UNIT-2

Linear Model (Already in Unit-1 notes).

Non Linear (Logistic Regression)

→ It is one of the ~~people~~ most popular ML algorithms, which comes under the Supervised Learning technique.

→ It is used for predicting the categorical dependent variable using a given set of independent variables.

→ It predicts the output of a categorical dependent variable.

→ Therefore the out come must be a categorical or discrete value. It can be either Yes or No, ~~or~~, 0 or 1, true or False etc. but instead of giving the exact value as 0 & 1, it gives the probabilistic value which lie between 0 & 1.

→ It is much similar between to the Linear Regression except that how they are used.

→ Linear Regression is used for solving Regression problems, whereas Logistic Regression is used for solving the classification problems.

→ Instead of fitting a regression line, we fit an "S" shaped logistic function which predicts two Maximum values (0 & 1).

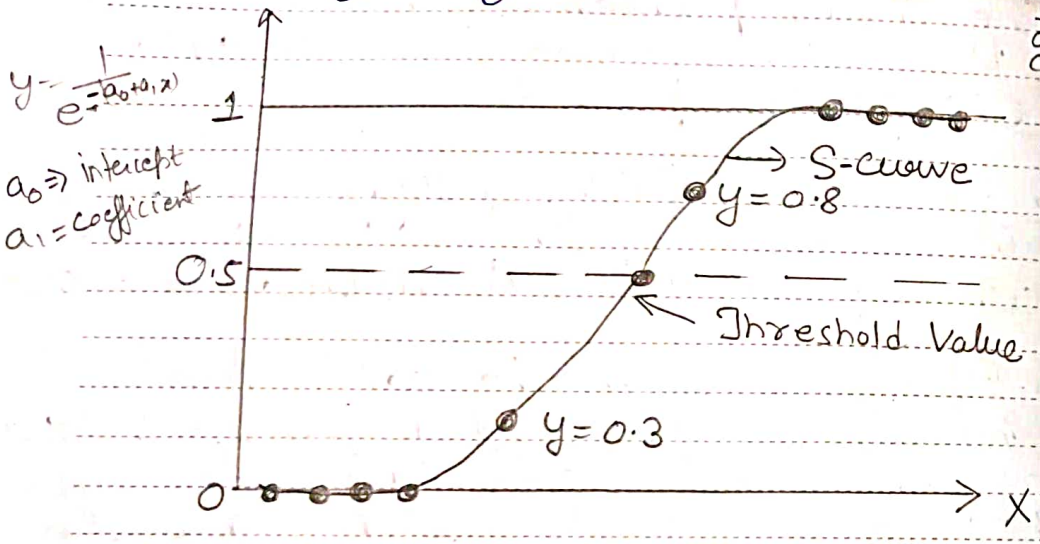
→ The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight etc.

→ It is a significant Machine learning algorithm because it has the ability to provide probabilities & classify new data using continuous & discrete datasets.

→ It can be used to classify the observations using different types of data & can easily determine the most effective variables used for the classification.

Max likelihood ~~or~~ func for estimation find out a_1 & a_2 then cal that func then optimize

→ The below image is showing the logistic function:



then gradient descent then $a_1 = a_2$ = Calculate

→ It uses the concept of predictive modeling as regression, therefore, it is called logistic regression but is used to classify samples. Therefore, it falls under the classification algorithm.

✱ Assumptions for Logistic Regression

→ The dependent variable must be categorical in nature

→ The independent variable should not have multi-collinearity.

✱ Logistic Regression Equation

It can be obtained from linear regression equation. The mathematical steps to get the logistic Regression equation as ~~is~~ given below:

i) Straight Line can be written as

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

ii) y can be between 0 & 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}; \text{ 0 for } y=0, \text{ & infinity for } y=1$$

iii) But we need range between -[infinity to +[infinity], then take logarithm of the equation it will become:

$$\log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + \cdots + b_n$$

The above equation is the final equation of logistic regression.

Types of Logistic Regression

It can be classified into 3 categories

A) Binomial:
There can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail etc.

B) Multinomial:
There can be 3 or more possible unordered types of the dependent variable such as "cat", "dog", or "sheep"

C) Ordinal:
There can be 3 or more possible ordered types of dependent variables, such as "low, Medium or High.

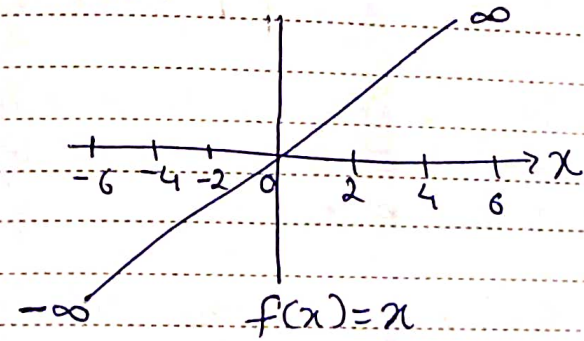
Activation Function

→ It is used to get the output of a node
→ It convert a input signal of a node in network than output signal
→ It is used to map the resulting values in between 0 to 1 or -1 to 1 depending upon the type of function used.

Activation Function

- Linear Activation Function
- Non-Linear Function

A) Linear Activation Function:-
It is also called Identity Activation function



→ The output of the function will not be confined between any range.

→ ANN not be able to learn anything.

B) Non-Linear Activation Function

→ It makes it easy for the model to generalize or adapt with variety of data & to differentiate between the output

→ It makes the network more powerful & add ability to it to learn something complex & complicated from data.

→ Many types of logistic non-linear functions are Logistic, Sigmoid, Tanh, ReLu etc.

* Activation Function (I/P → AF → N/L)

→ The activation function help the network use the important information & supress the irrelevant data points.

→ It decides whether a neuron should be activated or not by calculating the weighted sum & further adding bias to it.

→ It introduce non-linearity into the output of a neuron.

→ If we do not apply activation function then the output signal would be simply linear function (1 degree polynomial).

→ Non-Linear function degree more than the linear function is easy to solve, but they are limited in the complexity & have less power.

→ Without Activation function our model cannot learn & model complicated data such as image, videos, audios, speech etc.

→ We need ANN to learn & represent almost anything & any arbitrary complex function that maps an input to output.

→ Neural network without activation function work as linear regression model.

Types of Activation Function

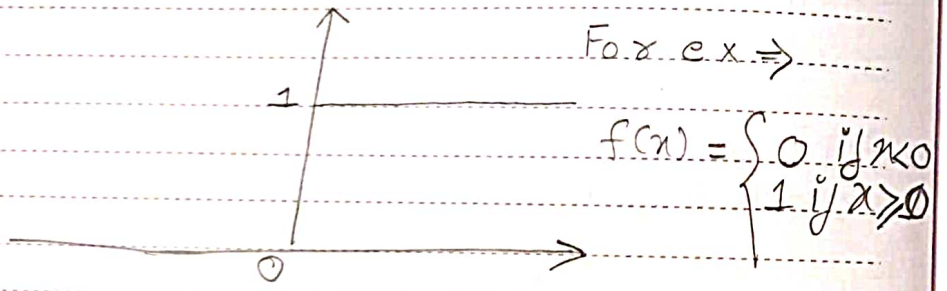
- Threshold Activation function / Binary Step
- Sigmoid / logistic Activation function
- Tanh / Hyperbolic Tangent activation function
- ReLu (Rectified Linear Unit) function
- Leaky ReL

1) Threshold AF (Binary Step function)

→ It is a threshold based activation function
→ Also known as Heaviside function

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

$\theta \rightarrow$ threshold



→ If the input to the BSF, is greater than a threshold than the neuron is activated else it is deactivated.

→ BSF are useful for binary classification scheme.

→ This activation function is useful when the input pattern can only belong to one or two groups that is binary classification.

Limitation

→ This function will not be used when there are multiple classes in the target variable

2) Sigmoid [Logistic function]

→ It is one of the most widely used non-linear activation function.

→ It transforms the values between the range 0 & 1

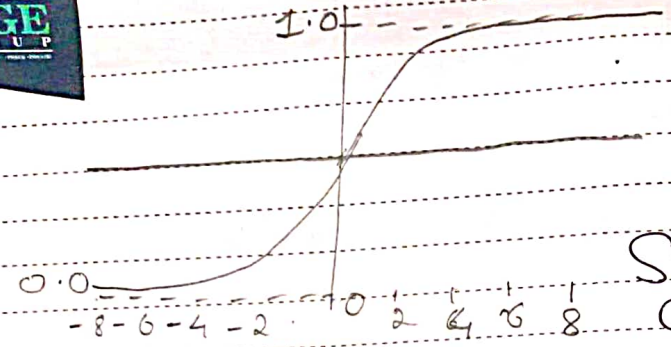
→ Mathematical Expression for Sigmoid is

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

where,

$x \rightarrow -\infty$, $f(x) \rightarrow 0$
$x \rightarrow \infty$, $f(x) = 1$
$x \rightarrow 0$, $f(x) = 0.5$

→ It is a mathematical function having S shaped curve or Sigmoid curve which ranges between 0 & 1



Sigmoid Curve

→ The output is not zero centered

→ It is used for models where we need to predict the probablity as an output

→ It is mostly used for ~~multiple~~ multi-class classification

→ It is used in major machine learning algorithm such as logistic regression & neural network

$$\text{Probablity of event } (x) = \frac{1}{1+e^{x}}$$

→ It converts a value x into a probablity of something happening, such as, the probablity 'p' of rainfall given two weather conditions.

i) If $P >= 0.5$, then output is true

ii) If $P < 0.5$, then output is false

3) Tanh (Hyperbolic Tangent function)

→ It is similar to sigmoid but better in performance

→ It is also a non-linear activation algorithm

→ The function ranges between (-1 to 1)

→ It is symmetric around the origin

→ Formula used:

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

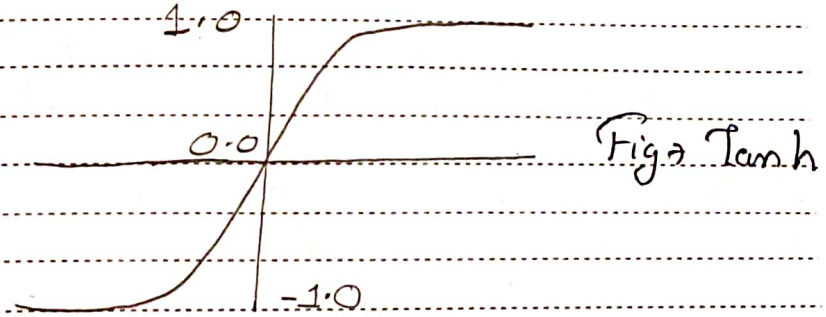


Fig: Tanh

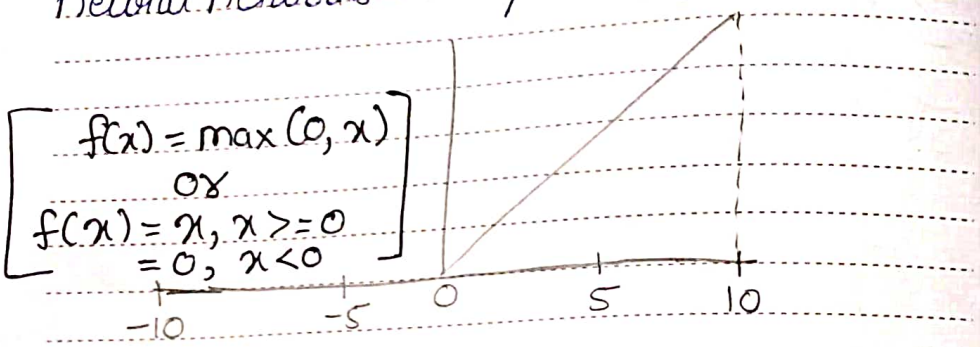
→ The advantage is that negative inputs will be mapped strongly negative & the zero inputs will be mapped near zero in the tanh graph

→ The tanh function is mainly used classification between two classes

4) ReLU (Rectified Linear Unit)

→ Mostly used activation function, since it is used in almost all the convolutional neural networks or deep learning

$$f(x) = \max(0, x)$$
or
$$f(x) = x, \quad x >= 0$$
$$= 0, \quad x < 0$$



→ It gives an output x if x is positive & 0 otherwise 0

→ It ranges between 0 to ∞

→ It is half rectified

→ $f(x) = x$ when x is above or equal to zero

Limitations

→ All the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly.

→ It is less computationally expensive than ~~tanh~~ & sigm tanh & Sigmoid because it involves simpler mathematical operations

→ Its main advantage of this function over other ~~ReLU~~ activation functions is that it does not activate all the neurons at the same time.

→ This means that the neurons will only for deactivated if the output of the linear transformation is less than 0.

→ In backpropagation process, the weight & bias for some neurons are not updated

→ This can create dead neurons which never get activated.

→ This is taken care by the 'Leaky' ReLU function.

5) Leaky ReLU:-

→ It's an improved version of ReLU function

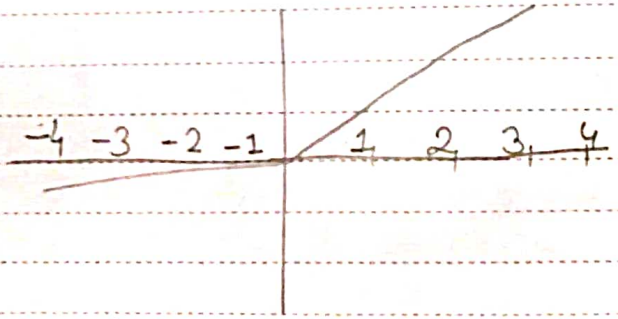
→ For ReLU function the gradient is 0 for $x<0$, which would deactivate the neurons in that region.

→ Leaky ReLU is defined to address this problem

→ Instead of defining the ReLU function as 0 for negative values of x, we define it as an extremely small linear component of x

* The mathematical expression

$$f(x) = 0.01x, \quad x<0$$
$$= x, \quad x \geq 0$$



By making this small modification, the gradient of the left side of graph comes out to be a non-zero value

→ So there would no longer encounter dead neurons in that region.

Artificial Neural Network (ANN)

Google translate, Google Assistant

→ An ANN is a computational nonlinear model that is inspired from the brain (i.e. neurons)

→ Like people, it learn by example

→ ANN can perform tasks like classification, prediction, decision making, visualization & others just by considering examples

→ ANN consist of large collection of artificial neurons or processing elements which operates in parallel

→ Every neuron is connected with other neuron through a connection link

→ Each connection link is associated with a weight that has information about the input signal.

→ Weight is the most useful information for neuron to solve a particular problem because it usually

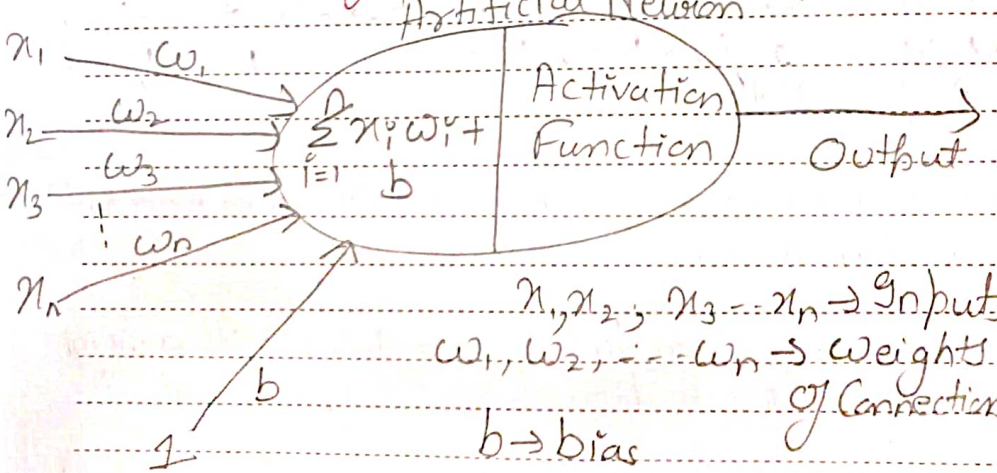
excites or inhibits the signal that is being communicated.

→ Every neuron has weighted inputs (synapses) an activation function that defines the output given an input & one output.

→ The neuron / nodes can take input data & perform simple operations on the data & the result of these operations is passed to other neurons

→ ANNs are capable of learning, which take place by altering weight values.

Model of ANN



$x_1, x_2, x_3 \ldots x_n \rightarrow$ Inputs

$w_1, w_2, \ldots w_n \rightarrow$ weights of Connections

$b \rightarrow$ bias

Fig → Perceptron (Single layer ANN)

→ ANNs can be viewed as weighted directed graphs in which artificial neurons are nodes & directed edges with weights are connections between neuron outputs & neuron inputs.

→ It is inspired by a biological neuron.

→ A single layer neural network is called perceptron. It gives a simple output.

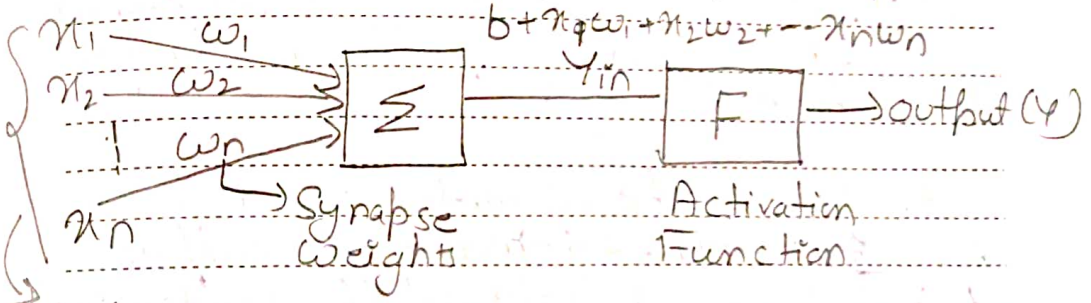


Fig → Perceptron (General Model of ANN)

→ In above figure, $x_1, x_2 \ldots x_n$ are the various inputs (independent variables) to the network.

→ Each of these inputs is multiplied by a connection weight or synapse.

→ Weights $w_1, w_2 \dots w_n$ shows the strength of a particular node

→ b → bias value, it allows you to shift the activation function up or down

→ Products of inputs & weight are summed & fed to a activation function to generate a result & this result is sent as output

→ ANN receives input from the external world in the form of pattern & image in vector form

→ In case of the weighted sum is zero, bias (b) is added to make the output non-zero or to scale up the system response.

→ Weighted sum corresponds to any numerical value ranging from 0 to ∞

→ ANN helps you to conduct image understanding, human learning, computer speech etc.

→ In order to limit the response to arrive at desired value, the threshold value is set up

→ For this, the sum is passed through activation function

→ The activation function is set of the transfer function used to get desired output

Range [-1 to 1]

→ Activation Function can be linear or non-linear.

→ The main purpose of AF is to convert input signal of a node in an ANN to an output signal. This output signal is used as input to the next layer layer., in the network

From the above general model of ANN, the net input can be calculated as

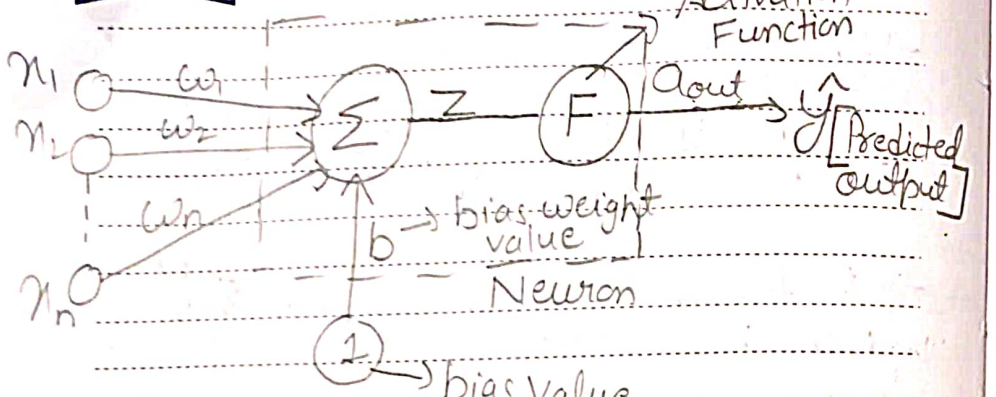
$$Y_{in} = x_1 w_1 + x_2 w_2 + \dots x_n w_n$$

$$Y_{in} = \sum_{i=1}^{n} x_i w_i$$

The output can be calculated by applying the AF over the net input

$$Y = F(\Sigma x_i w_i + \text{bias})$$

Operation at one Neuron of a Neural Network



i) Neuron / Node:-

→ It is the basic unit of a neural network

→ It gets certain ~~or~~ number of inputs & bias values.

$$z = x_1w_1 + x_2w_2 + \dots x_nw_n + bx1$$

$$z = \sum_{i=1} x_i \cdot w_i + b$$

$$a_{out} = F(z) = \frac{1}{1+e^{-z}} \quad \text{[for Sigmoid Activation function]}$$

$$\therefore \hat{y} = \frac{1}{1+e^{-z}} \quad \text{Predicted Output}$$

Operation of Multilayer Neural Network (ANN)

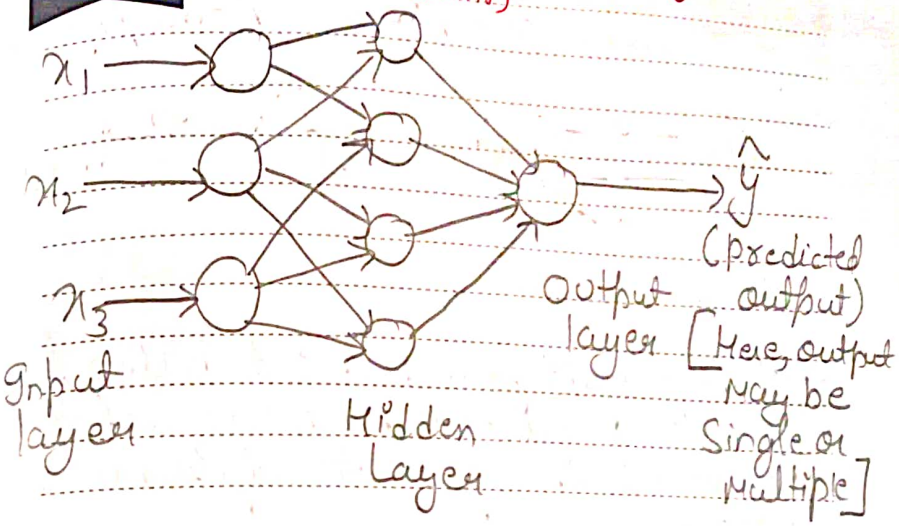


Fig → Multilayer Neural Network with Single output

ii) Connections

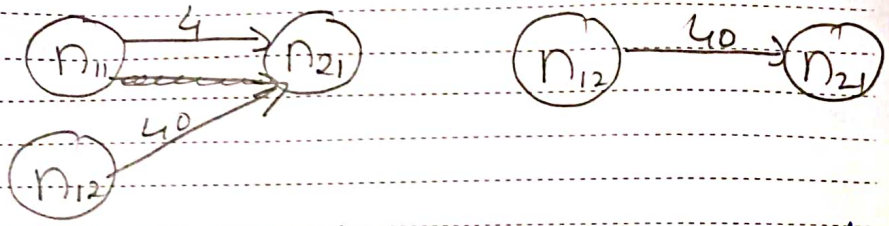
It connects one neuron in one layer to another neuron in another layer

→ A connection always has a weight value associated with it

→ Goal of ANN training is to update Connections weight values to decrease the loss/error

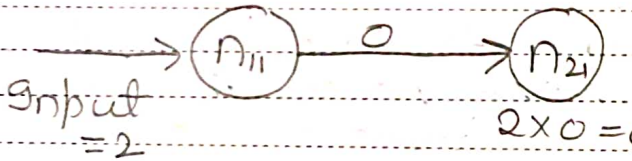
iii) Weights / Parameters :-

A weight represent the strength of the connection between nodes / neurons

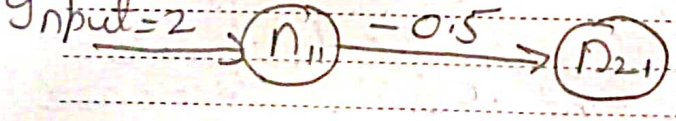


→ If the weight from node 1 to node 2 has greater magnitude, it means that neuron 1 has greater influence over neuron 2.

→ A weight brings down the importance of the input value.



→ Weights near zero means changing this input will not change the output.



→ Negative weights means increasing this input will decrease the output

→ So, weight decides how much influence the input will have on the output

Multilayer Artificial Neural Network

→ An ANN consists of artificial neurons or processing elements & is organized in three interconnected layers

i) Input layer (Single)
ii) Hidden layer (One or more)
iii) Output layer (Single ~~or more~~)

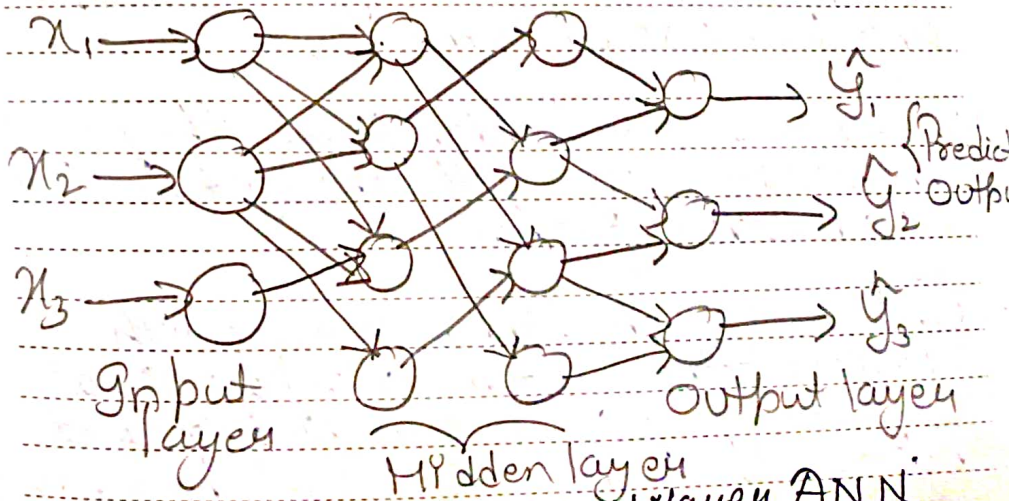


Fig:- Multilayer ANN (Multiple outputs)

i) Input Layer:-

→ 1st layer in the neural network

→ This layer contains neurons ~~which~~ which receive input from the outside world & passes them on to the next layer (hidden layer)

→ It doesn't apply any operations on the input values & has no weights & biases value ~~asso~~ associated

ii) Hidden Layer:-

→ These units (neurons) are in between input & output layers.

→ The job of hidden layer is to transform the input into something that output unit can use in some way.

→ These neurons are hidden from the people who are interfacing with the system & acts as a blackbox to them

→ On increasing the hidden layers with neurons, the system's computational & processing power can be increased but the training process of the System gets more complex at the same time

→ Hidden layer ~~se~~ neurons send information to output layer

iii) Output Layer

→ It is the last layer in the neural network & it receives the input from the last hidden layer

→ With this layer we can get desired number of values (outputs) & in a desired range

→ Most ANN are fully connected that means each hidden neuron is fully connected to every neuron in its previous layer (input) & to the next layer (output) layer

Loss Function:-

→ It is a method of evaluating how well machine learning algorithm models given datasets

→ If the prediction of models are wrong (deviates to meet from actual result) loss function will output a higher number. If predictions are good, it will output a lower number.

→ To improve the output (predictions) of machine learning algorithm, loss function help us.

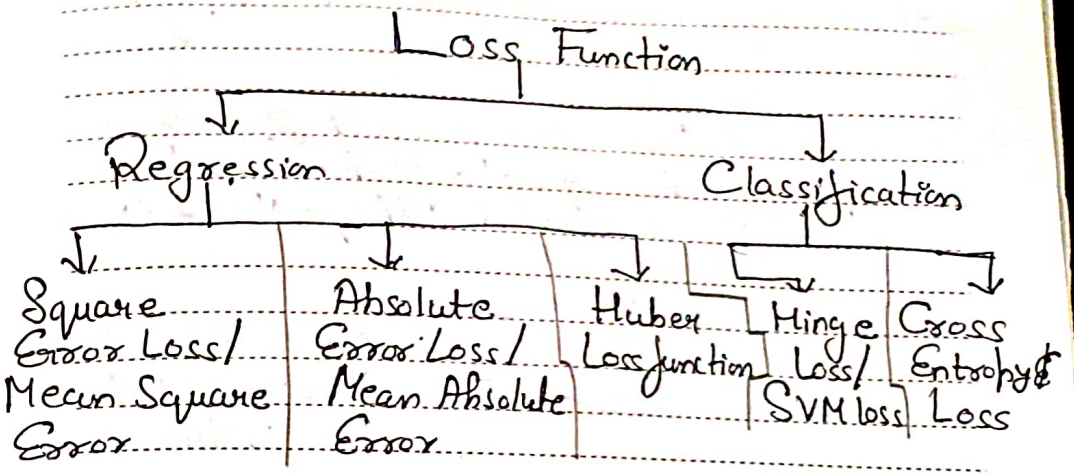
→ Machine learn by means of loss function

→ With the help of some optimization function like gradient descent, loss function learns to reduce the error in prediction

→ A loss function / error function is for a single training example

→ A cost function is the average loss over the entire training dataset

→ The optimization strategies like gradient descent aim at minimizing the cost function



i) Regression

It deals with predicting a continuous value for example future salary of an employee or future price of house etc.

ii) Classification:-

It refers to assigning an object into one or more than one classes.

Ex → Classifying an email as spam or not

* Regression Loss Function

1) Squared Error Loss / (L2 Loss) / Quadratic Loss

→ Squared Error loss for each training example is the square of the difference between the actual & the predicted values

$$L = (y - f(x))^2$$

OR

$$L = (y - \hat{y})^2$$

y → Actual value
$\hat{y}$ → Predicted value

→ The corresponding cost function is the mean of these Squared Errors (MSE)

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

n → nos of datapoints or observation

→ Mean Square error is measured as the average of squared difference between prediction & actual observation.

→ The lesser the value of MSE, the better are the predictions

→ Easier to solve

→ In regression Machine learning algorithm, our goal is to minimize this mean, which will provide us with the best fit line that goes through all the points.

→ Due to squaring, predictions which are far away from actual values are penalized heavily in comparison to less deviated predictions

→ But if our data is prone to many ~~either~~ outliers then we should not use MSE because Squaring a large ~~quat~~ quantity makes it even large.

→ MSE cost function is less Robust to outliers

2) Absolute Error Loss:-

→ Absolute Error for each training example is the distance between the predicted & the actual values, irrespective of the sign

$$L = |y - f(x)| \text{ or } L = |y - \hat{y}|$$

y → actual value, $\hat{y}$ → predicted value

The corresponding cost function is the mean of these absolute errors (MAE)

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

$n \rightarrow$ nos. of observations / samples

→ Mean absolute error is measured as the average of sum of absolute difference between prediction & actual observations

Advantage

→ Unlike MSE, MAE is more robust to outliers since it does not make use of square

Disadvantage:-

→ Handling the absolute or modules operator in mathematical equation is not easy Ex

Error	Absolute Error
0	0
1	1
2	1
3	1
4	2
5	5

Ex

	Error	Absolute Error	Error²
1	0	0	0
2	1	1	1
3	1	1	1
4	-2	2	4
5	15	15	225 → outlier

3) Huber Loss

→ It combines the best properties of MSE & MAE

→ It is quadratic for smaller errors & is linear otherwise.

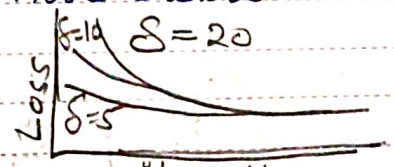
→ It is identified by its delta parameter.

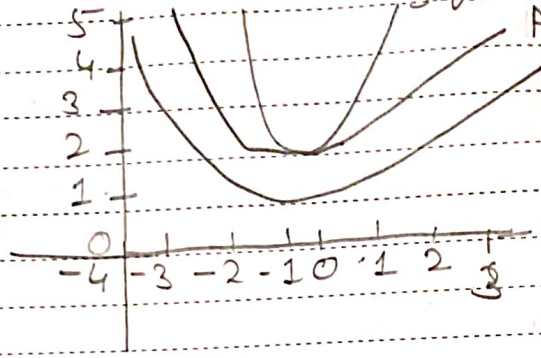
$$L_\delta = \begin{cases} \frac{1}{2}(y - f(x))^2, & \text{if } |y - f(x)| \le \delta \\ \delta |y - f(x)| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases}$$

↳ Quadratic for small value

↳ Linear for large value

→ Huber loss is more robust to outliers than MSE.





Drawback

We might need to train hyper-parameter delta which is an iteration process

✱ CLASSIFICATION LOSS FUNCTION

1) Hinge Loss / Multiclass SVM Loss:-

→ It is popular with support vector Machine (SVMs)

→ These are used for training the classifiers

→ Binary classification Loss function

Loss function

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } S_{y_i} \geq S_j + 1 \\ S_j - S_{y_i} + 1 & \text{otherwise} \end{cases}$$

(correct predicted value; wrong predicted value)

$$= \sum_{j=y_i} \max(0, S_j - S_{y_i} + 1)$$

$(S_j \neq S_{y_i})$

→ For each given training input, we compute each of its associated class scores.

→ In the sum above, we compare the score of right category against those for the wrong category

→ This is done by subtracting the wrong class value & the right class value if the score difference is greater than some fudge factor (usually 1) otherwise the error is zero

Ex

	Cat	Car	Frog
	Predicted score		
Cat	(3.2)	1.3	2.2
Car	5.1	(4.9)	2.5
Frog	-1.7	2.0	(-3.1)
Losses	2.9	0	12.9

3 training examples & 3 classes to predict

Soln Cat: $5.1 - 3.2 + 1 = 2.9$

$-1.7 - 3.2 + 1 = -3.9 \approx 0$ (compare to 1)

$\max(0,$

→ In simple term, the score of correct category should be greater than sum of scores of all incorrect categories by some safety margin (usually one)

2) Cross Entropy Loss function

Ex

Animal Image	cat	Dog	Frog
Label	[1 0 0]	[0 1 0]	[0 0 1]

↑ Probablity distribution of each image

Each image is labelled with the corresponding animal using the one hot encoding

Let we designed a machine learning model that classifier those images

Intially Prediction

First it classify the first image as

$\hat{P} = [0.4 \; 0.3 \; 0.3]$

It means, model says that the first image is

40% for a cat
30% for a dog
30% for a frog

But this estimation is not very precise about what animal the image has

Label for cat (P) = [1 0 0]

$\therefore$ Cross Entropy $= -[1 \times \log 0.4 + 0 \times \log 0.3 + 0 \times \log 0.3]$

$= -\log 0.4$

≈ 0.916 (>0 - Error/loss)

→ When model is well trained, let it predict the following values for 1st image

$\hat{P} = [0.98 \; 0.01 \; 0.01]$

$\therefore$ Cross Entropy $= -(1 \times \log 0.98 + 0 \times \log 0.01 + 0 \times \log 0.01)$

$= -\log 0.98$

≈ 0.02 (>0 - error/loss)

Comparison to first prediction of image the cross entropy for second prediction is much lower.

That means the cross entropy goes down as the machine learning model predictions gets more & more accurate.

Cross entropy = 0 (for perfect prediction)

→ That means we can use cross entropy as a loss function to train a classification model.

* Binary Cross Entropy Loss Function

→ It is cross entropy with two classes
→ It is used for binary classification (1/0 or Yes/No)

Formula for binary cross entropy

$$L = -P \log \hat{P} - (1-P) \log(1-\hat{P})$$

$$L = -[P \log \hat{P} + (1-P) \log(1-\hat{P})]$$

Cat — Dog

Loss Function $L = -\sum_{i=1}^{2} P_i \log \hat{P}_i$

Cat = P-60%
Dog = (1-P)-40%

$$L = P_1 \log \hat{P}_1 + P_2 \log \hat{P}_2$$

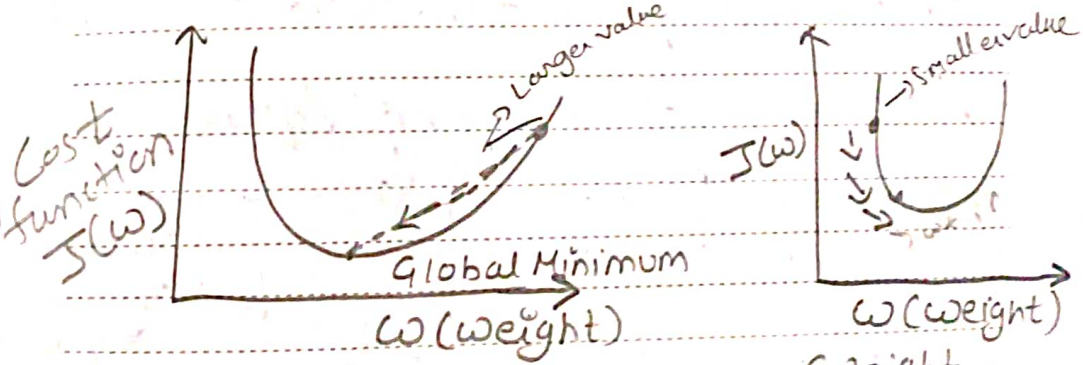
→ For multi class classification

$$L = -\sum_{i=1}^{m} P_i \log \hat{P}_i$$

→ Usually first a sigmoid activation function is applied to get the predicted output in probabilities & then cross entropy loss is ~~call~~ Calculated.

Gradient Descent

↳ ~~Sad~~ Slope ↳ act of moving downwards



→ It is one of the most popular & widely used optimization algorithm used for

training machine learning models.

→ The optimizer is a search technique which is used to update weights (Parameters) in the model

→ For a given machine learning model with parameters (weights & biases) & a cost function (used to evaluate the model) & for Linear regression → coefficients evaluate

→ The aim of our learning problem (ANN) is to find a good set of weights for our model which minimizes the cost function

→ So, Gradient descent is an optimization algorithm that finds the optimal weights that reduces prediction error (cost function)

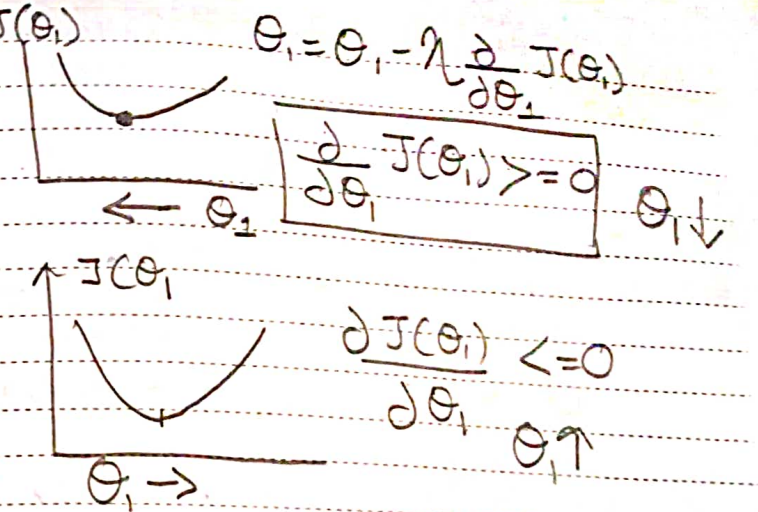
Let cost function is Mean of Squared Error

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

y_i → actual output

$\hat{y}_i$ → Predicted output

$$J(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$



$$\theta_1 = \theta_1 - \eta \frac{\partial}{\partial \theta_1} J(\theta_1)$$

$\frac{\partial}{\partial \theta_1} J(\theta_1) >= 0$ → $\theta_1 \downarrow$

$\frac{\partial J(\theta_1)}{\partial \theta_1} <= 0$ → $\theta_1 \uparrow$

→ Gradient descent is an optimization algorithm (iteration) used to minimize cost function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient

→ During training of Neural Network, our aim is to find a set of values for W such that $(y_i - \hat{y}_i)^2$ is small. This means prediction of learning model will close to the target/actual output.

→ In this, instead of going through every weight one at a time, we look at the angle of the cost function line (↳ derivative)

→ Gradient descent works well when we have a convex curve

Updated weight: $w' = w - \eta \frac{\partial J}{\partial w}$ (old weight = w)

$\eta \rightarrow$ Learning rate

When $\frac{\partial J}{\partial w} = -ve$: $w' = w + \eta \frac{\partial J}{\partial w}$

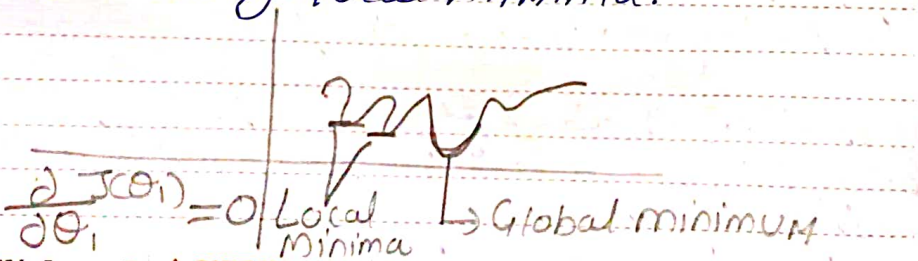
When $\frac{\partial J}{\partial w} = +ve$: $w' = w - \eta \frac{\partial J}{\partial w}$

→ It works only for problem which have a well defined convex optimization problem.

→ Convex optimization problem may also consist of minimal points

Global minimum — lowest point
Local ——"—— — other points

→ In this, our aim is to find global minimum while avoiding local minima.



Gradient Descent

Full Batch | MiniBatch | Stochastic

i) Full Batch:- Entire training data (Whole data) is used at once to ~~complete~~ Compute the gradient.

→ It is slow algorithm.

~~→ We calculate gradients by differentiation of cost function~~

ii) Stochastic:-
A sample data (Single training data) is used to compute the gradient

→ It is fast algorithm

iii) Mini Batch:-
We calculate the gradient for each small (mini) batch of training data.

→ It used in compilers, CPUs optimized for performing ~~to~~ Vector addition & multiplication

Batches = m samples

→ Gradient Descent algorithm calculates gradient by differentiation of cost function.

→ Gradient Descent only works for problems which have a well defined convex optimization problem.

Pseudocode / Algorithm for Gradient descent

Aim → It is used to minimize a cost function $J(w)$ parameterized by model parameters w

Step-1: Initialize the weight 'w' randomly

Step-2: Calculate Error (Sum of Squared Error)

Step-3: Calculate the gradient G of cost function w.r.t. 'w' i.e change in Sum of Squared Error when the weights 'w' are changed by a very small value from their original randomly initialized value.
This helps us move the values of 'w' in the direction where SSE is minimized

Step 4: Adjust / Update the weights with the gradients to reach the optimal values where SSE is minimized.

$$w = w - \eta \frac{\partial J}{\partial w}$$

where; η → learning rate, Gradient $(G) = \frac{\partial J}{\partial w}$

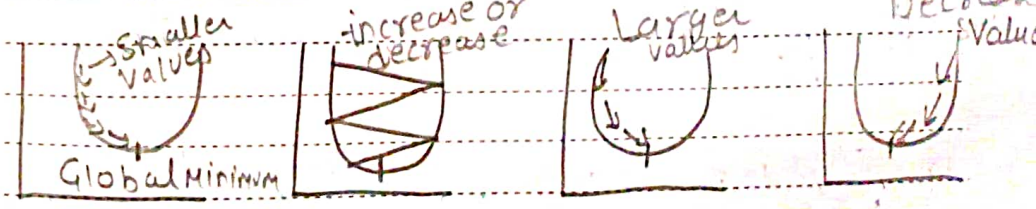
Step-5: Use the new weight for prediction & to calculate the new SSE.

→ Repeat Step 3 & 4 till further adjustments to weights doesn't significantly, reduce the error $J(w)$

Several passes are performed over the training data before the termination criteria is met.

→ Each pass is called an Epoch.

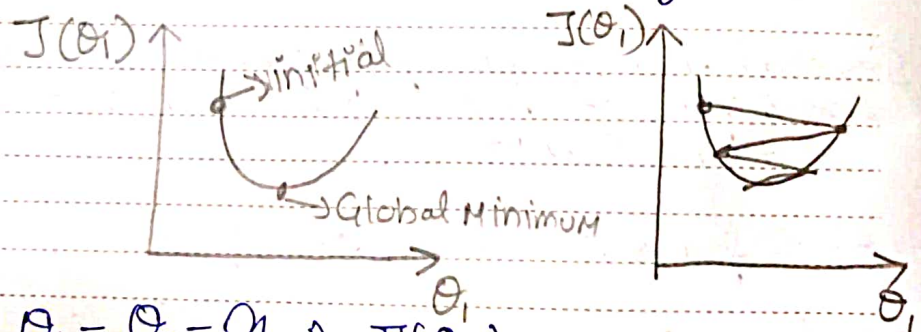
Learning rate determines the size of steps we take to reach a minimum

→ 

→ This parameter is called hyperparameter in ANN. We need to be very careful about this parameter.

→ Since high values of η many overshoot the minimum & very low value will reach minimum very slowly.

→ We usually start with a small value of learning such as 0.1, 0.01, or 0.001 & change it based on whether the cost function is reducing very slowly → increase learning rate or if exploding / being erratic ⊕ → decrease learning rate.



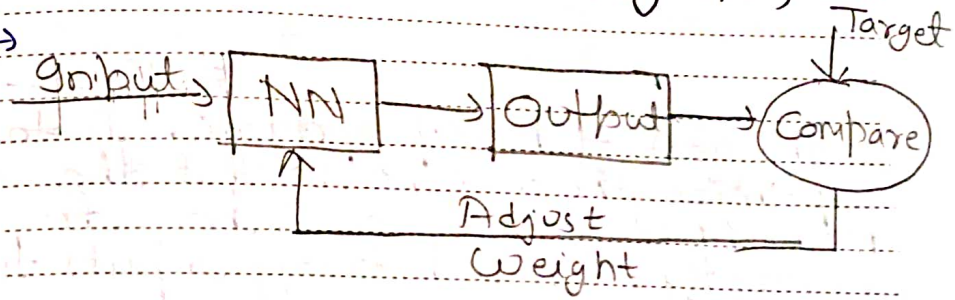
i) $\theta_1 = \theta_1 - \eta \frac{\partial}{\partial \theta_1} J(\theta_1)$

If η is too small, gradient descent can be slow

ii) If η is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge

Backpropagation (Supervised Learning Algorithm)

→

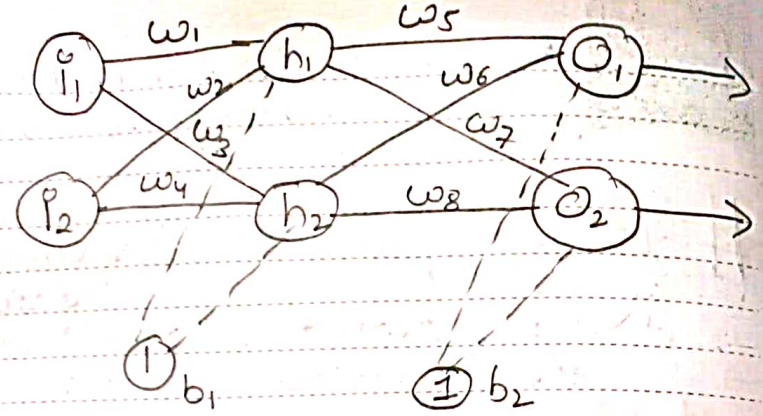


→ It is the training or learning algorithm

→ It is a standard method of training artificial neural networks.

→ It learns by example. If you submit to the algorithm the example of what you want the network to do, it changes the network's weights, so that it can produce desired output for a particular input on finishing the training

→ These networks are used for simple pattern recognition & mapping tasks.

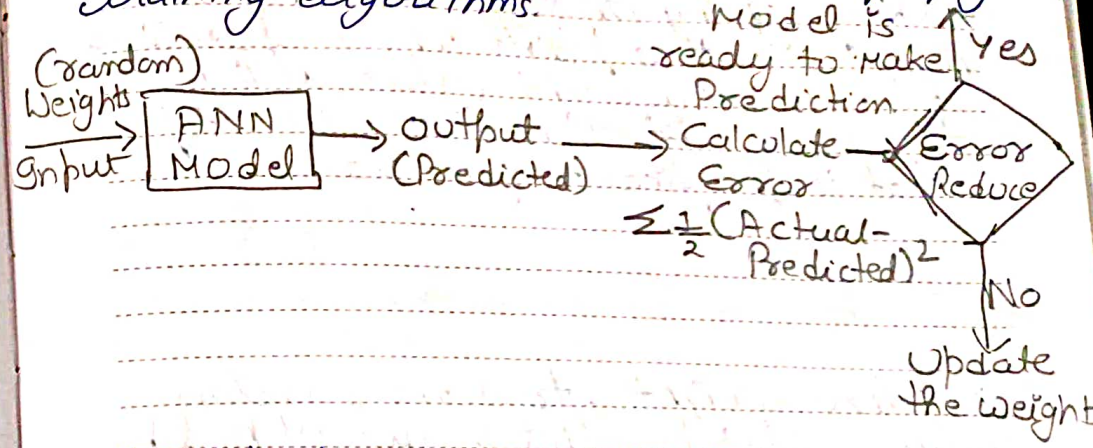


→ The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs

→ While designing a neural network, in the begining we initialize weights with some random values, It's not necessary that whatever weight values we have selected will be correct & it may happen that with these weights

→ Our model output is very much different than our actual output that means the error value is high.

→ To reduce the error, we need to somehow explain the model to change the weights such that error become minimum. For this, we use backpropagation training algorithms.



→ Backpropagation algorithm uses gradient descent technique to minimize the value of error function

→ Here, we are trying to get the value of weight such that error becomes minimum

→ We have to find out whether we need to increase or decrease the value of weight. If by increasing value of weight error reduces, we keep on increasing the weight value otherwise we

decrease the weight value to get minimum error.

→ keep going on right direction we might reach a point where further update of weight value will increase the error. At that time, we stop & that is the final weight value.

Steps involved in Backpropogation

→ Intialize weights to some random value, & also predict value of b_1 & b_2 bias

Step 1: Forward Propagation

The inputs from a training set are passed through the network

We find the total net input to each hidden layer neuron h_1 & b_2 using following formula

$$net_{h_1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h_2} = w_3 * i_1 + w_4 * i_2 + b_1 * 1$$

then using sigmoid activation function, calculate the output of h_1 & h_2

$$Out_{h_1} = \frac{1}{1+e^{-net_{h_1}}} \quad \& \quad Out_{h_2} = \frac{1}{1+e^{-net_{h_2}}}$$

→ The output of hidden layers forwarded as input to output layer, again using above formula, we calculate the output of output neurons O_1 & O_2

$$net_{O_1} = w_5 * Out_{h_1} + w_6 * Out_{h_2} + b_2 * 1$$

$$net_{O_2} = w_7 * Out_{h_1} + w_8 * Out_{h_2} + b_2 * 1$$

Step 2 Calculating the total error

→ As we are working with a training set, the correct output is known so we calculate the error for each output neuron O_1 & O_2 using the Square Error function.

$$E_{O_1} = \frac{1}{2}(target_{O_1} - Out_{O_1})^2$$

$$E_{O_2} = \frac{1}{2}(target_{O_2} - Out_{O_2})^2$$

The total error for the neural network is the sum of these errors.

$$E_{total} = E_{O1} + E_{O2}$$

Step 3:- Backward Propagation

→ The objective of backpropagation is to change the weights & biases for the neurons, in order to bring the error function to a minimum

Now, we will propogate backwards

first update weight w_5

Firstly, we have to calculate how much a change in w_5 affects the total error.

Calculate $\frac{\partial E_{total}}{\partial w_5}$ → The partial derivate of Error w.r.t. w_5

To decrease the error we then subtract this value from the current weight w_5

$$w_5' = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5}$$

η → Learning rate

→ We repeat this process to get the new weights w_6', w_7' & w_8'

[Note Chain rule $\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E}{\partial O_1} * \frac{\partial O_1}{\partial h_1} * \frac{\partial h_1}{\partial w_5}$]

Next we will continue backwards to calculate new values of w_1, w_2, w_3 & w_4 as

$$w_1' = w_1 - \eta \frac{\partial E_{total}}{\partial w_1}, \quad w_2' = w_2 - \eta \frac{\partial E_{total}}{\partial w_2}$$

$$w_3' = w_3 - \eta \frac{\partial E_{total}}{\partial w_3}, \quad w_4' = w_4 - \eta \frac{\partial E_{total}}{\partial w_4}$$

Finally, we have updated all of our weight

After that we will again propagate forward & calculate the output.

Again we will calculate error.

→ If the error is minimum we will stop else we will again propogate backwards & update the weight values

→ This process will keep on repeating until error becomes minimum

→ At the end of this process, the model is ~~to~~ ready to make the predictions for unknown input data.

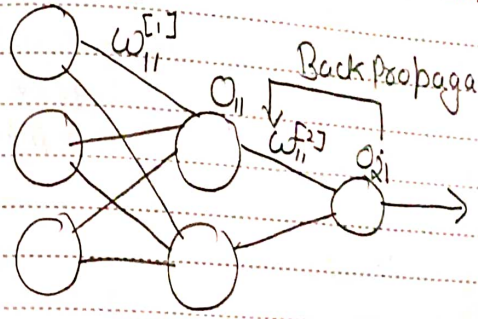
Weight Initialization & types of Gradient Problem

→ First step for building a neural network is the initialization of parameters (weights), if correct initialization is done then optimization will be achieved in the least time otherwise Converging to a ~~mini~~ minima will be impossible using Optimization technique like gradient Descent.

→ Intially we assign random values to weight, while fitting a deep network it can lead two problems

* Let we use sigmoid activation function to produce output

1) Vanishing Gradient Problem



~~$w = w - \eta \frac{dJ}{dw} \rightarrow w_{11}^{[1]'} < w_{11}^{[1]} - \eta \partial_{01}$~~

$$w' = w - \eta \frac{\partial J}{\partial w} \Rightarrow w_{11}^{[1]'} = w_{11}^{[1]} - \eta \frac{\partial J}{\partial w_{11}^{[1]}}$$

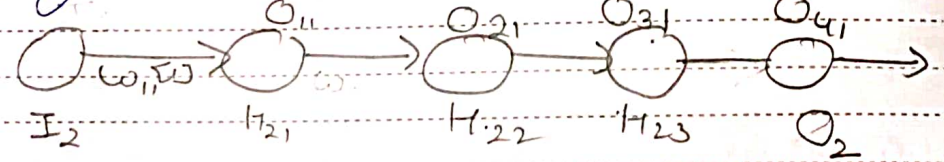
$$\frac{\partial J}{\partial w_{11}^{[1]}} = \frac{\partial J}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial O_{11}} \cdot \frac{\partial O_{11}}{\partial w_{11}^{[1]}}$$

⇒ In case of deep networks using Sigmoid activation functions, if weights are small then the gradient will be vanishingly small

[The gradient of the cost with respect to the weights are too small]

→ During the backpropagation stage, the error is calculated & gradient values are determined. The gradient are sent back to hidden layers & the weights are updated accordingly.

→ The process of gradient determination & sending it back to the next hidden layer is ~~contained~~. Continued until the input layer is reached

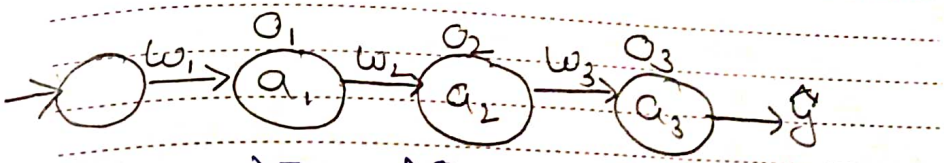


$$\frac{\partial J}{\partial w_{11}^{[1]}} = \frac{\partial J}{\partial O_{41}} \cdot \frac{\partial O_{41}}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial O_{11}} \cdot \frac{\partial O_{11}}{\partial w_{11}^{[1]}}$$

→ The gradient becomes smaller & smaller as it reaches the bottom of the network. Therefore, the weight of the initial layers will either update very slowly or remains the same.

In other words, the initial layers of the network won't learn effectively. Hence it can lead to overall inaccuracy of the whole network.

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



$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial O_3} * \frac{\partial O_3}{\partial O_2} * \frac{\partial O_2}{\partial O_1} * \frac{\partial O_1}{\partial w_1} \quad \text{— (1)}$$

$$\frac{\partial O_3}{\partial O_2} = \frac{\partial \text{Sigmoid}(O_2)}{\partial O_2} * w_3$$

$$= \frac{1}{4} * \frac{1}{3} = \frac{1}{12} \text{ (less value)}$$

$$\frac{\partial O_2}{\partial O_1} = \frac{\partial \text{Sigmoid}(O_1)}{\partial O_1} * w_2$$

{ less value }

→ Weights in the neural network usually between −1 & 1

→ Output of derivative of sigmoid function lies between 0 & 1/4

In eqn ① we multiply values between 0 & 1, so this gradient $\left(\frac{\partial J}{\partial w_1}\right)$ will be smaller

* Vanishing Gradient problem Definition

→ As we move backward (backpropagation) in the ANN, gradient becomes smaller & smaller in every layer. & it becomes tiny in the early layers (1st layer). This is called vanishing gradient problem

→ As input layer is farthest from the ~~at~~ output layer, its derivatives will be the layer expression (Chain rule). Hence it contain more Sigmoid derivatives and it will have smallest derivative, this makes lower layers slowest to train

~~*~~ Problem

→ Because of small step changes, ANN may convert at a local minimum instead of global minimum

→ If early layers are not accurate then later cycles build on this inaccuracy & the entire neural network gets corrupted.

ii) Exploding Gradient Problem

→ If the gradients were bigger than one (that means multiplying numbers greater than 1 always gives huge ~~unit~~ results). & because of this some layers may get instantly large weights & the algorithm diverges. This is called exploding gradient problem.

→ At an extreme, the values of weights can become so large as to overflow & result in "Not An Number" values

→ Exploding gradients can results in an unstable network that at best cannot learn from the learning data & at worst results, in 'NAN' weight values that can not longer be updated

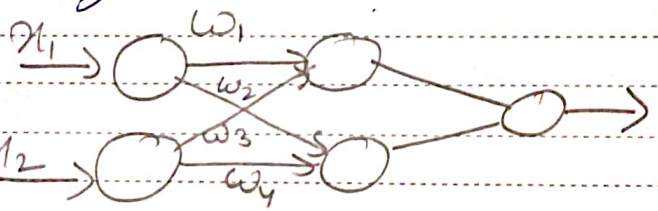
Weight Initialization Techniques

→ There are a ~~bigger~~ large number of weights in a neural network leading to a large search space for minima of the loss function

→ Weights must be effectively randomly initialized to prevent convergence to false minima

→ If all the weights are intialized with 0 in ANN training, it does not make model better than a linear model

→ Zero initialization is not successful in classification



$$z_1^{[1]} = x_1 w_{11} + x_2 w_{12} + b$$

$$z_2^{[1]} = x_1 w_{21} + x_2 w_{22} + b$$

$$z_1^{[1]} = z_2^{[1]} \Rightarrow a_1^{[1]} = a_2^{[1]}$$

→ Assigning random values to weight is better than just zero assignment

→ If weights are intialized with very high values → exploding gradient problem

→ If weights are initialized with very less values → vanishing gradient problem

→ The initialization step can be critical to the model's ultimate performance & it requires the right Method.

→ Choosing proper values for initialization is necessary for efficient training.

→ Weights should be random & different from each other

→ To prevent the gradients of the networks activations from vanishing or exploding

⇒ The mean of the activations should be zero ($\mu = 0$)

⇒ The variance of the activation should ~~be~~ stay same across every layer [$Var(a^{[l-1]}) = Var a^{[l]}$]

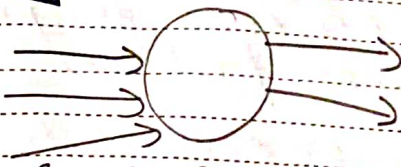
* Different weight initialization techniques

i) He initialization:-

It is useful technique for weight initialization in neural network which uses ReLU activation function

→ For the ReLU activation function, the best weight initialization strategy is to initialize the weight randomly but with the following variance-

$\sigma^2 = \frac{2}{N}$ (N → Nos. of input features)



$\sigma = \sqrt{\frac{2}{N}}$ or $\sqrt{\frac{2}{fan\text{-}in}}$

$W_{ij} \sim N(0, \sigma)$

N → Normal Distribution

ii) Xavier Initialization or Gorat Initialization

→ It works with ~~to~~ tanh activation function

→ It is same as He initialization but in this method 2 is replaced with 1

$\sigma^2 = \frac{1}{N} \Rightarrow \sigma = \sqrt{\frac{1}{N}}$ or

$\sigma = \sqrt{\frac{1}{fan\text{-}in}}$

$W_{ij} \sim N(0, \sigma)$ N → Normal distribution

N → Nos. of input features.

→ The above methods serve as good starting points for initialization & reduces the chances of vanishing & exploding gradients problem.

→ They set the weights neither too much bigger than one nor too much less than 1.

→ They help to avoid slow convergence also ensuring that we do not keep oscillating off the minima.

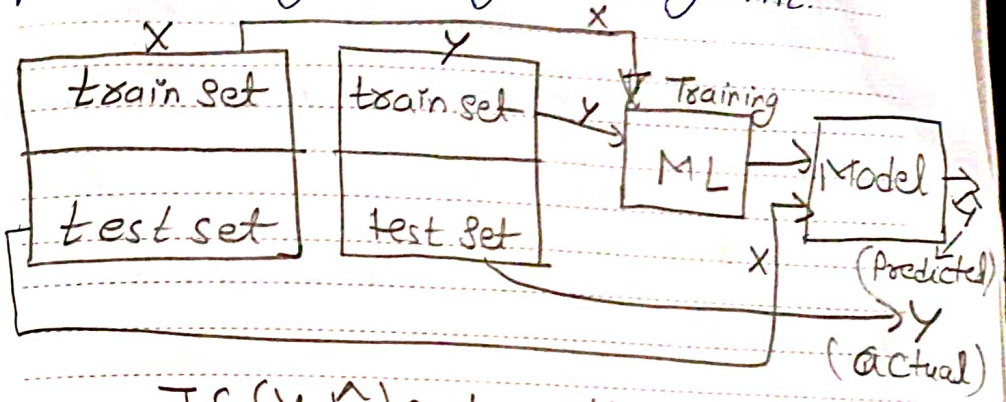
Training & Testing in Machine Learning

→ In ML, we basically try to create a model to predict on the test data

→ Training data is used to fit the model & testing data is used to test the model

→ The models generated are to predict the results unknown which is named as the test set.

→ Dataset is divided into train & test set in order to check accuracies, precision by training & testing on it.



If $(Y - \hat{Y}) \approx$ less then more accurate model

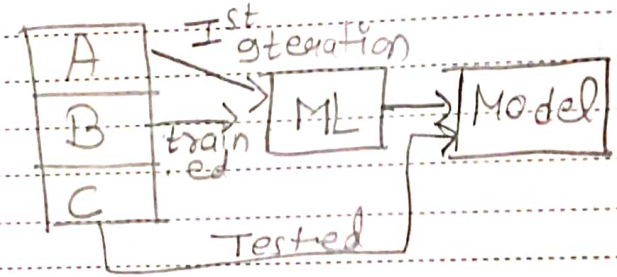
→ In addition to the training & test data, a third set of observations called a validation or hold-out set is required.

→ The validation set is used to tune variables called hyperparameters, which control how the model is learned.

→ It is common to allocate 50% or more data to training set
i) 25% to the test set
ii) 25% to the validation set

→ An algorithm trained on a large collection of noise, irrelevant or incorrect, labeled data will not perform better than an algorithm trained on a small set of data that is more representative of problem in the real world.

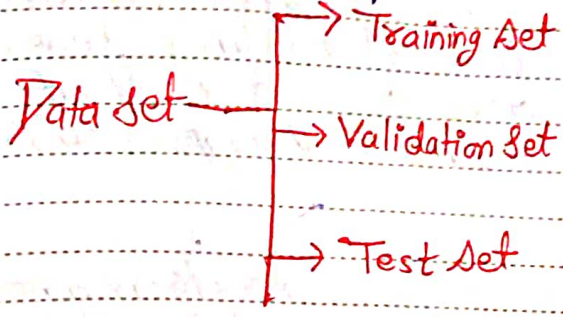
→ Cross validation can be used to train & validate an algorithm on the same data, so that the algorithm is trained & evaluated on all of the data.



II nd iteration
A, C → trained
B → tested

III rd iteration
B, C → trained
A → tested

→ Cross validation provides a more accurate estimate of the model's performance than testing a single partition of data.



i) Training Set:-
A set of examples used for learning of neural network

ii) Validation Set:-
A set of examples used to tune parameters of Neural Network [nos. of hidden layer in Neural network, η → learning rate]

iii) Test Set:-
A set of examples used to assess the performance of Neural Network.

Overfitting

Problem in Machine learning:-

→ The main goal of Machine learning Model is to give correct outputs to sets of input that it has never seen before

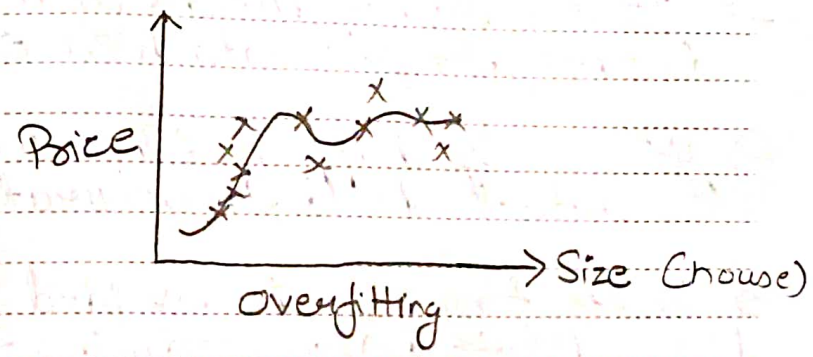
→ The course of poor performance in machine learning is either overfitting or underfitting the data

1) Overfitting in Machine learning

→ It is machine learning problem in which our model doesn't generalize well from our training data to unseen data

→ A model that has learned the noise instead of the signal is considered Overfit because it fits the training dataset but has poor fit with new datasets

→ A Machine Model is said to be overfitted When we train it with a lot of data When a model gets trained with so much of data, it starts learning from the noise & inaccurate data entries in our data set & build an unrealistic model



→ Overfitting is a phenomena which occurs when a model learns the detail & noise in the training data to an extent that it negatively impacts the performance of the model on new data.

→ It can be avoided by using Cross-validation, Early stopping, pruning & Regularization. Dropout etc.

2) Underfitting:-

→ It refers to a model that can neither model the training data nor generalize to new data.

→ Its occurence means that model does not fit the data well enough

→ It usually happens when we have less data to build an accurate model

→ Underfitting can be avoided by using more data & also reducing the features by feature selection technique

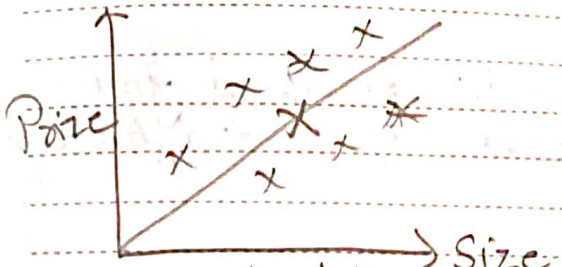
Regularization [to make things regular or acceptable]

→ When a machine learning model, models the training data well but fails to perform well on the testing data i.e. was not able to predict test data

→ It is called as overfitting & this Situation can be dealt with Regularization in Machine learning

→ Preventing Overfitting is necessary to improve the performance of our Machine learning Model.

→ Regularization adds a regularization term to prevent the coefficient to fit so perfectly extra features

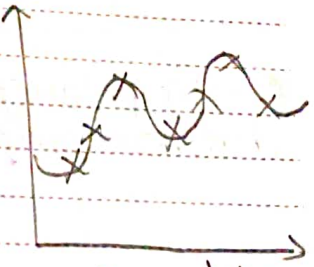


Underfitting

⇒ Train error ↑
Test error ↑

⇒ $\theta + \theta_1 x$

⇒ High bias Underfitting.



Overfitting

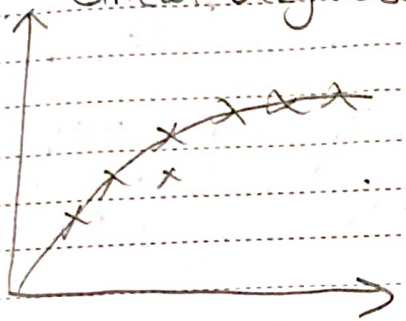
⇒ Train err
Test er

⇒ $\theta + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 - - \theta_7 x^7$

⇒ High Varian

⇒ $J(\theta) \approx 0$ — fail to gener new examples to high fail

Linear Regression



: Fit right

$\theta + \theta_1 x + \theta_2 x^2$

⇒ Best fit polynomial of degree K

→ Regularization is a method for automatically penalizing extra features

To reduce Overfitting

1) Reduce number of features, manually or by model selection algorithm
But disadvantage → some important features thrown out

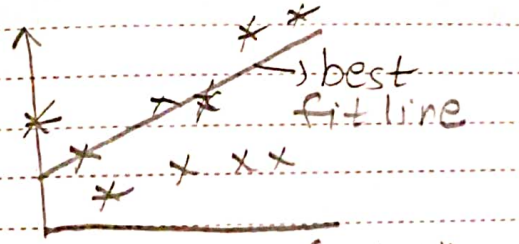
2) Regularization:- keep all the features but reduce magnitude or values of parameters θ_j

→ It works well when we have a lot of features, each of which contributes a bit to predicting y (target output)

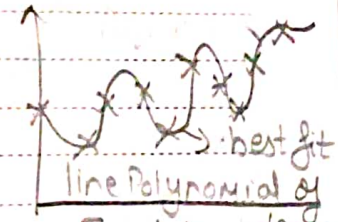
→ Regularization regularizes or shrinks the coefficient estimates towards zero.

→ These are techniques used to reduce error by fitting a function appropriately on the given training set & avoid overfitting.

→ It is a technique used for tuning the machine learning model by adding an additional penalty term in the error function



Simple (underfitting Problem)



Complex

$\theta + \theta_1 x_1 + \theta_2 x^2 + \theta_3 x^3 + \dots \theta_9 x^{19}$

→ The addition term (penalty) controls the excessively fluctuating function such that coefficient don't take extreme values

→ The algorithm on the right (figure) is fitting the noise. In above example, a way to reduce overfitting is then to artificially penalize higher degree polynomials.

→ This ensure that a higher degree polynomial is selected only if it reduces the error significantly compared to simpler model, to overcome the penalty. This is regularization.

✱ Types of Regularization

It is used when the dataset is large

1) L1 regularization or Lasso regularization

It adds a penalty to the error function & the penalty is the sum of the absolute values of weights

For Neural Network

$$\text{Min}\left(\sum_{i=1}^{n}(y_i - w_i x_i)^2 + p\sum_{i=1}^{n}|w_i|\right)$$

(loss function) (Penalty)

where p is the tuning parameter which decides how much we want to penalize the model.

→ Cost function $= \Sigma(y_i - \hat{y}_i)^2 + \alpha\Sigma|\beta_i|$

(Residual Square function)

For Linear Regression

$y_i = \beta_0 + \beta_1 x_1 + \cdots \beta_p x_p$

high $\beta \to 0$

⇒ Lasso stands for Least Absolute Shrinkage & Selection Operator

→ Lasso shrinks the less important features coefficient to zero, thus removing some feature altogether

→ Lasso works well for feature selection in case we have a huge number of features.

→ Lasso adds absolute value of magnitude of coefficients as penalty term to the loss function

→ Cross Validation work well with a small set of feature but for large set of features this technique is beneficial.

→ Lasso regularization makes some features with zero weight that means these features are essentially ignored completely in the model

→ So only a subset of the most important features are left with non-zero weights, it also makes the model easier to interpret

→ As the value of p/α increases, it reduces the value of coefficients, hence avoiding overfitting but after certain value, the model starts loosing important properties of the data (underfitting problem)

→ Therefore, the value of P/α should be carefully selected by using some hyper parameters tuning techniques.

2) L2 Regularization / Ridge Regularization

→ It also adds a penalty to the error function

→ But in L2 Regularization, the penalty is the sum of the squared values of weights

→ $Min\left(\sum_{i=1}^{n}(y_i - w_i x_i)^2 + p\sum_{i=1}^{n}(w_i)^2\right)$

where p is tuning parameter which decides how much we want to penalize the model

→ It adds squared magnitude of coefficient as penalty term to the loss function

$$\text{Cost function} = \sum (y_i - \hat{y}_i)^2 + \alpha \sum B_j^2$$

If $\alpha = 0$ → no penalty – regression without regularization

→ Ridge Regression enforces the B coefficient to be lower but it does not enforce them to be zero. That is will not remove irrelevant features but rather minimize their impact on the trained model

DROPOUT:-

* Methods to avoid overfitting
→ Reduce nos. of features
→ Add more training data
→ Regularization
→ Dropout

P.T.O.

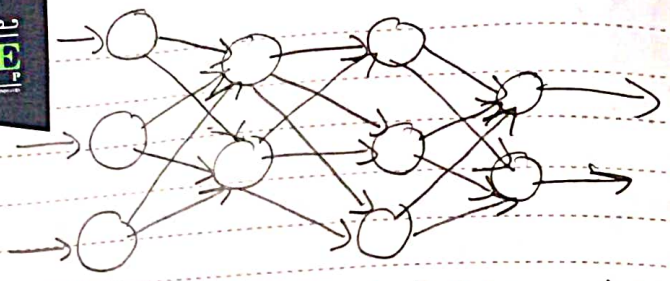


Fig:- Original Neural N/w

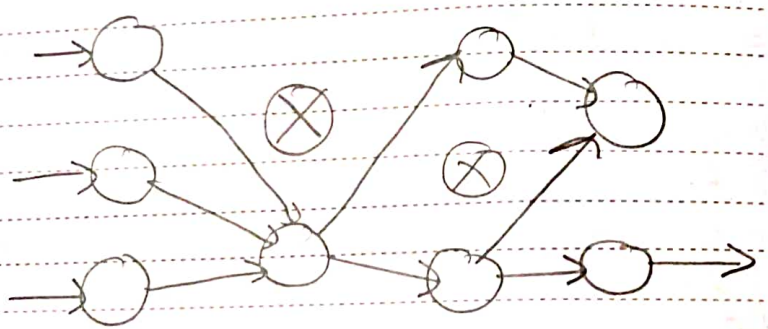


Fig Dropout Neural Network

⇒ Randomly drop units (neurons) along with their connections from the neural network during training

→ Dropout is a technique in which some nodes of the network are temporarily deactivated

→ This technique is applied in the phase to reduce overfitting effects

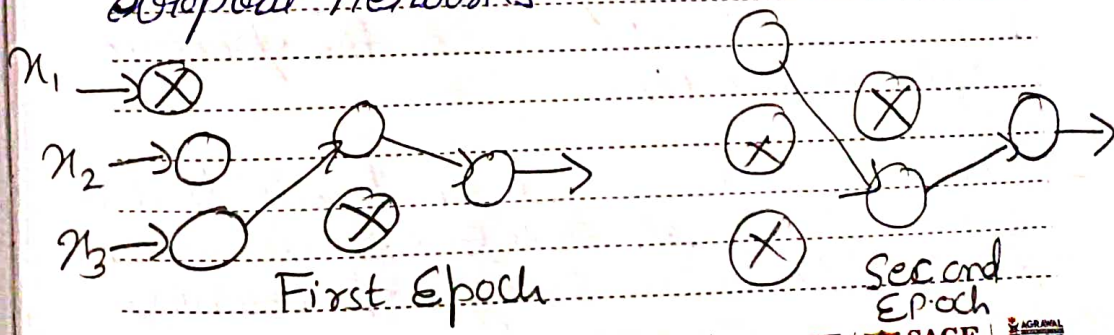
→ The basic idea behind dropout neural network to dropout nodes so that the network can concentrate on other features

→ In dropout approach we randomly choose a certain number of nodes from the input & the hidden layers which remain active & turn off the other nodes of these layers.

→ After this we can train a part of our learn set with this network.

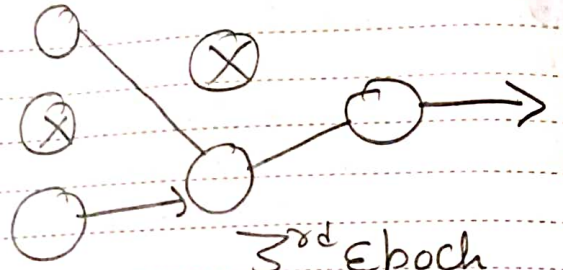
→ The next step consists in activating all the nodes again & randomly chose other nodes

→ It is also possible to train the whole training set with the randomly created dropout networks



First Epoch

Second Epoch



3rd Epoch

→ At each training stage, individual nodes are either dropped out of the network with probability (1-p). Or kept with probability p, so that a reduced network is left, incoming & outgoing edges to a dropped out node are also removed (p=0.5 works)

→ Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.

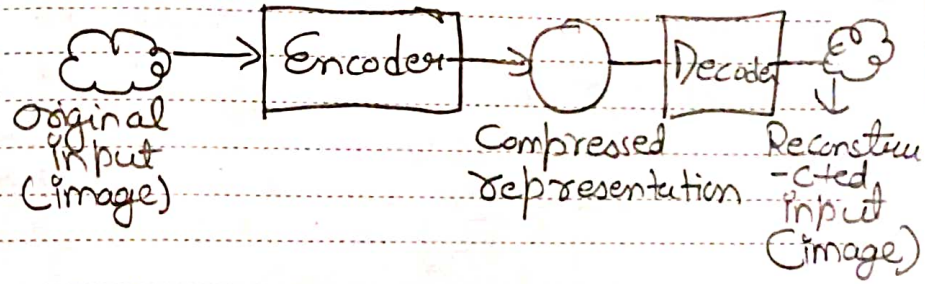
→ Using dropout approach the network become less sensitive to the specific weights of neurons.

→ This is turn results in a network that is ~~table~~ capable of better generalization & is less likely to overfit the training

data

→ Dropout roughly doubles the number of iteration required to converge. However training time for each epoch is less.

Auto Encoders



→ An autoencoder is a neural network that tries to reconstruct its input

→ These are unsupervised deep learning neural network algorithms that reduce the number of dimensions in the data to encode it

→ Once the data has been encoded through the algorithm it is then decoded on the other side

→ The system is completed when the data on both sides of the encoding matches.

→ An autoencoder is a neural network that learns to copy its input to its output.

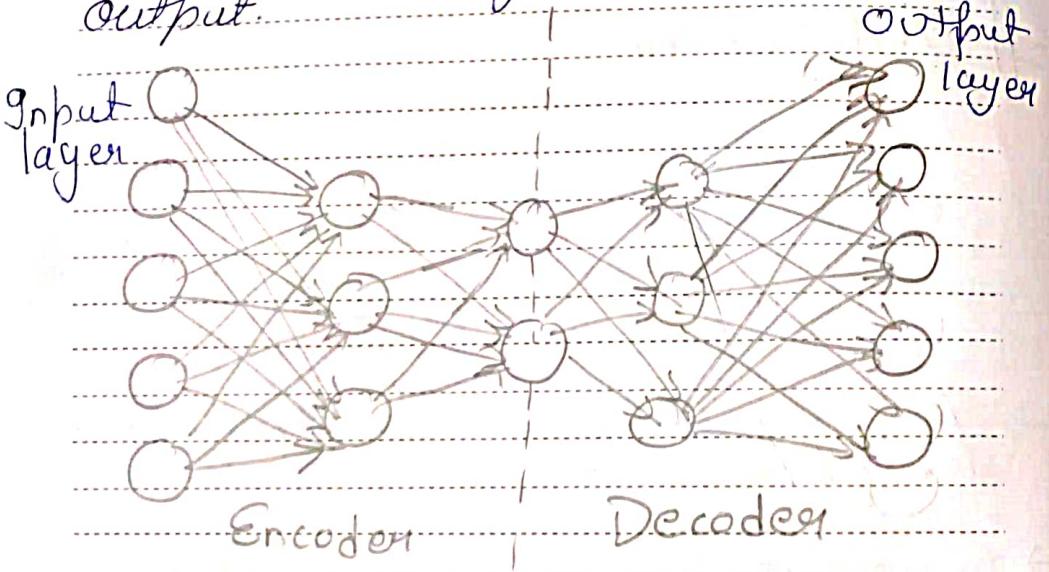


Fig:- Structure of an Autoencoder

→ Let we have only a set of unlabelled training examples $\{x^{(1)}, x^{(2)}, x^{(3)} \ldots\}$

→ An autoencoder neural network is an unsupervised learning algorithm that applies back propagation, setting the target values to be equal to the inputs. i.e. it uses.

$$y^{(i)} = x^{(i)}$$

→ The autoencoder tries to learn a function

$$f_{w,b}(x) \approx x$$

→ These consist of an input layer, an output layer, one or more hidden layer connecting them where the output layer has the same number of neurons as the input layer

→ An auto encoder consists of two parts:-

1) Encoder:-
It is a network that takes the input & output a feature map/vector. These feature vector hold the information, that represents the input

2) Decoder:-
It is a network usually the same network structure as encoder but in

opposite orientation, it takes the feature vector from the encoder & gives the best closet match to the actual input or intended output

→ The encoders are trained with the decoders.
i) The loss function is calculated between actual & reconstructed input.
ii) The optimizer try to train both encoder & decoder to lower this reconstruction loss.
iii) Finally the error is back-propagated through the system.

→ The autoencoder updates the algorithm & tries again forward & back until solution is found. Once the variables on each side match, the cycle stops.

→ We will design a neural network architecture such that we impose a bottleneck in the network which forces a compressed knowledge representation of the original input

Forex

(1) (1,0,0,1,0) → (X, Y) → (1,0,0,1,0)
5 features → 2 features → 5 features
Input layer → Hidden layer → Output layer

(2) 10 x 10 = 100 neurons → 50 neurons → 100 neurons

✱ Application of Autoencoder

1) Data Storage:-
The encoding processes are able to compress down large quantities of data. So it has big benefits for data storage

2) Feature Detection:-
The process used to encode the data identifies features of the data that can be used to identify it.
This list of features is used in multiple systems to understand the data like in convolutional neural network

which is used for feature detection in images.

3) Recommendation Systems:-
These are the systems that identify films or TV series you are likely to enjoy on your favorite streaming services

Batch Normalization:-

→ Normalization: In a dataset, all the features (columns) may not be in same range
Ex→ Price of house Age of house
5000000 25 yrs

→ Neural network takes lots of time to train for these kinds of dataset.

→ In simpler machine learning algorithm like linear regression, the input is normalized before training to make them into single distribution

→ Normalization is to convert the distribution of all inputs to have mean=0 & Standard deviation=1. So most of the values lie between -1 & 1

Batch:-
As we know that ML algorithm require training data set (group of examples) based on which it can learn.

→ Data needs to be in numerical form (vector) so algorithm can understand it

	feature 1	feature 2	feature n	Vector Y
Sample 1:	[number,	number	number]	[expected result]
Sample 2:	[number,	number	number]	
Sample m:	[number,	number,	number]	

X Matrix

Stochastic Gradient Descent
When only one sample per iteration is used to update model parameters this is called as Stochastic Gradient Descent

Mini Batch Gradient Descent:-
When more than one simple per

iteration but less than all available samples are used to update model parameters this technique is called as Mini Batch gradient descent

Batch Gradient descent:-
When all available samples are used to update model paramters this technique is called Batch Gradient descent

Batch:-
It is a number of samples fed to Neural network during single iteration

Batch Normalization:-

→ It is a technique for training very deep neural networks that standardizes the input to a layer for each mini batch.

→ This has the effect of stabilizing the learning ~~problem~~ process & dramatically reducing the number of training epochs required to train deep networks.

→ As we know that neural networks learn the problem using Back propagation algorithm.

→ During back propagation of errors to the weights & biases each layer is trying to correct itself for the error made up during the forward propagation.

→ But every single layer acts separately to convert itself for the error made up

→ For ex:- The 2nd layer adjusts its weights & biases to correct for the output but due to this ~~ready~~ readjustments, the output of 2nd layer (input of 3rd layer) is changed for same initial input

→ So the 3rd layer has to learn from scratch to produce the correct outputs for the same data.

→ This creates a problem because 3rd layer learns after 2nd finished, 4th starts learning after 3rd layer.

→ If neural network 1000 layers deep, then it would mostly takes lots of time to train.

→ This is happen because due to changes in weights of previous layers, the distribution of input values for current layer changes & forcing it to learn from new input distribution.

→ To overcome from this problem, batch normalization adds normalization layer between each layers

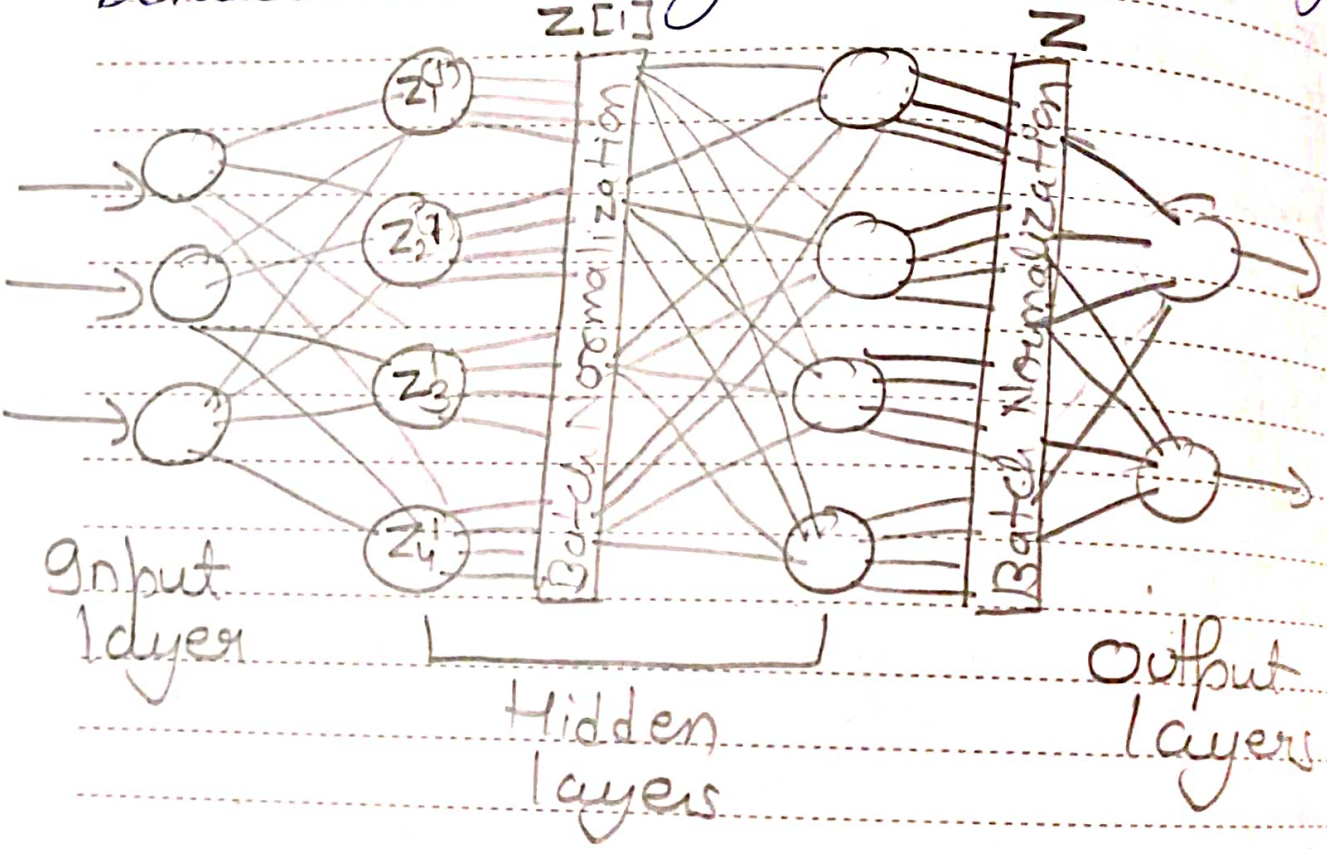


Fig → Batch Normalization

→ Due to this normalization layers between each fully connected layers, the range of input distribution of each layer

stays the same, no matter the changes in previous layer.

$$Z^{n}_{norm} = \frac{Z^{n} - Mean(batch)}{\sqrt{Variance(batch)}}$$

Adding new training parameter

$$\hat{Z}^{n} = \gamma^{n} Z^{n}_{norm} + \beta^{n}$$

$$a^{n}_{norm} = g(\hat{Z}^{n})$$

Randomly Sampled batch

↓

Input layer → Linear → Batch Normalization

Hidden layer 1 ⋮

Activation ← $Ca'_{norm} = g'(Z^{n})$

⋮

Hidden layer n: Activation → Linear → Batch Normalization

Output layer ←

→ Normalization brings all the inputs centered around 0. So there is not much change in each layer input. So there is no need

for layers to wait for the previous layers to learn, hence it fastens up the training of networks

→ If Batch normalization is applied before applying activation function (like Sigmoid & tanh, ReLU) then same transformation has to be done to move the distribution away from zero

→ A scaling factor γ & shifting factor β are used to do this

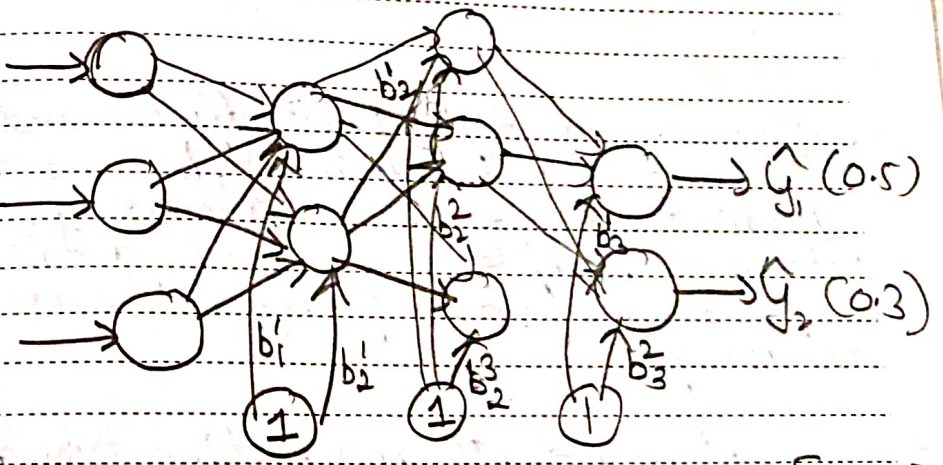
→ As training progresses, along with weights, γ & β also learn through backpropagation so as to improve accuracy of neural network

→ If Batch Normalization is applied after activation function then set γ=1 & β=0

Momentum:

→ It is a simple technique that often improves both training speed & accuracy of neural network

→ Training a neural network is the process of finding values for the weights & biases so that for a set of input values, the computed output values closely match the known target values.



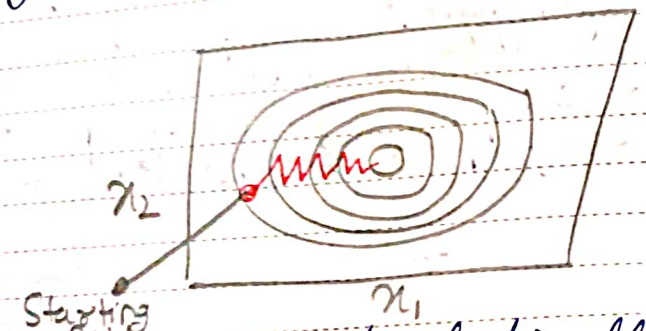
Actual [1 0]
Predicted [0.5, 0.3]

Nos. of weights = 3x2 + 2x3 + 3x2
= 6 + 6 + 6 = 18

Nos. of biases = 2 + 3 + 2 = 7

→ Stochastic gradient descent optimizer.

The figure shows the each step of SGD



→ The lines not point directly towards the centre (global minimum), because the gradient estimates in SGD are noisy, due to small sample size. So the gradient steps are noisy even if they are correct on average

→ So SGD wastes too much time swinging back & forth along the direction in parallel with x_2-axis which advancing too slowly along the direction of the x_1-axis

→ Therefore, total nos. of parameters $= 18 + 7 = 25$

→ Back propagation algorithm is used to train most of the neural networks

→ So update all the weights & biases after processing a single training item, we use

$$\Delta w_{ij} = \eta * \frac{\partial E}{\partial w_{ij}}$$

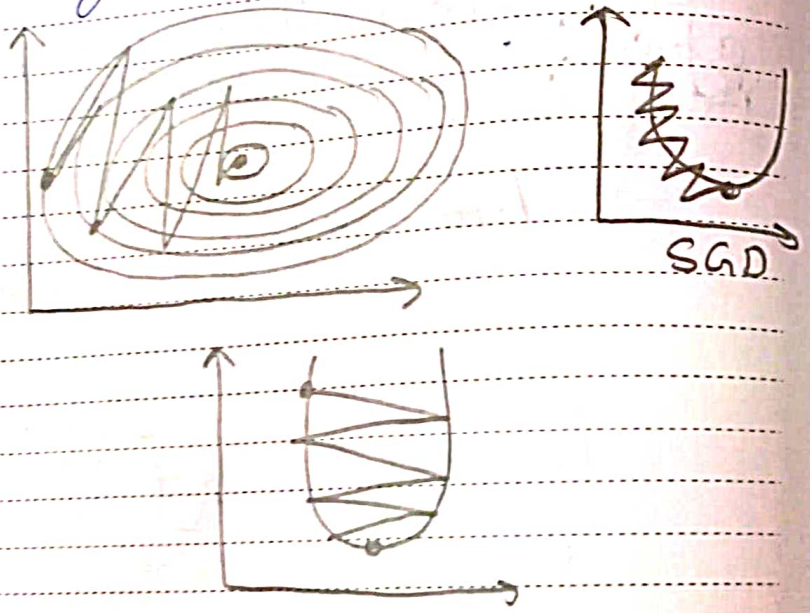
(η: learning rate; Δw_{ij}: weight increment connection node i to j; $\frac{\partial E}{\partial w_{ij}}$: weight gradient)

$$w_{ij}' = w_{ij} - \underbrace{\lambda * \frac{\partial E}{\partial w_{ij}}}_{\Delta w_{ij}}$$

→ Gradient descent is mostly used optimization algorithm in machine learning

→ But one of the problem with gradient descent is that it oscillates too much on vertical axis & this oscillations slows down the convergence to global minimum

→ We want less oscillation on vertical axis & faster movement towards the global minimum, to make further faster learning & convergence to the global minimum



→ Gradient descent with Momentum is an optimizer algorithm which helps to achieve this.

Now, learning rate $0 \le \beta \le 1$

$$\Delta w_{ij} = \eta * \frac{\partial E}{\partial w_{ij}} + \left(\beta * \Delta w_{ij}^{t-1}\right)$$

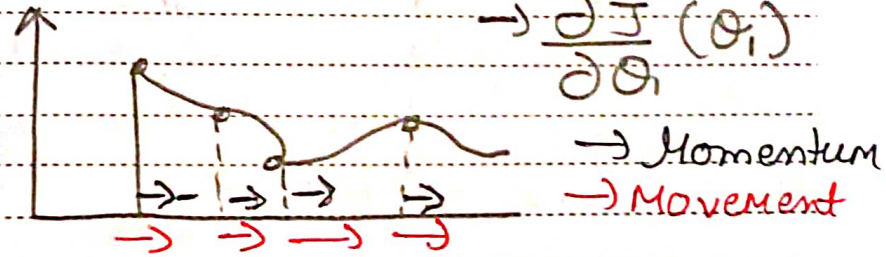
Weight increment | Weight | Momentum factor | Weight Increment (Previous update)

Gradient

→ Here another hyperparameter β is used

→ The derivative represents acceleration whereas momentum terms are velocity

→ By adding $\beta * \Delta w_{ij}^{t-1}$, it smooth out gradient descent & the vertical oscillation will average out to zero, because of this gradient descent will to converge quickly as it is not taking every step independent of previous ones. It is rather taking previous steps into account to gain momentum for them



$$\text{Movement} = -\frac{\partial J(\theta_1)}{\partial \theta_1} + \text{Momentum}$$

$$w_{new} = w_{old} - \eta \frac{\partial E}{\partial w_{old}} \quad \{\text{Without Momentum}\}$$

$$w_{new_t} = w_{old_t} - \left\{\eta \frac{\partial E}{\partial w_{old_t}} + \beta w_{t-1}\right\} \quad [\text{With Momentum}]$$

t-1 → t-2 } Exponential
t-2 → t-3 } increases

Generally $\beta = 0.9$ { Mostly used value } for reduce noise.

→ The idea of Momentum based optimizer is to remember the previous gradients from recent optimization steps & to use them to help to do a better job of choosing the direction to move next, where the objective cost function can be reduced fastest on a given example.

Tuning Hyper Parameters

Parameters

i) Model Parameters	ii) Model hyper parameters
Ex→ Weight, bias etc, weights in ANN, Support vectors in SVM, Coefficients in Linear & Logistic regression	Ex→ Learning rate for training a Neural Network, → C & Sigma for SVM → K in K-nearest neighbour
→ They can be directly trained from data	→ They cannot be directly trained from the data
	→ Its value cannot be estimated from data
→ Model parameters are learned during training when we optimize a loss function using method like gradient descent.	→ There is no way Calculate or Update hyperparameter to reduce the loss in order to find the optimal model architecture.

Model parameter	Model hyper parameters
→ It specify how to transform the input data into the desired output	→ It define an Machine learning Model is actually structured.
→	⇒ We generally perform experiments to find out which value works best.
→ These are the properties of training data that will learn on its own during training by the Machine learning Model.	→ It is a parameter whose value is used to control the learning process. → Used in processes to keep estimate Model parameters
→ Learned during training program	→ They are usually fixed before the actual training process begins

Hyperparameter Tuning

→ Hyperparameter tuning is a process of finding the hyperparameter values of a learning algorithm that produce the best model.

→ Hyperparameter tuning → Best hyperparameters setting
↓
Best Model

→ It makes the process of determining the best hyperparameter setting easier & less tedious.

→ The outcome of hyperparameter tuning is the best hyperparameter setting & the outcome of model training is the best model parameter setting.

→ Machine learning Model can have many hyperparameters & finding the best Combination of parameters can be treated as a search problem.

→ We can not know the best value for model hyperparameter on a given problem.

→ We can search the best value by trial & error.

Hyperparameter tuning Methods

1) Grid Search:-

→ In this approach, machine learning model is evaluated for a range of hyperparameter values.

→ It is called as Grid search because it searches for best set of hyperparameters from a grid of hyperparameter values

→ It brute force all combination

→ It is an exhaustive searching through a manually specified subset of the hyper parameter space of a learning algorithm

→ With this technique, we simply build a model for each possible combination of all of the hyperparameter values provided evaluating each model & selecting architecture which produces the best result.

Ex Machine learning Model M

2 hyperparameters

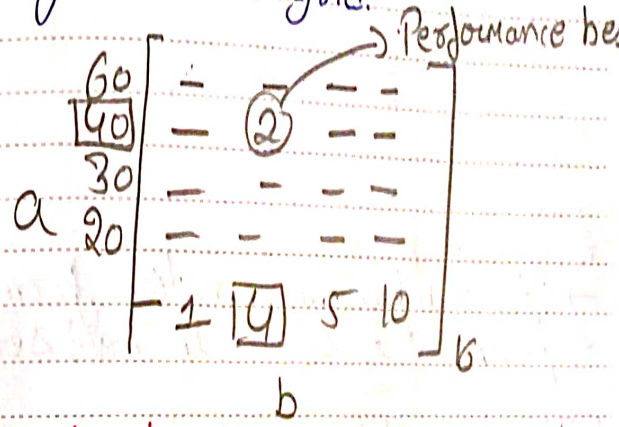
a = [20, 60, 30, 40] [20, 30, 40, 60]

b = [1, 4, 10, 5] [1, 4, 5, 10]

defined grid

→ Grid search technique will construct many versions of M with all possible combinations of (a, b) values that you defined in the first place.

→ This range of hyperparameter values is referred as grid.



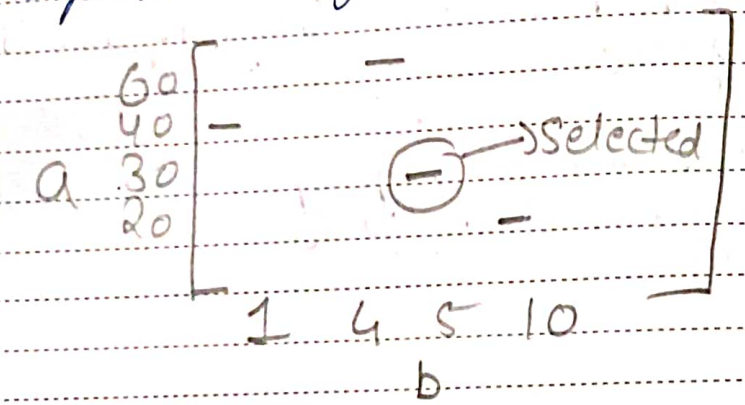
Disadvantage

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

ii) Random Search

→ Instead of searching over the entire grid, random search only evaluates a random sample of points on the grid.

→ This makes random search a lot cheaper than grid search.



Drawback

→ It doesn't use information from prior experiment to select the next set

→ It is very difficult to predict the next set for experiment

Softmax Activation Function