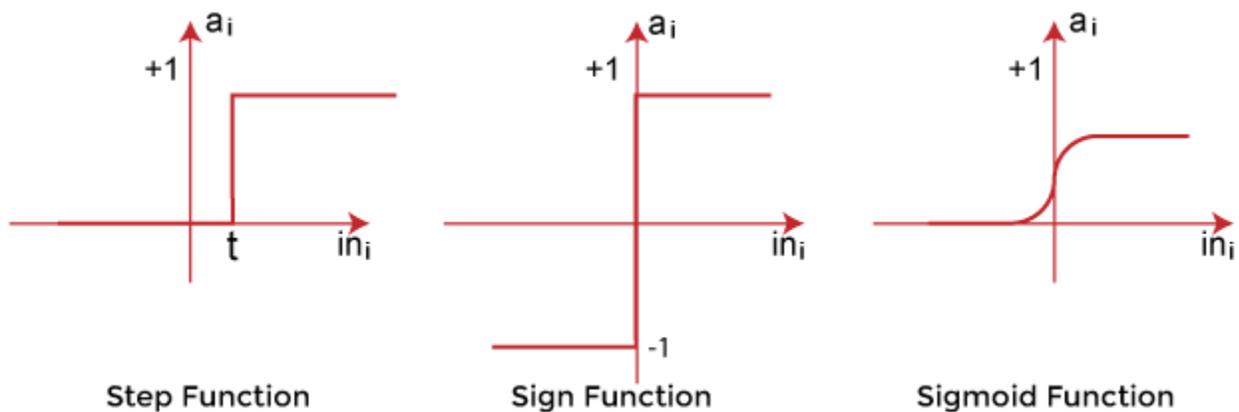
 o Step function, and

 o Sigmoid function

Fig:5.2

The data scientist uses the activation function to take a subjective decision based on various problem statements and forms the desired outputs. Activation function may differ (e.g., Sign, Step, and Sigmoid) in perceptron models by checking whether the learning process is slow or has vanishing or exploding gradients.

How does Perceptron work?

In Machine Learning, Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and Bias, net sum, and an activation function. The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum. Then this weighted sum is applied to the activation function 'f' to obtain the desired output. This activation function is also known as the **step function** and is represented by **'f'**.
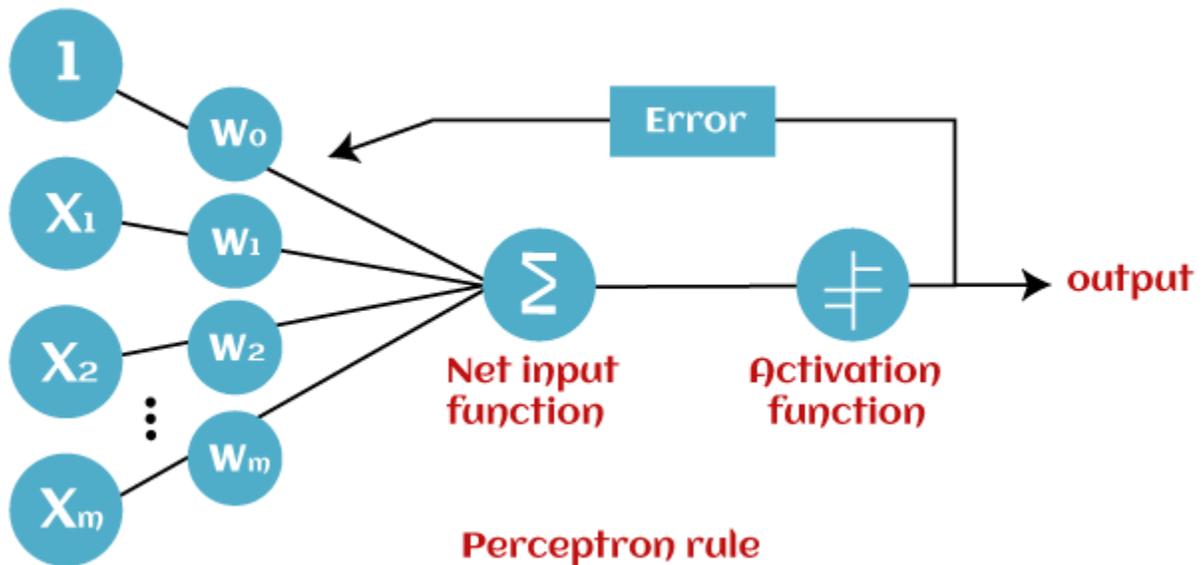

Fig:5.3

This step function or Activation function plays a vital role in ensuring that output is mapped between required values (0,1) or (-1,1). It is important to note that the weight of input is indicative of the strength of a node. Similarly, an input's bias value gives the ability to shift the activation function curve up or down.

Perceptron model works in two important steps as follows:

Step-1

In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

$$\sum wi*xi = x1*w1 + x2*w2 + ...wn*xn$$

Add a special term called **bias 'b'** to this weighted sum to improve the model's performance.

$$\sum wi*xi + b$$

Step-2

In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

$$Y = f(\textstyle\sum wi*xi + b)$$

Types of Perceptron Models

Based on the layers, Perceptron models are divided into two types. These are as follows:

1. Single-layer Perceptron Model
2. Multi-layer Perceptron model

Single Layer Perceptron Model:

This is one of the easiest Artificial neural networks (ANN) types. A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.

In a single layer perceptron model, its algorithms do not contain recorded data, so it begins with inconstantly allocated input for weight parameters. Further, it sums up all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.

If the outcome is same as pre-determined or threshold value, then the performance of this model is stated as satisfied, and weight demand does not change. However, this model consists of a few discrepancies triggered when multiple weight inputs values are fed into the model. Hence, to find desired output and minimize errors, some changes should be necessary for the weights input.

Multi-Layered Perceptron Model:

Like a single-layer perceptron model, a multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.

The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:

- o **Forward Stage:** Activation functions start from the input layer in the forward stage and terminate on the output layer.
- o **Backward Stage:** In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.

5.2 Multilayer Perceptron:

- Multilayer perceptron is one of the most commonly used machine learning method.
- The Multi-layer Perceptron network, consisting of multiple layers of connected neurons.
- Multilayer perceptron is an artificial neural network structure and is a non parametric estimator that can be used for classification and regression.
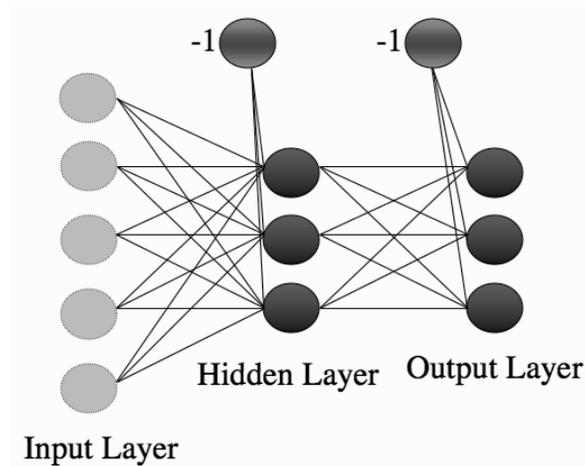

Fig: 5.4 The Multi-layer Perceptron network

- In the multi-layer perceptron diagram above, we can see that there are three inputs and thus three input nodes and the hidden layer has three nodes.
- The output layer gives two outputs, therefore there are two output nodes.
- The nodes in the input layer take input and forward it for further process, in the diagram above the nodes in the input layer forwards their output to each of the three nodes in the hidden layer, and in the same way, the hidden layer processes the information and passes it to the output layer.
- Every node in the multi-layer perception uses a sigmoid activation function. The sigmoid activation function takes real values as input and converts them to numbers between 0 and 1 using the sigmoid formula.
- The most commonly used form of this function (where β is some positive parameter) is:

$$a = g(h) = \frac{1}{1 + \exp(-\beta h)}.$$

- The multi-layer perceptron is also known as back propagation algorithm, which executes in two stages as follows:

i.   Forward stage:

In Figure 5.1, we start at the left by filling in the values for the inputs. We then use these inputs and the first level of weights to calculate the activations of the hidden layer, and then we use those activations and the next set of weights to calculate the activations of the output

layer. Now that we've got the outputs of the network, we can compare them to the targets and compute the error.

ii.   Backward stage: BACK-PROPAGATION OF ERROR

Backpropagation, or backward propagation of errors, is an algorithm that is designed to test for errors working back from output nodes to input nodes. The error function that we used for the Perceptron was

$$\sum_{k=1}^{N} E_k = \sum_{k=1}^{N} y_k - t_k.$$

Where N is the number of output nodes.

5.2.1 The Multi-layer Perceptron Algorithm:

The MLP training algorithm using back-propagation of error is described below:
1. an input vector is put into the input nodes
2. the inputs are fed *forward* through the network
    • the inputs and the first-layer weights (here labelled as v) are used to decide whether the hidden nodes fire or not. The activation function $g(\cdot)$ is the sigmoid function given in

$$a = g(h) = \frac{1}{1 + \exp(-\beta h)}.$$

    • the outputs of these neurons and the second-layer weights (labelled as w) are used to decide if the output neurons fire or not
3. the *error* is computed as the sum-of-squares difference between the network outputs and the targets

$$E(\mathbf{t}, \mathbf{y}) = \frac{1}{2} \sum_{k=1}^{N} (y_k - t_k)^2.$$

4. this error is fed *backwards* through the network in order to
    • first update the second-layer weights
    • and then afterwards, the first-layer weights

The Multi-layer Perceptron Algorithm

- **Initialisation**

    - initialise all weights to small (positive and negative) random values

- **Training**
    - repeat:
        * for each input vector:
        Forwards phase:
        - compute the activation of each neuron j in the hidden layer(s) using:

$$h_\zeta = \sum_{i=0}^{L} x_i v_{i\zeta}$$

$$a_\zeta = g(h_\zeta) = \frac{1}{1 + \exp(-\beta h_\zeta)}$$

        - work through the network until you get to the output layer neurons, which have activations

$$h_\kappa = \sum_{j} a_j w_{j\kappa}$$

$$y_\kappa = g(h_\kappa) = \frac{1}{1 + \exp(-\beta h_\kappa)}$$

        Backwards phase:
        - compute the error at the output using:

$$\delta_o(\kappa) = (y_\kappa - t_\kappa) y_\kappa (1 - y_\kappa)$$

        - compute the error in the hidden layer(s) using:

$$\delta_h(\zeta) = a_\zeta(1 - a_\zeta) \sum_{k=1}^{N} w_\zeta \delta_o(k)$$

        - update the output layer weights using:

$$w_{\zeta\kappa} \leftarrow w_{\zeta\kappa} - \eta \delta_o(\kappa) a_\zeta^{\text{hidden}}$$

        - update the hidden layer weights using:

$$v_i \leftarrow v_i - \eta \delta_h(\kappa) x_i$$

        * (if using sequential updating) randomise the order of the input vectors so that you don't train in exactly the same order each iteration
    - until learning stops
- **Recall**
    - use the Forwards phase in the training section above

Advantages of Multi-layer perceptron:

- It can be used to solve complex nonlinear problems.
- It handles large amounts of input data well.
- Makes quick predictions after training.
- The same accuracy ratio can be achieved even with smaller samples.

Disadvantages of Multi-layer perceptron:

- In Multi-layer perceptron, computations are difficult and time-consuming.

- In multi-layer Perceptron, it is difficult to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of the training.

5.3 Activation Functions:

- Artificial neurons are elementary units in an artificial neural network. The artificial neuron receives one or more inputs and sums them to produce an output. Each input is separately weighted, and the sum is passed through a function known as an activation function or transfer function.
- In an artificial neural network, the function which takes the incoming signals as input and produces the output signal is known as the activation function.
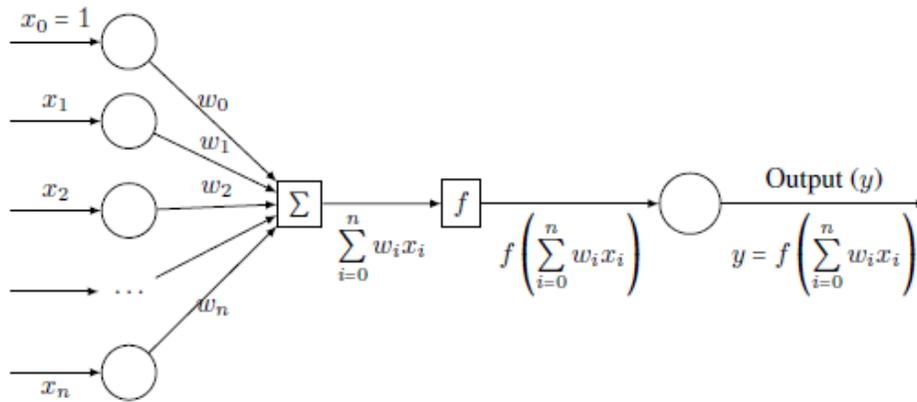

Fig: 5.5 Artificial neuron

x1, x2,......,xn	:	input signals
w1,w2,......,wn	:	weights associated with input signals
x0	:	input signal taking the constant value 1
w0	:	weight associated with x0 (called bias)
Σ	:	indicates summation of input signals
f	:	function which produces the output
y	:	output signal

- The function f can be expressed in the following form:

$$y = f\left(\sum_{i=0}^{n} w_i x_i\right)$$

Some simple activation functions
The following are some of the simple activation functions.

1. **Threshold activation function**
   - The threshold activation function is defined by

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x \leq 0 \end{cases}$$

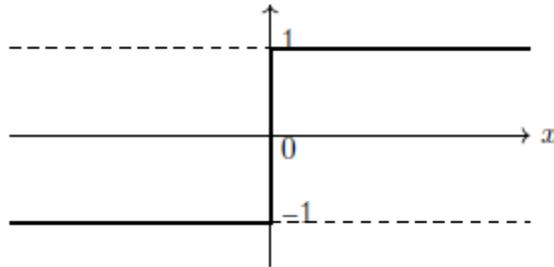   - The graph of this function is shown as follows:

Fig: 5.6 Threshold activation function

2. Unit step functions:
   - Sometimes, the threshold activation function is also defined as a unit step function in which case it is called a unit-step activation function.
   - This is defined as follows:

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

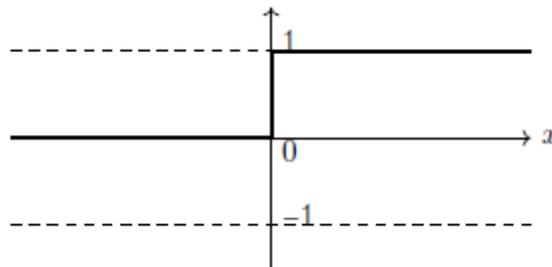   - The graph of this function is shown as follows:

Fig: 5.7 Unit step functions

3. Sigmoid activation function (logistic function):
   - One of the most commonly used activation functions is the sigmoid activation function.

   - It is a function which is plotted as 'S' shaped graph
   - This is defined as follows:

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

- Value Range :- 0to +1
- Nature :- non-linear
- 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.
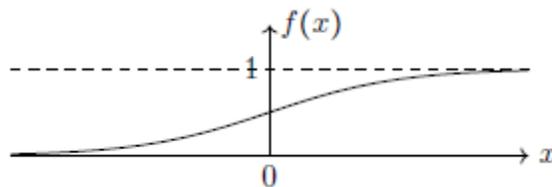- The graph of this function is shown as follows:

Fig: 5.8 Sigmoid activation function

4. Linear activation function
- The linear activation function is defined by
$$F(x) = mx + c$$
- This defines a straight line in the xy-plane.

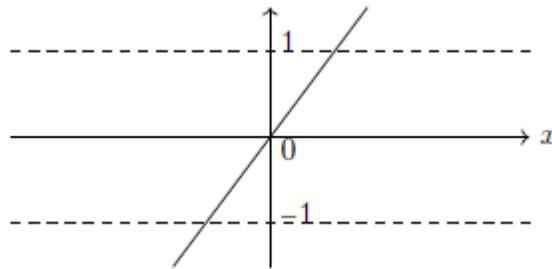
Fig: 5.9 Linear activation function

5. Tanh or Hyperbolic tangential activation function
- 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.
- 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.

- This is defined by

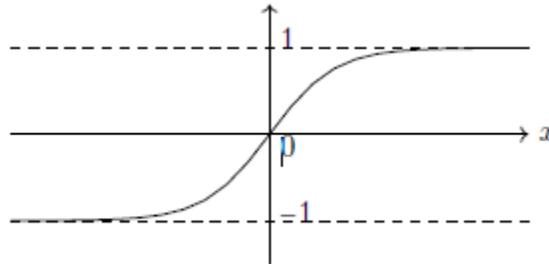$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Fig: 5.10 Hyperbolic tangent activation function

6. RELU Activation 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.
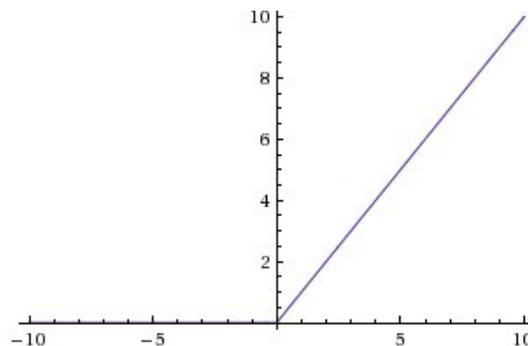

Fig: 5.11 RELU activation Function

5.4 Gradient Descent Optimization:
- Gradient Descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems.
- The general idea is to tweak parameters iteratively in order to minimize the cost function.
- An important parameter of Gradient Descent (GD) is the size of the steps, determined by the learning rate hyperparameters. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time, and if it is too high we may jump the optimal value.

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:

- 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 total number of samples from a dataset that is used for calculating the gradient for each iteration.
- In typical Gradient 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, the problem arises when our dataset 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 are reached. Hence, it becomes computationally very expensive to perform.
- This problem is solved by Stochastic Gradient Descent. In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration.
- The sample is randomly shuffled and selected for performing the iteration.
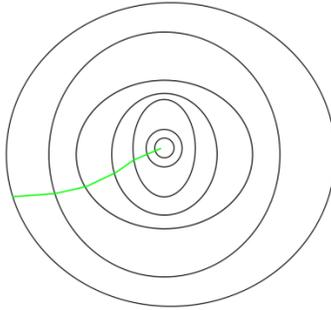
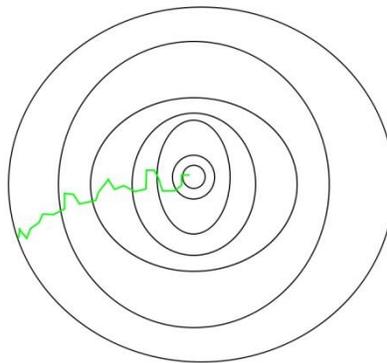Fig: 5.12 the path taken by Batch Gradient Descent

Fig: 5.13 the path taken by Stochastic Gradient Descent

The steps of the algorithm are

1. Find the slope of the objective function **with respect to each parameter/feature**. In other words, compute the gradient of the function.

2. Pick a random initial value for the parameters. (To clarify, in the parabola example, differentiate "y" with respect to "x". If we had more features like x1, x2 etc., we take the partial derivative of "y" with respect to each of the features.)

3. Update the gradient function by plugging in the parameter values.

4. Calculate the step sizes for each feature as : **step size = gradient * learning rate.**

5. Calculate the new parameters as : **new params = old params -step size**

6. Repeat steps 3 to 5 until gradient is almost 0.

5.5 Error backpropagation, from shallow networks to deep networks:

Backpropagation is one of the important concepts of a neural network. Our task is to classify our data best. For this, we have 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.

Backpropagation defines the whole process encompassing both the calculation of the gradient and its need in the stochastic gradient descent. Technically, backpropagation is used to calculate the gradient of the error of the network concerning the network's modifiable weights. The characteristics of Backpropagation are the iterative, recursive and effective approach through which it computes the updated weight to increase the network until it is not able to implement the service for which it is being trained. Derivatives of the activation service to be known at network design time are needed for Backpropagation.

Backpropagation is widely used in neural network training and calculates the loss function for the weights of the network. Its service with a multi-layer neural network and discover the internal description of input-output mapping. It is a standard form of artificial network training, which supports computing gradient loss function concerning all weights in the network. The backpropagation algorithm is used to train a neural network more effectively through a chain rule method. This gradient is used in a simple stochastic gradient descent algorithm to find weights that minimize the error. The error propagates backward from the output nodes to the inner nodes.

The training algorithm of backpropagation involves four stages which are as follows

- **Initialization of weights-** There are some small random values are assigned.
- **Feed-forward** - Each unit X receives an input signal and transmits this signal to each of the hidden unit Z1, Z2, ... Zn. Each hidden unit calculates the activation function and sends its signal Z, to each output unit. The output unit calculates the activation function to form the response of the given input pattern.
- **Backpropagation of errors -** Each output unit compares activation Y, with the target value T, to determine the associated error for that unit. It is based on the error, the factor 88 (K = 1, ... m) is computed and is used to distribute the error at the output unit Y, back to all units in the previous layer. Similarly the factor 88,(j = 1, ... p) is compared for each hidden unit Z.
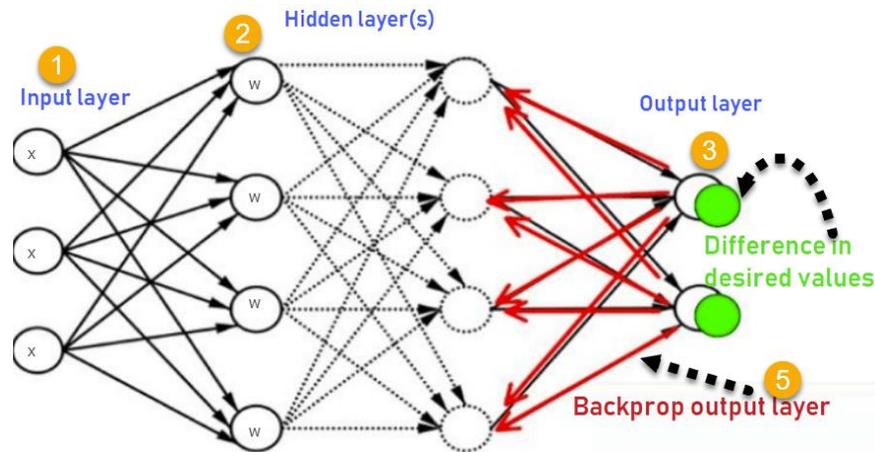- It can update the weights and biases.

Fig: 5.14 Back propagation neural network

Consider the above Back propagation neural network example diagram to understand.

The General Algorithm

The backpropagation algorithm proceeds in the following steps, assuming a suitable learning rate alpha a and random initialization of the parameters w_{ij}^k:wijk:

Definition

1. Calculate the forward phase for each input-output pair $(\vec{x}_d, y_d)$ and store the results $\hat{y}_d$, a_j^k and o_j^k for each node j in layer k by proceeding from layer 0, the input layer, to layer m, the output layer.

2. Calculate the backward phase for each input-output pair $(\vec{x}_d, y_d)$ and store the result $\frac{\partial E_d}{\partial w_{ij}^k}$ for each weight w_{ij}^k, connecting node in layer k − 1 store the results to node j in layer k by proceeding from layer m, the output layer, to layer1, the input layer.

(a) Evaluate the error term for the final layer ∂_l^m by using the second equation.

(b) Backpropogate the error terms for the hidden layers ∂_j^k, working backwards from the final hidden layer k = m-1, by repeatedly using the third equation.

(c) Evaluate the partial derivatives of the individual error E_d, with respect to w_{ij}^k by using the first equation.

3. Combine the individual gradients for each input-output pair $\frac{\partial E_d}{\partial w_{ij}^k}$ to get the total gradient $\frac{\partial E(X,\theta)}{\partial w_{ij}^k}$ for the entire set of input-output pairs $X = \{(\vec{x}_1, y_1),..., (\vec{x}_N, y_N)\}$ by using the fourth equation (a simple average of the individual gradients).

4. Update the weights according to the learning rate α and total gradient $\frac{\partial E(X,\theta)}{\partial w_{ij}^k}$ by using the fifth equation (moving in the direction of the negative gradient).

How Backpropagation Algorithm Works

Inputs X, arrive through the preconnected path

   1. Input is modeled using real weights W. The weights are usually randomly selected.

   2. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.

   3. Calculate the error in the outputs

$$\text{Error B = Actual Output - Desired Output}$$

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

   5. 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.

Types of Backpropagation

   There are two types of Backpropagation which are as follows -

Static Back Propagation - In this type of backpropagation, the static output is created because of the mapping of static input. It is used to resolve static classification problems like optical character recognition.

Recurrent Backpropagation - The Recurrent Propagation is directed forward or directed until a specific determined value or threshold value is acquired. After the certain value, the error is evaluated and propagated backward.

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.

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.

5.6 Unit saturation (aka the vanishing gradient problem)

The vanishing gradient problem is an issue that sometimes arises when training machine learning algorithms through gradient descent. This most often occurs in neural networks that have several neuronal layers such as in a deep learning system, but also occurs in recurrent neural networks.

The key point is that the calculated partial derivatives used to compute the gradient as one goes deeper into the network. Since the gradients control how much the network learns during training, the gradients are very small or zero, then little to no training can take place, leading to poor predictive performance.

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.
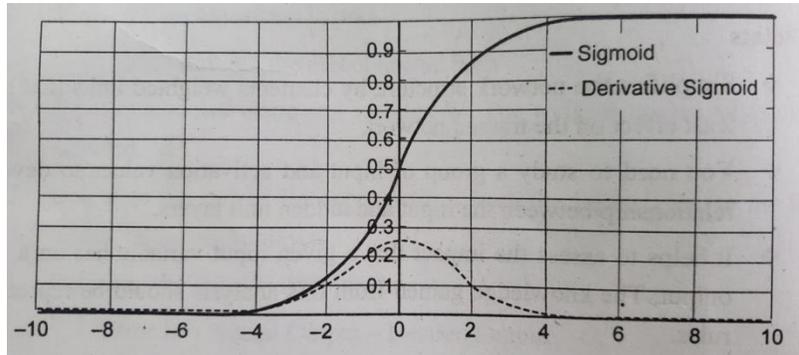

Fig:5.16

The sigmoid function and its derivative

As an example, the above image 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 activation like the sigmoid an 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.

Solution:

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. 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.
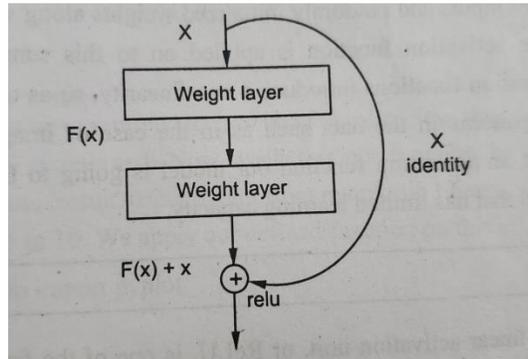

Fig:5.17

What is an activation function?

Activation function is a simple mathematical function that transforms the given input to the required output that has a certain range. From their name they activate the neuron when output reaches the set threshold value of the function. Basically are responsible for switching the neuron ON/OFF. The neuron receives the sum of the product of inputs and randomly initialized weights along with a static bias for each layer. The activation function is applied on to this sum, and an output is generated. Activation functions introduce a non-linearity, so as to make the network learn complex patterns in the data such as in the case of images, text, videos or sounds. Without an activation function our model is going to behave like a linear regression model that has limited learning capacity.

5.7 ReLU

The rectified linear activation unit, or ReLU, is one of the few landmarks in the deep learning revolution. It's simple, yet it's far superior to previous activation functions like sigmoid or tanh.

ReLU formula is: $f(x) = max(0,x)$

Both the ReLU function and its derivative are monotonic. If the function receives any negative input, it returns 0; however, if the function receives any positive value x, it returns that value. As a result, the output has a range of 0 to infinite. ReLU is the most often used activation function in neural networks, especially CNNs, and is utilized as the default activation function.
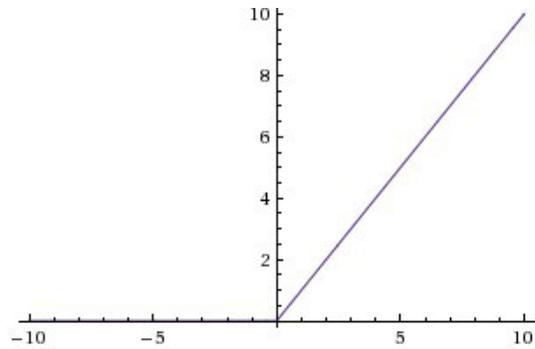
Fig: 5.18 RELU activation Function

Implementing ReLu function in Python

The code for ReLu is as follows :

```
def relu(x):

    return max(0.0, x)
```

To test the function, let's run it on a few inputs.

```
x = 1.0

print('Applying Relu on (%.1f) gives %.1f' % (x, relu(x)))

x = -10.0

print('Applying Relu on (%.1f) gives %.1f' % (x, relu(x)))

x = 0.0

print('Applying Relu on (%.1f) gives %.1f' % (x, relu(x)))

x = 15.0

print('Applying Relu on (%.1f) gives %.1f' % (x, relu(x)))

x = -20.0

print('Applying Relu on (%.1f) gives %.1f' % (x, relu(x)))
```
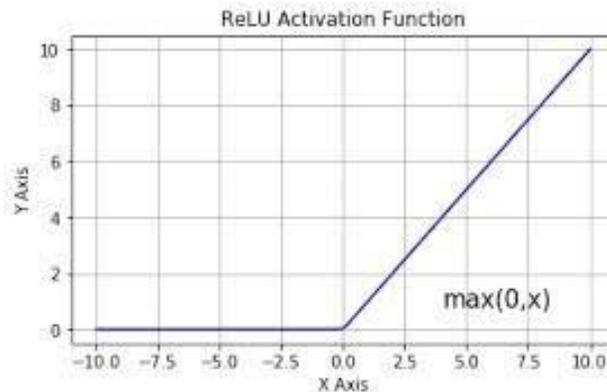
Fig: 5.19

We see from the plot that all the negative values have been set to zero, and the positive values are returned as it is. Note that we've given a set of consecutively increasing numbers as input, so we've a linear output with an increasing slope.

Advantages of ReLU:

ReLU is used in the hidden layers instead of Sigmoid or tanh as using sigmoid or tanh in the hidden layers leads to the infamous problem of "Vanishing Gradient". The "Vanishing Gradient" prevents the earlier layers from learning important information when the network is backpropagating. The sigmoid which is a logistic function is more preferrable to be used in regression or binary classification related problems and that too only in the output layer, as the output of a sigmoid function ranges from 0 to 1. Also and tanh saturate and have lesser sensitivity.

Some of the advantages of ReLU are:

→ **Simpler Computation:** Derivative remains constant i.e 1 for a positive input and thus reduces the time taken for the model to learn and in minimizing the errors.

→ **Representational Sparsity:** It is capable of outputting a true zero value.

→ **Linearity:** Linear activation functions are easier to optimize and allow for a smooth flow. So, it is best suited for supervised tasks on large sets of labelled data.

Disadvantages of ReLU:

- **Exploding Gradient:** This occurs when the gradient gets accumulated, this causes a large differences in the subsequent weight updates. This as a result causes instability when converging to the global minima and causes instability in the learning too.
- **Dying ReLU:** The problem of "dead neurons" occurs when the neuron gets stuck in the negative side and constantly outputs zero. Because gradient of 0 is also 0, it's unlikely for the neuron to ever recover. This happens when the learning rate is too high or negative bias is quite large.

5.8 Hyperparameter tuning

A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters, However, there is another kind of parameter, known as Hyperparameters, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Some examples of model hyperparameters include:

- The penalty in Logistic Regression Classifier i.e. L, or L, regularization
- The learning rate for training a neural network.
- The C and sigma hyperparameters for support vector machines.
- The k in k-nearest neighbors.

Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. The two best strategies for Hyperparameter tuning are:

Grid Search CV

In GridSearchCV approach, the machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for the best set of hyperparameters from a grid of hyperparameters values. For example, if we want to set two hyperparameters C and Alpha of the Logistic Regression Classifier model, with different sets of values. The grid search technique will construct many versions of the model with all possible combinations of hyperparameters and will return the best one.

0.5	0.701	0.703	0.697	0.696
0.4	0.699	0.702	0.698	0.702
0.3	0.721	0.726	0.713	0.703
0.2	0.706	0.705	0.704	0.701
C 0.1	0.698	0.692	0.688	0.675
	0.1	0.2	0.3	0.5

Alpha

Fig: 5.20

As in the image, for C = [0.1, 0.2, 0.3, 0.4, 0.5] and Alpha= [0.1, 0.2, 0.3, 0.4]. For a combination of C = 0.3 and Alpha = 0.2, the performance score comes out to be 0.726(Highest), therefore it is selected.

The following code illustrates how to use GridSearchCV

```
# Necessary imports

from sklearn.linear_model import Logistic Regression

from sklearn.model_selection import GridSearchCV

# Creating the hyperparameter grid

c_space = np.logspace(-5, 8, 15) param_grid = {'C': c_space}

#Instantiating logistic regression classifier

logreg = Logistic Regression()

# Instantiating the GridSearchCV object

logreg_cv= GridSearchCV(logreg, param_grid, cv = 5)

logreg_cv.fit(X,y)

# Print the tuned parameters and score

print("Tuned Logistic Regression Parameters:{}".format(logreg_cv.best_params_))

print("Best score is {}".format(logrcg_cv.best_score_))
```

Output:

Tuned Logistic Regression Parameters: {'C': 3.7275937203149381) Best score is 0.7708333333333334

Drawback:

GridSearch CV will go through all the intermediate combinations of hyperparameters which makes grid search computationally very expensive.

5.9 Batch Normalization

Normalization is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape. -Generally, when we input the data to a machine or deep learning algorithm we tend to change the values to a balanced scale. The reason we normalize is partly to ensure that our model can generalize appropriately. Now coming back to Batch normalization, it is a process to make neural networks faster and more stable through adding extra layers in a deep neural network. The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer. A typical neural network is trained using a collected set

of input data called batch. Similarly, the normalizing process in batch normalization takes place in batches, not as a single input.
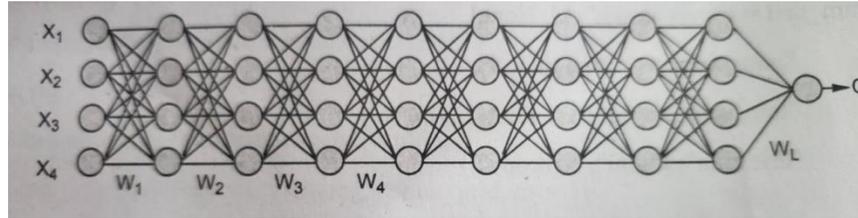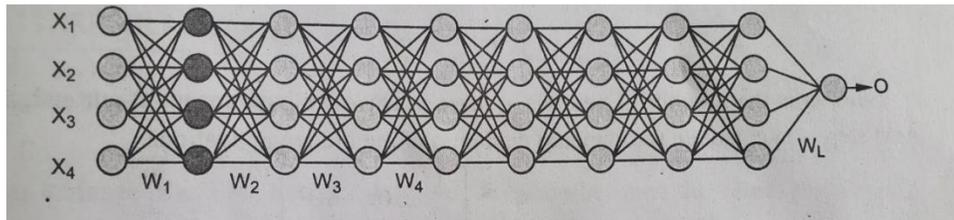

Fig: 5.21

L = Number of layers

Bias = 0

Activation Function = Sigmoid

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

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

Fig: 5.22

Similarly, this transformation will take place for the second layer and go till the last layer L as shown in the following image.

$$h_1 = \sigma(W_1 X)$$
$$h_2 = \sigma(W_2 h_1) = \sigma(W_2 \sigma(W_1 X))$$
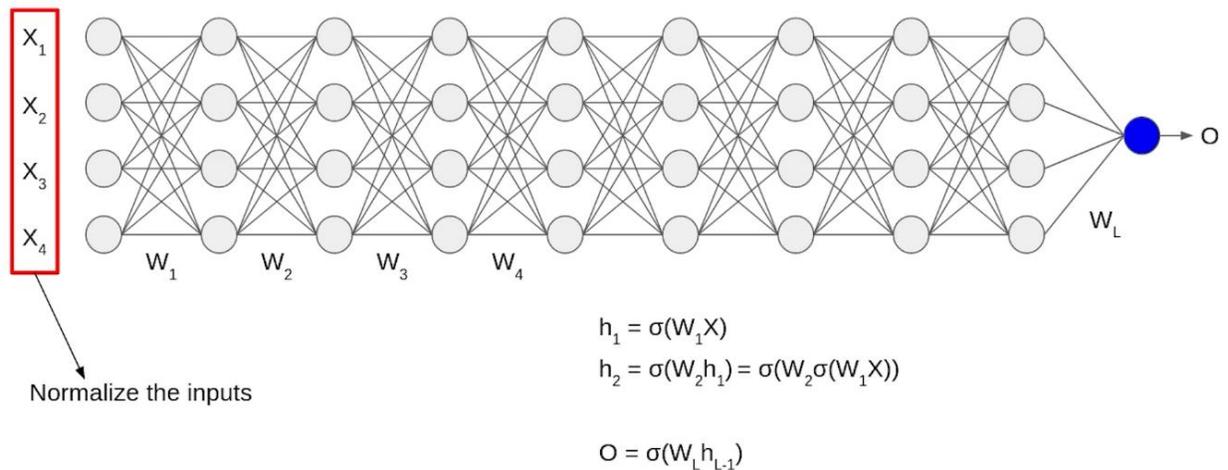
$$O = \sigma(W_L h_{L-1})$$

Fig:5.23

Although, our input X was normalized with time the output will no longer be on the same scale. As the data go through multiple layers of the neural network and L activation functions are applied, it leads to an internal co-variate shift in the data.

How does Batch Normalization work?

Since by now we have a clear idea of why we need Batch normalization, let's understand how it works. It is a two-step process. First, the input is normalized, and later rescaling and offsetting is performed.

Normalization of the Input

Normalization is the process of transforming the data to have a mean zero and standard deviation one. In this step we have our batch input from layer h, first, we need to calculate the mean of this hidden activation.

$$\mu = \frac{1}{m} \Sigma h_i$$

Here, m is the number of neurons at layer h. Once we have meant at our end, the next step is to calculate the standard deviation of the hidden activations.

$$\mu = \sqrt{\frac{1}{m} \Sigma (hi - \mu)2}$$

Further, as we have the mean and the standard deviation ready. We will normalize the hidden activations using these values. For this, we will subtract the mean from each input and divide the whole value with the sum of standard deviation and the smoothing term (ε). The smoothing term(ε) assures numerical stability within the operation by stopping a division by a zero value.

$$h_{i(norm)} = \frac{(h_i - \mu)}{\sigma + \varepsilon}$$

Advantages of Batch Normalization

Now let's look into the advantages the BN process offers.

Speed Up the Training

By Normalizing the hidden layer activation the Batch normalization speeds up the training process.

Handles internal covariate shift

It solves the problem of internal covariate shift. Through this, we ensure that the input for every layer is distributed around the same mean and standard deviation. If you are unaware of what is an internal covariate shift, look at the following example.

Internal covariate shift

Suppose we are training an image classification model, that classifies the images into Dog or Not Dog. Let's say we have the images of white dogs only, these images will have certain distribution as well. Using these images model will update its parameters.

Smoothens the Loss Function

Batch normalization smoothens the loss function that in turn by optimizing the model parameters improves the training speed of the model.

5.10   Regularization

The Problem of Overfitting

So, before diving into regularization, let's take a step back to understand what bias-variance is and its impact. Bias is the deviation between the values predicted by the model and the actual values whereas, variance is the difference between the predictions when the model fits different datasets.

When a model performs well on the training data and does not perform well on the testing data, then the model is said to have high generalization error. In other words, in such a scenario, the model has low bias and high variance and is too complex. This is called overfitting. Overfitting means that the model is a good fit on the train data compared to the data, as illustrated in the graph above. Overfitting is also a result of the model being too complex.

What Is Regularization in Machine Learning?

Regularization is one of the key concepts in Machine learning as it helps choose a simple model rather than a complex one. We want our model to perform well both on the train and the new unseen data, meaning the model must have the ability to be generalized. Generalization error is "a

measure of how accurately an algorithm can predict outcome values for previously unseen data." Regularization refers to the modifications that can be made to a leaming algorithm that helps to reduce this generalization error and not the training error. It reduces by ignoring the less important features. It also helps prevent overfitting, making the model more robust and decreasing the complexity of a model.

How Does Regularization Work?

Regularization works by shrinking the beta coefficients of a regression model. To understand why we need to shrink the coefficients, let us see the below example:
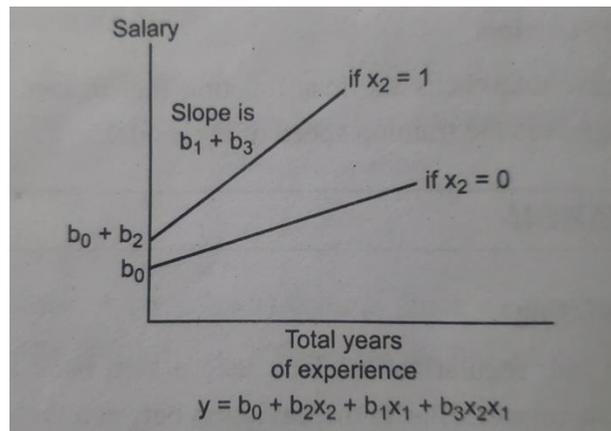

Fig: 5.24

In the above graph, the two lines represent the relationship between total years of experience and salary, where salary is the target variable. These are slopes indicating the change in salary per unit change in total years of experience. As the slope b_1+b_3 decreases to slope b_1 , we see that the salary is less sensitive to the total years of experience. By decreasing the slope, the target variable (salary) became less sensitive to the change in the independent X variables, which increases the bias into the model. Remember, bias is the difference between the predicted and the actual values.

With the increase in bias to the model, the variance (which is the difference between the predictions when the model fits different datasets.) decreases. And, by decreasing the variance, the overfitting gets reduced. The models having the higher variance leads to overfitting, and we saw above, we will shrink or reduce the beta coefficients to overcome the overfitting. The beta coefficients or the weights of the features converge towards zero, which is known as shrinkage.

What Is the Regularization Parameter?

For linear regression, the regularization has two terms in the loss function:

The Ordinary Least Squares (OLS) function, and

The penalty term

It becomes :

Loss function _{regularization} **=Loss function** _{ols}**+ Penalty term**

The goal of the linear regression model is to minimize the loss function. Now for Regularization, the goal becomes to minimize the following cost function:

$$\sum_{i=1}^{n} (\mathbf{y_{act}-\ y_{pred}})\mathbf{2} + \mathbf{penalty}$$

Where, the penalty term comprises the regularization parameter and the weights associated with the variables. Hence, the penalty term is:

$$penalty = \ \lambda * w$$

where,

λ = Regularization parameter

w = weight associated with the variables; generally considered to be L-p norms

The regularization parameter in machine learning is λ: It imposes a higher penalty on the variable having higher values, and hence, it controls the strength of the penalty term. This tuning parameter controls the bias-variance trade-off.

λ can take values 0 to infinity. If λ = 0, then means there is no difference between a model with and without regularization.

Regularization Techniques in Machine Learning

Each of the following techniques uses different regularization norms (L-p) based on the mathematical methodology that creates different kinds of regularization. These methodologies have different effects on the beta coefficients of the features. The regularization techniques in machine learning as follows:

(a) Ridge Regression

The Ridge regression technique is used to analyze the model where the variables may be having multicollinearity. It reduces the insignificant independent variables though it does not remove them completely. This type of regularization uses the L_2 norm for regularization

$$cost\ function = \sum_{i=1}^{n} (\mathbf{y_{act}-\ y_{pred}})\mathbf{2} + \lambda \|\mathbf{w}\|_2^2$$

(b) Lasso Regression

Least Absolute Shrinkage and Selection Operator (or LASSO) Regression penalizes the coefficients to the extent that it becomes zero. It eliminates the insignificant independent variables. This regularization technique uses the L1 norm for regularization.

$$cost\ function = \sum_{i=1}^{n}(y_{act} - y_{pred})2 + \lambda\|w\|^2$$

(c) Elastic Net Regression

The Elastic Net Regression technique is a combination of the Ridge and Lasso regression technique. It is the linear combination of penalties for both the L, -norm and L_2 -norm regularization.

The model using elastic net regression allows the learning of the sparse model where some of the points are zero, similar to Lasso regularization, and yet maintains the Ridge regression properties. Therefore, the model is trained on both the L, and L_2 norms.

The cost function of Elastic Net Regression is:

$$cost\ function = \sum_{i=1}^{n}(y_{act} - y_{pred})2 + \lambda_{ridge}\|w\|_2^2 + \lambda_{lasso}\|w\|_2^2$$

When to Use Which Regularization Technique?

The regularization in machine learning is used in following scenarios:

- Ridge regression is used when it is important to consider all the independent variables in the model or when many interactions are present. That is where collinearity or codependency is present amongst the variables.
- Lasso regression is applied when there are many predictors available and would want the model to make feature selection as well for us.

When many variables are present, and we can't determine whether to use Ridge or Lasso regression, then the Elastic-Net regression is your safe bet.

5.11 DROUPOUT:

"Dropout" in machine learning refers to the process of randomly ignoring certain nodes in a layer during training. In the figure below, the neural network on the left represents a typical neural network where all units are activated. On the right, the red units have been dropped out of the model- the values of their weights and biases are not considered during training.
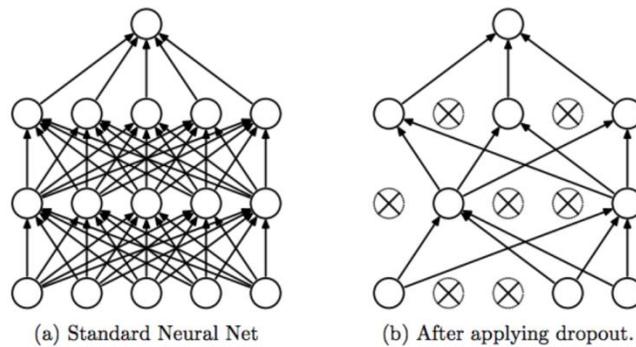
(a) Standard Neural Net   (b) After applying dropout.

Fig:5.25

Dropout is used as a regularization technique - it prevents overfitting by ensuring that no units are codependent.

Common Regularization Methods

Common regularization techniques include:

Early stopping: stop training automatically when a specific performance measure (eg. Validation loss, accuracy) stops improving

Weight decay: incentivize the network to use smaller weights by adding a penalty to the loss function (this ensures that the norms of the weights are relatively evenly distributed amongst all the weights in the networks, which prevents just a few weights from heavily influencing network output)

Noise: allow some random fluctuations in the data through augmentation (which makes the network robust to a larger distribution of inputs and hence improves generalization)

Model combination: average the outputs of separately trained neural networks (requires a lot of computational power, data, and time)

Dropout remains an extremely popular protective measure against overfitting because of its efficiency and effectiveness.

How Does Dropout Work?

When we apply dropout to a neural network, we're creating a "thinned" network with unique combinations of the units in the hidden layers being dropped randomly at different points in time during training. Each time the gradient of our model is updated, we generate a new thinned neural network with different units dropped based on a probability hyperparameter p. Training a network using dropout can thus be viewed as training loads of different thinned neural networks and merging them into one network that picks up the key properties of each thinned network. This process allows dropout to reduce the overfitting of models on training data.

This graph, taken from the paper "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" by Srivastava et al., compares the change in classification error of models without dropout to the same models with dropout (keeping all other hyperparameters constant). All the models have been trained on the MNIST dataset.
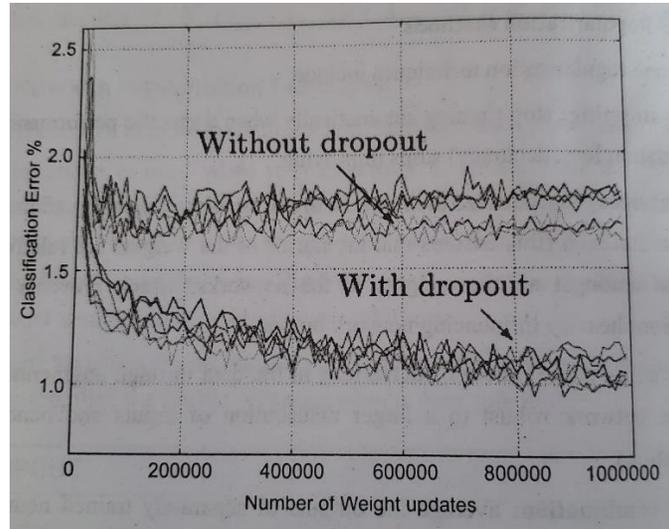

Fig:5.26

It is observed that the models with dropout had a lower classification error than the same models without dropout at any given point in time. A similar trend was observed when the models were used to train other datasets in vision, as well as speech recognition and text analysis. The lower error is because dropout helps prevent overfitting on the training data by reducing the reliance of each unit in the hidden layer on other units in the hidden layers.

The Downside of Dropout

Although dropout is clearly a highly effective tool, it comes with certain drawbacks. A network with dropout can take 2-3 times longer to train than a standard network. One way to attain the benefits of dropout without slowing down training is by finding a regularizer that is essentially equivalent to a dropout layer. For linear regression, this regularizer has been proven to be a modified form of L2 regularization.

<div align="center">

TWO MARKS (PART A)

</div>

1. **What is perceptron model in machine learning?**

Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence. Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks.

2. List down the advantages of multilayer perceptron?

- It can be used to solve complex nonlinear problems.
- It handles large amounts of input data well.
- Makes quick predictions after training.
- The same accuracy ratio can be achieved even with smaller samples.

3. Define activation function.

In an artificial neural network, the function which takes the incoming signals as input and produces the output signal is known as the activation function.

4. How do you train a neural model?

In the process of training, we want to strat with a bad performing neural network and wind up with network with high accuracy, In terms of loss function, we want our loss function to much lower in the end of training. Improving the network is possible, because we can change its function by adjusting weights. We want to find another function that performs better than the initial one.

5. List the types of gradient descent?

1.Batch Gradient Descent

2.Stochastic Gradient Descent

3.Mini-batch Gradient Descent

6. Define SGD.

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 total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset.

7. List the types of Backpropagation

Static Back Propagation - In this type of backpropagation, the static output is created because of the mapping of static input. It is used to resolve static classification problems like optical character recognition.

Recurrent Backpropagation - The Recurrent Propagation is directed forward or directed until a specific determined value or threshold value is acquired. After the certain value, the error is evaluated and propagated backward.

8. What is the downside of dropout?

Although dropout is clearly a highly effective tool, it comes with certain drawbacks. A network with dropout can take 2-3 times longer to train than a standard network. One way to attain the benefits of dropout without slowing down training is by finding a regularizer that is essentially equivalent to a dropout layer. For linear regression, this regularizer has been proven to be a modified form of L2 regularization.

9. Define ReLU

The rectified linear activation unit, or ReLU, is one of the few landmarks in the deep learning revolution. It's simple, yet it's far superior to previous activation functions like sigmoid or tanh.

$$\textbf{ReLU formula is: } f(x) = max(0,x)$$

Both the ReLU function and its derivative are monotonic.

10. What is the objective of gradient descent

Gradient Descent is an optimizing algorithm used in Machine/ Deep Learning algorithms. The goal of Gradient Descent is to minimize the slope to reach the lowest point of the surface.

PART B & C

1. **Explain about the multilayer perceptron model.**
2. **Elaborate the activation function in detail.**
3. **How does dropout works? Discuss**
4. **Explain backpropagation in detail.**
5. **Classify how to train a neural network.**