UNIT-II

Fundamentals of Deep Learning

Anatomy of Neural Networks :

Deep learning is in fact a new name for an approach to artificial intelligence called neural networks, Neural Networks were first proposed in 1944 by Warren Mccullough and Walter Pitts.

What is Neural Network

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain process information.

⇒ It consists of large no. of highly interconnected neurons in it to carry information

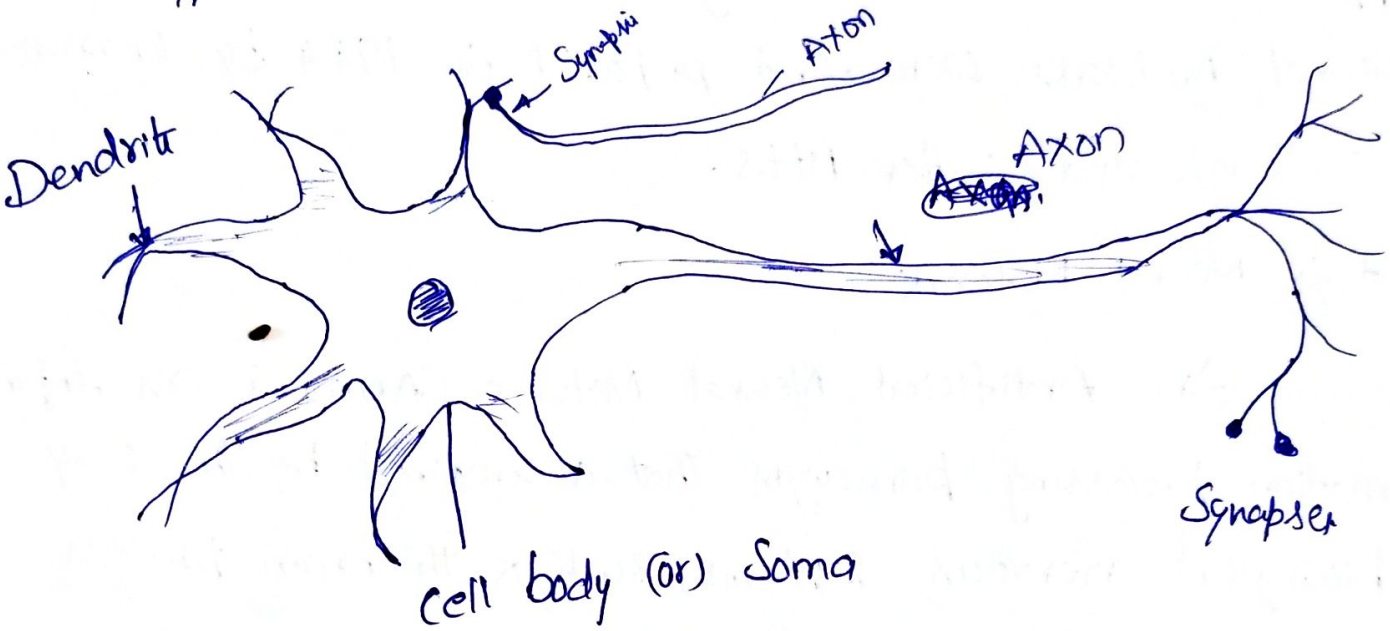
Example : Pattern recognition (or) data classification, Through a learning process.

Why use Neural Network

* Neural Networks, with their remarkable ability to derive and detect trends that are too complex to be noticed by either humans or other computer techniques.
* A trained neural network can be thought of an "expert" in the category of information it has been given to analyse.

How do our brain work.

* The Brain is a massively parallel - information processing system
* Our Brains are high network of preprocessing elements. A typical brain Contains a network of 10 billion neurons.



A Processing element

Dendrites :- Input
Cellbody :- Processor.
Synaptic :- Link.
Axon :- Output.

* A neuron is Connected to other neuron Through about 10,000 Synapses.
* A neuron receive input from other neurons, Inputs are Combined.
* Once input exceeds a critical level, the neuron discharg

a Spike an electrical pulse that travels from the body, down the axon, to the next neuron(s).

Anatomy of neural Networks

* Layers
* Neurons (Nodes)
* Weights and Biases
* Activation Function
* Loss Function
* Optimization Algorithm
* Learning Rate

Layers

- Input layer
- Hidden layers
- Output layer.

⇒ The First layer of the neural network, where the input data is fed. Each node in this layer represents a feature of the input data.

* Hidden layers between the input and output layers. Each node in a hidden layer represents a learned feature or Combination of features.

Deep neural networks have multiple hidden layers, giving rise to the term "Deep learning"

=> Output layers is the final layer that produces the network's output.

The no. of nodes in this layer depends on the type of problem. For instance, a binary classification problem would have one output node, while a multi class classification problem might have multiple nodes

Neurons (Nodes)

* Fundamental units that receive input, perform computation and produce an output.
* Each connection to a node has an associated weight that is adjusted during training to learn patterns in the data.
* The output of a node is often passed through an activation function to introduce non-linearity into the network, enabling it to learn complex relationships.

Weights and Biases :

* Weights represents the strength of connections between neurons.
* Bias are additional parameters that are added to the weighted sum of inputs before applying the activation function.

* These parameters are adjusted during the training process using optimization algorithms like gradient descent to minimize the error (or) loss.

Activation Function :-

* Introduces non-linearity to the model, enabling it to learn complex relationships in the data.
* Common activation functions include Sigmoid, hyperbolic tangent (tanh) and rectified linear unit (ReLU)

Loss Function :-

* Measures the difference between the predicted output and actual target values.
* The goal during training is to minimize this loss function

Optimization Algorithm :-

* Determine how the weights and biases are updated during training to minimize the loss function.
* Common optimization algorithms includes gradient descent and its variants (Adam, Gradient descent etc).

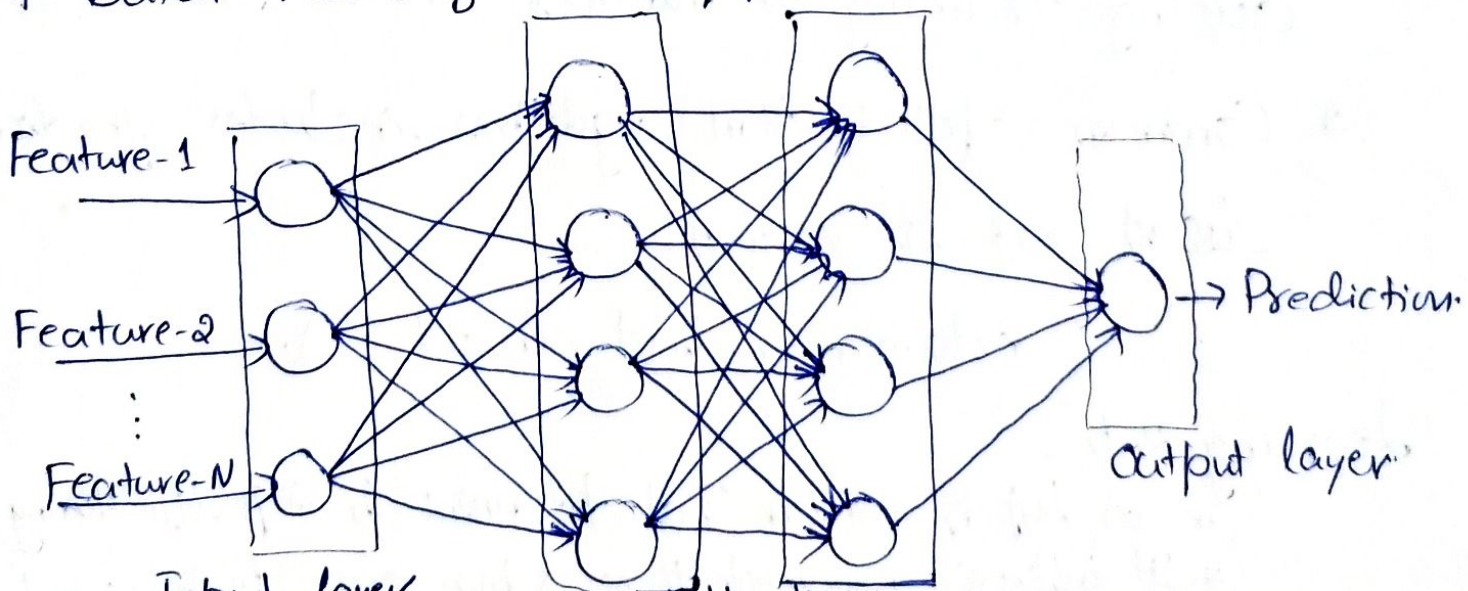
Learning Rate :-

* A hyperparameter that determine the step size during opti
* It influences how much the wt & bias are adjusted in each iter

Layers in Neural Network

Neural Network consist of various types of layers, each serving a specific purpose in information processing. The architecture of a neural network is defined by the arrangement and connectivity of these layers.

1. Input layer
2. Hidden layers
   - → Fully Connected (Dense) layer
   - → Convolutional layer (CNN)
   - → Recurrent layer (RNN)
3. Output layer
4. Activation layers.
5. Pooling layers.
6. Dropout layers.
7. Batch Normalization layer.



1. Input layer :-

* The first layer in the neural network
* It represents the features of the input data.
* Each node in this layer corresponds to a feature in the input.

2. Hidden layers

* Layers between the input and output layers.
* Nodes in These layers perform Computations based on the input data.
* Multiple hidden layers enable the network to learn hierarchical and Complex representations of the data.

Different types of hidden layers :-

⇒ Fully Connected (Dense) layer
⇒ Convolutional Layer
⇒ Recurrent Layer.

Fully Connected (Dense) layer:-
Each node in the layer is connected to every node in the previous and next layers.

Convolutional layer :- It is used for processing grid-structured data like images. It applies Convolution operations to capture spatial hierarchies.

Recurrent layer :-

A suitable for sequential data. It maintains hidden states that capture information from previous steps in the sequence.

3. output layer :

→ The final layer that produces the network's output

→ The no. of nodes in this layer depends on the nature of the problem (Eg:- binary classification, multi-class classification, regression).

→ The activation function used in the output layer is chosen based on the task :

* Sigmoid – Commonly used for binary classification problem

* Softmax – used for multi-class classification problem.

* Linear – used for regression tasks.

4. Activation Layers

Introduce non-linearity to the network, allowing it to learn complex patterns.

⇒ Commonly activation functions include :-

* Sigmoid :- Map values to the range (0,1)
* Hyperbolic Tangent (tanh) :- Maps values to the range (-1, 1)
* Rectified Linear Unit (ReLU) :- Sets negative values to zero, allowing only positive values to pass through

5. Pooling Layers (in CNN)

* Used to reduce spatial dimensions of the input data in Convolutional neural networks.
* Max pooling and average pooling are common techniques for selecting the maximum or average value from a group of values in certain region.

6. Dropout layer

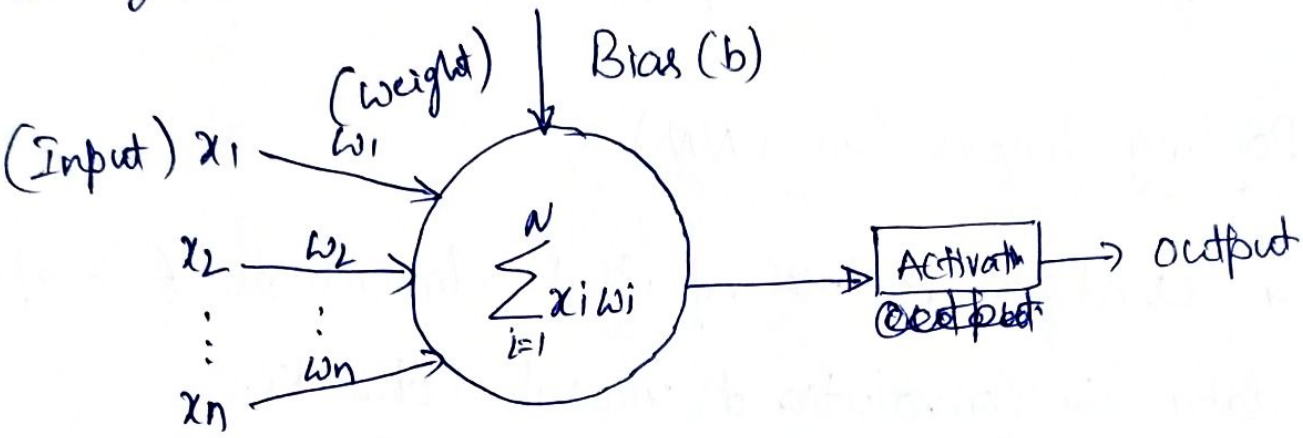
* It Helps prevent overfitting by randomly setting a fraction of input units to zero during training.
* It introduces a form of regularization, improving the model's generalization.

7. Batch Normalization Layer

* Normalizes the inputs of a layer to have zero mean & unit vari
* Helps with training stability and accelerates convergence.

Neuron

Neurons in a neural network are the basic computational units that process and transmit information. They are inspired by the structure and functioning of biological neurons in the human brain.



The process of information flow through a single neuron can be summarized as

$$\text{output} = \text{Activation}\left(\sum_{i=1}^{n} (\text{Input} \times \text{weight}) + \text{Bias}\right)$$

weights

⇒ Each input to a neuron is associated with a weight. Weights represent the strength of the connection between neurons.

⇒ During training, these weights are adjusted to learn patterns and relationships in the input data.

Input : Neuron receive input from other neurons or from the external environment. In the case of I/p layer, neurons directly receives the features of I/P Data.

Bias:

⇒ A bias term is added to the weighted sum before passing through an activation function.

⇒ Bias helps the neural network account for situations where all input values are zero (or) when the neuron needs to output a non-zero value even if all inputs are zero.

Activation function

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.

why do we need non-linearity:

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 task.

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

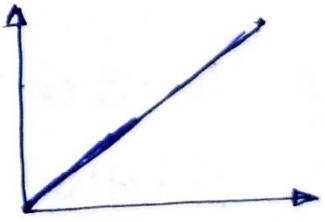
Variants of Activation Function / Types of Activation Function

1. Linear
2. Sigmoid
3. Hyperbolic Tangent
4. Rectified Linear Unit (ReLU)
5. Softmax Activation Function.

6.
- Leaky ReLU Activation Function (LReLU)
- Parametric ReLU Activation Function (PReLU)
- Exponential Linear Unit (ELU) Activation Function

Linear Function:

* Linear function has the Equation similar to as of a straight line $y = ax$
* 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 from $-\infty$ to $+\infty$



* It is used at just one place i.e output layer.

for Example :- Calculation of price 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

2. Sigmoid Function

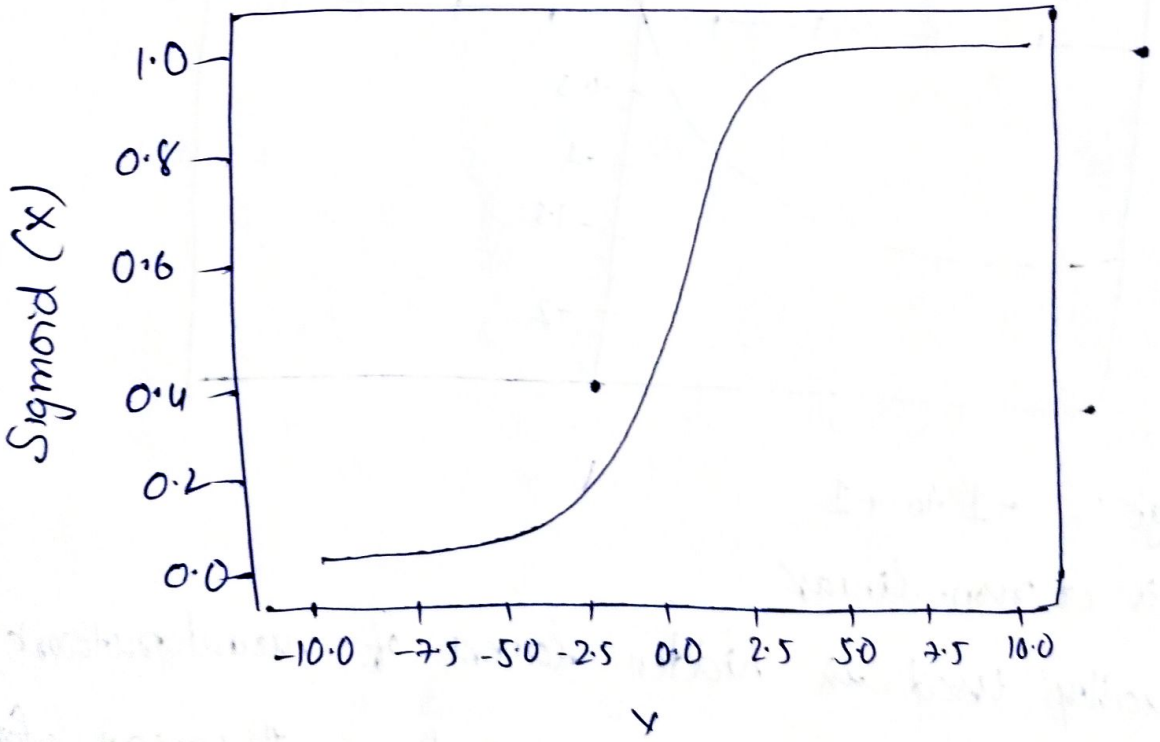
* It is a function which is plotted as 'S' shaped graph.

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

* Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x could also bring about large changes in the value of Y.

* Value Range : 0 to 1.

* 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 predicted easily to be 1 if value is greater than 0.5 and 0 otherwise

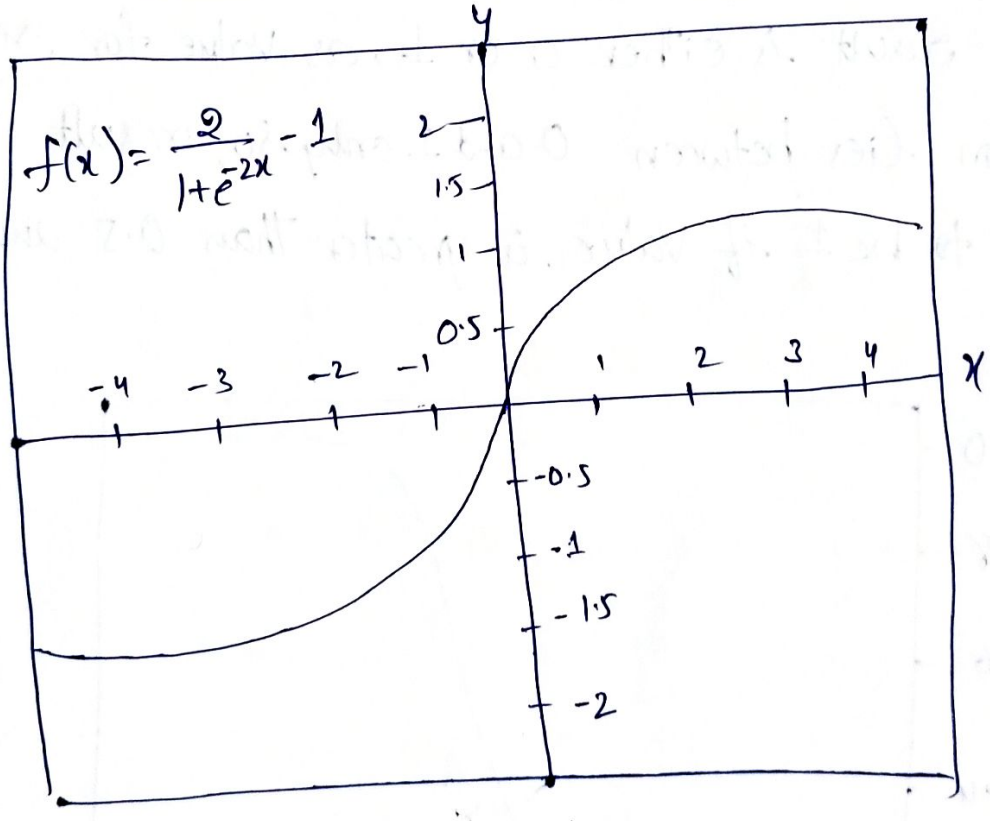


* To predict probabilities, e.g logistic regression.

3. Hyperbolic Tangent (tanh) Activation function

* The activation that works almost always better than Sigmoid function is Tanh function also known as Hyperbolic tangent function.
* It is actually mathematically shifted version of the Sigmoid function. Both are similar and can be derived from each other.

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1.$$ (or) $2 * \text{Sigmoid}(2x) - 1$



* Range: -1 to +1
* It is a non-linear
* Usually used in hidden layers of 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

4) RELU function

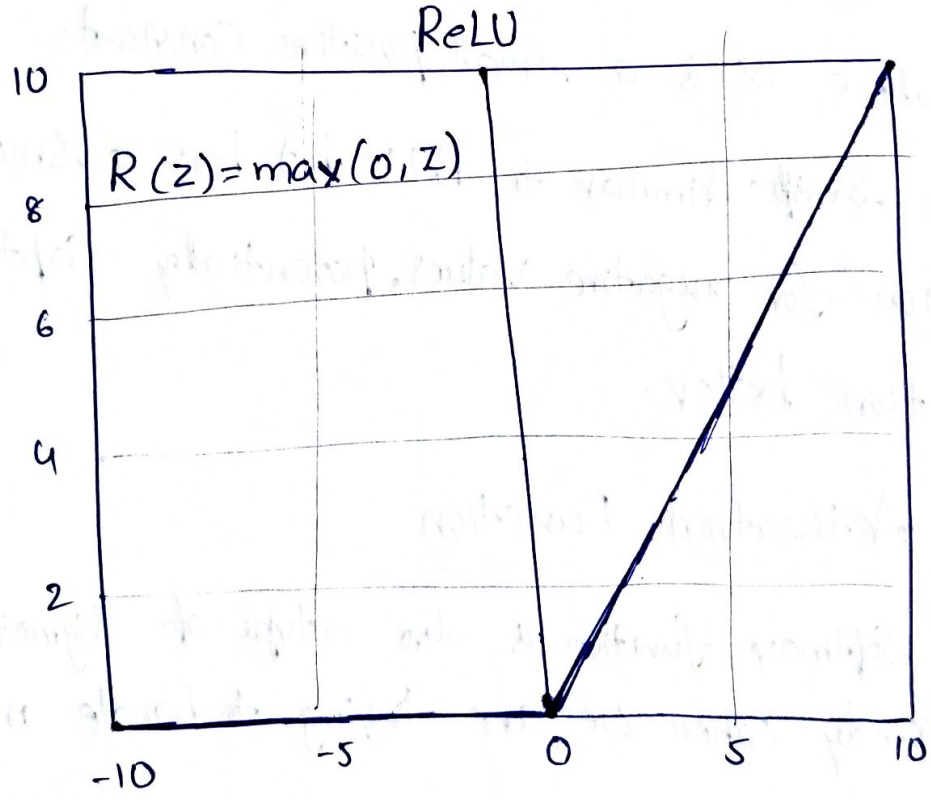
* It Stands for Rectified linear unit, it is the most widely used activation function. chiefly implemented in hidden layers of Neural network.

* ~~A~~($f(x) = \max(0, x)$ it gives an output x if x is positive & 0 otherwise

range $[0, \infty)$

* Non-linear, which means we can easily backpropa-gate the errors and have multiple layers of neurons being activated by the ReLU function.

*



[It is much faster than tanh & Sigmoid].

* ReLU is less Computationally expensive than tanh & 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.

⇒ LReLU (Leaky ReLU Activation Function) :-

* Formula : $f(x) = \max(\alpha x, x)$ where α is small positive Constant
* Range :- $(-\infty, \infty)$
* Example :- Similar to ReLU but allows a small negative slope for the negative input Values, preventing dead neurons

⇒ Parametric ReLU (PReLU) Activation Function) :-

Similar to leaky ReLU but allows the slope to be learned during training rather than being a fixed parameters

⇒ Exponential Linear Unit (ELU) Activation Function)

* $f(x) = x$ for $x > 0$ and $f(x) = \alpha(e^x - 1)$ for $x \leq 0$ where α is a small positive Constant.

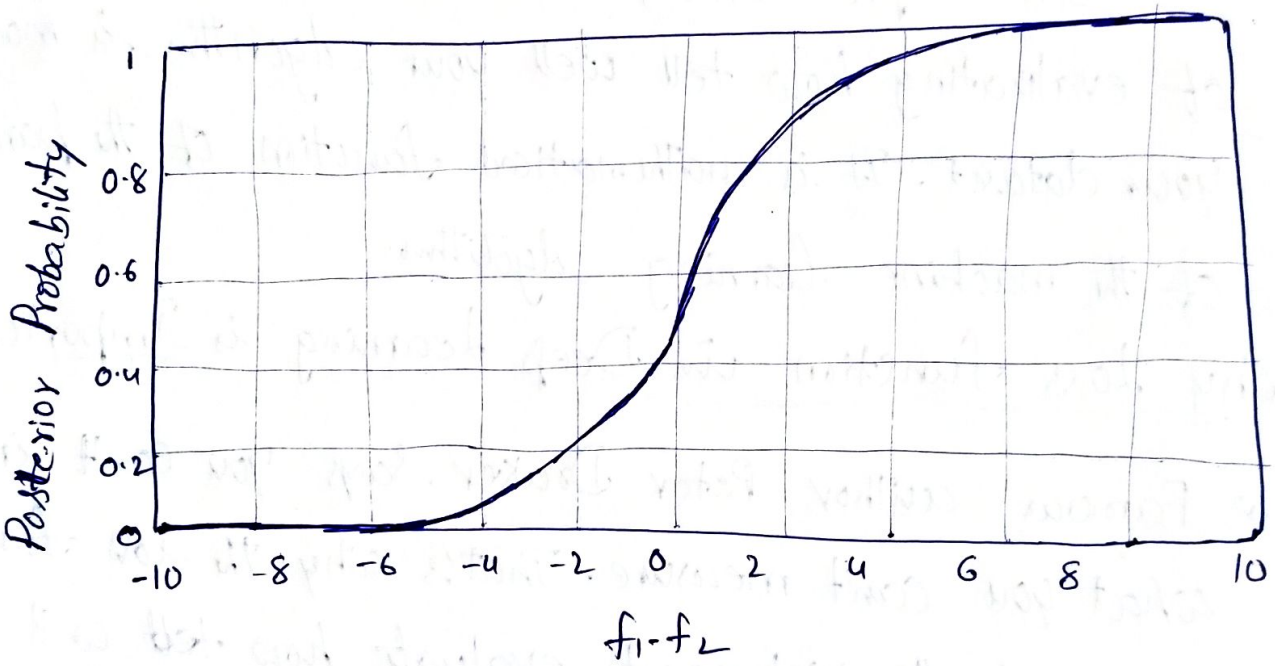
Example :- Similar to ReLU but has a smoother transition for negative Values, potentially capturing information better.

5. Softmax Activation Function

* The Softmax function is also a type of Sigmoid function but is handy when we are trying to handle multi-class classification.
* It is a 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 could squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.

∴ Outputs : The Softmax function is ideally used by in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.



* Simply use RELU as it is general activation function in hidden layers and is used in most cases these days.

* If your output is 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 class

LOSS FUNCTIONS

The loss function is very important in machine learning (or) deep learning.

* In mathematical optimization and decision theory, a loss (or) cost function (Sometimes also called an error function) is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event.

In simple terms, the loss function is a method of evaluating how tell well your algorithm is modeling your dataset. It is mathematical function of the parameter of the machine learning algorithm.

Why loss function in Deep Learning is Important

* Famous author Peter Drucker says you can't improve what you can't measure. That's why the loss function comes into the picture to evaluate how ~~tell~~ well your algorithm is modeling your dataset.

* If the value of the loss function is lower then it's a good model otherwise, we have to change the parameter of the model and minimize the loss.

classification loss function

A loss function (also known as a cost function (or) objective function). quantifiers the difference between the predicted outputs of a model and the actual target values in the training data.

* During training process, the goal is to minimize the value of the loss function, which reflects how well the model is performing on the training data.

Common loss Functions for classification:

* Mean Squared Error (MSE) Loss (Regression)
* Binary Crossentropy Loss (Binary classification)
* Categorical Crossentropy Loss (Multiclass classification)
* Sparse Categorical Crossentropy Loss (Multiclass classification)
* Hinge Loss (Regression)
* Custom loss functions.

Mean Squared Error (MSE) :

It calculates the average squared difference between the predicted values ($\hat{y_i}$) and the actual values (y_i)

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y_i})^2$$

* It is used for regression Problems.

Binary Crossentropy loss (Binary classification):

Appropriate for binary classification problems, it measure the difference between the true binary lables (y_i) and the predicted probabilities ($\hat{y_i}$)

$$\text{Binary Crossentropy} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y_i}) + (1-y_i)\log(1-\hat{y_i})\right]$$

Categorical Crossentropy loss (Multiclass classification):

It extends binary crossentropy to multiple classes by summing over all classes.

$$\text{Categorical Crossentropy} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(\hat{y_{ij}})$$

* It is used for multiclass classification problems.

Sparse Categorical Crossentropy loss (Multiclass classification)

Similar to Categorical Crossentropy but more efficient when dealing with integerencoded class labels.
(Eg. when classes are represented by integers)

Hinge Loss (SVM- Support Vector Machines)

It is used in the context of Support Vector Machines, particularly for binary classification. It encourages correct classification with a margin.

Huber Loss (Regression)

A combination of mean squared error and mean absolute error. It is less sensitive to outliers than MSE.

Custom Loss Functions :

In some cases, especially in advanced scenarios or specific problem domains, researchers and practioners may design custom loss functions tailored to the characteristics of their data and objectives.

OPTIMIZERS

In deep learning, optimizers are algorithms or methods used to minimize the cost or loss function during the training of neural network.

The Goal of training a neural network is to find optimal set of weights and biases that minimize the difference between the predicted output and the actual target values.

Optimizers play a crucial role in this process by adjusting the model parameters iteratively based on the gradients of the loss function with respect to those parameters.

Before proceeding to different types of optimizers, let's get familiar with some of their important terms.

Sample :- It depicts a single row of a dataset.

Epoch :- It denotes the number of times the algorithm operates on the entire training dataset.

Batch :- It is the number of samples to be considered for updating the model parameters.

Cost Function / Loss Function :

A cost function helps you calculate the cost, representing the difference between the actual value and the predicted value.

Learning rate : It offers a degree that denotes how much the model weights should be updated.

weights/Bias : They are learnable parameters that control the signal between two neurons in deep learning model.

Optimizers are various types.

* Gradient Descent (GD)
* Stochastic Gradient Descent (SGD)
* Mini-Batch Gradient Descent

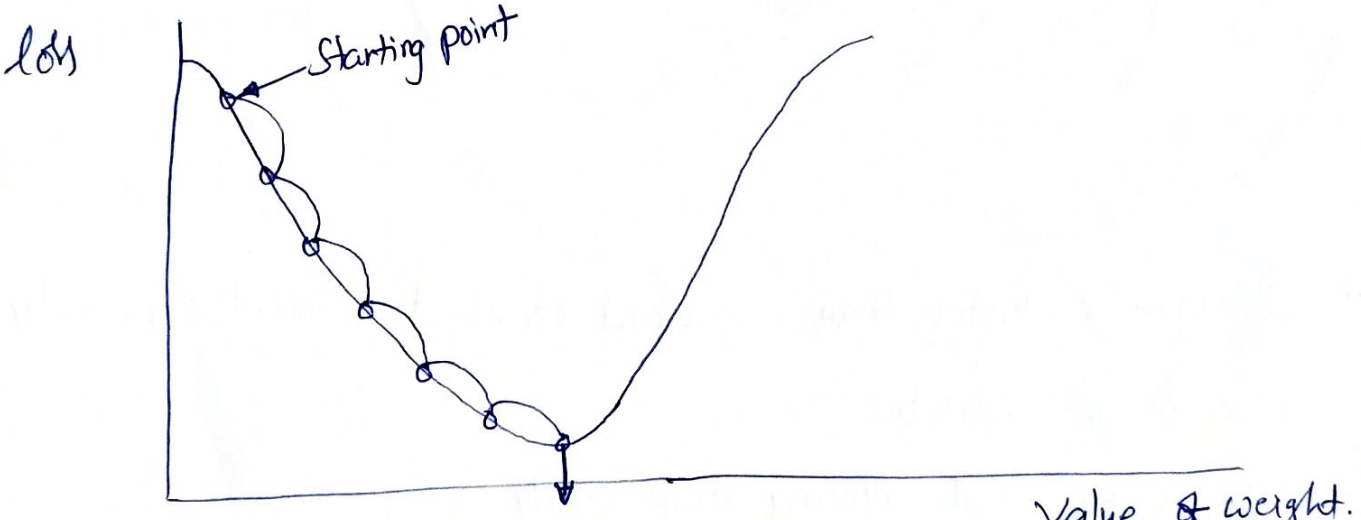
* Adam (Adaptive Moment Estimation)
* RMSprop (Root Mean Square Propagation)
* Adagrad (Adaptive Gradient Algorithm)

Gradient Descent

It is a simple optimization algorithm that updates the model's parameters to minimize the loss function.

$$\theta = \theta - \alpha \cdot \nabla_\theta L(\theta)$$

θ – model parameter
$L(\theta)$ is the loss function.
α is the learning rate.



point of convergence i.e
where the cost function is at its minimum.

* Simple to implement
* Can work well with a well-tuned learning rate.

→ It converge slowly, especially for complex models or large datasets.
→ Sensitive to the choice of learning rate.

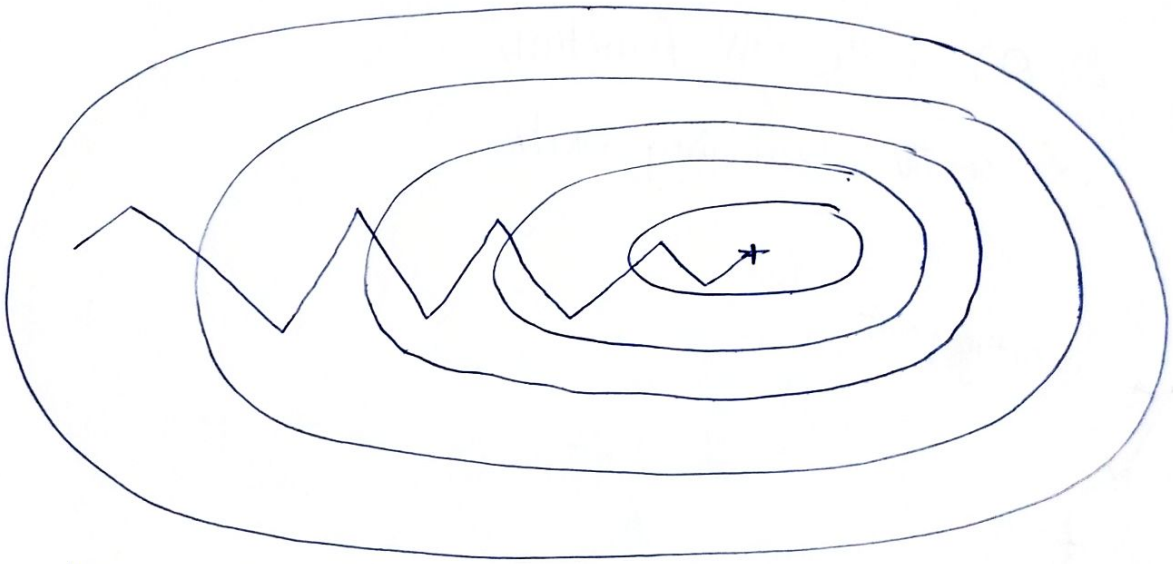
Stochastic gradient descent (SGD):

It is a variant of gradient descent that involves updating the parameters based on a small, randomly-selected subset of the data (i.e a "mini-batch") rather than the full dataset.

basic form of the algorithm

$$\theta = \theta - \alpha \cdot \nabla_\theta L(\theta; x^{(i)}, y^{(i)})$$

where $(x^{(i)}, y^{(i)})$ is a mini batch of data.



* It can be faster than standard Gradient descent, especially for large datasets.
* Can escape local minima more easily

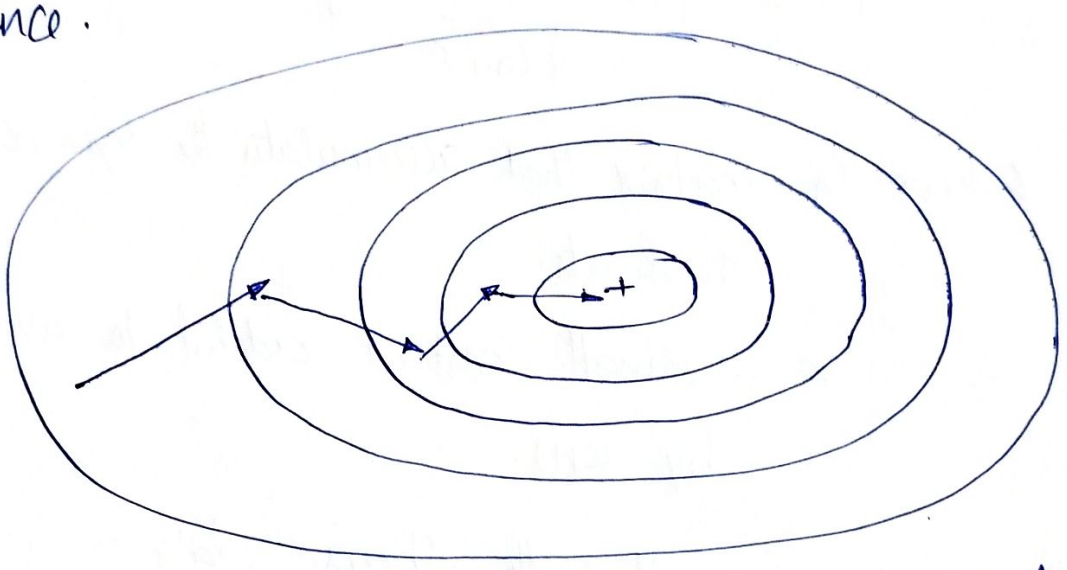
→ It can be noisy, leading to less stability

→ It may require more hyperparameter tuning to get good performance.

Mini-Batch Gradient Descent:

Mini-batch gradient descent is similar to SGD, but instead of using a single sample to compute the gradient, it uses a small, fixed-size "mini-batch" of samples.

The update rule is the same as for SGD, except that the gradient is averaged over the mini-batch. This can reduce noise in the updates and improve convergence.



* It can be faster that standard gradient descent, especially for large datasets.
* Can escape local minima more easily.
* Can reduce noise in updates, leading to more stable convergence

→ Can be sensitive to the choice of mini-batch size.

Adagrad

It is an optimization algorithm that uses an adaptive learning rate per parameter. The learning rate is updated

based on the historical gradient information so that parameters that receive many updates have a lower learning rate, and parameters that receive fewer updates have a larger learning rate.

$$g = \nabla_\theta L(\theta; x^{(i)}; y^{(i)})$$

$$G = G + g \odot g$$

$$\theta = \theta - \frac{\alpha}{\sqrt{G+\epsilon}} \odot g$$

where G - matrix that accumulates the squares of the gradients.

ϵ - small constant added to avoid dividing by zero.

* It can work well with sparse data.
* Automatically adjusts learning rate based on parameter updates.

→ Can converge too slowly for some problems.

→ Can stop learning altogether if the learning rates become too small.

RMSProp (Root Mean Square Propagation)

RMSProp is an optimization algorithm similar to Adagard, but it uses an exponentially decaying average of the squares of the gradients rather than the sum. This helps to reduce the monotonic learning rate decay of Adagrad and improve convergence.

$$g = \nabla_\theta L(\theta; x^{(i)}; y^{(i)})$$

$$G = \beta \cdot G + (1-\beta) \cdot g \odot g$$

$$\theta = \theta - \frac{\alpha}{\sqrt{G+\epsilon}} \odot g$$

where G is a matrix that accumulates the squares of gradients, ϵ is a small constant added to avoid division by zero, and β is a decay rate hyperparameter.

* It can work well with sparse data
* Automatically adjusts learning rate based on parameters updates.
* Can converge faster than Adagrad.

→ It can still converge too slowly for some problems

→ Requires tuning of the decay rate hyperparameter.

Adam

It short for "Adaptive moment estimation". It is an optimization algorithm that combines the ideas of SGD with momentum and RMSProp.

It uses an exponentially decaying average of the gradient and the squares of the gradients to determine the updated scale, similar to RMSProp. It also uses a momentum term to help the optimizer move more efficiently through the loss function.

The update rule can be written as follows.

$$g = \nabla_\theta L(\theta; x^{(i)}; y^{(i)})$$

$$m = \beta_1 \cdot m + (1-\beta_1) \cdot g$$

$$v = \beta_2 \cdot v + (1-\beta_2) \cdot g \odot g$$

$$\hat{m} = \frac{m}{1-\beta_1^t}$$

$$\hat{v} = \frac{v}{1-\beta_2^t}$$

$$\theta = \theta - \frac{\alpha}{\sqrt{\hat{v}+\epsilon}} \odot \hat{m}$$

where m & v are momentum and velocity vectors, ~~especially~~ respectively, and β_1 and β_2 are decay rates for the momentum and velocity.

* Can converge faster than other optimization algorithm
* Can work well with noisy data.

→ It may require more tuning of hyperparameters than other algorithm

→ May perform better on some types of problems.