Teacher's Sign/Remarks

11/7/22

Artificial Neural Network (ANN)

Introduction :-

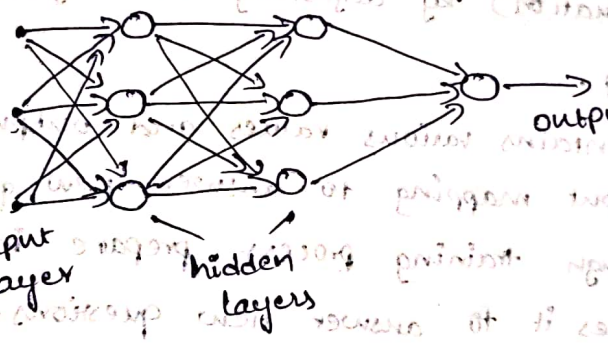
1. Artificial Intelligence
2. Machine Learning
   * Sub domain of AI
   * data + Algorithms

Ex: Loan approval systems.

Drawbacks :

* Cannot handle huge amounts of data.
  (Performance degrades when data is huge)
* Cannot answer the questions or tasks to which it is not trained.

3. Deep learning
   * Sub domain of ML
   * Artificial Neural Networks are building blocks (Artificial neurons)



* Advantages :-
  * Can handle huge amounts of data.
  * Can handle new situation for which it haven't trained.
* Artificial Neural Networks :
  * Definition :- It is a massive distributed parallel processor.
  * History :-

1943 - Mcculloch, pitt's developed first ANN model. (Artificial Neuron model).

1958 - Modified version of the above model is created by Rosenblatt and is named as perceptron

model.

1960's - Proposed modification for his model by introducing the back propagation algorithm.

* Benifits :

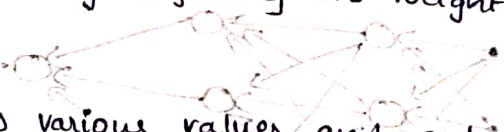
(1) Non-linearity.

- ANN has non-linear behaviour with the help of activation functions.
- It solves small problems using linear behaviour and also large or complex problems using non-linear behaviour.

(2) Adaptive.

- In ANN inputs have weights.
- Whenever ANN sees a new environment, then it adjusts inputs weights to handle it (or) answer to input.
- It can adopt new environment (or) it can work in new environment (new situation) by adjusting its weights.

(3) Input-output mapping.



- If training data contains various values and output, then ANN makes input-output mapping to answer new questions.
- ANN by going through training process prepare input output mapping and uses it to answer new questions.

(4) Evidential Response :- Along with o/p ANN will also produce evedential response i.e. at what probability, the o/p is produced.

(5) Contextual Information :- Knowledge is represented by the structure of ANN & state of ANN.

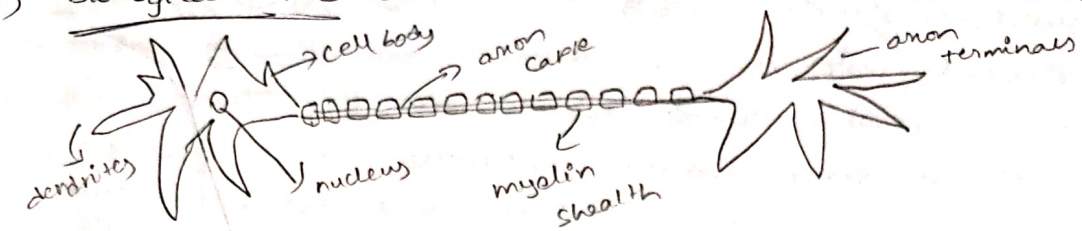
(6) uniformness :- Every neuron is equally capable of processing data.

(7) VLSI compatibility :- As ANN consists of many processing elements (neurons) is comparible with VLSI.

8.) Fault tolerant :- Failure of one of the neuron won't affects the entire ANN.

9.) Neuro Biological Analogy :- Neurons are analogous to the neurons in the brain.

10.) Biological Neuron :- 10 neurons in human brain



Biological Neuron consists of

1.) dendrites act as i/p units providers to neurons.

2.) cell body
Cell nucleus - performs aggregration of electrical impulses reaeved the different neuron.

3.) axon cable

4.) axon terminals

It also performs thresholding operation. Aggregated sum is called induced local field value it compares with threshold value.

* If induced local field values > threshold value then the info is transferred from one neuron to another neuron
if $IFV < TV \Rightarrow$ stays idle.

3.) axon cable

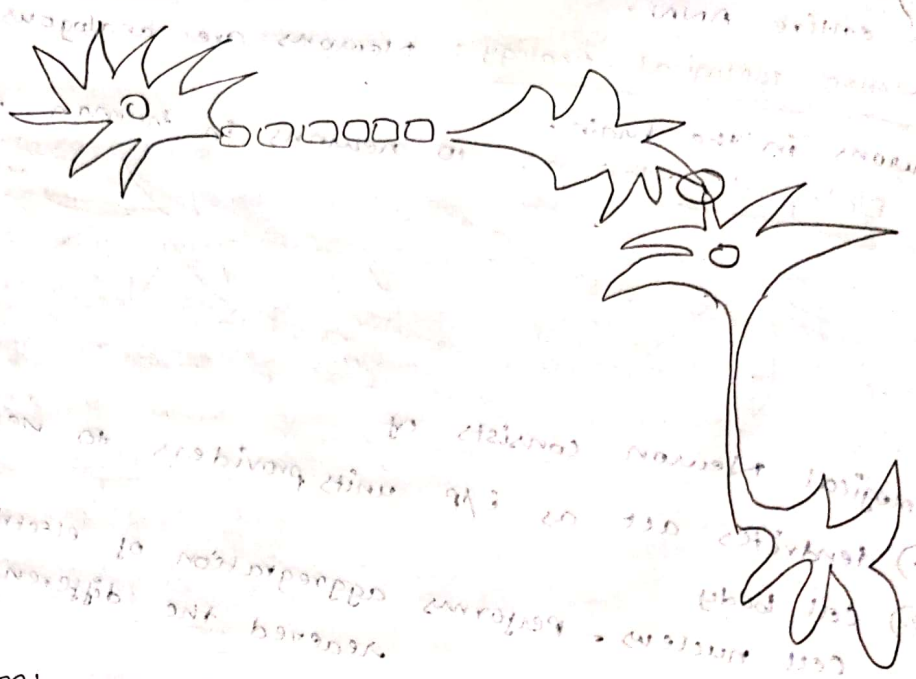
i) Sheduled with myelin sheath.

ii) If thickness of myelin sheath is more than very less energy loss in electrical signal otherwise more energy loss.

4.) axon terminals :

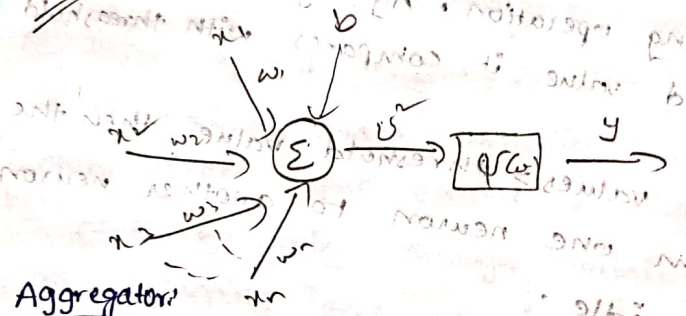
Continued on next page

ii) these are the output units.



Artificial neuron:

14/9



Aggregator:

$$v = \sum_{i=1}^{n} x_i w_i + b.$$

* Activation function:
  - mathematical function.
  - decides whether to activate (or) deactivate the neuron based on induced local field value i.e., v.

Benifits:

1) Activation function add non-linearing to the artificial neuron.
2) Activation function makes neuron to handle complex problems.

3) It also produces evidences in the form of probability.
4) It makes neuron to perform multiclass classification
5) Map output to certain range of values along with evidence.

* If there is no activation function, neuron will become linear unit, linear unit can perform binary classification.
* If there is activation function, it gives non-linearity to the neuron. It can handle complex tasks.
* If there is no activation function there is no change in input and output.
* sigmoid value of activation function produces probability value.

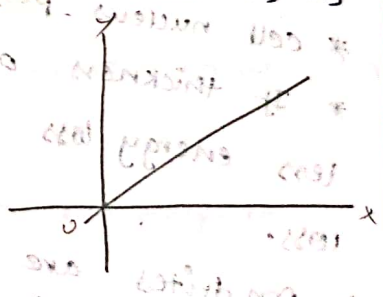
Different types of Activation functions :-

* Based on their classification and behaviour it was divided into 3 types.
  1) linear activation function.
  2) Binary threshold activation function.
  3) Non-linear activation function.

1) linear activation function: Activation function is a mathematical function.

* Linear activation function, it produces output same as its input.

$f(v) = v$. (Mathematical notation).



* Range of the output is $-\infty$ to $+\infty$.

Advantages:

* L.A.F is suitable for performing simple binary classification only.

Drawbacks:

* It is not suitable for error correction process as derivation is a constant.

$f'(v) = 1$, where 1 is constant.

Artificial neuron:

* It is a mathematical model of a biological neuron.
* It is inspired by biological neuron in many ways.

v - induced local field
(Activation state) (transfer signals).

$v \geq$ threshold $\Rightarrow y = 1$
$v <$ threshold $\Rightarrow y = 0$
(sit idle)
(inactive state)

* $x_1, x_2, \ldots, x_n$ is input to neuron.
* Input is weighted $w_1, w_2, \ldots, w_n$ (multiplicative factor).
* b - bias (resting potential)
* v is induced local field.

$$v = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n + b.$$
$$= \sum_{i=1} x_i w_i + b.$$

* y is output

$y = \varphi(v)$ where φ is activation function.
$y = 1$ if $v \geq$ threshold.
$y = 0$ if $v <$ threshold.

Biological Neuron:

* Dendrites acts as input providers to neurons.
* cell nucleus performs aggregation, thresholding.
* If thickness of myelin sheath is more than very less energy loss in electrical signal otherwise more energy loss.
* Dendrites are input units and axon terminates are output units of neuron.
* The connecting point of two neurons is called the synoptic structure.
* one neuron axon terminal is connected to dendrites of other neuron for transfer of signals. There is some gap between axon terminal of first neuron and dendrite of 2nd neuron.

* This gap is called synoptic structure.

15/9

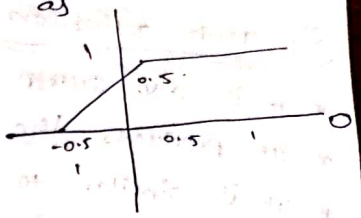
* Piece wise linear activation function :

* It allow us to have a neuron.

* It has real value boundaries.

* In mathematical notation it is defined as

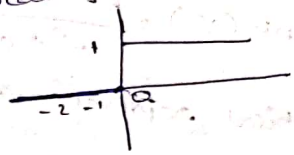
$$f(v) = \begin{cases} 1 & v \geq +1/2 \\ v & -1/2 < v < +1/2 \\ 0 & v \leq -1/2 \end{cases}$$



* Act Binary threshold :

Binary threshold is a A.F it produces output as 1 (or) 0 based on input value (or) induced local field value.

$$f(v) = \begin{cases} 1 & v \geq 0 \\ 0 & v < 0. \end{cases}$$



* It is not suitable for using in Back propagation.

* Non-linear Activation function :

* It introduced and provide artificial neuron with non-linear behaviour (or) non-linear functionality.

① Sigmoid / logistic activation function :

* It is a probabilistic activation function.

* It produces the output in the range 0 to 1 only.

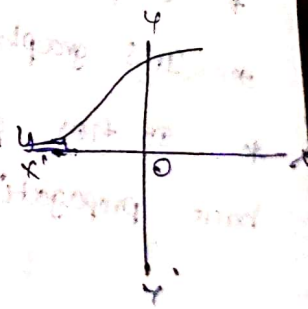
* It is most widely used activation function.

* we can use this activation function in hidden layers (and) output layers.

* When we pass an input to this function it computes Probability within the range 0 to 1.

* Mathematical notation for sigmoid activation function is.

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



* The graph for sigmoid / logistic activation function

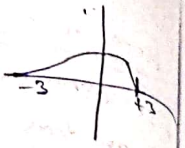
* these can be used in when we have a derivation interms of input. i

Adv

* And it has a gradient sigmoid (v) * (1 - sigmoid (v)), and it is used in the back propagation.

* ~~Drawbacks~~ * Drawbacks

* It suffers with the vanishing gradient problem.

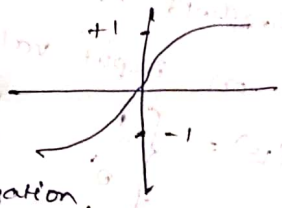


(2.) Tanh activation function :-

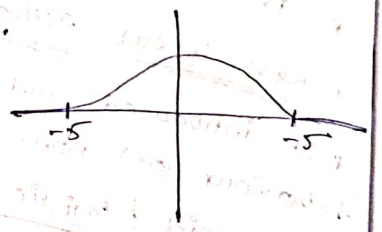
* It is also called as hyperbolic tanh activation function.
* It produces the output within a range -1 to 1.
* It is similar to sigmoid activation function.
* It is zero centered.
* Mathematical function for tanh activation function is

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

* Graphical representation of tanh is



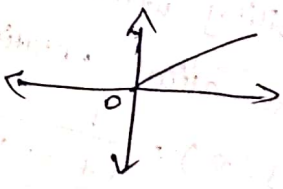
* has gradient 1 - $\tanh^2(v)$.
* And it is suitable for back propagation.
* Drawbacks :
  - It is vanishing gradient problem.
* In this below -5 and above +5 the gradient is 0.



* Relu activation function :-
* It is rectified linear unit.
* It is most widely used in Hidden layers of Artificial Neural Networks.
* It's mathematical function is

$$\text{relu}(v) = \max(0, v)$$

* Output ranges from 0 to +∞.
* It's graphical representation is



* It has gradient hence used in back propagation for error correction.

Drawbacks :-

* since, i
deactivatio
where all
* dying

(4.) Leaky

* similar
is -ve.
* It's math
LRelu (v)
-a is c
a =

* It avoids
multiplied
* output ra
* It is use
* Advantages :
* It doesn't
* It is si
* It produce
* No dying

(5.) Softmax

* It is used
* It is most
* Similar to
* It's mat
Softmax

* It compu
to
0 and 1.
* It has

Drawbacks :-

* since, i/p is -ve, when output is 0, neuron is in deactivation state so can't be used in solving a problem. where all neuron's should be in Activation state.
* dying relu problem.

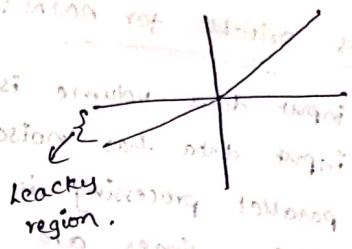
④ Leacky Relu Activation function :-

* similar to Relu but it produces an o/p when i/p is -ve.
* It's mathematical function is

LRelu (v) = max (av, v)

-a is constant.

$a = 0.01$



* It avoids dying relu problem using a variable a which is multiplied to input and max of av & i/p is considered as o/p.
* output range is $(-\infty, \infty)$
* It is used in hidden layers.
* Advantages :-
  * It doesn't activate the neuron.
  * It is simple activation function.
  * It produces output for -ve values too.
  * No dying relu Problem.

⑤ Softmax Activation function :-

* It is used at output layer.
* It is most widely used in A.F.
* similar to sigmoid AF and called as sigmoid of sigmoids.
* It's mathematical function is

$$\text{softmax}(v) = \frac{\text{sigmoid}_i(v)}{\sum_{i=1}^{K} \text{sigmoid}_j(v)}$$

* It computes relative probability so it output range is between 0 to 1.
* It has gradient hence used in back propagation for the error correction.

* Which function to use ?

* Simple problems Applications at all levels - sigmoidal (i.e., for binary classification),
* If all neurons should be in active state - Leacky relu - use at hidden layers.
* If deactivation of neuron's is needed use relu.
* Use softmax for output layer - (complex applications) multi-class classification.

* Applications suitable for ANN :- (drawbacks of ML),

1.) when input data volume is very large.
2.) when input data has noise.
3.) when parallel processing is require for applications.
4.) long training times are accepted.
5.) Suitable for image processing applications, objects identification, language processing, voice synthesizing.

* Which function to use ?

* Simple problems Applications at all levels : sigmoidal (i.e., for binary classification)

* If all neurons should be in active state - Leacky relu - use at hidden layers.

* If deactivation of neuron's is needed use relu.

* use softmax for output layer - (complex applications) multi-class classification

* Applications suitable for ANN :- (drawbacks of ML),

1.) when input data volume is very large.
2.) when input data has noise.
3.) when parallel processing is require for applications.
4.) long training times are accepted.
5.) Suitable for image processing applications, objects identification, language processing, voice synthesizing.

20/7/22

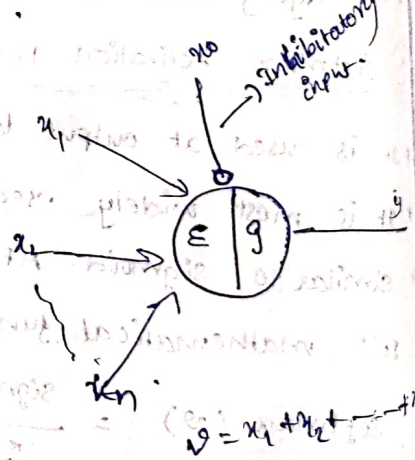
* Artificial Neuron Models :-

* there are 3 different types.

1.) Mclulloch and pits Neuron model.
2.) Perception model.
3.) Adaline Model.

1.) Mclulloch and pits Neuron model :-

* this is the first Artificial Neuron Model.
* this model is invented in 1943.
* Inspired by biological Neuron
* g - thresholding element, Σ = Aggregator.
* Activation function in mclulloch and pits Neuron model is threshold Activation function.



$y = g(v)$.

$y = 1 \quad v \geq 0$

$y = 0 \quad v < 0$

Where, θ is threshold value.

* It accepts only binary inputs only and produces binary outputs.
* It is not suitable for real value inputs.
* It will accepts 2 types of inputs.
  1.) Excited Inputs 2.) Inhibitatory inputs.

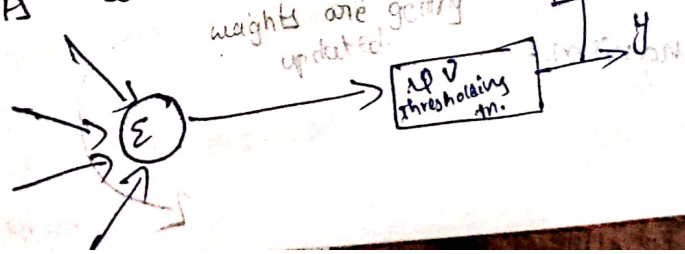
2.) * If it is active then make neuron to produce output as 0 respective of value of excitatory inputs.
* If it is turned off then neuron produce 0 and computes output by passing it to g.

1.) * these are normal inputs which can make neuron to go to excitatory state.
* excitatory inpws are positively weighted.
* Inhibitatory inputs are negatively weighted.

* Uses :-
* It is used for binary classification only.

Drawbacks :-
* there is no learning mechanism.
* It is not suitable for complex problems like multi-class classification.

2.) Perceptron model :-
* Rossen ball proposed perceptron.
* It was invented in 1958.
* perceptron requires huge competition efficiency.
* Electrical circuits Rossumball prepared perception model.
* He invented bias factor.
* He invented learning mechanism for neuron.
* the perceptron accepts the real values of inputs.
* with this changes the rossen ball invented perceptron.
* Inputs are weightened
* weights are assaigned with random values.



* $v = x_1 w_1 + x_2 w_2 + \dots + x_n w_n + b$.

* In mathematical notation,

$$v = \sum_{i=1}^{n} x_i w_i + b$$

* the output 'y' communicated as a function of v.

$$y = \varphi(v)$$

* There are 2 types of perceptron.

1.) Single layer perceptron.
2.) Multi- layer perceptron.

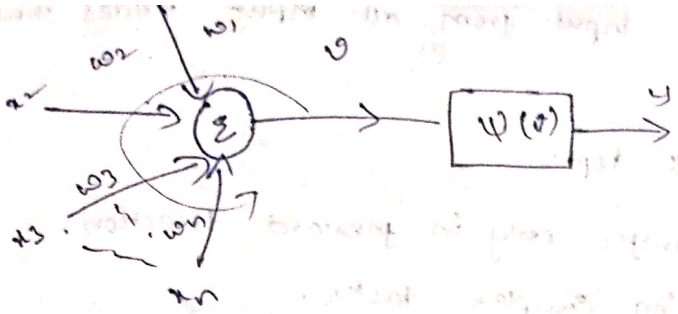
1.) In this there are only one output layer.

* Advantages:-

* In this it contains real- valued inputs.
* It contains multi- class classification.
* Learning mechanism will be used.

* Adaline model:-

* It is also known as widrow and half model.
* It was developed in stanford university in 1960.
* Adaline - Adaptive linear element.
* It is the model and used to predict the transmission line (means) next incoming bit.
* Madaline, It is used to reduce the voice breaks.
* Madaline is induced.
* these two models are prepared by the electrical circuits.
* It is accroming the adaptive linear element.
* It just look like perceptron but there is a change in the learning mechanism.



* Among these three perceptron is very useful and the

1.) perceptron has a single layer of artificial neurons.

2.) perceptron has multi-layers of artificial neurons containing multi-layers.

* Network Architectures (or) Network models :-

* Network is a interconnected structure of processing elements.

* In ANN, Processing element is artificial neuron.

* ANN is a interconnected structure of artificial neurons.

* Neurons are arranged in layers.

* There are 3 types of layers.

1.) Input layer

* It contains only input nodes.

2.) Hidden layer.

* It contains artificial neurons.

3.) output layer.

* It contains artificial neurons.

* There are 3 different architectural models.

1.) Single layer feed forward network.

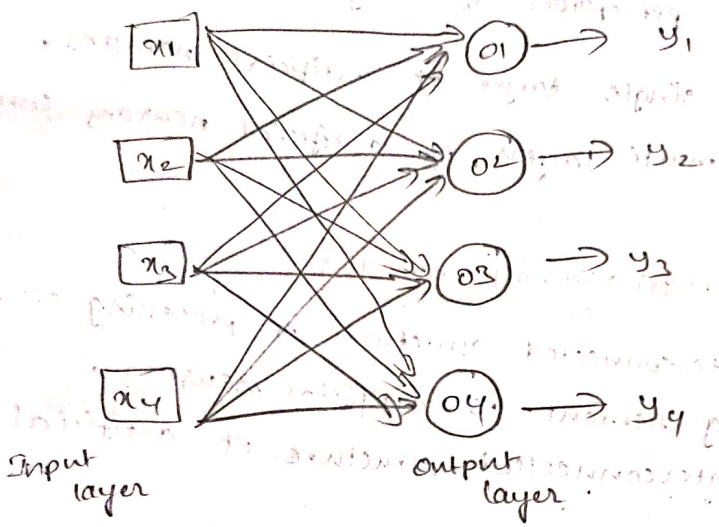
2.) Multi-layer feed forward network.

3.) Recurrent network.

* Single layer feed forward network :-

* In this architectural model, ANN consists of 1 output layer.

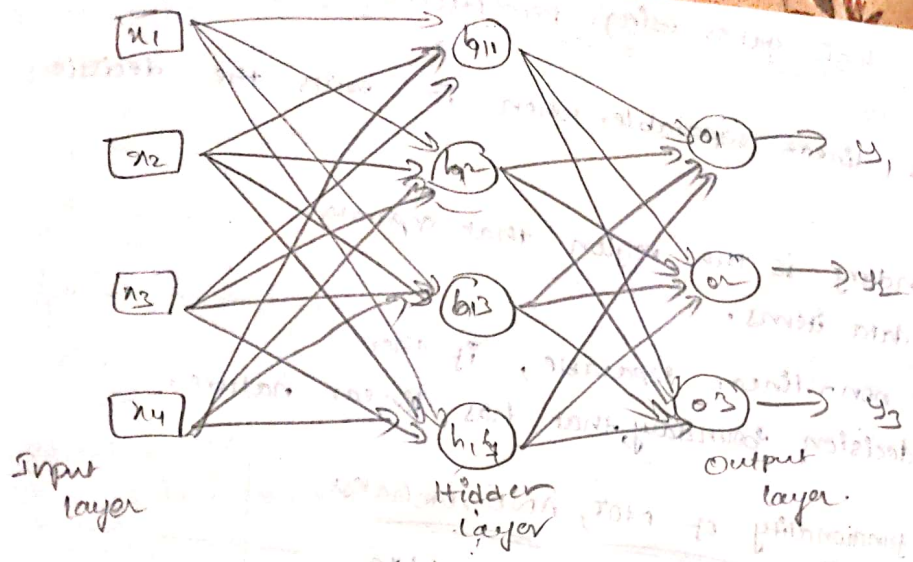
* each neuron gets same input from all input nodes but weights are different.
* there is no feed-back loops.
* The data (or) inputs transfer only in forward direction.
* Not suitable for handling complex tasks.



* Multi-layer feed forward Network:
* Along with 1 output layer, it also contain 1 (or) more hidden layers.
* Hidden layers let ANN to handle complex tasks.
* 1st level hidden layers can perform some low level tasks.
* 2nd level hidden layers can peform a task higher than previous layer.
* output layer neuron can handle high level tasks to produce the output.
* Neurons in all layers are connected with previous layer neurons.

Input layer → Hidden layer 1 → output layer → y

* Information flows in forward direction only.
* there exists no feed-back loops.



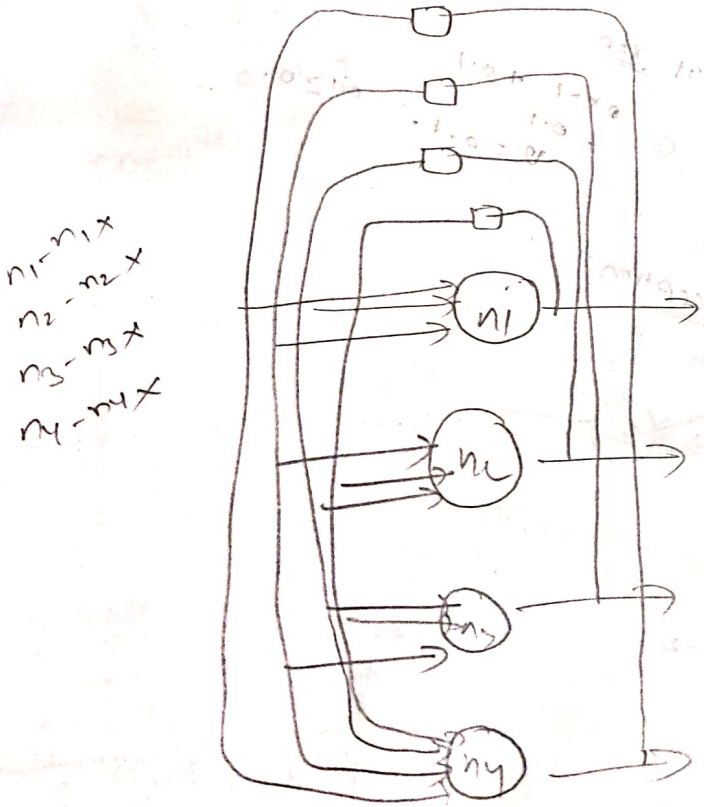
* Recurrent network :-

* widely used in text and language processing, speech recognition etc---

* There exists feed - back loops.

* i.e., output of neuron fed as input.

* Consist of unit delay operators (memory elements) to store previous output state of neurons.



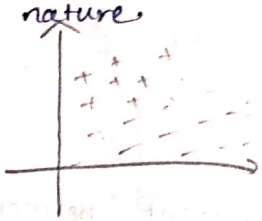
n1 - n1 x
n2 - n2 x
n3 - n3 x
n4 - n4 x

22/1/19 * Realizing logic gates using perception:

* Data set is linear separable, when it exists the decission boundary.

* Decission boundary is the region that separate two sets of data items.

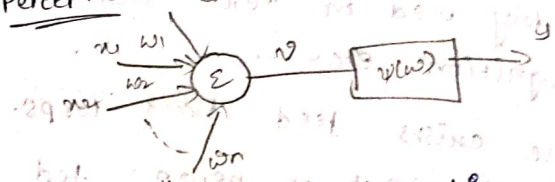
* If data is non-linear separable, if there is no exists decision boundary, that has linear nature.



* Realising the functionality of NOT, AND, OR Gates:

* Binary function is also called as medicide function.

Perceptron:



* NOT GATES:

x	o
1	0
0	1

* activation function - binary step

$y = 1 \quad v \geq 0$

$y = 0 \quad v < 0$

$v = xw + b$

If $x=1$ $\quad w=-1, b=0.1$

$1 \times -1 + 0.1$ so

$-1 + 0.1$

$= -0.9 < 0$

If $x=0$

$0 \times -1 + 0.1$

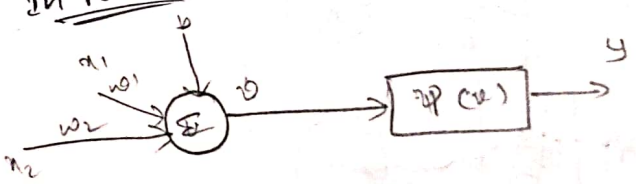
$= 0.1$

$v = 0.1$. $\quad 0.1 \geq 0$.

* AND GATES:

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

In Perceptron:



$v = x_1 w_1 + x_2 w_2 + b$.

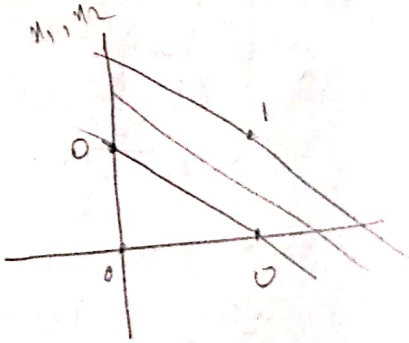
$w_1 = 1 \quad w_2 = 1 \quad b = -2$.

$0 \times 1 + 0 \times 1 + -2 \Rightarrow -2$

$0 \times 1 + 1 \times 1 + -2 \Rightarrow -1$

$1 \times 1 + 0 \times 1 + -2 \Rightarrow -1$

$1 \times 1 + 1 \times 1 - 2 \Rightarrow 0$.



the decission

re.

y

inary step

* $w_1 = 1$, $w_2 = 1$, $b = -2$, $x_1 = 0$, $x_2 = 0$,

$v = 0 \times 1 + 0 \times 1 - 2 = -2$

$y = 0$

$x_1 = 0$, $x_2 = 1$

$v = 0 \times 1 + 1 \times 1 - 2 = -1$

$y = 0$

$x_1 = 1$ $x_2 = 0$

$v = 1 \times 1 + 0 \times 1 - 2 = -1$.

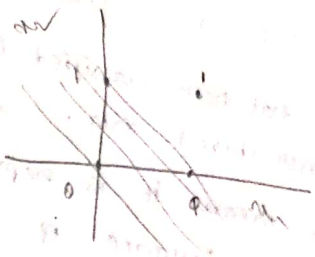
$y = 0$

$x_1 = 1$, $x_2 = 1$

$v = 1 \times 1 + 1 \times 1 - 2 = 0$

$y = 1$.

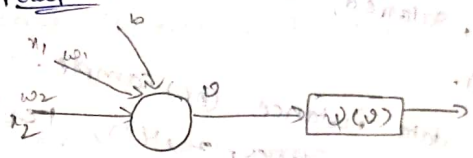
OR GRAPH



* OR GATES:

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Perceptron



$v = x_1 w_1 + x_2 w_2 + b$.

$w_1 = 1$, $w_2 = 1$, $b = -1$ $x_1 = 0$, $x_2 = 0$.

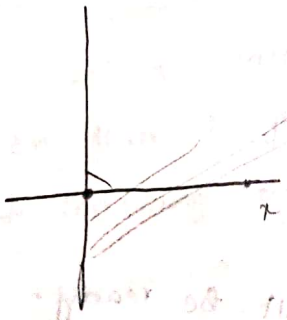
$v = 0 \times 0 + 0 \times 0 - 1$

$= -1$) $v = 0 \times 1 + 0 \times 1 - 1 = -1$.

The binary function will return 0

$y = 0$.

NOT GRAPH



⇒ $x_2 = 1$, $x_1 = 0$.

$v = 0 \times 1 + 1 \times 1 - 1 = 0$

$y = 1$.

=) $x_1 = 1$, $x_2 = 0$

$v = 1 \times 1 + 0 \times 1 - 1 = 0$

$y = 1$

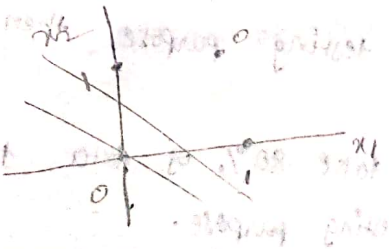
=) $x_1 = 1$, $x_2 = 1$

$v = 1 \times 1 + 1 \times 1 - 1 = 1$.

$y = 1$.

XOR GATES :

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

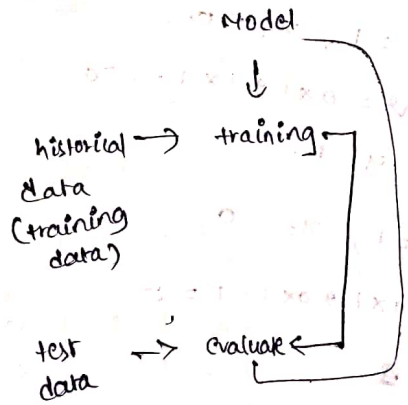


Implement XOR gate functionality using perceptron.

NOTE: (Use multiple perceptrons to implement functionality).

UNIT-I

* Data has been classified into 2 types.
  1.) unstructured data. 2.) structured data.
* Data means it is represented using set of attributes (features).
* Particular instance of all values for attributes we will represent data of (~~incidente~~) instance. (or) Sample (or) example.
* Ex: Customer information in loan management system.
  (account, salaried, annual income, dependents, assests, assets value).
* Example for data instance (or) sample.
  (3695771659, yes, 20lacks, 2, yes, 1cr).
* Data can be used to perform predictions.
* A model is trained with some portion of data and it is called as training data.
* The portion of the data used to evaluate for performance of the model called as test data.



* After completion of this process the model will be ready.
* 80 : 20.

when we have some data, and we have to use for training and testing purpose then we will use this 80:20 rule.

* In this, take 80% of data for training purpose and 20% of data for testing purpose.

* we need to randomize the selection of data members for training data, testing data to avoid bias.
* In labelled data,
* Labelled data :- A label is nothing but target output for the data instance.
* labelled data will be used for,
→ * data for which every data sample has target output associated with it.
* customer information

(369767513, yes, 20Lakhs, 2, yes, 1cr, sanctioned)
Input labels — label

* Learning process :- Learning process is a process for acquiring the knowledge, by making corrections to the information already known.
* ML/DL models i.e., ANN learn from the data by making corrections to free parameters.

* Types of learning :
* It was classified into 3 types.
1.) supervised learning.
2.) unsupervised learning.
3.) Reinforcement learning.

1.) Supervised learning :- In supervised learning, we use the labelled data set to train the machines and based on that training the machine will perform the outputs.
* learning with a teacher.

2.) unsupervised learning :
* learning without a teacher.
* uses data without labels means unlabelled data set will be used.

Reinforcement learning :
* learns everything by interacting with environment.

Im + Linear Algebra :

* 3 types of values (or) objects for mathematical functions
  1. Scalar. (1D entity).
  2. Matrices (2D entity)
  3. Tensors.
  4. vectors.

(2) Matrices:) * Collection of elements which are arranged in rows & columns.

2) we can define tensor of dimension two.

* Relation between Neuron and linear algebra is mathematical of biological neuron.

*

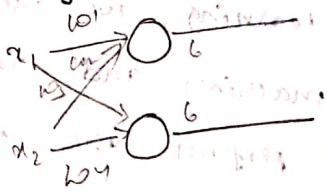
* System of linear Equations :-

Ex: $2x + 3y = 6$
$4x - 2y = 4$.

$2x + 3y = 6$
$4x - 2y = 4$

$$\begin{bmatrix} 2 & 3 \\ 4 & -2 \end{bmatrix}_{m_1 \times n_1} \begin{bmatrix} x \\ y \end{bmatrix}_{m_2 \times n_2} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}.$$

$$\Rightarrow \begin{bmatrix} 2x + 3y \\ 4x - 2y \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}.$$



* Artificial neuron perpms mathematical neuron.
* Given a system of linear equations

$$a_1 x + b_1 y = c_1$$
$$a_2 x + b_2 y = c_2.$$

it can be represented in terms of matrices and vectors as

$$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}.$$

* In this case linear equations are solved by using operations on matrices and vectors.

* In ANN, ... neuron can ... represented as ... * v is comput... which denote ...
* ANN is ...
  * linear func...
  * Inputs wi... vectors, ...
  * The com... matrices.
* operations
* Scalar, ma...
* In ANN ...
* Ex: ... K *
* operations
* Scalar
* These ... be ...
* Num...
* Multip...
* Divisi... indire...
* Add...
* Sup...

* In ANN, the neurons computes v as linear computation i.e., neuron can produce linear equations. The input, weights represented as matrices and vectors.

* v is computed by performing operations on matrices, vectors which denote input, weight.

* ANN is mathematical model.
  * linear function.
  * Inputs will be scalar, vectors, matrices - - -
  * The computation performed by neuron is in b/w vectors and matrices.

* operations on matrices :-

* Scalar, matrices. In b/w we can perform +, *, -, ÷.

* In ANN we can used this in the bias.

* Ex: $K * \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \Rightarrow \begin{bmatrix} Ka_1 & Kb_1 \\ Ka_2 & Kb_2 \end{bmatrix}$.

$K + \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \Rightarrow \begin{bmatrix} K+a_1 & K+b_1 \\ K+a_2 & K+b_2 \end{bmatrix}$.

* operations on matrices :-

* Scalar operations : we can perform +, -, *, ÷.

* These are most frequently operations and these can be mostly used in deep learning and ANN.

* Numpy will support all operations indirectly.

* Multiplication will directly allow numpy.

* Division support on the matrices and it supports indirectly. And it multiply $\frac{1}{K}A$.

* Addition and subtraction are not directly allowed.

* Suppose add (or) sub of scalar value to A, then, A+k, A-k.

* then the operations will be $A + K\,J_d$, $A - K\,J_d$.

* These scalars are multiplies with some

unique matrix like '1'. 'd' means dimensions of this unique matrix which is same as the unit.

* scalars can be used in matrices.

* Add, sub, not directly because, we want +, - want requires two matrices of same size (or) dimensions. means 2×2 (or) 3×3.

* Matrices + (Addition).

two operations A, B. ⇒ A+B.

* If A and B are of same size.

we can perform '+' operation is on same size only.

(Sub):

we want to sub one matrix to other, then it will support when, it has same size.

Transpose: In, this we can perform, A^T (or) A'.

convert rows into columns and columns into rows.

simply, we can write.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

* diagonal won't change and remaining will change.

Inverse: Important **

$$A^{-1} = \frac{\text{Adj } A}{\det A}.$$

Ex: $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$.

$\det A = ad - bc$

$$\text{adj } A = \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$\det A = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}.$$

$$\text{Adj } A = \begin{bmatrix} +\begin{vmatrix} e & f \\ h & i \end{vmatrix} & -\begin{vmatrix} d & f \\ g & i \end{vmatrix} & +\begin{vmatrix} d & e \\ g & h \end{vmatrix} \\ -\begin{vmatrix} b & c \\ h & i \end{vmatrix} & +\begin{vmatrix} a & c \\ g & i \end{vmatrix} & -\begin{vmatrix} a & b \\ g & h \end{vmatrix} \\ +\begin{vmatrix} b & c \\ e & f \end{vmatrix} & -\begin{vmatrix} a & c \\ d & f \end{vmatrix} & +\begin{vmatrix} a & b \\ d & e \end{vmatrix} \end{bmatrix}^T$$

* **Multiplication:-**

$A_{n_1 m_1}$ $B_{n_2 m_2}$ $A * B$

iff $n_1 == m_2$

* It will multiply just like, row of A with column of B.
* Inputs are represented using vector.
* weighted are represented using matrix.

$$\boxed{WX} + b.$$

* Bias is a scalar value.
* In multiplication, in programming languages it will directly support. But not supported in mathematics for some time.

* **Vector:** vector is another important object. Mathematics contains:
* vector is a one-dimensional unit and it is called as 1-dimensional tensor.
* Tensor means it is a part of matrix more than 2. [3×3].
* Basic operations are +, - and ×.
* Add & Sub are having same rule.
* In mul we can perform
  a) Element wise multpli operation.

2.) Multiplication used in matrices, dot product.

* Mostly dot product will be used in vectors.

* vectors:

* Input always represent in the form of vector.

* It is a 1-dimensional tensor.

* In mathematical terminology vectors means it has both magnitude and directions.

* we have 2 types of vectors,

1.) Row vector 2.) column vector.

1.) Row wise representation is called row vector.

2.) column wise representation is called column vector

Ex: $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ column vector

$[x_1, x_2, \dots, x_n]$ row vector.

* operations in vectors:

1.) Addition

2.) Subtraction

3.) Multiplication.

1.) Addition: when the dimensionality is same we can perform addition operation, $x+y$

dimension x = dimension y.

$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{n\times 1} \cdot [x_1 \; x_2 \; \dots \; x_n]_{1\times n}$

2.) Subtraction: when the dimensionality is same we can perform Subtraction operation, $x-y$.

dimension x = dimension y.

Ex: $x = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $y = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$

$x+y$

$\begin{bmatrix} 2+4 \\ 1+3 \end{bmatrix}$

$x-y$

$\begin{bmatrix} 2-4 \\ 1-3 \end{bmatrix}$

* Multiplication

In this two

1.) Elemen

$x =$

2.) x ⊙

2.) Dot · Pr

$x \cdot y$

* The

$x =$

$[x_1$

This is

* If

final

$x + y$

$$\begin{bmatrix} 2+4 \\ 1+3 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

$x - y$

$$\begin{bmatrix} 2-4 \\ 1-3 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \end{bmatrix}.$$

* Multiplication:

In this two types of operations are allowed.

1.) Element-wise multiplication.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

$$\Rightarrow x \odot y$$

$$\begin{bmatrix} x_1 * y_1 \\ x_2 * y_2 \\ \vdots \\ x_n * y_n \end{bmatrix}.$$

2.) Dot product:- (mostly used in the multiplication).

$x \cdot y$

* The result is between $x^T y$.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$\begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}_{1 \times n} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{n \times 1}.$$

$$= x_1 y_1 + x_2 y_2 + \cdots + x_n y_n.$$

This is the result of the dot product x and y.

* If we apply a dot product scalar value will be final result

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$w \cdot x \Rightarrow w^T x \Rightarrow w_1 x_1 + w_2 x_2 + \cdots + w_n x_n.$

* Scalar operations on vectors :-

In mathematically, we can multiply all elements of scalars with vectors.

* But in programming language, it supports by the python.

* only multiplication is allowed between scalar and a vector,

K - scalar.

x - vector.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \xrightarrow{Kx} \begin{bmatrix} Kx_1 \\ Kx_2 \\ | \\ Kx_n \end{bmatrix}$$

Kx

* Addition :-

K - scalar

x - vector

y - temporary with i's.

=) $x + Ky$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad y = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ 1_n \end{bmatrix}$$

$$x + Ky = \begin{bmatrix} x_1 \\ x_2 \\ | \\ x_n \end{bmatrix} + \begin{bmatrix} K \\ K \\ | \\ K_n \end{bmatrix}$$

$$= \begin{bmatrix} x_1 + K \\ x_2 + K \\ \vdots \\ x_n + K \end{bmatrix}$$

* Norm :- Distance between two vectors.

* Length of a vector :-

-> compute the length between two vectors.

* 3 different types of norms.

1.) l_1 norm

2.) l_2 norm ***

3.) l_{max} norm.

* l_2 norm is also called the Euclidean norm.

* l_{max} norm is also known as infinity norm.

l_1 norm :

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

* Representation of l_1 norm $= |x_1| + |x_2| + \cdots + |x_n|$.

* l_1 norms means computing sum of the absolute values of individual ~~squares~~ vectors.

Ex: $x = \begin{bmatrix} -2 \\ 3 \\ 6 \end{bmatrix}$

l_1 Norm $= |-2| + |3| + |6|$

$= 11$.

l_2 Norm : It is also known as Eucledean norm.

* The l_2 norm of vector is simply computed by taking

$$l_2 \text{ norm} = \sqrt{x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2}.$$

Ex: $x = \begin{bmatrix} -4 \\ 3 \\ 2 \end{bmatrix}$.

$$l_2 \text{ norm} = \sqrt{(-4)^2 + (3)^2 + (2)^2}$$

$$= \sqrt{16+9+4}$$

$$= \sqrt{29} = 5.36.$$

l_{max} Norm : It is also known as infinity norm.

* we can compute l_{max} norm as.

l_{max} norm $= \max(|x_1|, |x_2|, \cdots |x_n|)$.

* Simply we can say that maximum of absolute of x.

l_{max} norm $= \max(|-4|, |3|, |2|)$.

l_{max} norm $= 4$.

Because, it is largest among these.

* Numpy:

* Installation Numpy:

pip install numpy

pip - Python installation package

(or)

conda installation numpy

* Imparting numpy

import numpy as np

* creating array

np.array (list of elements, dtype = type)

a = np.array ([1,2,3,4,5], d type = np.int32)

a_2 = np.array([[1,2,3],[4,5,6]])

* Array with zeroes:

np.zeros (dimension, dtype)

- creates arrays with zero's

Ex: np.zeros ((1,3))
np.zeros ((3,3))

* Array with ones:

np.ones (dimension, dtype)

- creates array with 1's

Ex: np.ones ((2,2))
np.ones ((1,3))

* Identity matrix:

np.eye (dimension)

* returns identity matrix.

Ex: np.eye ((2,2))

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

np.eye ((3,3))

* Filling with

np.full (di

- creates num

Ex: np.full

np.full ((

np.full ((

finding ma

min (Array

max (Array

* Indexing n

* Forward i

* Backward

* Accessing

* [ ] is used

* to acce

A[i] to acc

* 2D arra

at ith row

* Slicing:

* To ext

array

Ex:

* Filling with required value:

np.full (dimension, value)

- creates numpy array with value passed.

Ex: ~~np.full (dimension, value)~~

np.full ((2,2), 0)

np.full ((2,2), 5).

finding max, min:

min (Array, axis)

max (Array, axis)

axis = 0 ⇒ row

axis = 1 ⇒ column.

* Indexing numpy arrays:

* Forward indexing (0 to n-1) (left to right)

* Backward indexing (last element to first element)
(-1 to -n) (right to left).

* Accessing elements of an array:

* [ ] is used to access elements.

* to access 1d array we have one level of indexing.
A [i] to access ith element of array A.

* 2D array - 2 level indexing A[i, j] to access element at ith row, jth column,

* Slicing:

* To extract the portion of array ':' - slicing operator.

array [start : end]

Ex: A = [1, 2, 3, 4]
0, 1, 2, 3

A [1:3] ⇒ 2, 3

A [:4]

A [:].

3/8/22 * Indexing and Slicing 3D Arrays :-

* There are 3 levels of indexing arrays.
* 3 levels of indexing required to access any element in 3D numpy array.

Ex:- a [i, j, k] (:' 3D array numpy)

i -> refers to particular array.
j -> refers to row in an array i.
k -> refers to column in array i.

Ex:- a = np.array ([[[1,2],[3,4],[5,6]], [[7,8],[9,10],[11,12]], [[13,14],[15,16],[17,18]]]);

* 3D array is an array of 2D arrays.

a =>

	0	1
0	1	2
1	3	4
2	5	6

0

	0	1
0	7	8
1	9	10
2	11	12

1

	0	1
0	13	14
1	15	16
2	17	18

2.

* To access 1st array 2nd row 2nd element,
  a [0, 1, 1]
  refers to 4.
* To access 2nd array 1st row 1st element
  a [1, 0, 0].
  refers to 7.

* Slicing:- To extract the portion of 2D arrays.

arr [arrayrange, rowrange, columnrange]

Ex:- 1st, 2nd array's list two rows.

a [0:2, 0:2, :]

[[1,2] [3,4]]
[[7,8] [9,10]]

* Fetch or extract last row 2nd column elements from 2nd and 3rd column arrays.

* a [1:, 2:, 1:
* Sparse Matrix:-
* In python we
* Scipy:- (Scientific
* It is used to
  to import sparse.
* CSR- matrix:-
* It is a function
* we use function
* It returns Spar
Ex:- b = np.array
c = sparse.
O/P:- (1,2) 3
(2,0)
* Reshaping numpy
* Reshape():- It is
* It takes target
  array reshaped to
  arr.reshape
Ex:- a = np.array
b = a.resh
b [[1,2,3],
* Vectors and matrice
* Flattening numpy
* flatten() - used
* Row vector [2,4,6
* Raven()- only
  of original arr
  array.
np.flatten()- use

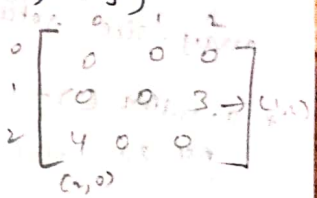
* a[1:, 2:, 1:]

* Sparse Matrix:-

* In python we have under package called as
* Scipy :- (Scientific python package)
* It is used to represent the sparse matrix, we use scipy to import sparse.
* csr_matrix :- Compressed sparse row matrix.
* It is a function to represesent sparse row matrix.
* we use function sparse.csr_matrix (numpy_ndarray).
* It returns Sparse matrix representation for numpy_ndarray.

Ex: b = np.array([[0,0,0], [0,0,3], [4,0,0]])

c = sparse.csr_matrix(m);



O/P:- (1,2) 3
(2,0) 4

* Reshaping numpy Arrays :-

* Reshape() : It is a method of numpy array object.
* It takes target shape as an argument and returns numpy array reshaped to specific shape.

arr.reshape(shape).

Ex: a = np.array([1,2,3,4,5,6,7,8,9]);
b = a.reshape(3,3);
b [[1,2,3], [4,5,6], [7,8,9]].

* Vectors and matrices:-

* Flattening numpy arrays:-

* flatten() - used to flaten numpy arrays.
* Row vector [2,4,6,]
* Raven() - only different is it has a duplicate copy of original array; changes made doesn't effect original array.

np.flatten() - used to flatten numpy array.

* Vectors and matrices intro :-

* Describing vectors and matrices :- Numpy array has shape attribute, size attribute.

shape - to know shape ; size - no. of elements present in numpy array

ndim - store dimensionality of numpy array.

* shape, size, ndim are the numpy array objects.

* a = np.array([[2,3,4],[5,6,7],[8,9,10]]);

a - numpy array object. (Reference of numpy array).

* a.shape() returns shape of numpy array shape is about rows and columns present in array.

* a.size() - Contains no. of elements present in numpy array. size returns value.

* a.ndim() - returns dimensionality of numpy array.

Ex: 2.

* Numpy array objects has 3 attributes a.shape(); a.ndim(), a.size().

* Addition, subtraction and multiplication :-

* we have two matrices A and B.

Add - A+B Sub - A-B Mul - A*B.

* In case of multiplication, need to perform A@B. It takes element wise multiplication.

* The numpy package has functions add and subtract.

np.add(arr1, arr2).

* Numpy has the feature called broadcasting. It extends the size. It means it streches the rows, hence it makes the size of the second operation

Ex: A = np.array([[2,4],[1,3]]);

B = np.array([[1,6],[5,1]]);

np.add(A,B); ([3,10],[4,9])

* when shape is

feature to make

c = np.array(

np.add(A,

[[7,11],

c = [5,7

* c is streched

to A.

$$C = \begin{bmatrix} 5 & 7 \\ 5 & 7 \end{bmatrix}$$

* Addition using

* Subtract operatio

matrices.

* Broad casting is

* If there is m

may not occurs

* Subtraction: It c

It means that

np.subtract

* Multiplication:

* It can be u

* It compute do

matrices.

* @ - dot product

* It can be u

to compute dot

* Element wise

* A - element

* A*A repres

* when shape is not equal to numpy uses broadcasting feature to make arrays equal in shape.

```
c = np.array((5,7));
np.add(A,c);
[[7,11],[4,10]]
c = [5,7]
```

* c is streched vertically to make a shape of c equal to A.

$$c = \begin{bmatrix} 5 & 7 \\ 5 & 7 \end{bmatrix}.$$

* Addition using '+' is also allowed on numpy arrays.
* Subtract operation can be used in both vectors and matrices.
* Broadcasting is not done allways.
* If there is more than 2 in matrices, then broadcasting may not occurs.

* **Subtraction:** It can be implemented by using arrays. It means that it can subtract one from another.

np.subtract(arr1, arr2)

* **Multiplication:** np.dot(arr1, arr2).
  * It can be used to multiply two matrices.
  * It compute dot product of the two vectors (or) the matrices.
* @ - dot product operator.
  * It can be used to multiply, the two matrices (or) to compute dot product of 2 vectors.
* **Element wise product:-**
  * A - Element wise multiplication operator.
  * A * B represents that elements of the dot product.

* Transpose of a matrix :-

* Matrix (or) a vector.
* Transpose of numpy array can be computed by using 'T' operators.
* Mostly we can use transpose operation.

Ex: arr.T
A.T.

* Linear Algebra package.
* It contains many functions used in linear algebra functions.
* we can solve easily.
* wrgt is is determined, b = Inverse.
* To compute the decrement and matrix simply as.

(* Function of ~~death~~ det).

* In numpy - we have linear Algebra package linalg().
* Determinent, Inverse to compute determinant of a matrix. lin.alg has a function det., usage is np.linalg.det(arr) np.linalg.det(A).
* To compute inverse of a matrix use linalg np.linalg.inv(arr)

Ex: np.linalg.inv(Arr)

* To compute eigen values, eigen vectors of a square matrix.
* use eigen function of linalg returns eigenval, vectors of a square matrix.

Ex: np.linalg.eig(arr)

eval, ev = np.linalg.eig(A)

* To solve linear equations use and solve functions of linalg.

Linalg.solve(arr1, arr2)

* Scalar oper
* using vec
* That appli
* It takes
* A function operation
* If = lamb
* we take
* creates elements i
Ex: a =
If =
* Now cr
vf = np
vf - iterativ
* vf(a) - vectors we functions.
* Numpy h
* working
* panel data
* It is all
* It is
* It is us
* Pandas
* Pandas
* Data - Stru
Series
Series
* In data
Columns
rows

* Scalar operations :-

* using vectorize function in numpy library.
* that applies on all elements of matrix vector.
* It takes function as argument.
* λ functions are used when we want to perform micro operation on variable.
* If : lambda a: a+1.
* we take np. vector (Element-function).
  * creates iterative function that can be applied to all elements in nd array.

Ex:- a = np.array([[2,4],[-1,3]])

lf = lambda b: b+10.

* Now creating a function using vectorize,

vf = np.vectorize(lf)

vf - iterative function (or) vectorized function.

* vf(a) - If we add any arithmetic operations (or) matrix (or) vectors we will get the exact result by this vectorized functions.
* Numpy has broad casting feature.

** working with pandas :-

* panel data.
* It is also treated as python and data analysis.
* It is used for built on top of numpy.
* It is used for data pre-processing operations.
* Pandas allows us to perform various data analytics operations.
* Pandas can perform data-visualizations.
* Data-structures supported by pandas.

1) Series 2) data frame.

1) Series can be organized only in the form of colums.

* In data frame it contains rows and columns, but in columns it contains similar data (same type of the value), rows (different types of values).

1.) Series
* It contains single column array.

2.) Data Frame
* 2D array where information organized as rows & columns.

* Installing pandas :-
pip install pandas.

* Importing pandas :-
import pandas as pd

* Series :-
* It is a column in 2D array.
* Creating Series
pd.Series(list).

Ex:- s = pd.Series([20, 30, 40, 50]).

0	20
1	30
2	40
3	50

* Index parameter of series method can be used to add user defined index.

Ex:- i = ['first', 'second', 'third', 'fourth']
s = pd.Series([20, 30, 40, 50], index = i);

* Creating from dictionary :-
* Series object can be created from the dictionary.
* Keys of dictionary used as indices.
* values are used as elements of series object.

Ex:- d = {'maths': 80, 'Physics': 75, 'chemistry': 100}.
s1 = pd.Series(d).

s1

maths	80
physics	75
chemistry	100

* Accessing in
* Series obje
1.) Inden
1.) Inden att
2.) It contain
Ex:- s1. ind
s1. valu
* NAN (Not
Point comp
* It is a
* Series a
1.) isnull
* s1.isnull
Ex:- FA
FAL
FAL
2.) It re
not NULL
* It a
* Handling
drop na
dropna -
fillna() -
Ex:- s1.
s1
ma
phy

* Accessing indices and elements :-

* Series object can have 2 attributes.
  1.) Index 2.) values.

1.) Index attribute contain list of indices.

2.) It contains element values.

Ex:- S1. index
S1. values.

* NAN (Not a Number) :- It is introduced as part of coding Point Computation.

* It is a floating point number.

* Series and data-frames (object) has 2 objects (methods)
  1.) isnull ( ) 2.) notnull ( ).

* S1.isnull ( ) - It contains boolean list ( ).

Ex:

FALSE
FALSE
FALSE

2.) It returns true when the element value is (not NULL ( )) not NAN.

* It returns false when the element value is NAN

* Handling NAN :

dropna ( ) fillna ( ).

dropna - It removes the element contains NAN value in series object.

fillna ( ) - It fills NAN elements with the value passed as parameter.

Ex: S1. dropna ( ).

S1	
maths	80
physics	75

S1. fillna (0)

S1	
maths	80
Physics	75
chemistry	0

* working with pandas :-

* Accessing of elements :

* Row index to access any element of series object.

* [ ] used to access elements.

Ex:- S1 ["maths"]

returns element, with index maths.

* Multiple elements can be accessed by specifying row indices separated by commas.

Ex:- S1 ["maths", "chemistry"]

It returns elements associated with indices maths & chemistry

* Arithmetic operations :-

* These are allowed on two series objects if two objects has same indices.

Ex:- S1 = Pd. series ([1, 2, 3, 4, 5])

S2 = Pd. series ([6, 7, 8, 9, 10])

S1 + S2.

* Scalar operations :-

* Scalar operations are allowed on series object directly.

Ex:- S1 + 3, S1 - 3, S1 * 3, S1 / 3.

- add 3 to every element of series object S1.

* Apply method :-

apply (function [, arguments])

method in series object can be used to apply specific function to series data.

Ex:- lf = lambda x : x + 3

S1 . apply (lf).

Ex:- S3 = pd . series (['yes', 'no', 'no', 'yes', 'no'])

lf = lambda r; 1 if x == 'yes'

else 0

S3. apply (lf)

[1, 0, 0, 1, 0]

* Data Frame :

* Data fram

data frame

* 2nd mari

* Creating data

using data

*) Pd . Data

Ex:-

data = [['G

[

[

[

df = P

12/5/22 'popu

* Summari

* Method

-> he

-> he

* Data Frame :- * It is in the form of table.

* Data frame constructor will be used to create new data frame.

* 2nd matrix with rows, column indices.

* Creating dataframe :

Using dataframe Constructor,

→ Pd . DataFrame (data_list [, columns, index]).

Ex :-

country	population.
Germany	30,00,000
France	25,00,000
UK	45,00,000.
USA	50,00,000

Datalist = [['Germany', '30,00,000'],
['France', 25,00,000],
['UK', 45,00,000],
['USA', 50,00,000]].

df = Pd. DataFrame (data_list, columns = ['country', 'population']), inden = ['country1', 'Country 2', 'country 3', 'country4']

12/5/22

* Summarizing data frame :-

* Methods of dataframe objects.

→ head ( ) - it fetchs first 5 rows of data.

→ head (n) - it fetches first n rows of data.

* It can be used as df. head ( ).

→ tail ( ) - it fetches last 5 rows of data.

→ tail (n) - it fetches last n rows.

* It can be used as df. tail ( ).

→ info ( ) - it writtens information about data frame.

- returns the data frame information object which consists of rows, columns and their types,

* this can be used as df.info().

→ describe() - It returns the statistical related information like max, min, mean, 25% (first quartile), 50%, 75%..

* This can be used as df.describe().

→ max, min.

* returns the max and min of each and every column in the data frame.

* this can be used as df.max() ; df.min().

→ count().

* returns the no. of non-NA values present in each and every columns. (or)

* returns count of non-NAN values for each column.

* It can be used as df.count().

→ mean() - It returns the mean of each and every column present in the data frame:

→ var()

* we can compute variance using var()

* It returns variance.

* It can be used as df.var().

→ std() - standard deviation

* It returns the standard deviation of the data present in the data frame.

* It can be used as df.std().

Transpose: It converts rows into columns and columns into rows.

* Numpy support transpose.

* It can apply the operator T can be used "df.T."

* concatenation and melting operation:-

* Concatenate(): Concatenate data frames either row wise (or) column wise.

* Axis parameters are used to specify axes.

0 - row wise.

1 - column wise.

melt():
columns.
columns
sort():
Sort_valu
one or)
* Parameter
1.) by:
be applie
2.) Ascendin
ascendi
* when
* when
3.) Find:
* specify
merge so
* Quick
merge
heap
* It can
* Extract
there
1.)
2.) [
3.) lo
(Thes
base
1.) : n
whe
singl
* Sin

melt(): If we have some data frame containing rows & columns. this melt() will convert rows into columns and columns to rows data.

sort(): we can sort the data based on specific column.

sort_values() - sort the data frame with respect to one or more columns.

* Parameters:- there are 3 important para

1.) by :- It specify the column on which sorting need to be applied.

2.) Ascending :- It specifies whether the sorting is in ascending order.

* when the statement is true - sorting in ascending order.
* when the statement is false - sorting in descending order.

3.) Find :- It can be done by using quick sort method.

* specify kind of sorting means either quick sort, merge sort, heap sort.
* Quick sort - to apply quick sort,
  merge sort - to apply merge sort,
  heap sort - to apply heap sort.

* It can be used as df.sort_values(by="column").

* Extracting Data From a dataframe:-

there are 3 different ways.

1.) . Notation
2.) [ ]
3.) loc, iloc,
(These are 2 rows which extracts rows and columns based on index positions).

1.) . notation:-

when we want to access a data frame in single column, . notation will be used,

* simply extract specific column,

df

	A	B	C
0	a	1	20
1	b	2	30
2	c	3	40

df.A

dataframe.column.

2. [ ] indexing

we can extract data from data frame by using [] (index) operator.

* In this we can extract row [] and column [].
* used to access data by giving row (or) column index

→ dataframe [rowindex]
→ returns a row

→ dataframe [row i : rown]
returns row i to row n-1.

→ we can access it by using columns also.
dataframe [col_name]
returns one single column.

* To access list of columns pass the list of column names.
* dataframe [[col1, col2, col3]]
* dataframe [col1 : col n],
→ returns range of columns, starting from col 1 to col n-1,

* "loc" is used to access the data from data frame by using labels. (labeled base operators).
dataframe.loc [row, column].
* dataframe.loc [:, col1].

If we want to access more than 1 columns
dataframe.loc [:, [col1, col2, col3]].

* Scikit - le
* Preprocessing
* In preproce
* Label Encod
numerical
* In order t

```
from sklearn.preprocessing import LabelEncoder.
// create label encoder object.
    l = LabelEncoder()
// fit data with label encoder
    l.fit_transform(df["Gender"])
```

(fit-transform takes input to convert numerical data and perform both sckit and transition).

* Data will be converted into numerical information.
* Label encoder will help us to convert categorical data into numerical data.
* After converting the data into numerical data (by taking example) need to drop old gender column from data frame.

```
df.drop("Gender", axis = 1, inplace = True).
```

* Now add encoded column to the dataset `df["x"] = a`

* Standarize the data : (Data standardization).
* It can be also called normalise the data.
* In preprocessing method
  - Standard scalar
  - Min-max scalar
* These can be used inorder to normalise the data.

* Standard scalar: is normalize the data, i.e., it performs unit varience.
* this will @x) it makes the data distribution as normal distribution. By this most of the data will be send to the min point.
* And remaining most of data sended arround in the mean point.
* Inorder to do this, it removes the mean value is removed from each and every value.

Min-Max scalar: It makes use of min and max of values inorder to perform normalization.
* once, data is normalize, then the column values will be in between min-max.

Standard scalar:
* from sklearn.preprocessing import StandardScalar
# create standard scalar object
s = StandardScalar()
# fit the data
s.fit(data)
s.transform()
(transform ~~word~~ returns and array to normalize the data).

Min-max scalar:
from scalar.preprocessing import MinMaxScalar
# create min max scalar object
m = MinMaxScalar()
# fit the data
m.fit(data)
# transform
m.transform()

* preprocessing modelle has another function one hot encoder.

One-hot Encoder:
* Inorder to convert categorical data into numerical form.
* when we apply one hot encoder we get list of the data, and we get the data in the form of vectors.

* when we apply one hot encoder to the preprocessing we get no of vectors.

[c1, c2, c3]

$\begin{bmatrix}1\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix}$

like this the vectors are placed which are encoded with 0's and 1's.

* usage of one-hot encoder in scikit-learn,

```
from sklearn.preprocessing import one hot Encoder
# create one hot encoder object
    e = onehotEncoder()
# fit data
    e.fit(data)
# transform
    e.transform()
```

(to transform the data into one hot encoder and it returns nd.array and no of vectors will produced).

* Remaining columns:-

* Data frame object method

rename (column = dict)

Ex: d = {"A": [100, 200, 300], "B" = [1, 2, 3]})

df = Pd.DataFrame(d)

df.rename(columns = {"A": "first", "B": "second"}, inplace = True)

* Train-test-split:-

sklearn.model_selection

↳ train.test split.

```
# splitting data
from sklearn.model_selection
        import train_test_split,
# split data by passing features data, response data
    : train_test_split(X, Y, test_size = 0.2,
                       random_state = True).
```

* Perceptron model :- perceptron is a (neural) Artificial neural network models

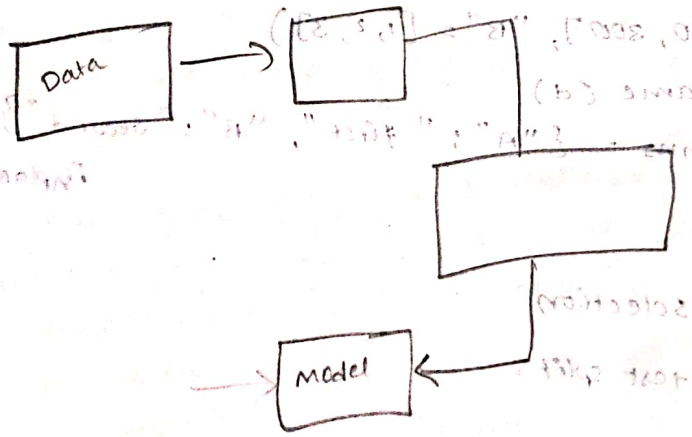
* In sk learn we have sub module called sk.learn. (linear model selection)

* It is used to construct the perceptron model.

* This (linear model selection) has perceptron and this can be implement the simple perceptron model.

* #perceptron classifier is used as learning algorithm and it provides when to stop the algorithm

```
# create perceptron model
from sk learn . linear - model . import Perceptron
pc = Perceptron ( tol = 1e-3 ) .
# fit the data (it passes input data to the training process).
pc . fit (x, y)
# check accuracy
pc . score ( ) (score represents the accuracy of the model).
# input new data to
# make predictions .
pc . predict ( x - new)
```



*Multi- layer perceptron :-

```
Sklearn . neural - network
        ↳ MLPClassifier
# create MLPClassifier
from sklearn . neural - network import MLPClassifier
an = MLPClassifier ( )
```

fit data
an . fit (y

prediction
an . predict

* Realising logic

```
def perceptron .
{
# read in
# convert
# read
x = np .
w = np .
v = np .
# define
(v - indu
o = activ
Print (o)
}
def activati
{
if v ≥ 0
retu
else :
ret
}
```

```
# fit data
an. fit (Y-train, Y-train)
# prediction
an. prediction (x.new).
```

UNIT - II

* Realising logic gates :-

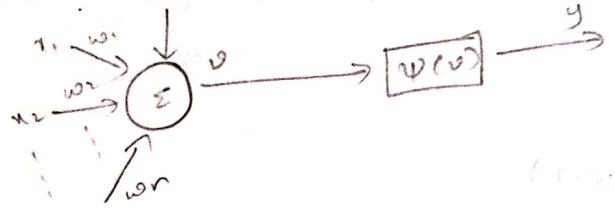
```
def perceptron_classifier ()
{
    # read inputs (x, w)
    # convert input into integer or float datatype & create
                                        list of inputs.

    # read bias.
    x = np. array (x)
    w = np. array (w)
    v = np. dot (x.T, w.T)
    # define activation function & pass v as parameter.
    (v - induced local field value).
    o = activation (v)
    Print (o)
}
def activation (v)
{
    if v ≥ 0:
        return 1
    else :
        return 0
}
```

* To implement AND, OR, ---- gated perceptron, without reading w values & bias, initialise w & b values in program.

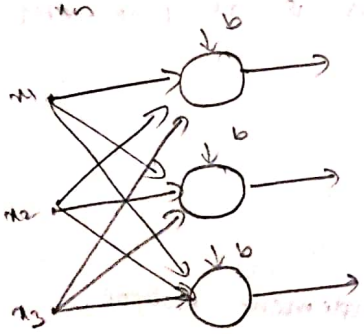
UNIT - III

Single layer perceptron :

* Perceptron is used to perform classification tasks.
* simple perceptron is used to perform binary classification only.
* Multilayer perceptron performs multiclass classification (complex classification)
* single layer perceptron is a perceptron with 1 (or) more artificial neurons organised in a single layer.



single layer perceptron with 1 neuron.



single layer perceptron with n no. of neurons.

* single layer perceptron has no hidden layer.
* simple perceptron can be used to perform binary classification.
* simple perceptron can classify data if it is linearly seperable (Data is classified based on decision boundary)
* If perceptron produces actual output y that is different from target response D then this situation is called error.

$e = D - y \Rightarrow e(n) = D - y(n)$

↳ error for iterative 'n',

* Reason for error is free parameters (these are initially initialised with random values).
* when there is an error, perceptron goes through learning mechanism (or) optimization procedure to minimize the error.

* for this iterat
* unconstrained

Generally ,

$v = \sum_{i=1}^{n} x_i w$

$= x^T \cdot w$

$y = \psi(v)$

$e = d - y$

↙ target output

$C(w) = \frac{1}{2}$

graph

* start with

$\nabla = \left[\frac{\partial}{\partial} \right.$

$\nabla C(w)$

$\nabla C(w)$

$\nabla C(w)$

* Gradie

* It is
* It is
* It

value

$w(n+1)$

gradient

informa

* for this iterative learning / optimization procedure is used.

* unconstrained optimization :-

Generally, $x = [x_1, x_2, \dots, x_n]$

$w = [w_1, w_2, \dots, w_n]$

b are inputs.

$$v = \sum_{i=1}^{n} x_i w_i + b$$

$$= x^T \cdot w + b$$

$$y = \varphi(v)$$

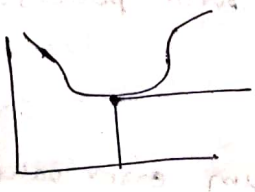
$e = d - y$ → actual output.

↙ target output

$$C(w) = \frac{1}{2} \sum_{i=1}^{K} e_i^2$$

⇒ minimizing errors results in minimizing cost function.

graph look like



global minimum error value position
(position where error value is min).

* start with $w^0, w^1, \dots, w^N$.

$$\nabla = \left[\frac{d}{dw_1}, \frac{d}{dw_2}, \dots, \frac{d}{dw_n} \right].$$

$$\nabla C(w) = \left[\frac{dC(w)}{dw_1}, \frac{dC(w)}{dw_2}, \dots, \frac{dC(w)}{dw_n} \right]$$

$$\boxed{\nabla C(w^*) = 0}.$$

$$\nabla C(w(n+1)) < \nabla C(w(n))$$

* Gradient descent algorithm (or) steepest descent algorithm:-

* It is unconstrained optimization algorithm.

* It is an iterative algorithm.

* It use gradient of cost function to determine update value for w.

$w(n+1) = w(n) + \Delta w(n)$

gradient of cost function is $g(n)$ used to get slope information and to calculate the new weight vectors.

$w(n+1) = w(n) + \Delta w(n)$

* Gradient of cost function is $g(n)$ used to get slope information and to calculate the new weight vectors.

$w(n+1) = w(n) - \eta g(n)$

η = learning parameter - It define a step size that needed to be taken.

- Its values is between 0 and 1 $\Rightarrow 0 \leq \eta \leq 1$.

* While computation, if error arises, then we calculate new weight function by subtracting $g(n)$ from old weight function and new weight function is used for further computations that minimizes the errors.

* Effect of n :-

* η value is very small :-

* It takes more time to move to global minimum error value position and algorithm traps in local minimum error value position.

* η value is large :-

* value position overshoot global minimum error points.

* Advantages :-

* convergence is very fast.

(movement from maximum error to minimum error value)

we should check

$\nabla c\,(w(n+1)) < \nabla c\,(w(n))$

use first order taylor series

$c(w(n)) + g(n)\,(w(n+1) - w(n))$

$w(n+1) = w(n) - \eta g(n)$ — (1)

$w(n+1) - w(n) = -\eta g(n)$

$\Delta w(n) = -\eta g(n)$

$\Rightarrow c(w(n)) + g(n) * -\eta g(n)$

$\Rightarrow c(w(n)) - \eta g(n) + g(n) *$

$\Rightarrow c(w(n)) - (\eta - x)\, g(n)$

$\Rightarrow c(w(n+1)) = c(w(n) - \eta (g(n))^2$

$\Rightarrow$ $c(w(n+1)) < c(w(n))$

compute g(

$\Rightarrow \frac{\partial c(w(n}{\partial w}$

$\Rightarrow \frac{\partial c(w}{\partial e}$

$\Rightarrow \frac{\partial e}{\partial w}$

$e = d - y$

$= d - \eta$

$= d - \psi$

$= d +$

$\Rightarrow \frac{\partial c(w}{\partial w}$

$\Rightarrow -\sum_{i=1}^{n} e$

$w(n+1) =$

$g(n) \rightarrow gr$

* we got

$\Rightarrow w(n+1)$

$w(n+1)$

Eg:- $x = \begin{bmatrix} 1 \\ \end{bmatrix}$

iteration

$w^0 =$

$v = \eta_1$

$= 1$

$= 1$

$y = 4$

$e =$

get slope
ectors.

e that

≤ 1.

calculate
old
used
errors.

um error
inimum

points.

rror value)

compute $g(n)$ => derivative of $C(w(n))$ w.r.to weight vector :

$$\Rightarrow \frac{\partial C(w(n))}{\partial w} = \frac{\partial C(w(n))}{\partial e} \cdot \frac{\partial e}{\partial w}$$

$$\Rightarrow \frac{\partial C(w(n))}{\partial e} = \frac{\partial}{\partial e} \frac{1}{2} \sum_{i=1}^{K} e_i^2 = \sum_{i=1}^{K} e_i$$

$$\Rightarrow \frac{\partial e}{\partial w} = \frac{\partial}{\partial w}(d - xw) = -x$$

$$e = d - y$$
$$= d - \varphi(v)$$
$$= d - \varphi(xw)$$
$$= d - xw \quad (\because \text{linear activation function})$$

$$\Rightarrow \frac{\partial C(w(n))}{\partial w} = \sum_{i=1}^{K} e_i \times (-x)$$

$$\Rightarrow -\sum_{i=1}^{K} e_i \times i \rightsquigarrow i\text{th input sample in a training set (denotes particular instance).}$$

$$w(n+1) = w(n) - \eta g(n)$$

$g(n)$ - gradient of cost function.

* we got $\frac{\partial C(w(n))}{\partial (w)} = -\sum_{i=1}^{n} e_i x_i$

$$\Rightarrow w(n+1) = w(n) - \eta x - \sum_{i=1}^{n} w_i x_i$$

$$\boxed{w(n+1) = w(n) + \eta \sum_{i=1}^{n} e_i x_i}$$

Eg: $x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ $w = \begin{bmatrix} 1.0 \\ 0.8 \end{bmatrix}$ $d = 1$ $b = 0.3$ $\eta = 0.03$

$v \geq 0 \Rightarrow 1$

$v < 0 \Rightarrow 0$

iteration 1:

$$w^0 = \begin{bmatrix} 1.0 \\ 0.8 \end{bmatrix}$$

$$v = x_1 w_1 + x_2 w_2 + b$$
$$= 1(1.0) + 2(-0.8) + 0.3$$
$$= 1 - 1.6 + 0.3 = -0.3$$

$$y = \varphi(v) = \varphi(-0.3) = 0$$

$$e = 1 - 0 = 1$$

Iteration 2 :

$$w' = w(0+1) = w(0) + \eta e x$$

$$= \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} + 0.03 [1] \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 1.03 \\ -0.74 \end{bmatrix}.$$

$$v = w_1 x_1 + w_2 x_2 + b$$

$$= (1.03)1 + (-0.74)2 + 0.3$$

$$= 1.03 - 1.48 + 0.3$$

$$= -0.15$$

$$\psi(-0.15) = 0$$

$$e = 1 - 0 = 1$$

Iteration 3 :

$$w^2 = w(1+1) = w(1) + \eta e x$$

$$= \begin{bmatrix} 1.03 \\ -0.74 \end{bmatrix} + 0.03 (1) \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 1.06 \\ -0.78 \end{bmatrix}$$

$$v = x_1 w_1 + x_2 w_2 + b$$

$$= 1.06 + 2(-0.68) + 0.3$$

$$= 1.36 - 1.36$$

$$= 0$$

$$y = \psi(0) = 1$$

$$e = 0$$

Q2 : Now,

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad w = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} \quad d = 1 \quad \eta = 0.5 \quad b = 0.3$$

iteration 1 :

$$w^0 = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix}$$

$$v = x_1 w_1 + x_2 w_2 + b$$

$$= 1.0 + 0.3 - 1.6$$

$$= -0.3$$

$$\psi(-0.3) = 0$$

iteration 2 :-

$$w' = w(0+1) = w(0) + \eta e x$$

$$= \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} + 0.5 (1) \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 1.5 \\ -0.8 \end{bmatrix}$$

$$v = x_1 w_1 + x_2 w_2 + b$$

$$= 1.5 - 0.8(2) + 0.3$$

$$= 0.2 .$$

$$\varphi(0.2) = 1$$

$$e = 0 .$$

* Drawbacks of gradient descent :-

* Trap into the local minima point.
* Convergence takes a long time when data size is large.

* Stochastic gradient descent :-

* uses only one data sample to compute the update.
* update rule

$$w(n+1) = w(n) - \eta e(n) x .$$

* slowly converges to the global minimum error point.

* Newtons method :-

* Approximation of quadratic cost function, then computes gradient of the approximated cost function.
* uses second order taylor series expansion to approximate the cost function.

$$\Delta w(n) = w(n+1) - w(n)$$

$$\because f(x) = f(a) + f'(a) f(x) + \frac{1}{2} f(x) H(x) f(x) + \cdots$$

$$\Delta w(n) = w(n+1) - w(n)$$

$$\simeq g(n) \Delta w(n) + \frac{1}{2} \Delta w(n) H(n) \Delta w(n) .$$

$H(n)$ → Hessian matrix.

$$H = \begin{bmatrix} \frac{\partial e}{\partial w_1^2} & \frac{\partial e}{\partial w_1 \partial w_2} & \frac{\partial e}{\partial w_1 \partial w_3} & \cdots & \frac{\partial e}{\partial w_1 \partial w_n} \\ \frac{\partial e}{\partial w_2 \partial w_2} & \frac{\partial e}{\partial w_2^2} & \frac{\partial e}{\partial w_2 \partial w_3} & \cdots & \frac{\partial e}{\partial w_2 \partial w_n} \\ \vdots & & & & \\ \frac{\partial e}{\partial w_m \partial w_1} & \frac{\partial e}{\partial w_m \partial w_2} & \cdots & & \frac{\partial e}{\partial w_n^2} \end{bmatrix}$$

$$\frac{\partial f}{\partial \Delta w(n)} \qquad g(n) + H(n)\,\Delta w(n) = 0$$

$$H(n) = \frac{-g(n)}{\Delta w(n)} \Rightarrow \Delta w(n) = \frac{-g(n)}{H(n)}$$

$$\Delta w(n) = -g(n) * H^{-1}(n)$$

$$w(n+1) = w(n) + \Delta w(n)$$

↳ update amount.

$$\boxed{w(n+1) = w(n) - \eta g(n) H^{-1}(n)}$$