Machine Learning

Unit - I

Learning - Types of Machine Learning - supervised Learning - The Brain and the Neuron - Design a Learning system - perspectives and Issues in Machine Learning - concept Learning Task - concept learning as search - Finding a Maximally specific Hypothesis - Version spaces and the Candidate Elimination Algorithm - Linear Discriminants : - Perceptron - Linear separability - Linear Regression.

Unit - II

Multi-layer perceptron - Going forwards - Going Backwards : Back propagation Error - Multi-layer perceptron in practice - Examples of using the MLP - Overview - Deriving Back-Propagation - Radial Basis Functions and Splines - concepts - RBF Network - Curse of Dimensionality - Interpolations & Basic Functions - support Vector Machines.

Unit - III

Learning with trees - Decision Trees - Constructing Decision Trees - Classification & Regression Trees - Ensemble Learning - Boosting - Bagging - Different ways to combine classifiers - Basic statistics -

Unit - IV

Dimensionality Reduction - Linear Discriminant Analysis - principal component Analysis - Factor Analysis - Independent component Analysis - Locally Linear Embedding - Isomap - Least squares optimization Evolutionary Learning - Genetic algorithms - Genetic Offspring :- Genetic operators - Using Genetic Algorithms.

Unit - V

Reinforcement Learning - Overview - Getting Lost Example Markov Chain Monte Carlo Methods - sampling - Proposal Distribution - Markov chain Monte Carlo - Graphical Models - Bayesian Networks - Markov Random Fields - Hidden Markov Models - Tracking Methods.

Machine Learning

⇒ ML is a branch of AI that enables algorithms to uncover hidden patterns within datasets, allowing them to make predictions on new, similar data without explicit programming for each task.

⇒ Machine learning's impact extends to autonomous vehicles, drones, robots.

⇒ This approach marks a break through where machines learn from data examples to generate accurate outcomes, closely interwined with data mining & data science

The field of machine learning was founded by computer scientist "Alan Turing" in the "1950s"

Learning:

Learning is a phenomena and process which has manifestations of various aspects. Learning Process includes gaining of new symbolic knowledge and development of cognitive skills through instructions and practise. It is also discovering of new facts and theories through observation and experience.

Machine Learning :

ML is about making computers modify or adopt their actions so that these actions get more accurate where accuracy is measured by how well the choosen actions reflect the correct ones.

Computer Program :

A computer program is a set to learn from Experiences "E" with respect to some classes of Tasks "T" and performance measure "P" improves with Experience "E"

example : A computer program that learnt play "checkers" Might improve its performance as Measure by its ability to win at the classes of Task involving playing checkers gam through experience obtained by playing game by against itself.

① Checkers Learning Problem :

Task 'T' : playing checkers

Performance 'P' : Practise of games won agai openents.

Experience 'E' : playing Practise against itse

② A Robot driving learning problem :

Task 'T' : Driving on public foreline highway using vision sensors.

Performance 'P' : Average distance travelled before and other (as judged by human obseive)

Experience 'E' : A sequence of images and steaing command recorded while obseiving a human driver.

Need of Machine Learning :

1⇒ Main goal of ML is to devise leaning algorithm that do the learning automatically without human interaction or assistance.

2⇒ The ML paradism/methodology can be viewed as programming by examples.

⇒ To develop competitional models of human leaning process and perform computer simultaneously.

3→ The goal of ML is to build computer system that can adopt and learn from experiences.

4→ ML Algorithms can figure out how to perform important tasks by generalising from examples.

5→ ML provides business inside and intelligence decision makes or provided with greater insights into their organisations.

6⇒ ML are discussed relationships blw the variables of a system from direct sample of the system.

Reasons for using Machine Learning:

→ Some tasks can't be defined well except by examples.

⇒ Relationships and co-relations can be hidd, large amount of data.

⇒ Human design us often produces Machine that do not work as well as desired in the environment in which they are used.

⇒ The effort of knowledge available about certain tasks might too large for explicit encoding by humans.

⇒ Environment change time to time.

⇒ New knowledge about tasks is consistently being discovered by humans.

Applications of Machine Learning:

⇒ Learning to recognise spoken words

⇒ Learning to drive an automonus vehicle.

⇒ Leaning to classify new structures.

⇒ Leaning to play games.

⇒ Spoken Language understanding.

Types Of Machine Learning :

There are 4 types of ML

* Supervised Leaning
* Unsupervised Leaning
* Reinforcement Learning
* Evolutionary Leaning

Before classifying the info. it might be useful and based on that form a training dataset Leaning as meaning getting better at some task. Through practise so according to leaning to judge how much it correct or wrong responses. Finally we might not have any correct we just want the algorithm to find inputs that have something in common.

*Supervised Learning :

A training set of examples with correct responses (target) is provided and based on this training set. The Algorithm generalises to respond correctly to all possible inputs. This is also called Learning from examples.

* Unsupervised Learning :

correct responses are not provided but instant the algorithm tries to identify similarities blw the input. so that inputs that have something in common are categorised together to statistical approach to unsupervised leaning is known as density estimation.

* Reinforcement Leaning :

The both supervised & Unsupervised leaning combined in this leaning. The algorithm gets told when the answer is wrong but doesnot get told how to correct it. It has to explore and write try out diff. possibilities until it works out how to get the answer wright / correct. Reinforcemen Leaning is sometime called leaning with a critic because of this monitor that store the answer but doesnot suggest improvements.

* Evolutionary Learning :

Biological Evolution can be seen as a leaning Process. Biological organisations adopt to improve their seuival rates and chance of having offspring in their environment. We will look at how we can model this in a computer using an idea of fitness. which correspondence to a score for how good the correct soln is.

Real Time Examples

Supervised Learning :

⇒ Image classification
⇒ spam Detection
⇒ sentiment Analysis
⇒ House price prediction
⇒ Medical Diagnosis

Unsupervised Learning :

⇒ customer segmentation
⇒ Anomaly Detection
⇒ Market Basket Analysis
⇒ Document clustering
⇒ Dimensionality Reduction

Reinforcement Learning :

⇒ Autonomous vehicles
⇒ Game playing
⇒ Robotics
⇒ Portfolio Management
⇒ Personalized Recommendations

Evolutionary Learning :

⇒ Art & Music Generation
⇒ structural optimization
⇒ Drug Discovery
⇒ Vehicle Routing Problem
⇒ Antenna Design

Supervised Learning :

The I/p and targets are should produce o/p (Sensable outputs)

⇒ The I/p are x_i & target is T_i

⇒ The 'i' index suggests that we have lot of pieces of data indexed by i running from 1 to some uppelimit N

⇒ The inputs & targets are written in "bold phase font" to signify vectors. Since eac phase of data has values for several dif features.

⇒ Every possible pieces of i/p data should be considered. Remaining no need for ML a all.

Types of Supervised Leaning :

* Regression
* Classification

The above two learning are inherit real world process. It is heard hard to speci generalisation.

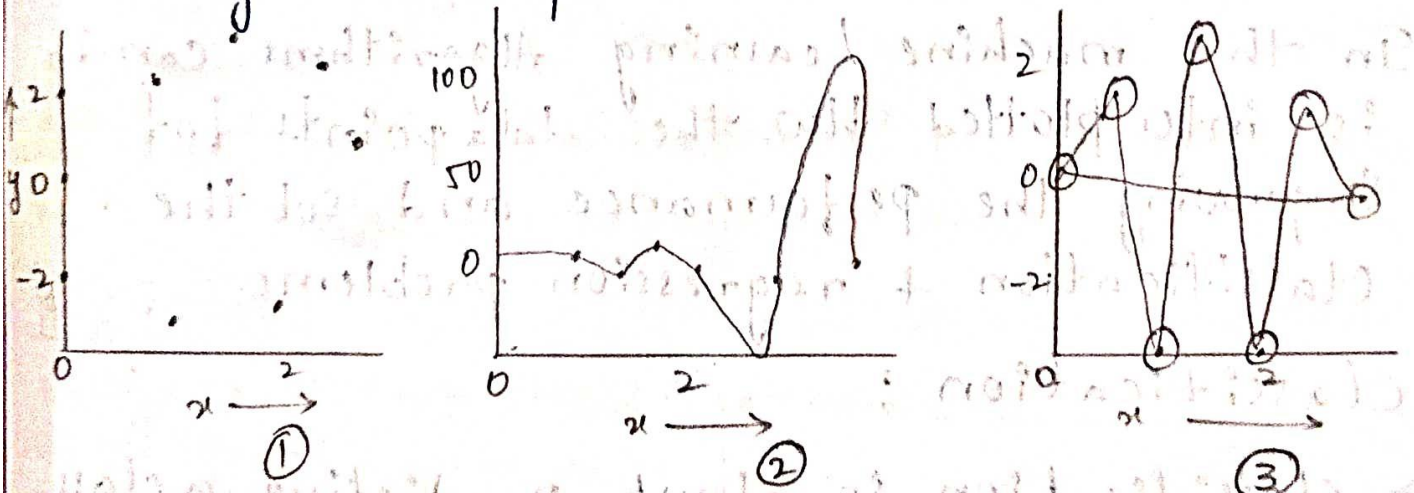
Regression :

It is about predicting a qualit quanti or contineous values

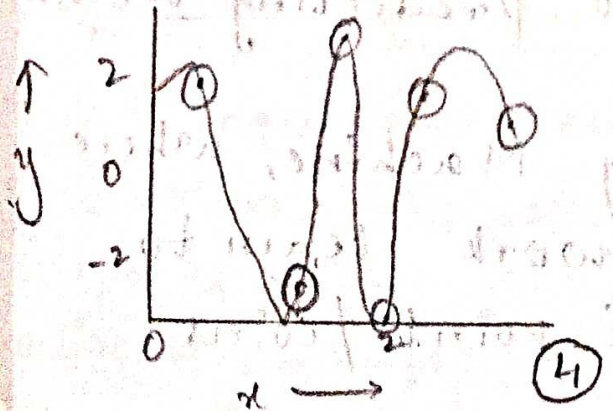
Ex: Where $x = 0.44$

x	y
0	0
0.5	1.5
1.04	-2.5
1.56	-2.54
2.6	-1.5
3.14	0

When $x = 0.44$ here x, t is there if not exist. we consider as our target 't' is 'y'. since the value $x = 0.44$ is not in the example. But to find/predict somewhere what value it has. for that the values comes from some sort function, find out what the fun. is. The o/p value 'y' for any given value of 'x' - this is known as regression problem in statistics.



① ② ③



④

- The 2nd img show a plot of 7 values of x & y in table. while the other plots shows the different attempts to fit a curve through the data points

- The 3rd img shows that 2 possible answers found by using st. lines to connect up the points.

And tries to use a cubic function

- The final image shows that sin x fun (sin x is combination of +ve & -ve values) so finally last img is the correct answer compaing to remaining images. That means it gives a better soln to test how well they generalise.

- In the machine Leaning Algorithms can do is into plotted blw the data points for improving the performance and sol the classification & regression problems.

Classification :

⇒ classification is about predicting a class (or) discrete values

Ex: Consider a vending Machine, where we use a linear network learn to recognise all the diff points / coins.

we train the classifier to recognize all coins. Either if it is circular or different shapes but it gives the decision based on boundaries. That means a set of st lines (or) not ① ② ③ ④

(10) (20) fig: coins

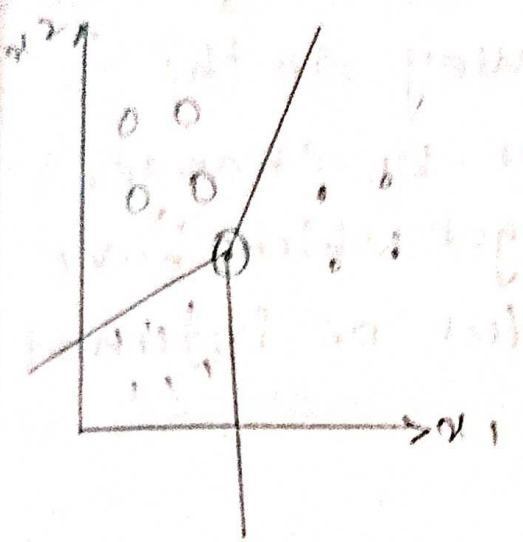


fig: Linear classification

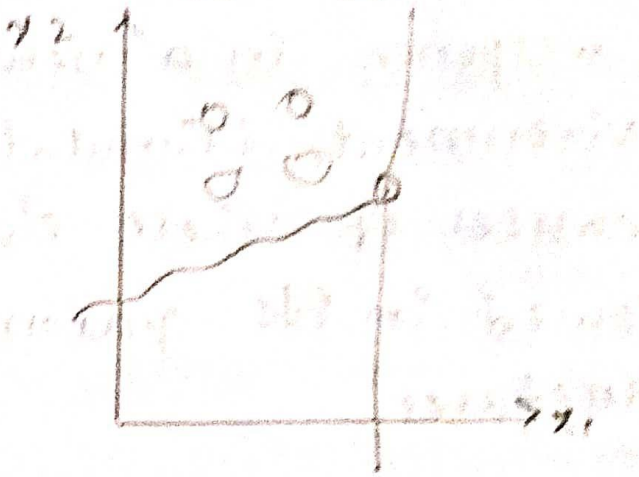


fig: Non linear classification

The Brain and neuron

Neuron Network:

Large no. of interconnection neurons called neuron networks

Neuron:

Nerve cells called neurons

Objectives of Neuron Networks:

The acquired new environment - new situation - new data when expose to such kind of new situation - new data how it adjusts, drafts itself and how it response

still get desired result

* The neuron network is stimulated by environment.
* It experiences changes in its parameters, internal structures as a result of stimulation
* It responce in a new way to the environment stimulation or changes because of these changes which have occured in its parametus or internal structures.

Dendroies :

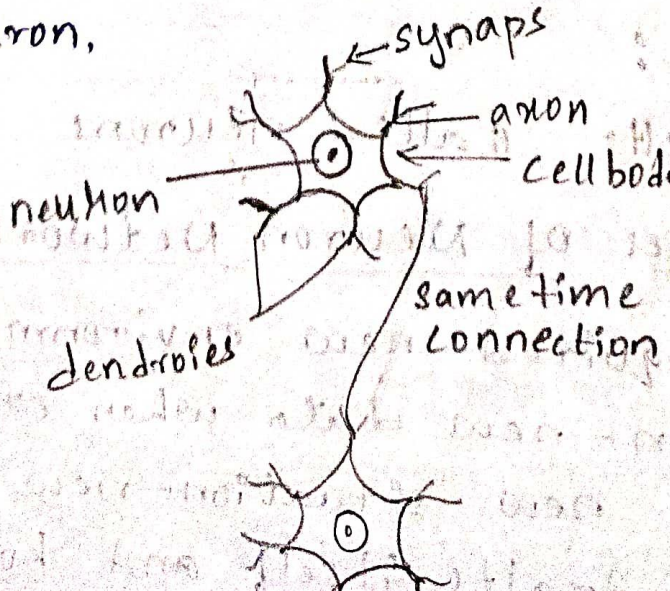
Small neves which connected to cellbody

Synapses :

Small bulb like organ

Axon :

Single long connection blw one neuron to another neuron.

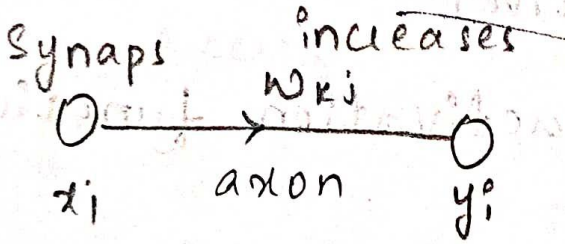


Hebb's Rule :

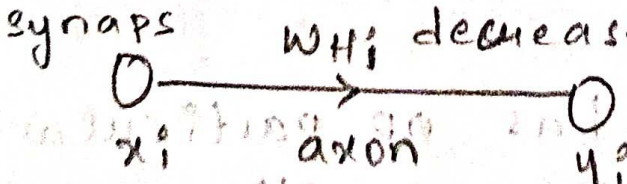
Hebbian learning rule is one of the earliest & simple learning for neuron network. Hebb in the year of 1945.

Rules :

① If 2 interconnected neurons on either side of a synaps are both on fired (or) activated at the same time (synchronous) then the synaptic way blw them should be increased.



② If 2 neurons on either side of synaps on are fired (or) activated different times (Asynchronously) then weight of the synoptic connection is decrease.



Mechelloch - Pitts Rule :

The Mecholloc - Pitts neuron was the earliest neuron network. It is usually called as M-P model

It was discovered in 1943

The neurons in M-P model are connected by directed weighted paths.

The weight associated can be of 2 types
1. excitatory (positive)
2. Inhibitory (Negative)

⇒ It uses binary activation function for each neuron.

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

where θ = thershold value.

⇒ The threshold places an important role in M-P model

⇒ The M-P neuron has no particular learning (or) training algorithm.

⇒ They are generally used for implementing logical fun.

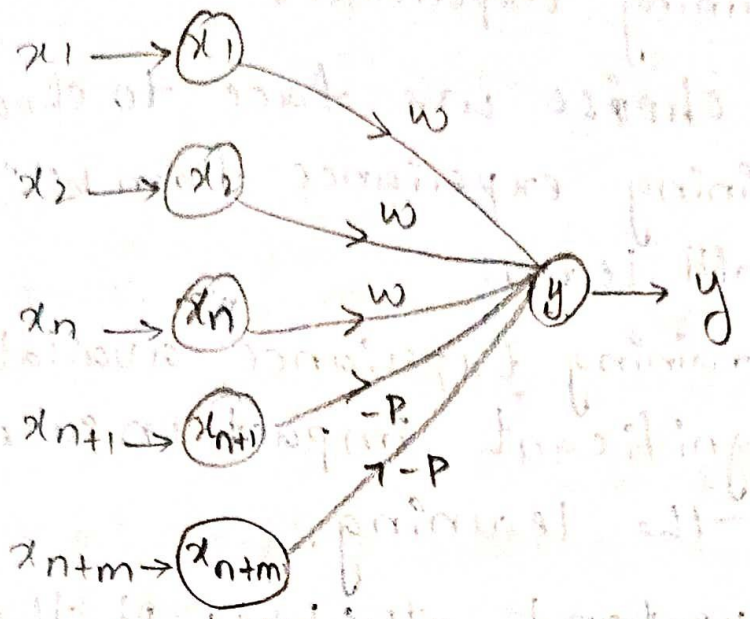


fig: M-P model

Design a learning system

To get a successful learning system it should be designed for a proper design for that we follow 5 steps. In order to view some of the basic design issues and approaches to machine learning let us consider a design program to learn to play checkers game with the goal of world tournament. we adopt the peformance measue i.e the persent of game to win this world tournament

1. Choosing the training experience
2. " the target function
3. " the representation for the target fun.
4. " a fun Approximation Algorithm.
5. The final Design.

1) Choosing the training Experiances

The first design choice we face to choose the type of training experiance from which how the sys will learn.

⇒ The type of "training Experiance" available can have a significant impact on success or failure of the learning.

⇒ A second importand attribute of the training experiance is the "Degree" to which the learner controls the sequence of training examples.

⇒ A 3rd imp attribute of the training experiance is how well it represents the distribution of examples over which the final sys performance 'P' must be measured.

⇒ In order to complete the design of learning system we must choose

* The exact type of knowledge to be learned
* A representation for this target knowledge
* A learning mechanism.

2) Choosing the target function:

The next design choice is to design/ determine exactly what type of knowledge will be learn. And how this will be used by the performance program.

→ Let us begin with a checkers playing program that can generate the target moves from any board state. The program needs only to learn how to choose the best moves from among these target moves.

→ This learning task is representively large class of task for which the legal moves.

→ That define some large search spaces are known as priors (important) but for which the best search strategy is not known.

⇒ The fun choose move & use the notation "B → M" to indicate that this fun accepts as I/p any board from the set of legal board. From the set of legal board states "B". And produces as O/p some move from the set of legal moves "M"

⇒ Let us define the target value "V(b)" for any arbitary board state "b in s" as follows

* If b is the final board state i.e won than V(b) = 100

* If b is the final board state that is lost the V(b) = -100

* If b is the final board state i.e drawn then V(b) = 0

* If b is a not a final state in the game then $v(b) = v(b')$

so, where b is the final best state that can be achieved from b and playing optimally untill the game win.

3) Choosing the representation for the target function:

Now that we have specified that target function 'v'. we must choose a representation that the leaning problem will use to describe the function '$\hat{v}$' that it will learn for any given boaud state the function '$\hat{v}$' will be calculated as leaner combination of the following boaud features.

x_1: The no. of black pieces on the boaud

x_2: The no. of red pieces on the boaud

x_3: The no. of black kings on the board

x_4: The no. of red piece on the board

x_5: The no. of black piece treated by red

x_6: The no. of red pieces treated by black.

Partial design of checkers leaning algorithm or Program.

1) Task: (T)

Playing checkers

2) Performance measure (P):

percentage of game won in the world tournament.

3) Experience (E):

Games played against itself

Target function :(V)

Based on R

$$O(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

Target function Representation:

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

4) Choosing a function approximation Algorithm:

In order to learn the target fun (V) we request a set of training examples each describes a specific board state (b) & the training value 'V train(b)' for b, in other words, each training example is an ordered pair of the form $\{b, V_{train}(b)\}$ for that

* Estimating the training value

The Ambuiguity inheit in estimating training values for intermediate boaud states, one simple approach has been found to be successful.

$$V_{train}(b) \leftarrow \hat{V}(Succesi(b))$$

* Adjusting the weights:

one common approach is to define the best hypothesis or set of weights as that which minimises the error b/w the training values & the predicted values by the hypothesis ($\hat{V}$)

$$E = \sum_{(b, V_{train}(b))} \left(V_{train}(b) - \hat{V}(b)\right)^2$$

So, least mean squares is one such algorithm.

5) The finalised Design:

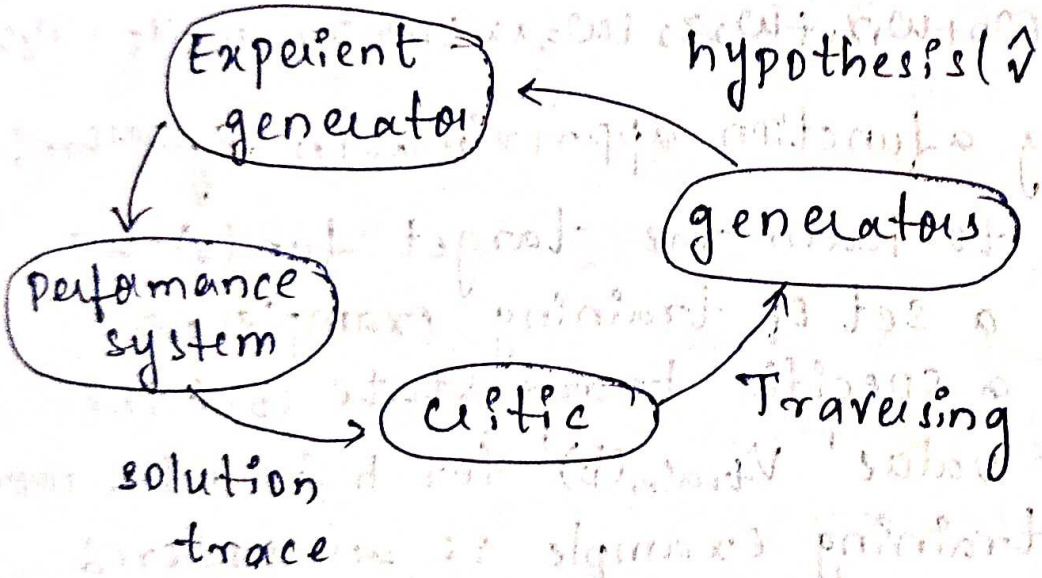


fig: Final design of checkers Learning system

⇒ Final design of checkers Learning sys can be naturally described by 4 diff. modules/ Program modules that represents the central component in many learning system.

⇒ The Experimental generator gives the new Problem to a Learning system.

⇒ The performance system is a module that must solve the given performance task. In these case playing checkers by using the learned target function (s)

⇒ The critic takes as input, the history or trace of the game and produces the output set of training examples of the target function.

⇒ The generalizer takes as input a training example and produces a o/p hypothesis, that is it estimates the target function.

Perspectives and Issues in ML:

Perspectives:

1) One useful perspective of ML is that it involves searching a very large space of possible hypothesis to determine one that best fits the observed data & any trial knowledge held by the learner.

Ex: consider the space of the hypotheris that could in principles the o/p by the above checkers learner.

2) The LMS (least mean square) for fitting weight achieves these goals by incrementally tuning the weight, adding each weight, each time the hypothesized evolution function predicts a value that differs from the evolution value.

3) Their algorithm works well when the hypoth. representation considered by the learner defin. a continuously. parameterized space of potent. hypotheses - that means the learning as a searc. problem with confidence we can have a hypothesis consistent with a training data.

Issues in Machine Learning:

1) What learning algorithm is used.
2) How much training data is sufficient.
3) When & How trail knowledge can guide the learning process.
4) What is the best strategy for choosing a next training experience.
5) What is the best way to reduce the learning task to one or more function approximation problems.
6) How can the learner automatically alter its representation to improve its learning ability.

Concept Learning:

Inferring a boolean valued fun from training examples of its i/p x o/p

Ex: The general concepts are categories such as bird, car situations in which I should study more in order to pass the exam etc. Each such concepts can be viewed as describing some subset of objects are events defined over a larger set that means function

defined overall animals whose value is true for birds and false for other animals.

→ We consider the problem of automatically inferring the general definition of some concept, given examples labelled as members or non members of that concept.

Concept learning task:

→ If some instances x satisfies the all the constraints of then h satisfies x or c(x) = 1 we examine i.e $[h(x) = 1]$ → In general any concept learning task can be described by the set of candidate hypotheses considered by learner and the set of training example

→ The representations of the concept learning task is:

1)-Any value is acceptable for this attribute it is indicated by "?"

2) Specify a single required value for an attributes or no value is accepted "∅".

Ex:

S.No	sky	Airtemp	Humidity	wind	water
1.	Sunny	warm	Normal	Strong	warm
2.	Sunny	warm	high	strong	warm
3.	Rainy	cold	high	strong	warm
4.	Sunny	warm	high	strong	cool

S.No	Forecast	Enjoy sport
1.	Same	Yes
2.	Same	Yes
3.	Change	No
4.	Change	Yes.

+ve & -ve training examples for the target concept sport attribute enjoysport indicates whether or not enjoy the

enjoy sport for an arbitary, day based on values of its other attributes. Each hypothesis be a vector of 6 constraints they are sky, Air Temp, Humidity, wind, water, forecast, The hypothesis enjoys his sports only an cold days with high humidity is represented by the expression.

The most general hypothesis that every day is a +ve example is represented by.

(?, ?, ?, ?, ?, ?)

The most specific +ve hypothesis that day is +ve example is represented by (∅, ∅, ∅, ∅, ∅, ∅)

The enjoysport concept learning fact requires learning the set of days for which enjoysport = Yes.

Notations

→ The set of items over which the concept is defined is called the set of instances indecated by "x"

→ All possible days in the "x" attributes are sky, airtemp, Humidity, 'c'. c can be any boolean valued function defined over the instanced x. $c: X \rightarrow \{0, 1\}$

→ In our current example the target concept corresponds to the value of attribute enjoyspork i.e $c(x) = 1$ If enjoy port is = (1) Yes and $c(x) = 0$ NO.

Given:

→ Instances X: possible days each described by the attributes

Sky (with possible values Sunny, Rainy)

Airtemp (with possible values warm, cold)

Humidity (" " " Normal, High)

Wind (" " " Strong, weak)

Forecast (" " " Same, Change)

→ Hypothesis H: Each hypothesis is described by a conjunction of constraints on the attributes Sky, Airtemp, Humidity, Wind, Forecast. The constraints may be ?

Target concept C: EnjoySport : X → {0,1}

Training examples D: A set of examples of the target function

Determine: A hypothesis h in H such that h(x) = c(x) for all x in X

Inductive Learning Hypothesis:

Any hypothesis found to approximating the target fun well over a sufficient large set of training examples will also approximate the target fun well over other unobserved examples (unseen)

(or)

Concept learning involves determining a mapping from a set of I/p variables to a boolean value such methods are known as Inductive learning methods.

Generalisation:

- If a fun can be found, which maps training data to correct classifications, then it will also well for unseen data this process is known as generalization.

Concept Learning as Seach:

- Concept learning can be viewed as the task of seaching through a large space of hypothesis implicitly defined by the hypothesises representation.
- The goal of this seach is to find the hypothesis that best fits the training examples.

General to Specific Ordering of Hypothesis:

→ many algorithms for concept learning organise the search through the hypothesis space by relaying (depending) on a very useful structure that exist for any concept learning problem. i.e general to specific ordering of hypothesis.

Advantages:

⇒ Naturally occuring structure over the hypothesis space.

⇒ Infinite hypothesis spaces.

Example:

Consider 2 hypothesis,

$h_1 = \{sunny, ?, ?, strong, ?, ?\}$

$h_2 = \{sunny, ?, ?, ?, ?, ?\}$

Let "h_1 & h_2" be the set of instances classified +vely because h_2 have only fewer constraints in H. classifies more instances as +ve in "h_2". Infact any instance classified +ve by h_1 will also be classified +ve by h_2. That means we can say that h_2 is more general than h_1.

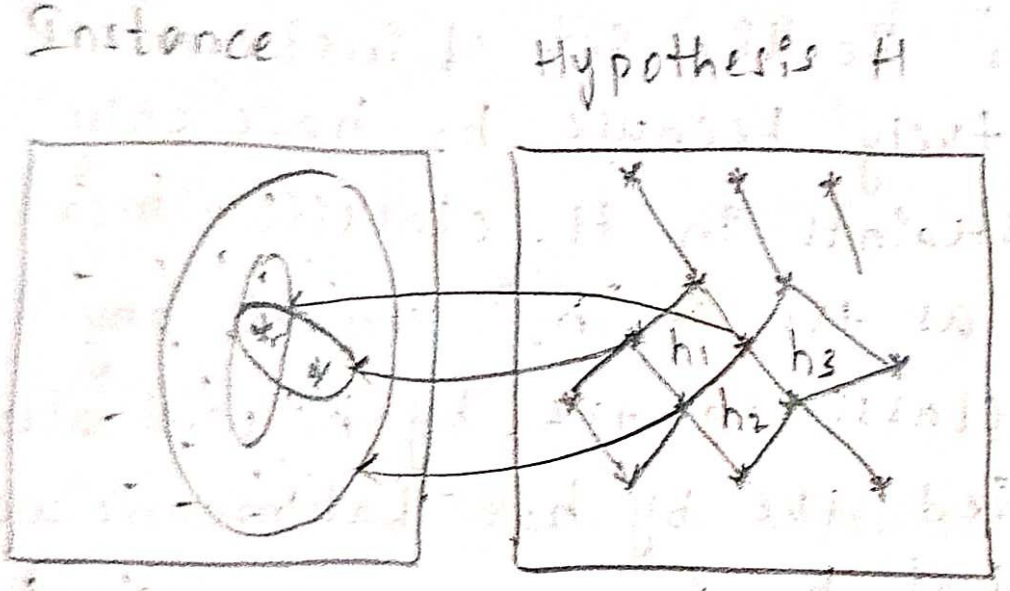
One leaning method is to determine the most specific hypothesis that matches all the training data.

"More - General - Than (or) Equal-to" relation h_1 & h_2 has to be 2 boolean value fun defined over "x" then h_2 is most general than (or) equal to h_1 ($h_2 \geq h_1$) written in expression

if and only if any expression that satisfi
h_2 also satisfies h_1

$\Rightarrow$ h_1 is more general than h_2
$(h_1 \geq h_2)$ if & only if $h_1 \geq h_2$ is true
and $h_2 \geq h_1$ is false we also say
h_2 is more specific than h_1

condition: $h_j \geq h_k \; \forall x \in X, h_k(x) = 1$
$h_j(x) = 1$



x_1 = [Sunny, Warm, high, strong, cool, Same]

x_2 = [Sunny, Warm, high, light, Warm, same]

h_1 = {Sunny, ?, ?, Strong, ?, ?}

h_2 = {Sunny, ?, ?, ?, ?, ?}

h_3 = {Sunny, ?, ?, ?, cool, ?}

Finding a maximally Specific Hypothesis (find - s)

- The key propeuty is the hypothesis space disciibed by conjuctions of attribute constrains. Find-s is guaranted to o/p the most specific hypothesis with in H i.e consistent with the +ve training examples. It's final hypothesis will also be consistent with the -ve examples provided the tauget concept. Is contain in H and provided the training examples aue correct.
- The find-s algorithm stauts fiom most specific hypothesis and generalized it considering only +ve examples.
- This algorithm ignores -ve examples as long as the hypothesis phase contains a hypothesis that disciibes the true taeget concepts. And the training data contains no errors, Ignoring -ve examples doesnot cause any problem.
- Find-s algorithm finds the most specific hypothesis within H. i.e consistent with the +ve training examples. The final hypothesis will also be consistent with -ve Examples if the ~~concept~~ correct

target concept is in H and the training examples are correct.

Algorithm

step ①

Initialize 'h' to the most specific hypothesis in "H"

$$h = \{\phi, \phi, \phi, \phi, \phi, \phi\}$$

step ②

for each +ve training instance "x"

for each attribute constrain "a_i" in "h"

⇒ if the constrain a_i is satisfied by x then do nothing.

⇒ else replace "a_i" in "h" by the next most general constrain that is satisfied by "x"

step ③ O/P hypothesis "h"

Example:

Example	Sky	Air Temp	Humidity	Wind	Water	forecast	enjoy sport
1	Sunny	warm	Normal	strong warm	warm	same	yes
2	sunny	warm	high	strong	warm	same	yes
3	Rainy	cold	high	strong	warm	change	NO
4	sunny	warm	high	strong	cool	change	yes

Sol:

step ①

Initialize "h" to the most specific hypothesis in "H"

$h_0 = \{\phi, \phi, \phi, \phi, \phi, \phi\}$

step ②

for each +ve training instance "x"
for each attribute constrain "a_i" in "h"
if the constrain a_i is satisfied by x then do nothing, else replace "a_i" in "h" by the next most general constrain that is satisfied by "x".

$h_0 = \langle \phi, \phi, \phi, \phi, \phi, \phi \rangle$

$x_1 = \langle$sunny, warm, Normal, strong, warm, same$\rangle$

Iteration I:

$h_1 = \langle$sunny, warm, Normal, strong, warm, same$\rangle$

Iteration II:

$x_2 = \langle$sunny, warm, high, strong, warm, same$\rangle$

$h_2 = \langle$sunny, warm, ?, strong, warm, same$\rangle$

Iteration III:

Ignore

$h_3 = \langle$sunny, warm, ?, strong, warm, same$\rangle$

Iteration IV:

x_4 = <sunny, warm, high, strong, cool, change>

h_4 = <sunny, warm, ?, strong, ?, ?>

Step ③ :

O/p hypothesis "h"

h_4 = <sunny, warm, ?, strong, ?, ?>

Version Space :

VS is a set of all hypothesis that are consistent with the training examples.

⇒ Vs denoted $VS_{H,D}$ with respect to hypothesis space 'H' & training example 'D' is the subset from 'H' consistent with the training example in 'D'.

⇒ A Vs is hierarchical representation of knowledge that enables you to keep track of all the useful info. supplied by a sequence of learning examples without remembering any of the examples.

$$VS_{H,D} = \{h \in H \mid \text{consistent}(h, D)\}$$

→ The version space method is a concept learning process accomplished by managing multiple models within a version space.

Limitations in Find-S Algorithm:

→ Find-S o/p a hypothesis from H that might fit the training data equally well.

→ The key idea in the candidate algorithm is to o/p description of the set of all hypotheses consistent with the training examples.

Representation of Version Space:

Definition:

A hypothesis 'h' is consistent with a set of training examples "D". If and only if $h(x) = c(x)$ for each example $(x, c(x))$ in D. [This "x" is instances, $c(x)$ is the target]

Notation is

$$\text{consistent}(h, D) = (\forall (x, c(x)) \in D),\ h(x) = c(x)$$

LIST-THEN-Eliminate Algorithm:

To represent the version space simply to list all of its members. The list-then eliminate algorithm first initialize the version space to contain all the hypothesis in "H", then eliminate any hypothesis found

inconsistent with any training examples.
→ The list-then-eliminate algorithm can be applied when ever the hypothesis space H is finite.

Algorithm:

1. Version space ← a list containing every hypothesis in 'H'
2. for each training example, (x, c(x)) remove from version space any hypothesis 'h' for which $h(x) \neq c(x)$
3. Output the list of hypothesis in version space.

Candidate Elimination Algorithm:

A more compact representation of versionspace:

A Candidate elimination Algorithm works on same algorithm i.e List-than-eliminate.
⇒ It employes much more compact representation of version space.
→ In the previous find-s alg. o/p of the hypothesis is < Sunny, warm, ?, strong, ?, ? > in that 6 diff hypothesis from h that are consistent with these training examples
⇒ They consistitute the V.S relatives to these set of data & these hypothesis.

→ The possible representations are G & S.
- The general boundary 'G' with respect to the hypothesis space H & training data D is the set of maximally general members of H consistent with D.
- The specific boundary 'S' with respect to the hypothesis space H & training data D is the set of minimally general members of H consistent with D.

Candidate Elimination Algorithm:

This alg. computes the V.S containing all hypothesis from H that are consistent with an observed sequence of training examples.
The key diff blw consistent & inconsistent of the data satisfies $h(x) = c(x)$.
- for example 'x' is said to satisfy hypothesis h when $h(x) = 1$. When ever x is a +ve (or) -ve example of the target concept however such an example consistent with "h" depends on the target concept in particular, whether $h(x) = c(x)$.

Algorithm:

⇒ The candidate elimination algorithm

$G_0 = < ?, ?, ?, ?, ?, ? >$

$S_0 = < \phi, \phi, \phi, \phi, \phi, \phi >$

Step ①

Initialize G to the set of maximally general hypothesis in H

Step ②

Initialize S to the set of maximally specific hypothesis in H

For each training example D, do

If D is +ve:

⇒ Remove from G any hypothesis inconsistent with d.

⇒ For each hypothesis s in S that is not consistent with D

⇒ Remove s from S

⇒ Add to S all minimal generalizations h of s. such that his consistent with d and some member of G is more general than h

⇒ Remove from s any hypothesis that is more general than another hypothesis in S

If 'd' is -ve:

⇒ Remove from S any hypothesis inconsistent with d

⇒ For each hypothesis g in G that is not consistent with d

⇒ Remove g from G

⇒ Add to g all minimal Specialization h of g. such that

⇒ h is consistent with d, & some member of s is more specific than h.

⇒ Remove from g any hypothesis that is less general than any other hypothesis in G.

Linear Discriminants - Perceptrons

Limitations of the Mcculloch-pitts rule

⇒ Neurons doesn't actually response as threshold devices but produces a graded o/p in contineous way.

⇒ The neurons are not updated sequentially according to a computer clock.

⇒ The weights can be +ve (or) -ve within the brain but with the mcculloch-pitts rule neurons the weights can change

from +ve to -ve vice versa.

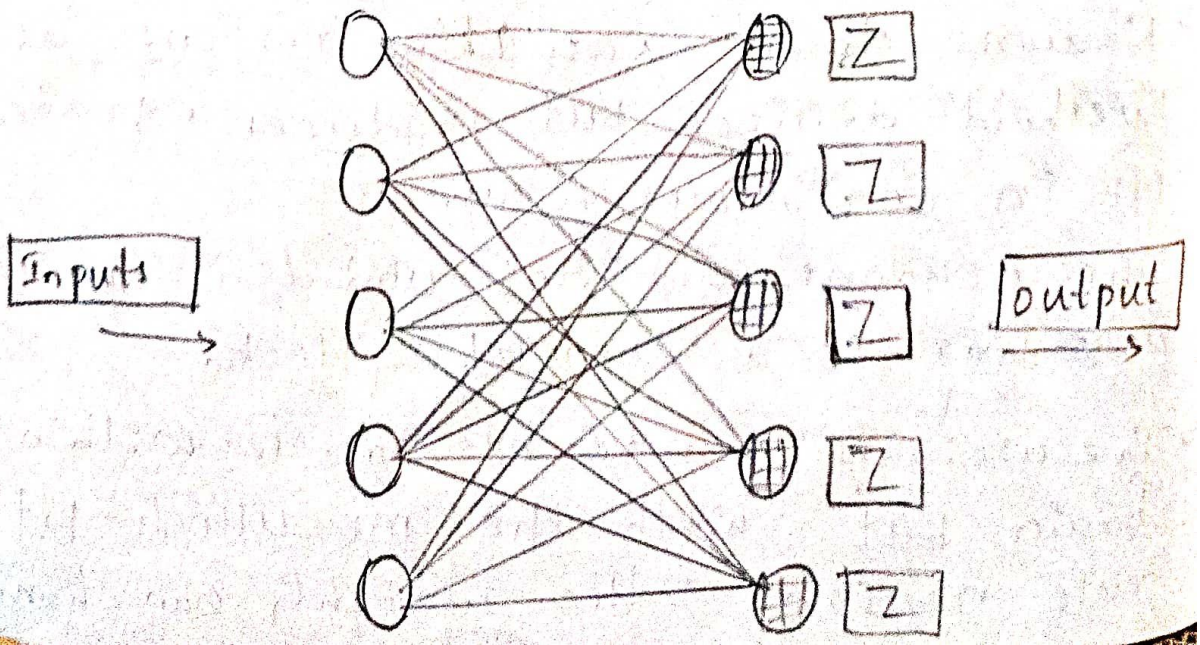
Perceptron :

The I/p to a real neuron are not necessarily summed linearly - they may be nonlinear summations.

However the most noticable diff is that real neurons do not output a single o/p response. for that to make a neuron learn change the weights & thresholds of the neurons using linear discriminant. That means the neurons are discriminated linearly using single access and single o/p.

Perceptron :

The perceptron is nothing more than a collection of Mcculloch & pitts neurons together with a set of inputs and some weights to fasten the inputs to the neuron.



- The neurons in the perceptron are completly independent of each other. The no. of inputs is the same as the no. of neurons in the above diagram.
- But this doesnot have to be the case, in general there will be "m" inputs & "n" neurons the perceptron to learn to reproduce a particular target i.e, a pattern of firing & nonfiring neurons for the given inputs.
- The Mcculloch & pitts neuron, the weights were labelled as "w_j" $\Rightarrow$ "j" indicates the index running over the no. of inputs.

$$[\Delta w_{ik} = -(y_k - t_k) \times x_i]$$

- for a neuron that is correct, we are happy does not use perceptron rule. But any neuron that fired when should not have done are failed to fire when it should needs to have it changed the weights. for that

$$\boxed{\Delta w_{ik} = -(y_k - t_k) \times x_i}$$

where,

Δw_{ik} - it produces the weights that are intersected in w_{ik}. where 'i' runs from 1 - m so, wrong as k.

$y_k - t_k$ - It computes the difference blw the output "y_k" which is what the neuron did and the target for that neuron. "t_k" which is what the neuron should have done. This is a "possible error function".

⇒ The learning rules need to be finished but we need to decide how much to change the weight by.

⇒ This is done by multiplying the value by a parameter called the learning rate is 'η' (neta).

⇒ The value of the learning rate decides how fast the n.w learns. So, it gets a little subsection of its own (next) but first lets write down the final rule for updating a weight. $w_{ij} \leftarrow w_{ij} - \eta (y_j - t_j) x_i$

→ The weights changes a lot when ever there is a wrong answer which tends the network unstable. So that, it never settles down. It will more stable & resistent to noise and inaquasis in the data then we use the modean learning rate is $0.1 < \eta < 0.4$

The Bias input :

Each neuron a firing threshold 'θ' that determined what value it needed before it should fired.

Perceptron Algorithm

⇒ The algorithm is seperated into 2 parts

* Training phase
* Recall phase

→ the recall phase used after training it is the one that should be fast to use since it is used far more often than the train phase.

Step 1 : Initialization

→ set all of the weights w_{ij} to small (positive & negative) random numbers.

Step 2 : Training

↦ for T iterations (or) until all the o/p. are correct

↦ for each input vector,

⇒ compute the activation of each neuron j using activation function g.

$$y_j = g\left(\sum_{i=0}^{m} w_{ij} x_i\right) = \begin{cases} 1 & \text{if } \sum_{i=0}^{m} w_{ij} x_i > 0 \\ 0 & \text{if } \sum_{i=0}^{m} w_{ij} x_i \leq 0 \end{cases}$$

• update each of weights individually using

$$w_{ij} \leftarrow w_{ij} - \eta (y_j - t_j) x_i$$

Step 3 : Recall

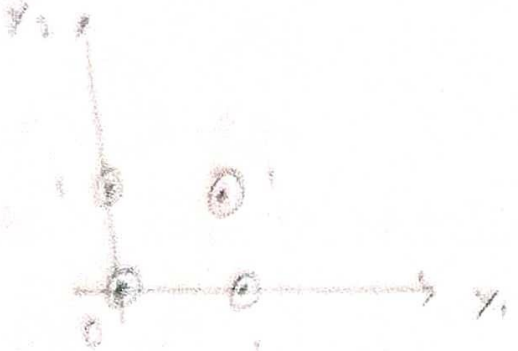
→ compute the activation of each neuron 'j' using

$$y_j = g\left(\sum_{i=0}^{m} w_{ij} x_i\right) = \begin{cases} 1 & \text{if } w_{ij} x_i > 0 \\ 0 & \text{if } w_{ij} x_i \leq 0 \end{cases}$$

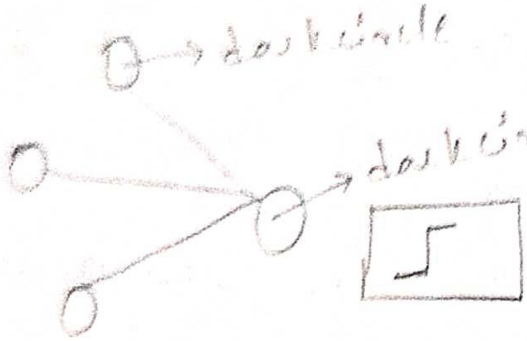
Example

Using "OR" GATE

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



The perceptron network for the example



The complexity of perceptron is big "O(mn)"

Example :

$w_0 = -0.05$, $w_1 = -0.02$, $w_2 = +0.02$

these are the inputs. Both inputs are (0,0) : 0 and the input weight is -1 so the neuron reaches $[-0.05 \times -1] + [0.02 \times 0] + [0.02 \times 0]$

$= 0.05$ (O/P)

The o/p produces = 1. since the input is going to above '0'. i.e the neuron fire/fail in the activation function.

$w_0 = -0.05 - 0.25 (1-0) \times -1 =$ $[\eta = 0.25]$ using updation eq $[w_{ij} - \eta (y_j - t_j) . x_i]$

$w_1 = -0.02 - 0.25 (1-0) \times 0 =$

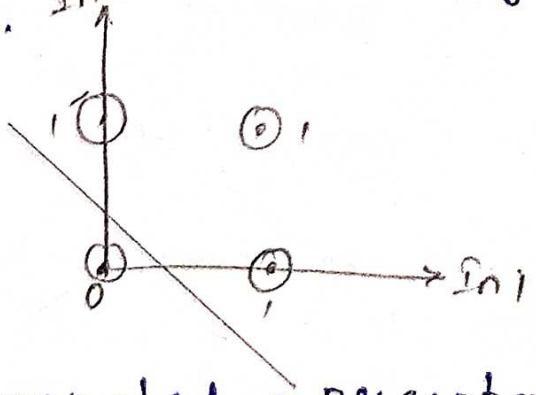
Implementation :

The perceptron using training & recall phases produces the neural network is not writing it we only show it one data point. For that implementation, Lang like 'c' & 'Java' iterating the loops using arrays is difficult for that python syntax iterating the loops using numerical libraries - numpy provides an alternative methods. Because can easily multiply arrays & matrices together.

Linear Separability

⇒ The perceptron does it tries to find a st. line where the neuron fires on one side of the line and doesn't on the other side.

⇒ This is called the decision boundary (or) Discriminant function.



⇒ Decision boundary computed a perceptron for the OR fun.

⇒ The perceptron, the boundary case is where we find an I/p vector x that has "$x_1 . w^T = 0$" Now suppose that we find another input vector x_2 that satisfies "$x_2 . w^T = 0$".

⇒ Then

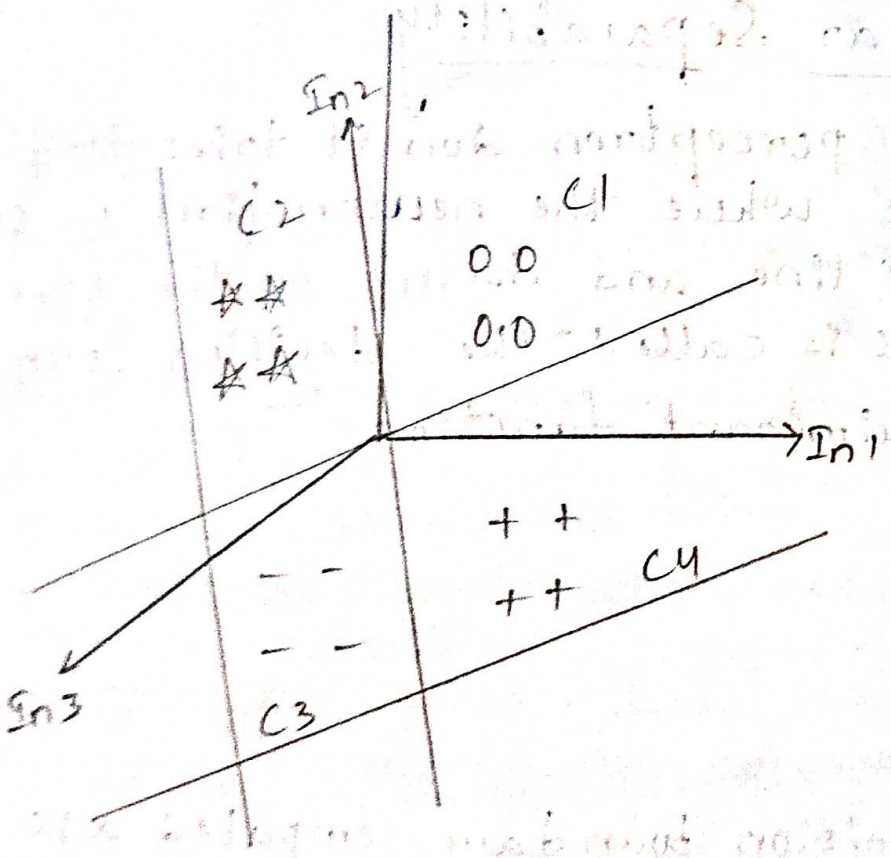
$$x_1 w^T = x_2 \cdot w^T$$

$$(x_1 - x_2) w^T = 0$$

⇒ If that st. line exists but it is a bit of problem otherwise. The cases where there is st. line of all linearly separable cases

⇒ If non linearly separable the weights for each neuron separatly describes a st. line

⇒ So, by putting together several neurons we get several st. lines. That each try to separate diff. parts of the spaces.



⇒ Different decision boundaries computed by perceptron by 4 neurons.

⇒ <u>The perceptrons convergens theorem</u>: The perceptron will coverage to a sol that separates the classes if that it will do it

after a finite no of iterations
the no of iterations is bounded by $\frac{1}{\gamma^2}$
where 'γ' is the distance btw the separating hyperplane and the closest datapoint to it.

⇒ If the weights are linearly separable then the algorithm will converge and that the time taken to run and of the distance btw the separable hyperplane and the nearest datapoint is called margin

XOR Function

⇒ The "XOR" has the same 4 input points as the "OR" fun.

⇒ In XOR can't draw a st. line on the graph that separates True from False. So, the XOR fun is not linearly separable.

In1	In2	?
0	0	1
0	1	0
1	0	0
1	1	1

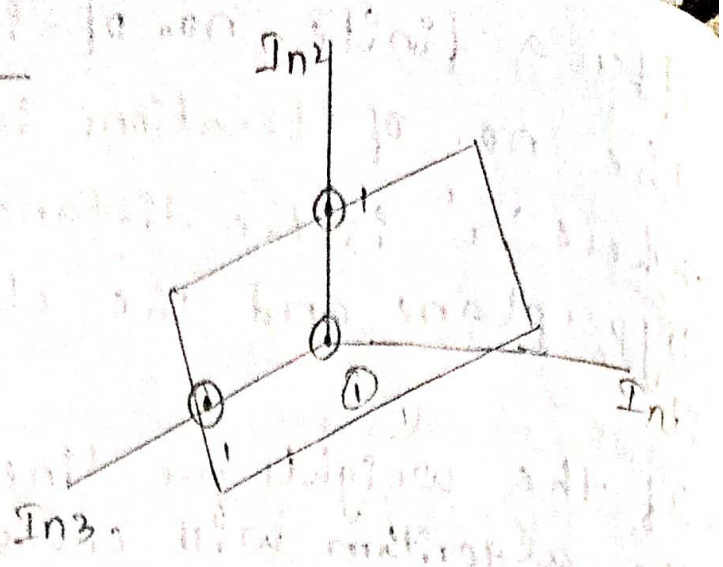


⇒ The perceptron was not capable of solving the above XOR fun.

A Useful Insight:

⇒ The XOR fun is impossible to solve a linear fun. If we rewrite the problem in 3 dimensions instead of 2. Then it is perfectly possible to find a plane that can separate the 2 classes.

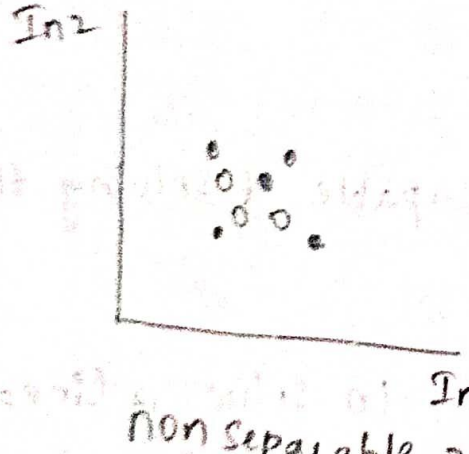
In1	In2	In3	i
0	0	1	0
0	1	0	0
1	0	0	0
1	1	1	1



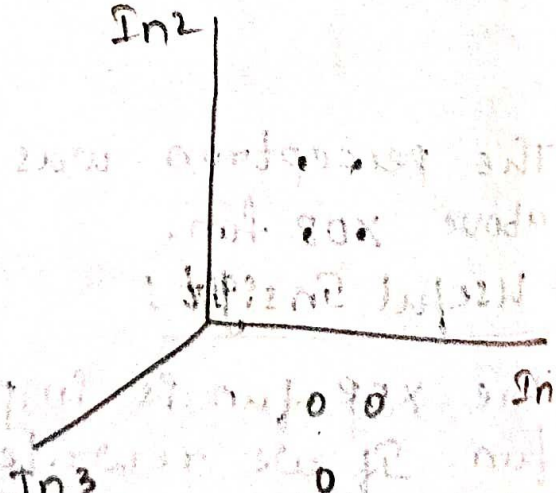
It is always possible to separate using 2 axis. There is a whole class of methods for doing this resonably, efficiently. So, we called "kernal classifier".

It is sufficient to pointout to make a lineal perceptron to non lineal.

For example, the coordinates are x_1 & x_2 while on the right side the coordinates are "$x_1 \cdot x_2$" and "$x_1 * x_2$" it is now eas to fit a plane that separates the data.



non separable. 2D



The same dataset with 3 coordinates, makes it separate.

Linear Regression

⇒ It is also a type of ML algorithm.

⇒ More specifically, it is supervised ML algorithm that learns from the labelled datasets, that maps the datapoints to most optimised data linear fun.

⇒ Labelled data means, the data set whose respective target value is already known.

⇒ Two types of supervised ~~regression~~ learning:
* classification
* Regression.

⇒ Def: It computes the linear relationship b/w the dependent variables and one (or) more independent features by fitting a linear eqn to observed data.

→ In regression only 1 independent feature it is simple independent regression (linear) Ex: salary.

⇒ One (or) more than 1 feature it is called multiple linear regression.

⇒ In this we need to separate regression problems where we fit a line to data from classification problem. where we find a line that separates the classes so that they can be differentiated. In this regression problem use "indicator variable" which is a class, each datapoint belongs to.

⟹ Regression or making a prediction about an unknown value 'y' (such as indicator variable for classes (or) a future value of some data) by computing some function known values 'xi'

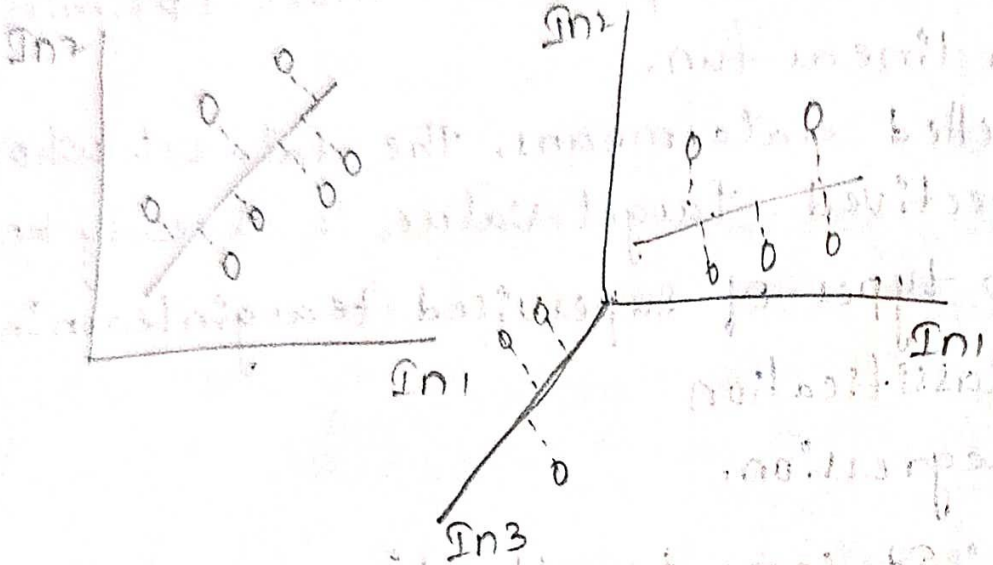


Fig:
Linear regression in two & 3 dimensions.

⟹ The solution of best fit data is to try to minimise the distance b/w each data points and line that fit.

$$\sum_{i=0}^{N}\left(t_j - \sum_{i=0}^{M} B_i x_{ij}\right)^2$$

$B_i \rightarrow$ st. line
$t \rightarrow$ column vector

Example:
Assume that data set consists of collection of a number of datapoints about certain cars {weight, speed, cost, ...} with the aim belong to predict the fuel efficiency in miles per Gallon.

Ex:

- This dataset has 1 problem there are missing values in it. that are labelled with '?' but it doesn't like this and we don't know what to do with them using. In python np.loadtxt().
- So, after downloading the data set manually edit the file and delete all the lines where there is a '?' is labelled.
- The linear regression can't do much with names of cars. It appears a comment like ever. So, now separate the data into training & testing sets then use the training set to recover 'B' vector.
- Then we get the predicted value on the test set.
- On remaining data use the sum of squares errors which consists of computing the difference b/w the prediction & true value. Squaring them so that they all are +ve not them adding then using linear regression.

5/9/24

Unit - 2

Multi-layer perceptron :

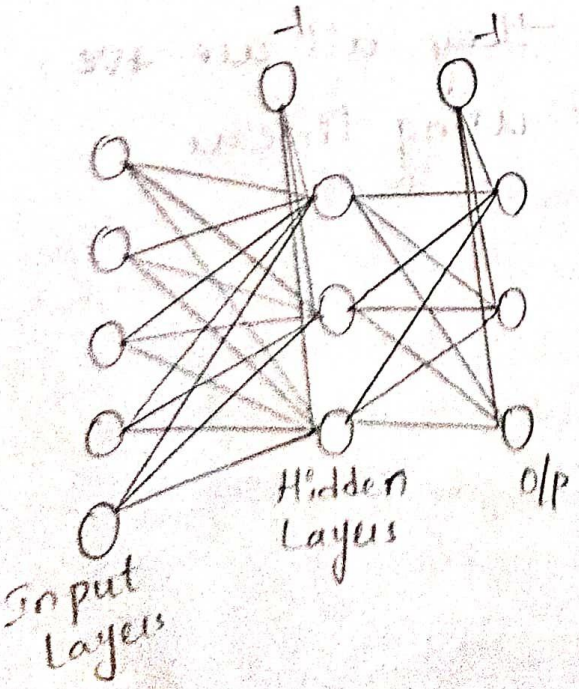
→ In the previous sessions we can made linearly separable problem that means it is not decided that the leaning in the neural network happens in the weights.

⇒ So, to peform more computation it seems sensible to add more weights, for that adding extra layers of notes makes a neural network more poweful.

⇒ In previous perceptron we check XOR problem and solve 2D XOR. But it is not possible for a linear model like the perceptron.

⇒ Because, to put an input and work through the nlw treating it as 2 diff perceptrons. Using "activation function".

⇒ To check that it gives the correct answer. All it is required.



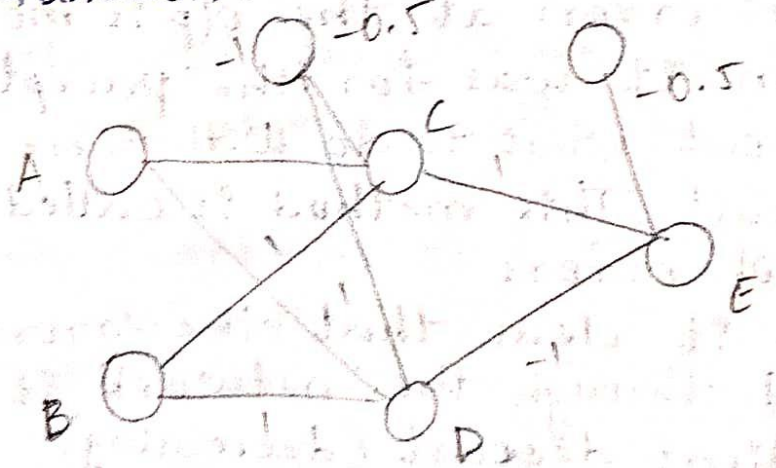
⇒ To compute the error at o/p find the difference between target and the o/p.

⇒ But we don't know the which weight were wrong in the 1st layer (or) the 2nd also.

⇒ So, the correct activations are for the neuron is the middle of the network. This fact gives the neurons in the middle of the n/w their names. They are called 'hidden layers'. Because it is not apossible to examine & correct their values directly.

Example:

⇒ The nodes are ABCDE. The I/P is 1,0 corresponds to node 'A' being 1 and node 'B' being 0. The input to neuron 'C' is therefore $-1 \times 0.5 + 1 \times 1 + 0 \times 1 = 0.5$ this is threshold above 0. So, neuron 'C' is fired giving o/p 1. For neuron 'D' the input is $-1 \times 1 + 1 \times 1 + 0 \times 1 = 0$ so it does not fired. Therefore the i/p neuron 'E' is $-1 \times 0.5 + 1 \times 1 + 0 \times -1 = 0.5$. So it will fire hence the o/p is 1. Checking the results of the inputs 'E' fires when inputs 'A & 'B' are different to eachother. But doesno fire when they are same. which is exactly the XOR function.



Going forward :

⇒ The perceptron training the MLP consisting of 2 parts.

* working out, what the o/ps are for the given inputs & current weights.
* Then updating the weights according to the errors which is a function of the difference blw the o/ps & targets. These are generally known as going foward through the network.

⇒ In the above XOR example, we saw the go foward for the MLP.

Bias :

⇒ A Bias input to each neuron that means we having an extra input that is permanently set to '-1' & adjusting the weights to each neuron as part of the training.

Going Backward ; Back propagation of Error

⇒ computing the errors at the o/p is more difficult than it was for the perceptron.

⇒ But working out what to do with these errors is more difficult. This method is called Back Propagation of errors.

⇒ which makes it clear that the errors are sent backward through the network it is a form of gradient descent (decreasing)

The error function for the perceptron

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

where,

N → The no. of o/p nodes

suppose, we make 2 errors

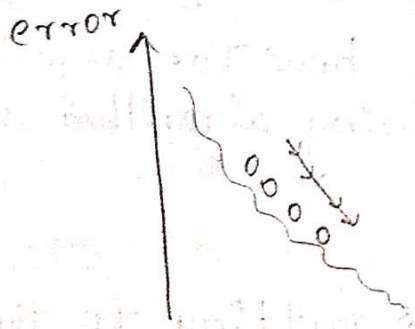
* The target is bigger than the o/p.
* The o/p is bigger than the target.

If these 2 errors are of same size, if we add them up and we could get '0'. which means that the error value suggests that no error was made.

otherwise,

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

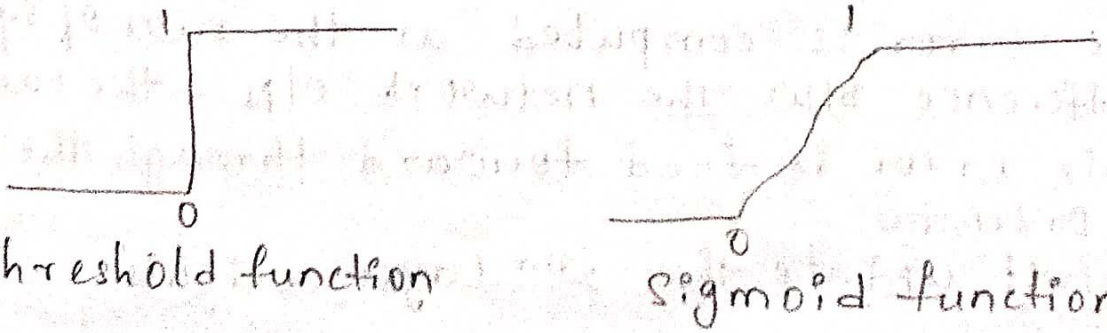
The above formula using graph representation it is goes to down hill.



that means the weights of the network are trained so that the error goes downhill until it reaches the local minima using 2 functions

* Threshold function - It changes the value from (0-1)
* Sigmoid function - It varies smoothly & differentially

⇒ Threshold function is completly lineal & sigmoid is non lineal.



Threshold function Sigmoid function

⇒ In backward we have
* For the o/p neuron we don't know the s/p.
* For the hidden neurons we don't know the targets.

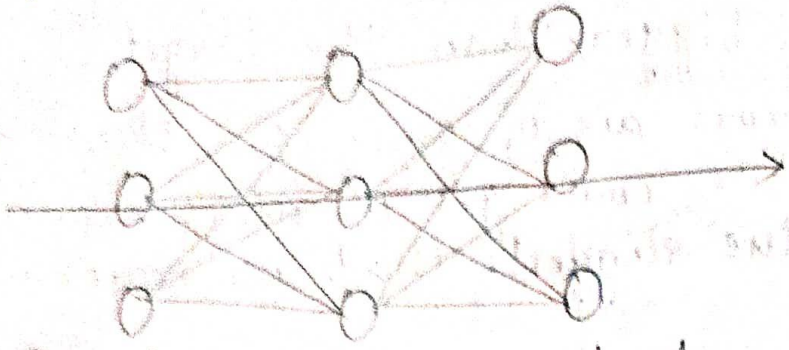


Fig: The forward direction of perceptron in MLP

The multi-layered perceptron algorithm

⇒ Quickly summarise how the algorithm works & the the full MLP training algorithm using back propagat of error is

Steps

1) Input vectors is putting to the input nodes.
2) The inputs are feed forward through the netw
   - The inputs & the 1st layered weights are used to decide whether the hidden nodes fired or not. The activation function $g()$ is the sigmoid function in the above equation.
   - The o/p of these neurons and the 2nd layer weights are used to decide if the o/p neuron fires (or) not.
3) The error is computed as the sum of squared difference b/w the network o/ps & the targets.
4) This error is feed forward through the n/w in order to
   - first update the 2nd layer weights.

• And then, afterwards the 1st layer weights.

Multi Layer Perceptron Algorithm

Step 1 : Initialization

→ Initialize all weights to small +ve & -ve random values.

Step 2 : Training

→ Repeat

• for each Ip vector

forward phase :

• compute the activation of each neuron j in the hidden layer using :

sigmoid - o/p
softmax

$$h_i = \sum_{i=0}^{k} x_i v_{ij}$$

$$x_i = g(h_i) = \frac{1}{1+\exp(-\beta h_i)}$$

• work through the network until you get to the o/p layer neuron which have activations

$$h_i = \sum_{i=0}^{k} a_i w_{ij}$$

$$y_i = g(h_i) = \frac{1}{1+\exp(-\beta h_i)}$$

Backward phase :

• compute the error at the o/p using

desired o/p, estimated o/p

$$\delta_o(k) = (y_k - t_k)\, y_k (1 - y_k)$$

• Compute the error on the hidden layer using

$$\delta_n(k) = a_i (1 - a_i) \sum_{k=1}^{N} w_i \delta_o(k)$$

• Update the o/p layer weights using

$$w_{ik} \leftarrow w_{ik} - \eta\, \delta_o(k)\, a_i(k)$$

• Update the hidden layer weights using

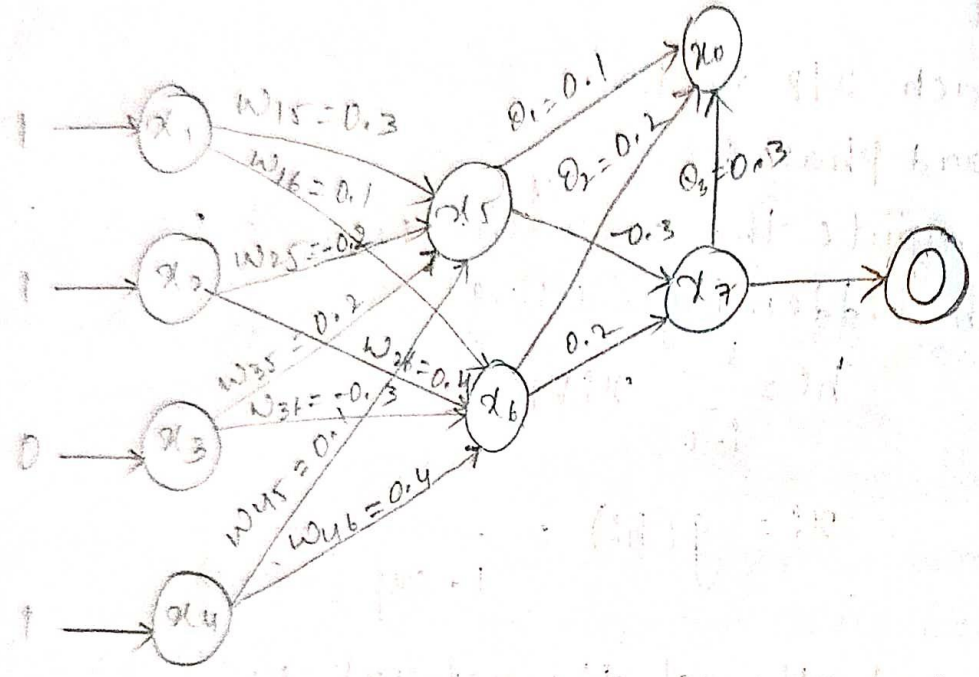
$$v_i \leftarrow v_i - \eta\, \delta_n(k)\, x_i$$

⇒ Randomise the order of the I/p vector so that you don't know train in exactly the same order in each iteration until learning stops.

Step 3: Recall

⇒ Use the forward phase in the training session.

Example:



Step 1 Initializing the weights

⇒ The MLP alg suggests that the weights are initialized to small random nos, both +ve & -ve.

Step 2

⇒ If the initial weight values are close to 1 (or) then the inputs to the sigmoid has saturated, reached to max (or) min value.

⇒ If the weights are initial values '0' (or) close to '0' then the I/p is '0' and o/p will be linear of the neuron.

⇒ $\boxed{\frac{-1}{\sqrt{n}} < w < \frac{1}{\sqrt{n}}}$ set the weights in this range.

Where 'n' is the no. of nodes in the I/p layer.

x_1	x_2	x_3	x_4	w_{15}	w_{25}	w_{35}	w_{45}	w_{16}	w_{26}	w_{36}	w_{46}
1	1	0	1	0.3	-0.2	0.2	0.1	0.1	0.4	-0.3	0.4

w_{56}	w_{67}	θ_1	θ_2	θ_3
-0.3	0.2	0.1	0.2	0.3

[x_0 value is always '1']

⇒ Differentiate o/p activation fun.

Step1

⇒ In activation fun used in the layer can be linear (or) non linear depending on the type of the problem.

⇒ The o/p neurons with linear nodes that sum the inputs & give that their activation function is $g(h) = h$

⇒ Otherwise, a sigmoid neuron activation function is used. If the problem is binary classification.

⇒ Softmax activation fun is used in a multiclass classification problem.

⇒ This doesnot mean that we changed the hidden layer neurons, this stay exactly the same. And we only modify the output nodes. i.e. netinput.

for x_5

$$x_1 \times w_{15} + x_2 \times w_{25} + x_3 \times w_{35} + x_4 \times w_{45} + \overset{1}{x_0} \times \theta_1$$

$$= 1 \times 0.3 + 1 \times -0.2 + 0 \times 0.2 + 1 \times 0.1 + 1 \times 0.1$$

$$= 0.3 - 0.2 + 0.1 + 0.1$$

$$= 0.5 - 0.2$$

$$= 0.3$$

$$x_i = \frac{1}{1 + \exp(-\text{Phi})} = \frac{1}{1 + \exp(-0.3)} = 0.46$$

Unit - 2

5/9/24

Multi-layer perceptron :

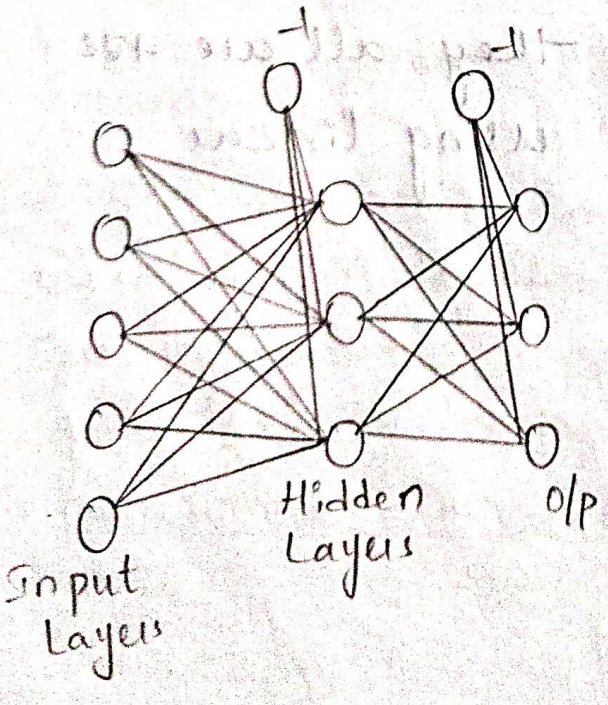
→ In the previous sessions we can made linearly separable problem that means it is not decided that the leaning in the neural network happens in the weights.

→ So, to perform more computation it seems sensible to add more weights, for that adding extra layers of notes makes a neural network more powerful.

⇒ In previous perceptron we check XOR problem and solve 2D XOR. But it is not possible for a linear model like the perceptron.

⇒ Because, to put an input and work through the nlw treating it as 2 diff perceptrons. Using "activation function".

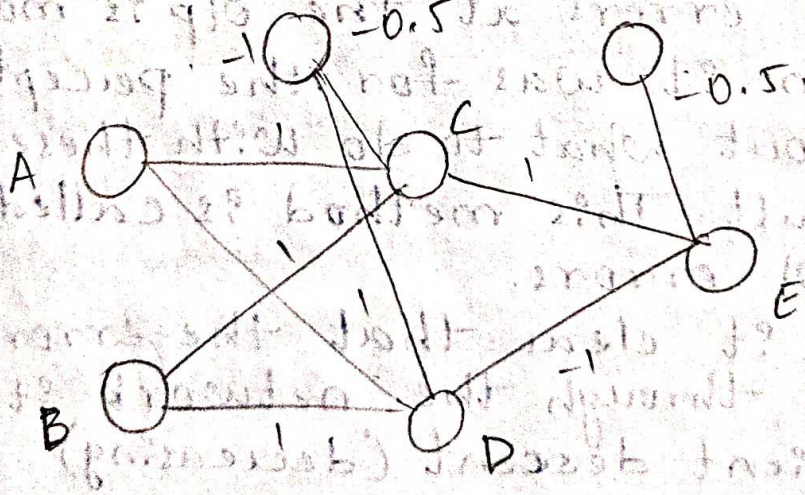
⇒ To check that it gives the correct answer All it is required.



→ To compute the error at the output we know the target and the o/ps.
→ But we don't know the which weight were wrong in the 1st layer (or) the 2nd also.
→ So, the correct activations are for the neuron is the middle of the network. This fact gives the neurons in the middle of the n/w their names. They are called 'hidden layers'. Because it is not apossible to examine & correct their values directly.

Example:

→ The nodes are ABCDE. The I/P is 1,0 corresponds to node 'A' being 1 and node 'B' being 0. The input to neuron 'C' is therefore $-1 \times 0.5 + 1 \times 1 + 0 \times 1 = 0.5$ this is threshold above 0. So, neuron 'C' is fired giving o/p 1. For neuron 'D' the input is $-1 \times 1 + 1 \times 1 + 0 \times 1 = 0$ so it does not fired. Therefore the I/P neuron 'E' is $-1 \times 0.5 + 1 \times 1 + 0 \times -1 = 0.5$. So it will fire hence the o/p is 1. checking the results of the inputs 'E' fires when inputs 'A & 'B' are different to eachother. But doesnot fire when they are same. which is exactly the XOR function.



Going forward :

⇒ The perceptron training the MLP consists of 2 parts.

* working out, what the o/ps are for the given inputs & current weights.
* Then updating the weights according to the errors which is a function of the difference blw the o/ps & targets. These are generally known as going foward through the network.

⇒ In the above XOR example, we saw the go foward for the MLP.

Bias :

⇒ A Bias input to each neuron that means we having an extra input that is permanent set to '-1' & adjusting the weights to each neuron as part of the training.

Going Backward ; Back propagation of Error

⇒ computing the errors at the o/p is more difficult than it was for the perceptron.

⇒ But working out what to do with these errors is more difficult. This method is called Back Propagation of errors.

⇒ which makes it clear that the errors are sent backward through the network it is a form of gradient descent (decreasing)

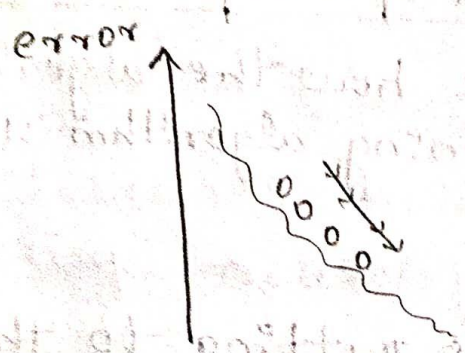
The error function for the perceptron

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

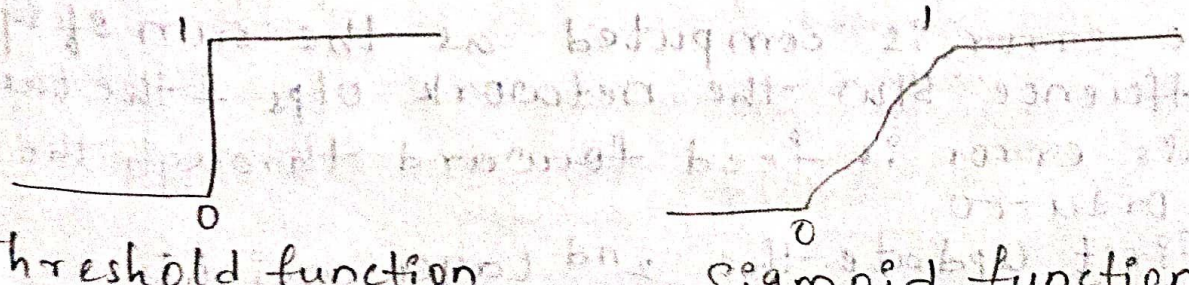
where,
N → The no. of o/p nodes
suppose, we make 2 errors
* the target is bigger than the o/p.
* The o/p is bigger than the target.
If these 2 errors are of same size, if we add them up and we could get '0', which means that the error value suggests that no error was made.
otherwise,

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

the above formula using graph representation it is goes to down hill.



that means the weights of the network are trained so that the error goes downhill until it reaches the local minima using 2 functions
* Threshold function - It changes the value from (0-1)
* Sigmoid function - It varies smoothly & differentially
⇒ Threshold function is completly lineau & sigmoid is non lineau.



Threshold function sigmoid function

⇒ In backward we have 2 problems
* For the o/p neuron we don't know the i/p.
* For the hidden neurons we don't know the targets.

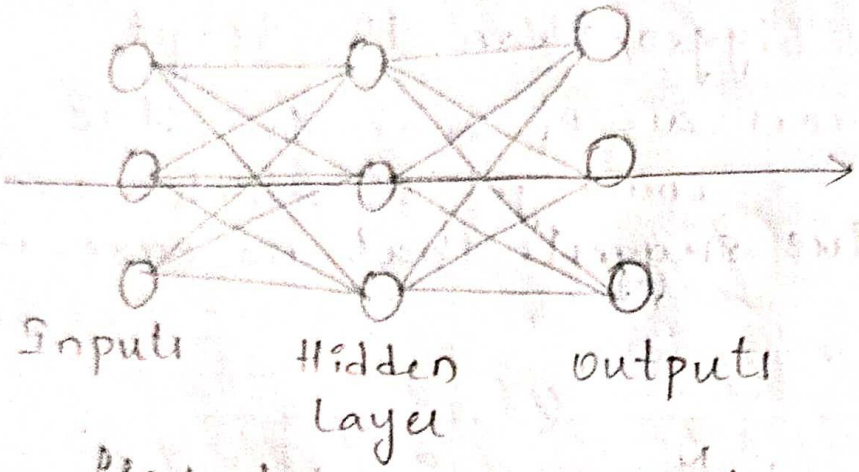


Fig: The forward direction of perceptron in MLP

The multi-layered perceptron algorithm

⇒ Quickly summarise how the algorithm works & the the full MLP training algorithm using back propagation of error is

Steps

1) Input vectors is putting to the input nodes
2) The inputs are feed forward through the network
   - The inputs & the 1st layered weights are used to decide whether the hidden nodes fired or not. The activation function $g()$ is the sigmoid function in the above equation.
   - The o/p of these neurons and the 2nd layer weights are used to decide if the o/p neuron fires (or) not.
3) The error is computed as the sum of squared difference b/w the network o/ps & the targets.
4) This error is feed forward through the n/w in order to
   - first update the 2nd layer

• And then, afterwards the 1st layer weights.

Multi Layer Perceptron Algorithm

Step 1 : Initialization

Initialize all weights to small +ve & -ve random values.

Step 2 : Training

• Repeat

• for each Ip vector

forward phase :

• compute the activation of each neuron j in the hidden layer using :

$$h_i = \sum_{i=0}^{k} x_i v_{ij}$$

sigmoid - 0 / 1
softmax -

$$x_i = g(h_i) = \frac{1}{1+\exp(-\beta h_i)}$$

• work through the network until you get to the olp layer neuron which have activations

$$h_i = \sum_{i=0}^{k} a_i w_{ij}$$

$$y_i = g(h_i) = \frac{1}{1+\exp(-\beta h_i)}$$

Backward phase :

desired olp — estimated olp

• compute the error at the olp using

$$S_o(k) = (y_k - t_k)\, y_k (1 - y_k)$$

• compute the error on the hidden layer using $S_n(k) = a_i (1-a_i) \sum_{k=1}^{N} w_i S_o(k)$

• update the olp layer weights using

$$w_{ik} \leftarrow w_{ik} - \eta S_o(k) a_i(k)$$

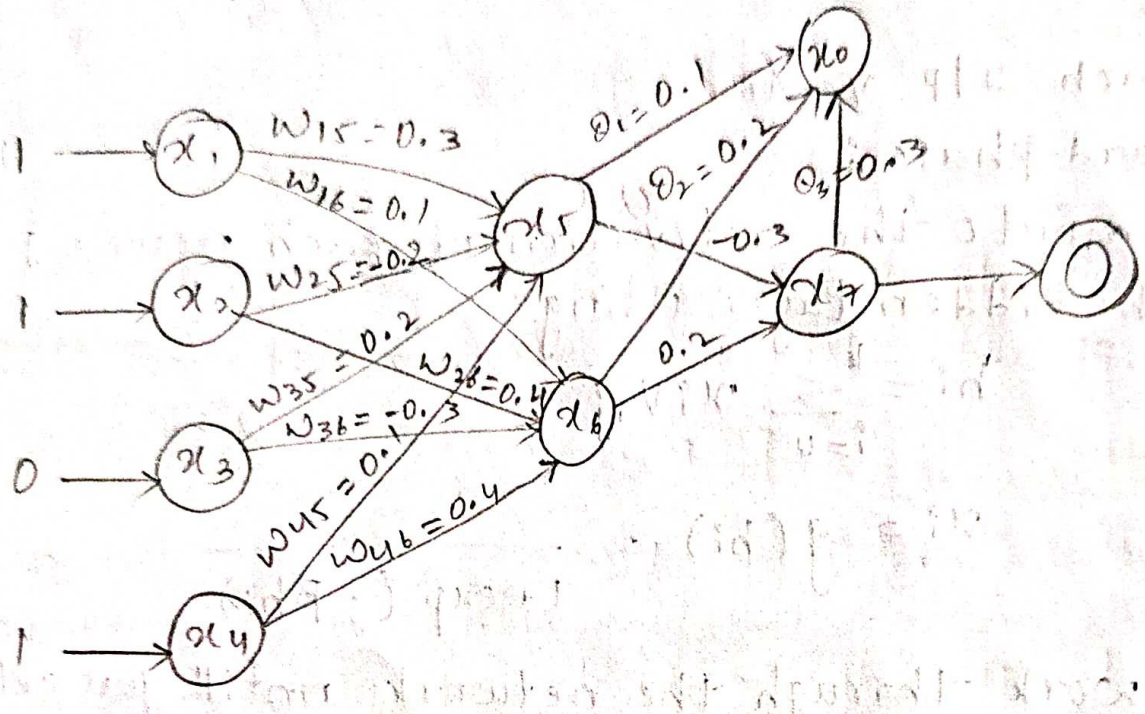
• update the hidden layer weights using

⇒ Randomise the order of the Ip vector so that you don't know train in exactly the same order in each situation until learning stops.

Step 3: Recall

⇒ Use the forward phase in the training session

Example:



Step 1 Initializing the weights

⇒ The MLP alg suggests that the weights are initialized to small random nos, both +ve & -ve.

Step 2

⇒ If the initial weight values are close to 1 (or) then the inputs to the sigmoid has saturated, reached to max (or) min value.

⇒ If the weights are initial values '0' (or) close to '0' then the Ip is '0' and olp will be linear of the neuron.

⇒ $\boxed{\frac{-1}{\sqrt{n}} < w < \frac{1}{\sqrt{n}}}$ set the weights in this range.

Where 'n' is the no. of nodes in the Ip layer.

⇒ Ba

	x_2	x_3	x_4	w_{15}	w_{25}	w_{35}	w_{45}	w_{16}	w_{26}	w_{36}	w_{46}
x_1	1	0	1	0.3	-0.2	0.2	0.1	0.1	0.4	-0.3	0.4
1											

	w_{67}	θ_1	θ_2	θ_3
w_{57}	0.2	0.1	0.2	0.3

[x_0 value is always '1']

⇒ Differentiate olp activation fun.:

Step1

→ In activation fun used in the layer can be linear (or) non linear depending on the type of the problem.

→ The olp neurons with linear nodes that sum the inputs & give that their activation function is

$g(h) = h$

⇒ Otherwise, a sigmoid neuron activation function is used. If the problem is binary classification.

⇒ Softmax activation fun is used in a multiclass classification problem.

⇒ This doesnot mean that we changed the hidden layer neurons, this stay exactly the same. And we only modify the output nodes. i.e. netinput.

for x_5

$$x_1 \times w_{15} + x_2 \times w_{25} + x_3 \times w_{35} + x_4 \times w_{45} + x_0 \times \theta_1$$

$$= 1 \times 0.3 + 1 \times -0.2 + 0 \times 0.2 + 1 \times 0.1 + 1 \times 0.1$$

$$= 0.3 - 0.2 + 0.1 + 0.1$$

$$= 0.5 - 0.2$$

$$= 0.3$$

$$x_i = \frac{1}{1 + \exp(-Bhi)} = \frac{1}{1 + \exp(-0.3)} = 0.46$$

→ The soft-max fun recall the o/p by calculating the exponential of inputs to the neurons & devided by total sum of the all the neurons. so that the activation sum to 1 and lie b/w '0 & '1'.

⇒ An activation fun it can be written as

$$y_k = g(h_k) = \frac{\exp(h_k)}{\sum_{k=1}^{N} \exp(h_k)} = \exp(0.3)$$

⇒ If we change the activation fun then the derivation of activation will also change & so the learn rule will also change.

Sequential & Batch training:

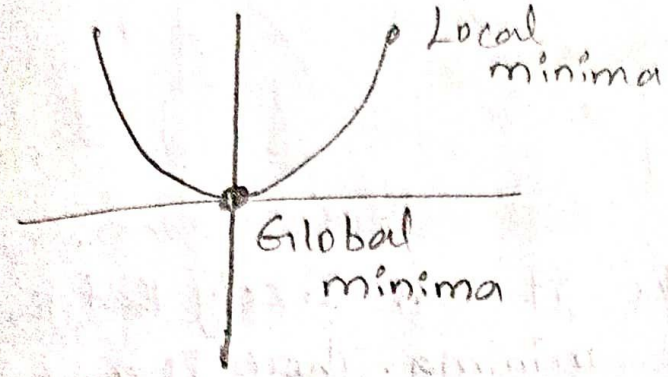
⇒ In all of the training examples are used to update the weights.

⇒ There is only 1 set of weight updates for each "epoch" (pass through all of the training examples) i.e sequential training.

⇒ The main draw back of sequential algorithm is avg sum of square error is the weights are moved in the directions that most of inputs want them to move. Rather than being pulled around by each input individually that means we only update the weights once for each iteration of the algorithm.

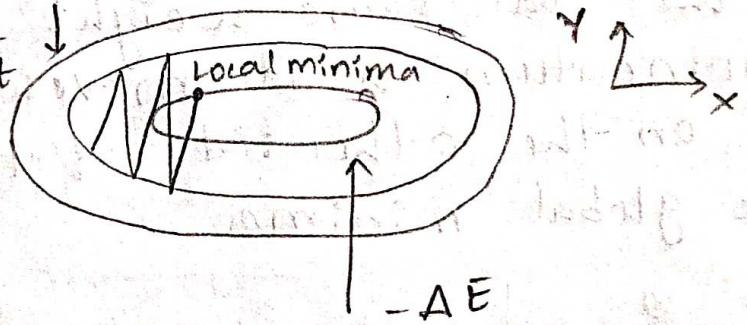
⇒ for that, the Batch method performs a more aquate estimate of the gradient & will converge to the local minima more quic

- The alg works as sequential version. This potentially reaching better soln.
- This can significantly improve the speed with which the alg runs. learns.

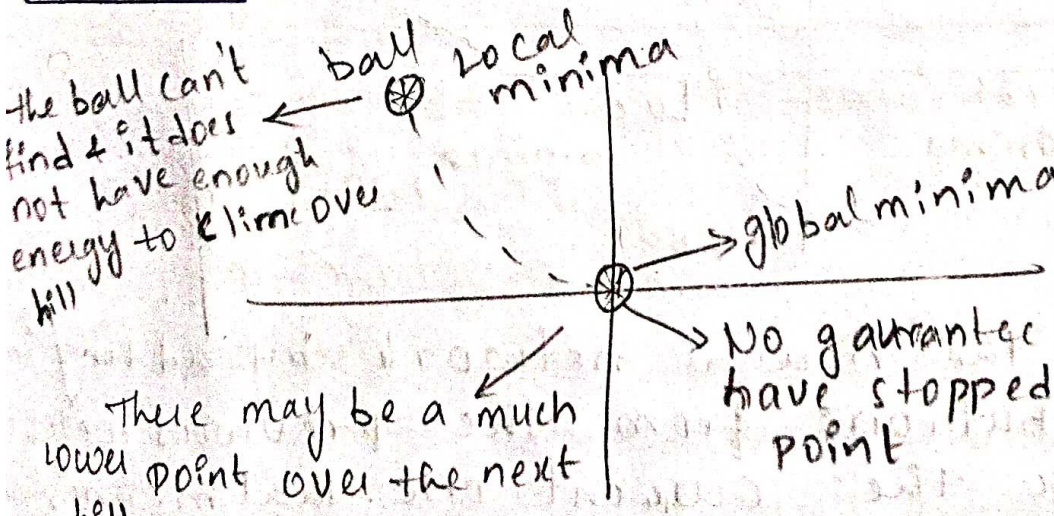


Local Minima:

In Local minima the driving force behind the learning rule is the minimization of the n/w error by the gradient descent. Here we are performing an optimization i.e the value of the weights inorder to minimize the error fun.



Example:



So, gradient decent works in the same way in 2 or 3 dimensions as similar problems.

In this we start at a local minima we will Probably end up there.

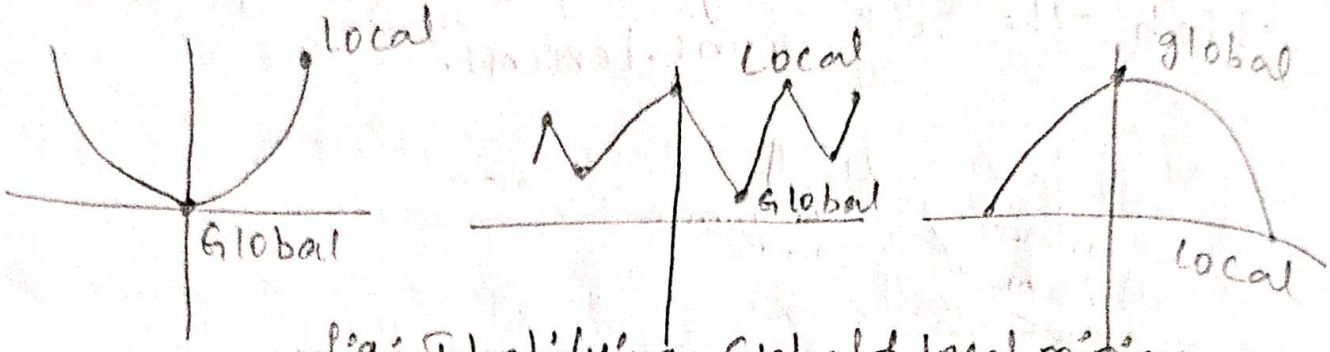
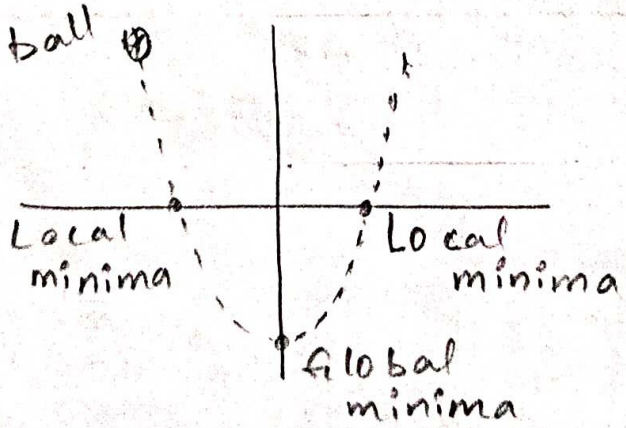


fig: Identifying Global & local minima.

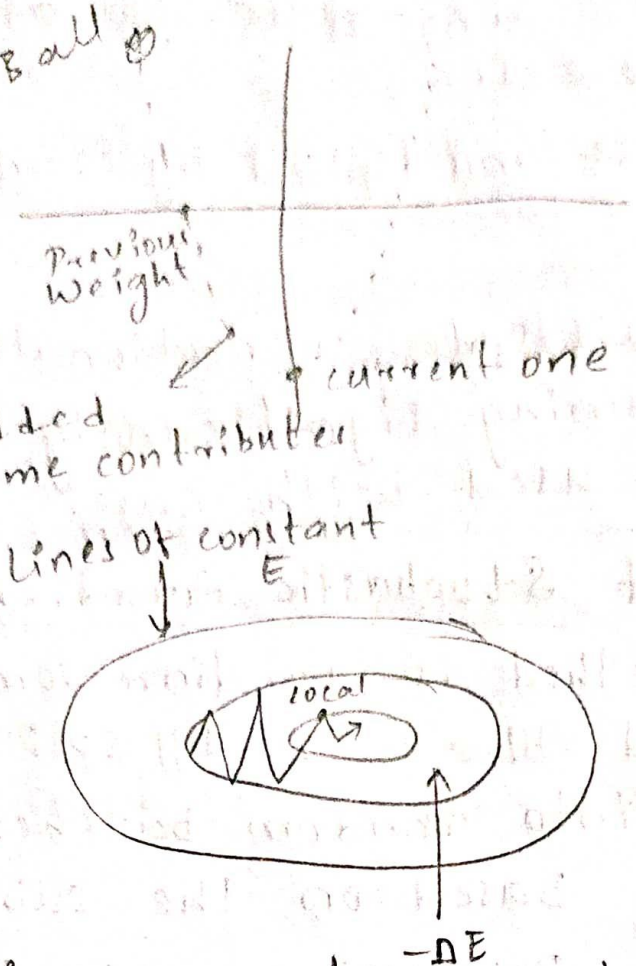
We can also try to make it less likely that the al will get stucked in local minima. There is a moderate effective.

Picking up momentum :

- In the above ex the ball stops rolling at the gradient because it runs out of energy at bottom of the dip.
- If we give the ball some weight it will generate momentum as it moves. TO overcome a small hill on the other side of the local minim to find the global minima.



- The idea in the neural network by adding some contributions from the previous weigh that we make the current one. That means,



→ Adding momentum can help to avoid local minima. It also makes the dynamics of the optimization more stable, improving convergency.

⇒ The MLP alg is an equation where we can add a second term to the weight update so that

$$w_{\beta k}^{t} \longleftarrow w_{\beta k}^{t-1} + \eta_{*\beta} so(k)\, a_{\beta}^{hidden} + \alpha \Delta w_{\beta k}^{t-1}$$

t → used to indicate the current update

$t-1$ → Previous one

$\Delta w_{\beta k}^{t-1}$ → Previous update

$\eta_{*\beta} so(k)\, a_{\beta}^{hidden}$ → we made to the weights $0<\alpha<1$ is the momentum constant.

$\alpha = 0.9$ is used. It can't improve the speed of the learning a lot.

In this we can add some weight is known as weight decay.

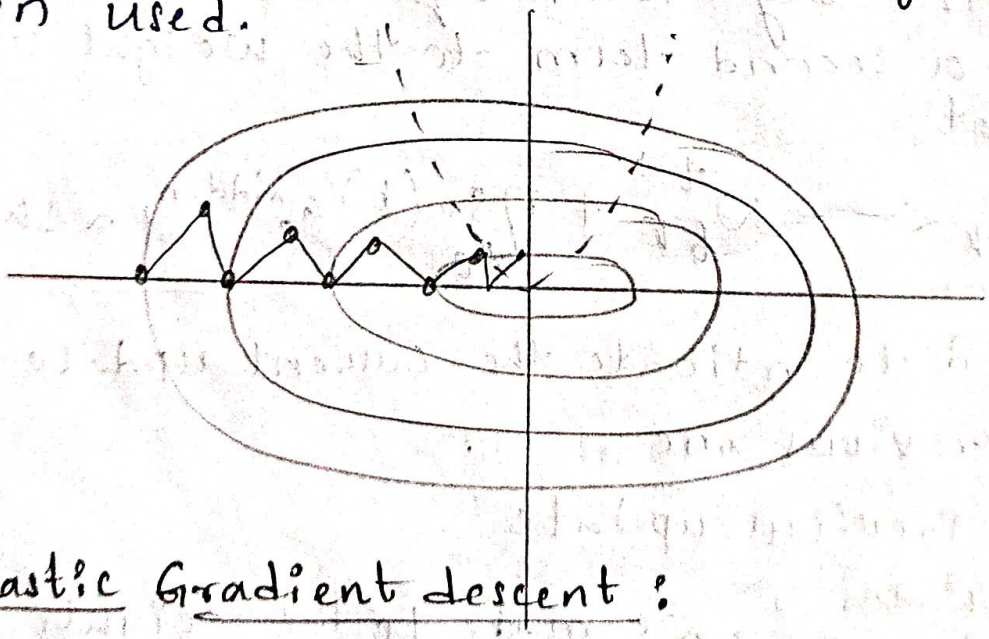
This reduces the size of weight as no. of iterations increases.

Every weight is multiplied by some constant. i.e $\boxed{0 < c < 1}$

This is not fail safe. Occationally it can make the learning significantly worst. So it should be used with care.

Min Batches & Stochastic Gradient descent:

⇒ Min batch method is to find some happy middel ground blw the 2 by spliting the training set into random batches, estimating the gradient based on the subset of the training set, performing a weight update, then using the next subset to estimate a new gradient & using for that weight update until all of the training set have been used.



Stochastic Gradient descent:

⇒ It can escape local minima due to noise update.

⇒ fastu update since it process only one training ex at a time.

A new random I/P vector is choosen & the process repeated. i.e stochastic Gradient descent not just the MPL problem. It is used for very large training sets.

It could be very expensive to use the whole dataset to estimate the gradient.

Other improvements:

In back propogation alg. few things that can be done to improve.

- convergency & behaviour of back propogation.
- Reduce the learning rate.
- Much larger performance gain
- Improve the network.

Multi-Layer perceptron in practise:

⇒ Using the MLP to find soln for 4 diff types of problems:

1) Regression
2) Classification
3) Time series prediction
4) Data compression

⇒ For the above problems we need to follow the bellow steps:

- Amount of training data -

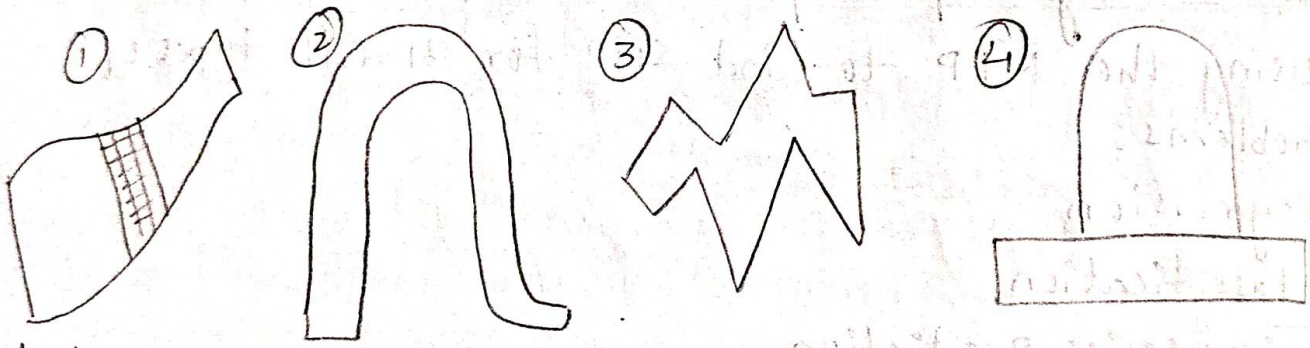
The more training data there is the better for learning. Although the time that the algorithm takes to learn increases. So, there is no way to compute what the min. amount of data required is since it depends on the problem. So, neural network training is a fairly, computationally, expensive operation. Because we need to show the n/w all of these I/P lots of time.

for the MLP, with 1 input layer there are "$(L+1)\times M+(M+1)\times N$" weights. where L, M, N are the nodes in the i/p's, hidden o/p layer. The extra '+1' come from the bias node. which also have adjustable weights.

- **Number of Hidden Layers -**

Which is the choise of no of hidden nodes, & no of hidden layers. For this we use "universal approximation theorem". For 1 hidden layer lots of hidden nodes is sufficient. It is guide the choise of the no of hidden nodes and then choosing the one that gives the best results. In this we are approximating a localised response and orbitary accuracy using a linear combination of such bumps.

Ex:

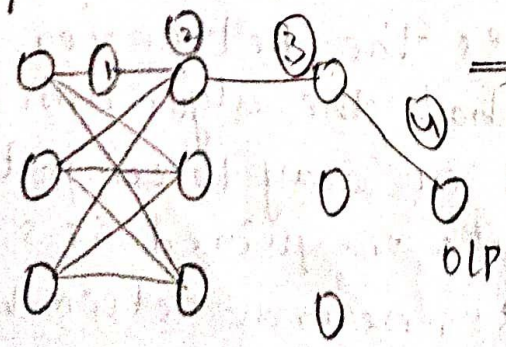


Which can be added to other including received once to get a hill shape.

adding another hill at 90° Produces a bump.

Adding another hill it produces a sharpen end.

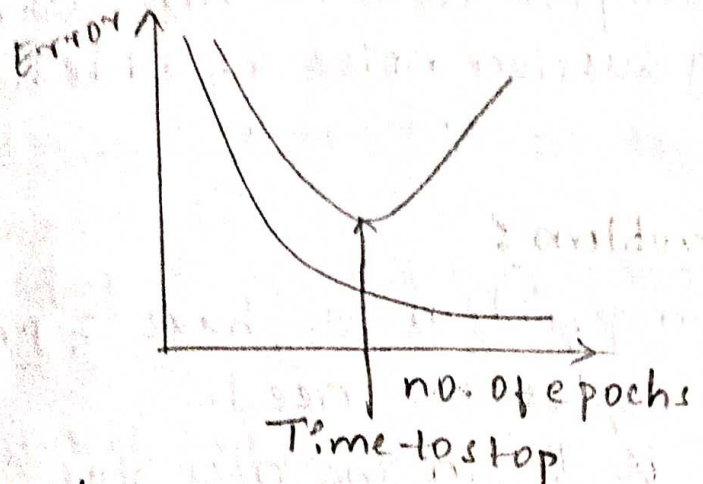
output layer.



⟹ Two hidden layers are sufficient to compute these bump funs for diff inputs. so, if the fun that we want to learn (approximation) is continuous the nlw+ compute it.

, when

some predefined no. of N iterations & running until that is reached runs the risk that the n/w has overfitted by them. (or) not learn sufficiently and only stoping when some predefined min. error is reached. That means the alg never terminate (or) that if overfits.



The effect of overfitting on the training data and validation error curves with the point at which early stopping the learning ~~rate~~ marked.

Examples of using the MLP:

In this we make the n/w learn about some data. The 4 types of problems that are generally solved using MLP.

* Regression:

⇒ Regression problem is very simple one.

⇒ A set of samples generated by a simple mathematical fun & try to learn the generating fun so that we can find the values of any inputs not just the ones we have training data for.

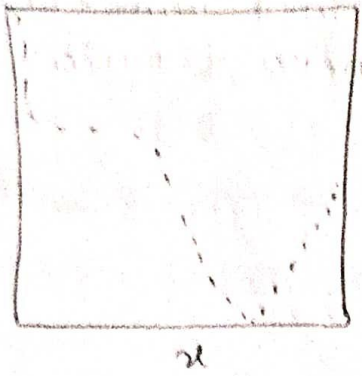
⇒ Example: We consider 40 data points. We can split data into 3 sets. 1 is training set, testing set, Validation set.

→ Suppose 50:25:25 that means half of them training set that is not very many & might not be enough for the algorithm to learn efficiently.

so iteration '0' error 12.370

" '100' error 8.2075

⇒ In the above sample of errors decreasing after 101 iterations.



The data that will learn using an MLP consistency of some samples from a sine wave with gaussion noise is added.

* Classification problem:

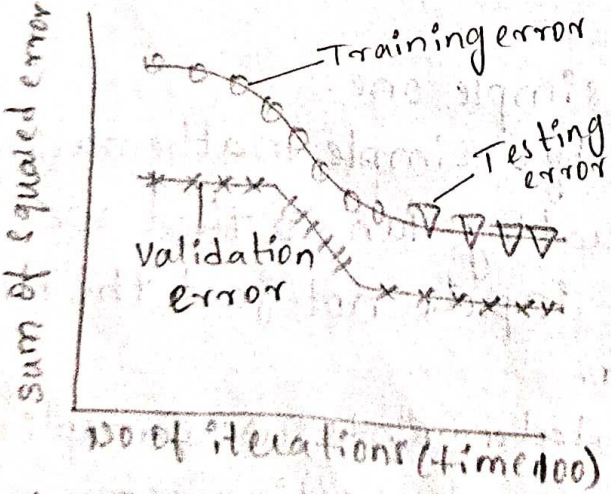
In above regression problem we have 2 problems.

- How many hidden nodes we need.
- Decide how long to train the nlw for.

⇒ We need to solve 1st problem to test out diff nlw & see which gets lower error.

⇒ 2nd problem is to implement eauly stopping. After few iterations then evaluate the validation set error by running the network forwaud.

⇒ keep track of all validation errors & stop when it is increases.



No of hidden nodes	1	2	3	4
mean error	2.21	0.52	0.52	0.52
standaid deviation	0.17	0.00	0.00	0.02
max error	2.31	0.53	0.54	0.54
min error	2.10	0.51	0.51	0.50

few diff sizes of N/W reporting the sum of squared validation error is —

2-4 is similar nodes to stop in one node by using the MLP for classification problem. using the MLP for class problem is not radically diff one that means the olp encoding has been worked out the first is to use a single layer node for the olp 'y' & put some threshold on to the activation value of that node

$$\text{classes} \begin{cases} C_1 & \text{if } y \le -0.5 \\ C_2 & -0.5 < y \le 0 \\ C_3 & 0 < y \le 0.5 \\ C_4 & y > 0.5 \end{cases}$$

we have 4 classes C_1, C_2, C_3, C_4 & $y = 0.5$ then the 4 class problem is given the soln $y = 0.5$ belongs to class C_3 but in neural nlw doesnot give us any info about how close it to the boundary of the clp. so, we don't know that a difficult ex.. to classify in MLP is 1 — of — N that means olp encoding. Two types of activation fun once the nlw has been trained, performing the activation fun $y_k > y_j \; \forall j \neq k$

Hardmax:

this generates an unabiguious decision. Since it is very unlikely that 2 olp neurons will have identically large olp. values that means the neuron with the highest activation is choosen to fight and remaining it ignores.

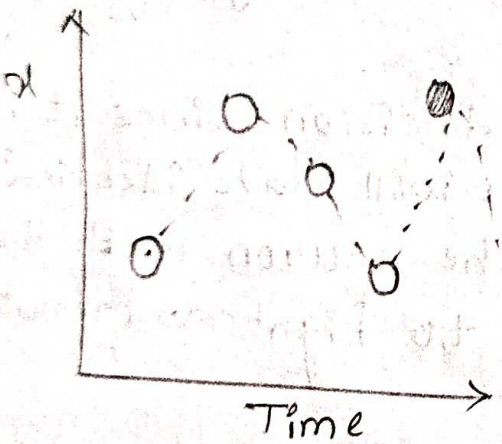
Softmax:

the effect of scaling the olp of each neuron according to how large it is. In compaision of each other making the total sum to 1, if there is one class it clears the winner. It will have a near '1'. If there are several values of classes that are close to each other. That represents in $\frac{1}{P}$ where P is the no. of olp neurons that have similar values.

⇒ Decreasing the lots of data from the over represented classes which may seem wasteful.
⇒ For this we have 1 soln Novelty detection which is train the data in the -ve classes only and to assume that anything that looks diff to that is a +ve value.
⇒ Example : Iris dataset.

* Timeseries problem :

⇒ Data analysis task is known as timeseries.
⇒ Prediction we have a set of data that shows how something vary time t we want to predict how the data will vary in the future.
⇒ Ex : Stock market, seasonal variation.
⇒ The problem is that, even if there is some regularity in the time series it can appear over many diff scales.



with T=3 & k=4 with set of datapoints in the above fig. White circles represent the Ip vectors & target is black circle.

Time series is same as regression problem that is the Op node need to have linear activations. So, we aim to minimize the sum of squared error.

* Data Compression / Auto association N/W :

⇒ Train the nlw to reproduce the Ip at the Op layer.
⇒ In this case at the Ip is reproduced at the output. which does not seen very useful

first: By suppose that we
that has fewer neurons than the slp layer we
use the below one. i.e Bottleneck.

- Bottleneck - It gives what the hidden layer of the MLP are doing, they are finding a diff (after lower dimensions) representation of the input data that extracts important components of the data and ignores the noise.

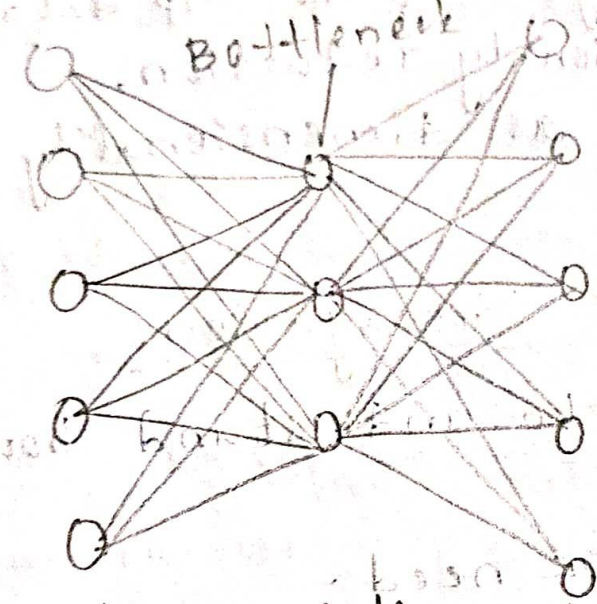
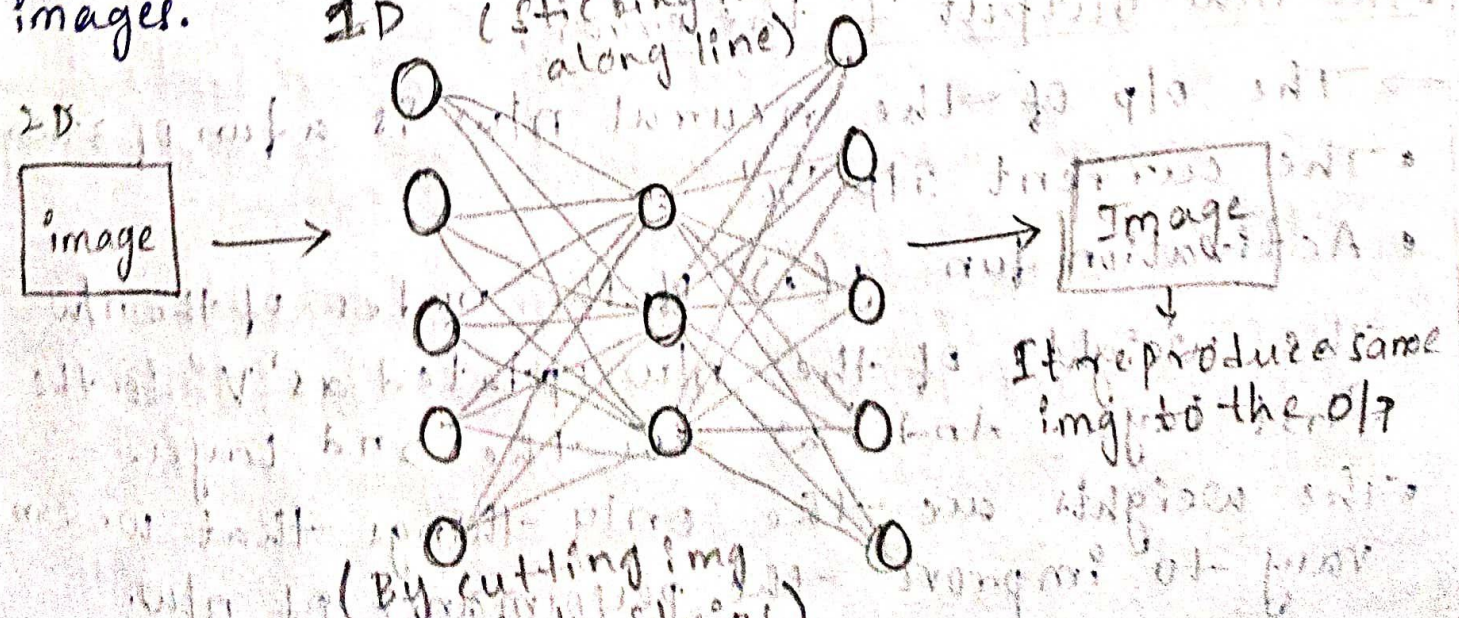


Fig: Auto associative network.

⇒ The NIW is trained to reproduce the Ips at the outputs passing them through the bottleneck hidden layer that compresses the data.

Auto associative NIW is also used to denoise images.



⇒ After training we can
nodes and first set to the weights of the n/w
→ Insert same values in the hidden layer then by
feeding the activation forward through the
2nd set of weights.
→ The correct img will be reproduced on the o/p
⇒ If the nodes are in linear activation then
what the n/w learns to compute the principle
component analysis of the I/p data (PCA) that
means dimensionaly reduction.
⇒ It's not work as dimensionality reduction
technique.

Deriving Back Propagation:

⇒ This explains to understand how and why
alg works.
⇒ Three things we need -

- The derivative with respect to 'x' of $\frac{1}{2}x^2$
- Chain rule i.e $\frac{dy}{dx} = \frac{dy}{dt}\frac{dt}{dx}$
- $\frac{dy}{dx} = 0$ if 'y' is not a fun of 'x'

The n/w output & the error:

⇒ The o/p of the neural n/w is a fun of 3 thing

- The current I/p 'x'
- Activation fun $g(\cdot)$ of the nodes of the n/w
- The weights of the n/w updated as 'V' for the first layer. And 'W' for the 2nd layer.
- The weights are the only things that we can vary to improve the performance of n/w.

The error of the n/w:

Minimize the error function.

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

↓ O/P ↓ Target

LP

if $t_k > y_k \Longrightarrow$ the sign of the error is different

if $y_k > t_k \Longrightarrow$ O/P node that are wrong.

But some have the +ve sign and -ve sign so we choose some of squares error fun which cal the difference blw y_k & t_k for each node of 'k'

ession

$$E(W) = \frac{1}{2} \sum_{k=1}^{N} (y_k - t_k)^2$$

:tf

$$= \frac{1}{2} \sum_{k=1}^{N} \left[g\left(\sum_{j=0}^{M} w_{jk} a_j \right) - t_k \right]^2$$

If the I/p node by 'i', output node by 'k' then

$$E(W) = \frac{1}{2} \sum_{k=1}^{N} \left[g\left(\sum_{i=0}^{k} w_{ik} x_i \right) - t_k \right]^2$$

We can't differentiat the threshold function if no hidden neurons the activation fun of O/p neuron is

$$g_{ki} = \sum_{i=0}^{L} w_{ik} - x_i$$

Using gradient decent alg:

It adjusts each weight & how the error fun changes withrespect to the diff. weights using partial deivation.

$$\frac{\partial E}{\partial w_{ik}} = \frac{\partial}{\partial w_{ik}} \left(\frac{1}{2} \sum_{k=1}^{N} (y_k - t_k)^2 \right)$$

$$\frac{\partial E}{\partial w_{ik}} = \frac{1}{2}\sum_{k=1}^{N} 2(y_k - t_k)\frac{\partial}{\partial w_{ik}}\left(y_k - \sum_{i=0}^{L} w_{ik}x_i\right)$$

Now t_k is not a fun,

so $\frac{\partial t_k}{\partial w_{ik}} = 0$ But the only $\sum_{i=0}^{L} w_{ik}x_i$ that is fun. when $i=1$. i.e w_{ik} is delivation of

$$\frac{\partial E}{\partial w_{ik}} = \sum_{k=1}^{N}(t_k - y_k)(-x_i).$$

⇒ So, the gradient decent rule direction is down hill then in weight update rule we include 'η'

$$w_{ik} \leftarrow w_{ik} + \eta(t_k - y_k)x_i$$

In the above eq" we ignore the threshold fun because neuron either fire (or) not.

Requirements of an activation function:

⇒ To model a neuron using Activation fun to follow this properties:

- It must be differenciable so that we can compute the gradient.
- It should saturated (become constant) at both ends of the range. so that the neuron either fired or does not fire.
- It should change blw the saturation values fairly quickly in the middle. All of this citerias perfectly satisfied by sigmoid function because they are S-shaped.

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

$\beta \rightarrow$ +ve parameter

$$= g'(h) = \frac{dg}{dh} = \frac{d}{dh}\left(1+e^{-\beta h}\right)^{-1}$$

$$= -1\left(1+e^{-\beta h}\right)^{-2}\frac{de^{-\beta h}}{dh}$$

$$= -1\left(1+\varepsilon^{-\beta h}\right)^{-2}\left(-\beta e^{-\beta h}\right)$$

$$= \frac{\beta e^{-\beta h}}{\left(1+e^{-\beta h}\right)^2}$$

$$= \beta\, g(h)\left(1-g(h)\right)$$

$$= \beta\, a(1-a)$$

Back propagation & Error : (using chain rule)

$$\Rightarrow \frac{\partial E}{\partial w_{ok}} = \frac{\partial E}{\partial h_k}\,\frac{\partial h_k}{\partial w_{ok}}$$

where,

$h_k = \sum_{j=0}^{M} w_{jk} a_o$ is the Σip to the O/p of neuron k. Then

$$\frac{\partial h_k}{\partial w_{ok}} = \frac{\partial \sum_{j=0}^{M} w_{jk} a_o}{\partial w_{ok}}$$

$$\frac{\partial w_{jk}}{\partial w_{ok}} = 0$$

If 'δ' for all values except $j = o$ when it is 1 i.e 'a o' then $\frac{\partial E}{\partial h_k}$ is error (or) Delta term

$$\delta(k) = \frac{\partial E}{\partial h_k}$$

$$\delta_o(k) = \frac{\partial E}{\partial h_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial h_k} \quad \text{(Chain rule)}$$

The o/p of the o/p layer, neuron 'k' is

$$y_k = g(h_k^{output})$$

$$= g\left(\sum_{j=0}^{M} w_{jk} a_j^{hidden}\right)$$

Sub. in this exp for the error at the o/p

$$\delta_o(k) = \frac{\partial E}{\partial g(h_k^{o/p})} \frac{\partial g(h_k^{o/p})}{\partial h_k^{o/p}}$$

$$= \frac{\partial E}{\partial g(h_k^{o/p})} g'(h_k^{o/p})$$

$$= \frac{\partial}{\partial g(h_k^{o/p})} \left[\frac{1}{2} \sum_{k=1}^{N} g(h_k^{o/p}) - t_k)^2\right] g'(h_k^{o/p})$$

$$= (g(h_k^{o/p}) - t_k)\, g'(h_k^{o/p})$$

$$= (y_k - t_k)\, g'(h_k^{o/p})$$

- **output layer - update function**

$$w_{ok} \leftarrow w_{ok} - \eta \frac{\partial E}{\partial w_{ok}}$$

$$= w_{ok} - \eta \delta_o(k) a_o$$

the '-' sign is downhill to min" the error.

Hidden node Activation of all the O/P node:

$$\delta_o(\sigma) = \sum_{k=1}^{N} \frac{\partial E}{\partial h_k^{O/P}} \frac{\partial h_k^{O/P}}{\partial h_\sigma^{hidden}}$$

$$= \sum_{k=1}^{N} \delta_o(k) \frac{\partial h_k^{O/P}}{\partial h_\sigma^{hidden}}$$

Input to the o/p layer, neurons come from the activation of the hidden layer neurons multiplied by the each layer weights. That means input to output the first hidden layer activation by the 2nd layer weight.

$$h_k^{O/P} = \sum_{j=0}^{M} w_{jk}\, g(h_j^{hidden})$$

$$\frac{\partial h_k^{O/P}}{\partial h_\sigma^{hidden}} = \frac{\partial g\left(\sum_{j=0}^{M} w_{jk} h_j^{hidden}\right)}{\partial h_\sigma^{hidden}}$$

then,

$$\frac{\partial h_\sigma}{\partial h_j} = 0 \quad \text{unless } j = \sigma \text{ when it is 1}$$

$$\frac{\partial h_k^{O/P}}{\partial h_\sigma^{hidden}} = w_{\sigma k}\, g'(a_\sigma) \quad \left[\because g' = \frac{\partial g(h_k^{O/P})}{\partial h_k^{O/P}}\right]$$

The hidden nodes have sigmoid activation fun.

$$g'(a_\sigma) = \beta a_\sigma (1 - a_\sigma)$$

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

Update rule v_i is

$$v_i \leftarrow v_i - \eta \frac{\partial E}{\partial v_0}$$

$$v_i - \eta\, a_\sigma (1 - a_\sigma) \left(\sum_{k=1}^{N} \delta_o(k)\, w_\sigma\right) x_i$$

If the NIw has extra hidden ... o/p ...
exactly the same contribution continuously.

The o/p activation functions :

We aimed at making the nodes act like neurons either firing (or) not firing then we use the sigmoid activatation fun" but in this we observed 2 cases.

- Regression - the o/p to be continuoued.
- Multiclass classification - only 1 of the o/p neuro to fire. so, we identified possible activatio fun to this cases using δ_0 (deltaterm) for them.

The 3 fun" are (using δ_0)

⇒ Lineal: $y_k = g(h_k) = h_k$

⇒ sigmoidal: $y_k = g(h_k) = \frac{1}{1+\exp -\beta h_k}$

⇒ Softmax : y_k

- for lineal o/ps

$$\delta_0(k) = (y_k - t(k))\, y_k$$

- for sigmoid o/ps

$$\delta_0(k) = \beta\,(y_k - t(k))\, y_k\,(1 - y_k \cdot t)$$

- for softmax o/ps

$$\frac{\partial}{\partial h_k} y_k = \frac{\partial}{\partial h_k} (\exp(h_k) (\sum_{k=1}^{N} \exp(h_k))^{-1}$$

the 'k' index runs over the o/p nodes we face 2 cases.

⇒ $K = k$

⇒ $k \neq k$

• If $k = k$, both same then expression of

$$\frac{\partial \exp(h_k)}{\partial h_k} = \exp(h_k)$$

use chain rule,

$$= \frac{\partial}{\partial h_k}(\exp(h_k))\left(\sum_{k=1}^{N} \exp(h_k)^{-1}\right)$$

$$= \exp(h_k)\left(\sum_{k=1}^{N} \exp(h_k)^{-1}\right) - \exp(h_k)\left(\sum_{k=1}^{N} \exp(h_k)\right)^{-2}$$

$$= y_k(1 - y_k)$$

• If $k \neq k$ then,

$$\frac{\partial}{\partial h_k} \exp(h_k)\left(\sum_{k=1}^{N} \exp(h_k)\right)^{-1}$$

$$= -\exp(h_k)\exp(h_k)\left(\sum_{k=1}^{N} \exp(h_k)\right)^{-2}$$

$$= -y_k y_k$$

⇒ Using Kronecker delta function:

• $i = j$ which is 1

• $i \neq j$ which is 0.

2 cases in 1 eqn to get the delta term

$$\delta_o(k) = (y_k - t_k)\, y_k(\delta_k k - y_k)$$

In the above all, last we think about whether or not the sum of squared fun. is always best one.

26/10/24

Unit-4

Dimensionality Reduction:

It can also remove noise, signifs cantly improve the results of the learning alg make the data set easier to work & make the results easier to understand especially the importance of the field for machine learning & other form of data analysis.

In this we see the 2 alg
- Locally lineally embeded
- Isomap

For dimensionality reduction we use 3 diff ways
- Feature selection
- Feature derivation
- Clustering

* Feature selection:

The features that are available & seeing whether or not they actually useful that means corelated the o/p variance.

* Feature derivation:

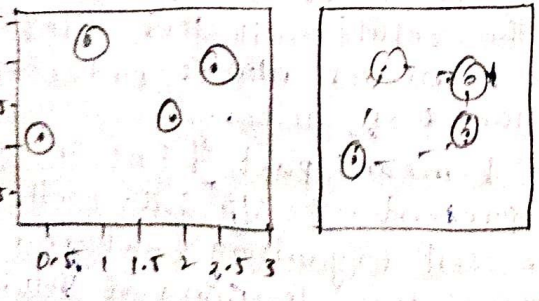
Deriving new features from alg that the old ones.

* Clustering:

Group together similar data points.

Ex:

x	y
2.00	-1.53
2.37	-2.80
1.00	-2.17
0.68	-1.8



The above 3 views of the same four points as no's where the links are nonclear. As 4 ploted points at centre as 4 points, that line on a circle.

⇒ combination of diff o/p's rather than the i/p's themselves are used.

⇒ It is inherit to the way that the decision tree works at each stage of the algorithm it decides which feature to add next.

⇒ In this dimensionality reduction testing errors at each stage to see whether or not it changes at all that features are added.

⇒ we are not then able to use the knowledge of the clases in order to reduce the problem further. so, we consider a method of dimensionality reduction i.e aimed at supervised learning is linear discriminant analysis.

⇒ This method is created to one of the best known statistics of 20th century, R.A fisher & dated from 1986.

Linear Discriminant Analysis:

A simple 2D dataset consisting of 2 classes we can compute various statistics about the data but we will settle for the mean of the 2 classes in the data μ_1, μ_2.

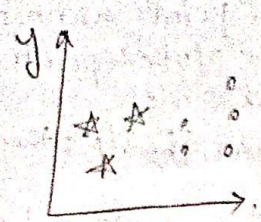
And the entire dataset mean is μ & the covariance of each class itself is

$$\Sigma_i (x_i - \mu)(x_j - \mu)^T$$

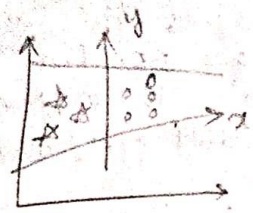
The principle of LDA is that the co-variance matrix can tell us about the scatter with in a dataset which is the amount of spread that there is within the data.

The scatter finds co-variances by probability of the

The Alternative Error functions:



A set of datapoints in 2D with two classes



The meaning of the blw class & within class scatter

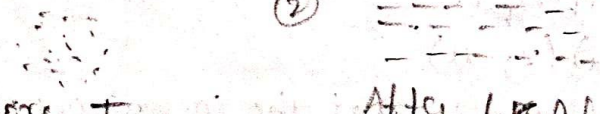
$$S_W = \sum_{\text{classes}} \sum_{\text{sec}} P_c \, (x_j - \mu_c)(x_j - \mu_c)^T$$

If we ov dataset is easy to seperate into classes then the within class scatter should be small so that each class is tightly clustered together. If we able to separate the data we also want distance blw the classes to be large. This is known as blw classes scatter.

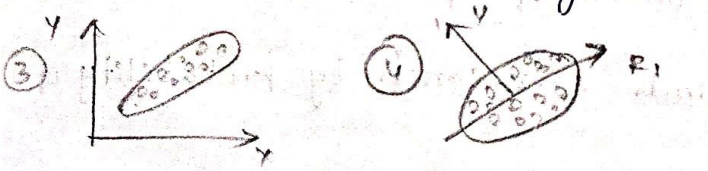
$$S_B = \sum_{\text{classes}} (\mu_c - \mu)(\mu_c - \mu)^T$$

Principal Component Analysis (PCA)

To identify a lower dimensional set of access. The idea is that by finding particular set of coordinate access it will become clear that some of dimensions are not required.



Plot of the iris data showing the 3 classes



2 different set of coordinate axies the second diagram consist of a rotation & translation of the 1st & was found using principle component analysis. The idea of the principle component is that it is a direction in the data with the largest variation. We have data matrix X & we want to rotate it so that the data lies along the directions of maximum variations. This means that we multiply our data matrix by rotation matrix. so that $Y = P^T X$, where P is choosen so that the covariance matrix of y is diagonal.

$$\boxed{COV(Y) = COV(P^T X)}$$

We can get a different handle on this by using some lineau algebra & definition of covariance is

$$\begin{aligned} COV(Y) &= E(YY^T) \\ &= E[(P^T X)(P^T X)^T] \\ &= E[(P^T X)(X^T P)] \\ &= P^T E(XX^T) P \\ &= P^T \, COV(X) \, P \end{aligned}$$

The 2 extra things that we needed to know $(P^T X)^T = X^T P^T = X^T P$ & $E(P) = P$ where P → covariance it is not a data dependent matrix. because,

$P \, COV(Y) = PP^T \, COV(X) \, P = COV(X) P$.

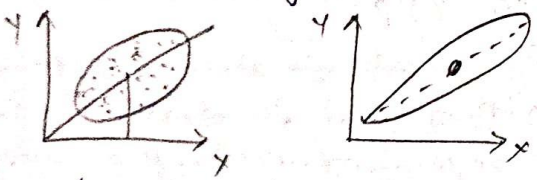
Rotation matrix is $P^T = P^{-1}$ as covariance Y is diagonal P as set of column vectors, then $P \, COV(Y) = [\lambda_1 P_1, \lambda_2 P_2, \ldots \lambda_n P_n]$

→ Intersecting eq is $\boxed{XP_i = zP_i}$ Analysis (PCA) λ is a column vector while z is a full matrix & it can be apply to each of the P_i vector

that makeup P. These directions are enough that they have a name igon vectors & the amount that they rescale the axises & known as eigen values.

Algorithm

⇒ write n data points $x_i = \{x_1, x_2 \dots x_m\}$
⇒ Put these vectors into a matrix X
⇒ Center the data points by subtracting the mean of each col, putting it into matrix B.
⇒ compute the co-variance matrix $C = \frac{1}{N} B^T B$
⇒ compute eigen values & vectors of C, so $V^{-1} C V = D$ where V holds the eigen vectors of C & D is the MM diagonal eigen matrix.
⇒ Sort the columns of D into order of decreasing eigen values & apply the same order to the col of B
⇒ Reject those with eigen values less than some n living dimensions in the data.



The above 2 diff examples of using PCA. 2D data from an eclipse being mapped into 1 principal component which lies along with the principal axis of the eclipse.

Relation with the MLP:

PCA has been used in the SOM alg to initialize the weights thus reducing the amt of learning that is req & that it is very useful for dimensionality reduction. The auto associ

something very similar to the principal component of the data in the hidden nodes. PCA is linear it just rotates & translate the axis it can't do anything more complicated. They can only prefer & perform linear tasks i.e you can use the 3 layers of the MLP 1st layer is computing some non-linear transformation of the data.

Kernel PCA:

⇒ One problem with PCA is, the directions of variations are all st. liner. so that we can use the auto-associater with multiple hidden layers.
⇒ But there is a very extension to PCA that uses the kernel trick.
⇒ This apply a function $\phi(\cdot)$ to each datapoint 'x' that transforms the data into kernel space. And then perform normal lines in the PCA space.

$$C = \frac{1}{N} \sum_{n=1}^{N} \phi(x_n)\, \phi(x_n)^T_N$$

Eigen vector equation:

$$\lambda(\phi(x_i) V) = (\phi(x_i)(V)) \left[i = 1 \dots N\right]$$

where,

$V = \sum_{j=1}^{N} \alpha_j \phi(x_j)$ are the eigen vectors of the original problem.

α_j → is the kernel problem. at this point we can apply the kernel trick and produces N×N matrix k.

$$k(i,s) = (\phi(x_i)\ \phi(x_s))$$

Putting those together we get the equation

$$N \lambda k\alpha = k^2 \alpha$$ — target point

Matrix

And left multiplied by k^{-1} to produce it to

$$N \lambda k\alpha = k\alpha$$

computing the projection of new datapoint x into the kernel PCA space requires

$$(V^k \cdot \phi(x)) = \sum_{i=1}^{N} \alpha_i^k \, (\phi(x_i) \cdot \phi(x_i))$$

Kernel PCA Algorithm:

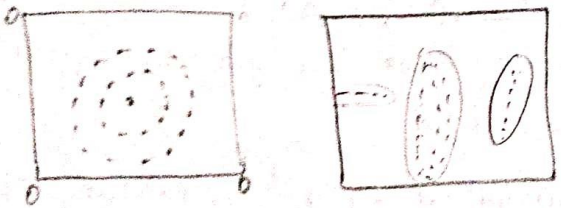
⇒ Choose a kernel and apply it to all pairs of points to get matrix 'K' of distance b/w the pairs of points in the transformed space.
⇒ compute the eigen values & eigen vectors of 'K'
⇒ Normalize the eigen vectors by the squareroot of the eigen values.
⇒ Retain the eigen vectors corresponding to the largest eigen values.

The whole kernel PCA only trick of implementation is diagonalization of 'K'

Example of PCA:

Plot of the 1st 2 non linear principal components of the iris data. showing that 3 classes are clearly differential.

A very non linear dataset consisting of 3 centroids i.e circles & the kernel PCA mapping of the data set which requires only one component to seperate the data.



The kernel alg leads to end $O(n^3)$.
⇒ The o/p shows the kernel PCA when applied to the dataset. It produces the only one component as input.

Factor Analysis:

⇒ The idea of the factor analysis is to ask whether the data is observed can be explained by a smaller no of uncorelated factors (or) Latent variables
⇒ The problem of factor analysis is to find these independent factors & the noise that is inherite in the measure of each factor.
ex: IQ Test
⇒ A data set in the N×M matrix x ie each row of x is an M dimensional datapoints & x has covariance matrix 'Σ'
⇒ With PCA we center the datapoints by subtracting of the mean of the each variable. The mean is $E[b_i] = 0$, the model is $\boxed{X = WY + \epsilon}$. where x is the observations & ε is the noise, b_i is the independent variable.
⇒ If the noise in the data the covariance will be zero. i.e the data come from a set of separate independent processes.
⇒ The aim of the factor analysis is to try to find out a set of factor loadings w_{ij} and the values for the valience of the noise parameter 'Ψ' so that the data in x can be reconstructed from the parameters (or) so that we can perform dimensionality reductions.
⇒ We first define the long likelyhood

$$Q(\theta_t | \theta_{t-1}) = \int P(x|y, \theta_{t-1}) \log(P(y|x, \theta_t) P(x)) dx$$

(x: axis, θ_{t-1}: data)

⇒ we replace several of the terms,

$$\delta(\theta_t|\theta_{t-1}) = \frac{1}{2}\int p(x|y,\theta_{t-1}) \log(\det(\psi^{-1})) - (y-wx)^T \psi^{-1}(y-wx)) dx$$

(noise)

⇒ we apply linear algebra we get,

$$w_{new} = (y E(x|y)^T)(E(xx^T|y))^{-1}$$

$$\psi_{new} = \frac{1}{N} \text{diagonal}\,(xx^T - wE(x|y)y^T)$$

All the diagonal (on the diagonal)

$$E(x|y) = w^T(ww^T+\psi)^{-1} b$$

$$E(xx^T|x) - E(x|y)E(x|y)^T = I - w^T(ww^T+\psi)^{-1}w$$

Independent Component Analysis: (ICA)

⇒ There is a related approach to Factor analysis that is known as ICA.

⇒ That means in PCA the components were choosen so that the we orthogeral uncorelated. so we required that the components are statistically independent.

⇒ The main problem in ICA is blind source separation i.e the data we see are actually created by a set of undelined physical processes that are independent.

⇒ we want to find a transformations that turns into a mixture of independent sources (or) components



⇒ The blind source separation is also known as the cocktail party problem. i.e if you are at a party then you are ears hears lots of diff sounds coming from lots of diff locations (People talking, click of glasses, background music) but you are some how able to focus on the voice of the people you are talking to and can infact separate of the sounds from all of the diff sources even though they are mixed together. Based on the above ex the cocktail party problem is the challenge of separate these sources although there is one wrinkle. For the alg to work you need as many ears as there are sources.

⇒ Suppose we have 2 noises s_1^t, s_2^t where the top index covers the fact that there are lots of datapoints appearing all time & 2 micriophones that hear things giving 2 i/ps x_1^t, x_2^t the sounds that are heared come from the source as

$$x_1 = as_1 + bs_2$$

$$x_2 = cs_1 + ds_2$$

In the form of matrix, $X = AS$

A → Mixing Matrix

We actually know about the sources & signals there are 3 things:

- The mixtures are not independent even though the sources are
- The mixtures will look like normal distributions even if the the sources are not.
- The mixture will look more complicated than sources

⇒ If we find factors that are independent of each other then they are probably sources.

⇒ We can measure the amount of independence blw 2 variables by using the mutual information.

Locally Linear Embedding: (LLE)

⇒ The locally linear alg, which is called locally linear Embedding.

⇒ The idea of the LLE is that by making linear approximations we will make some errors. So we should make those errors as small as possibly by making the patches small.

⇒ where there is lots of non-linearity in the data. That type of errors is called as reconstructor errors.

And simply the sum of squares of distance blw the original point & reconstruction.

$$E = \sum_{i=1}^{N} \left(x_i - \sum_{j=1} w_{ij} x_j \right)^2$$

the w_{ij} → weights of how much effects the j^{th} data point has on the reconstruction of i^{th} one.
Those points that are close to the current datapoint are used. for that there are 2 common ways to create neighbourhoods.

- Points that are less than some predefined distance 'd' to the current point / neighbourhood.
- The k-nearest points are neighbours.

⇒ To reapply the same basic cost function but minimize it according to the positions 'y'

$$y_i = \sum_{i=1}^{N} \left(y_i - \sum_{j=1} w_{ij} y_j \right)^2$$

Algorithm:

1. Decide on the neighbours of each points (k-nearest neighbour)
   - compute distance blw every pair of points.
   - find the k-smallest distance.
   - set $w_{ij} = 0$ for all other points.
   - for each point x_i, create a list of its neighbours locations z_i
   - compute $z_i = z_i - x_i$
2. Compute the weight matrix W that minimizes according to the constrains.
   - compute local covariance $C = ZZ^T$ where Z is the matrix of z_is
   - Solve $CW = I$ for W where I is the $N \times N$ identity matrix.
   - set $w_{ij} = 0$ for non neighbours
   - set other elements to $W / \Sigma(W)$
3. Compute the lower dimensional vectors Y_i that minimizes according to the constrains.
   - create $M = (I - W)^T (I - W)$
   - compute the eigen values & eigen vectors of M
   - sort the eigen vectors into order by size of eigen values.
   - set the q^{th} row of Y to be the $q+1$ eigen vector corresponding to the eigen value. that may be zero. q^{th} smallest

ISO Map:

⇒ It tries to minimize to global euros by looking at all of the pair wise distance & computing global distance. It is a variant of the standard multi dimensional scaling algorithm.

Multi Dimensional Scaling (MDS)

⇒ It tries to find a linear approximation to the full data space that embeds the data into a lower dimensionality.
⇒ In the case of MDS the embedding tries 2 preserve the distance blw all pairs of points.
⇒ Their are lots of choices for MDS cost function but the most common ones are "kruskal-shepshaid scaling". is known as "Least squares"

$$S_{KS}(z_1, z_2 \dots z_n) = \sum_{i \neq i'} (d_{ii'} - \|z_i - z_{i'}\|)^2$$

Sammon mapping:

$$S_{SM}(z_1, z_2 \dots z_m) = \sum_{i \neq i'} \left(\frac{d_{ii'} - \|z_i - z_{i'}\|}{d_{ii'}} \right)^2$$

This puts more weight on to short distance. so that neighbouring points stay. the current distance a path.
⇒ Gradient descent is also used to minimize the distance
⇒ Their is another version of the MDS is called classical MDS. That uses similarities blw datapoints rather than distances.

⇒ These can be constructed from a set of distances blw the centers inner product i.e $S_{ii'} = (x_i - \bar{x})(x_{i'} - \bar{x})^T$

MDs Algorithm:

⇒ Compute the matrix of squared pairwise similarities 'D'

$$D_{ij} = \|x_i - x_j\|^2$$

⇒ Compute $J = I_N - \frac{1}{N}$

where, I_N = $N \times N$ identity Matrix

N = no. of the datapoints

⇒ Compute $B = -\frac{1}{2} JDJ^T$

⇒ Find the L largest eigen values λ_i of B together with the corresponding eigen vectors e_i.

⇒ Put the eigen values into a diagonal matrix V & set of eigen vectors to be columns of matrix P.

⇒ Compute the embedding as $X = PV^{\frac{1}{2}}$

- This classical MDs alg works fine on flat manifolds (data points)
- IsoMap approximating them by assuming that the distance blw pair of points that are closed together since over a small distance the non linearity of the manifold home matter.

IsoMap Algorithm:

⇒ Construct the pair wise distance blw all pairwise points.

⇒ Identify the neighbours of each point to make a weighted 'G'.

⇒ Estimate the distance d_G by finding shortest paths.

⇒ Apply classical MDs to the set of d_G

- The Time complexities of alg (disasters kruskal) are $O(N^2)$. If it increase then time complexity is $O(N^3)$

Least Square Optimization:

⇒ For many of the alg that we have derived we have used a least square errors function. Such as the error of the MLP & the linear regression.

⇒ The levenberg-Marquardt alg which is a trust region optimization alg.

⇒ compared to the least square optimization it is specially derived alg.

The levenberg-Marquardth alg:

⇒ For least square problem the objective fun that we are optimizing is $f(x) = \frac{1}{2}\sum_{i=1}^{m} r_j^2(x) = \frac{1}{2}\|r(x)\|^2$, where, $\frac{1}{2}$ is the derivative $r(x)$ is linear fun that means $r(x) = (r_1(x), r_2(x) \ldots r_m(x))^T$ we can write the transpose of r is "Jacobin"

$$J^T(x) = \left\{ \frac{\partial r_1}{\partial x_1} \quad \frac{\partial r_2}{\partial x_2} \quad \cdots \quad \frac{\partial r_m}{\partial x_1} \right\} = \frac{\partial r_i}{\partial x_i}$$

If we use gradient descent we want to compute directly $\nabla f(x) = J(x)^T r(x)$. $\nabla^2 f(x)$

$$= J(x)^T J(x) + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x)$$

If $\nabla^2 r_j(x) = 0$ for all J then the soln is

$$\nabla f(x) = J^T(Jx + r) = 0 \Rightarrow J^T J = -J^T r(x)$$

This is a linear squared problem can be solved effectively $Ax = b$ where $A = J^T J, B = -J^T r(x)$. st directly as $x = -(J^T J)^{-1} J^T r$

In linear algebra to find x in variety of diff ways i.e cholesky factorization, QR factorization, singular value decomposition. SVD is the decomposition of matrix A of size $m \times n$. where U & V are the orthogonal matrices the inverse of the matrix of its transpose that means $U^T U = UV^T = I$ where I is the identity matrix & U is of size $m \times m$ & V is of size $n \times n$, 's' is diagonal matrix of size $m \times n$.

The levenberg - Marquardt Alg

1) Given Start point x

2) while $J^T r(x) >$ tolerance & maximum no. of iterations not exceeded
- Repeat
  - solve $(J^T J + vS)\,dx = -J^T r$ for dx using linear least square
  - set $x_{new} = x + dx$
  - compute the ratio of actual & prediction reductions
    - * actual = $\|f(x) - f(x_{new})\|$
    - * Predicted = $\nabla f^T(x) \times x_{new} - x$
    - * P = actual / predicted
  - if $0 < P < 0.25$
    - * accept step : $x = x_{new}$
    - * else if $P > 0.25$
      - → accept step : $x = x_{new}$
      - → increase trust region size (reduce v)
    - * else
      - → reject step
      - → reduce trust region (increase v)
      - → until x is updated or maximum no. of iterations are exceded.

→ The fun is mandatory, complicated that is definately not to fit linear least square.

$$y = f(P_1, P_2) = P\cos(P_2 x) + P_2 \sin(P_1 x)$$

Difference blw the data (y) & the current fit parameter estimates $\hat{P}_1$, $\hat{P}_2$ $r = y - \hat{P}_1 \cos(\hat{P}_2 x) + \hat{P}_2 \sin(\hat{P}_1 x)$

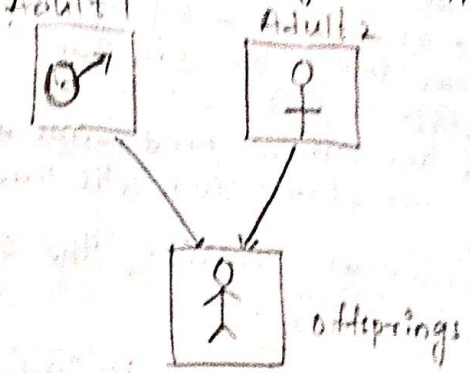
Evolutionary Learning

⇒ Evolution works on a population through an imaginary fitness landscape which has an implicit bias towards fitter (or) unfitter.

→ The Genetic alg models the genetic process that gives rise to evolution.

→ ex: Reproduction of the data.

Show both parents gives some genetic info to their offsprings



Each adult in the mating pair passes one of their 2 womosomes to their offsprings.

1) Genetic Algorithm (GA):

⇒ A genetic alg is a heuristic search alg. It will reflects the process of natural selection where the fitter individuals are selected for reproduction inorder to produce offspring of the nextgen.

→ The genetic Alg is a computational approximation to how evolution performs search which is why producing modifications of the parent 'genomes' in their offspring & thus producing new individuals with diff fitness.

⇒ For this principal, a simple genetic inside a computer to solve problems with in it are:
- A method for representing problems as chromosomes.
- A way to calculate the fitness of the solution.
- A selection method to choose parents.
- A way to generate offspring by breeding the parents. Ex: NP Complete, kNapsack etc.

* String Representation:

⇒ NP complete the first thing that we need is some way to represent the individual soln to the chromosomes.

⇒ chromosome is individual entity of the data.

⇒ In genetic alg, we a string, with each element of string

⇒ We make the string 'L' where L is the total no. of things we would like to take with us. And make each unit a binary digit. '0' — for the things we will take '1' for the not take (not fit in that)

⇒ To work this out we need some way to decide how well each string full fills the problem criteria.

⇒ This is known as fitness of the string.

Evolutionary fitness:

⇒ It is defined how to fit an individual is ?

⇒ The fitness fun can be encoding the string in the form of problem specific part of the genetic algorithm.

⇒ That means, clearly the best string should have the highest fitness & the fitness should be decreases as the strings do less well on the problem.

* Population :

⇒ We can now measure the fitness of any string the genetic alg works on a population of strings with the 1st generation usually being created randomly.

⇒ After the initial popn is choosen randomly the alg evolves to produce each successive generation with the hope being that there will be progressively fitter individuals in the population.

⇒ As the member of generations increases.

* Generating Offsprings: (parent selection)

⇒ Offspring is created by exchanging the genes of parents.

⇒ The idea is that avg fitness will improve if we select strings that already relatively fit compare to the other members of the populations which is exploitization of our current population.

⇒ There are 3 commonly employed ways to do this to produce better results.

- Tournament selection
- Truncation selection
- fitness proportion selection.

Tournament selection - repetedly pick 4 strings from the population, with replacement & then put the fittest 2 of them into the matting pool

Truncation selection - Pick some function 'f' of the best strings ~~ing~~ ignore the rest.

fitness proportion selection - The better option is to select strings probabilities with the probability of strings being selected being proportional to its fitness.

string is α

$$p^{\alpha} = \frac{f^{\alpha}}{\Sigma_{\alpha'} f^{\alpha'}}$$

when,

f^{α} is the fitness if the fitness is not +ve 'f' needs to replace by exp(sf)

$$p^{\alpha} = \frac{\exp(sf)^{\alpha}}{\Sigma_{\alpha'} \exp(sf)^{\alpha'} f}$$

where, s ⟶ selection strength.

Genetic Operators

⇒ Having selected how breeding pairs need to decide how to combine their 2 strings to generate the offspring which is the genetic part of the algorithm.

⇒ There are 2 genetic operators are used:

- cross over
- Mutation

Cross over: The most common way of doing this is. to pick one point at random in the string & use parent one for the first part of the one up to

the crossover point and parent 2 for the rest.

Ex:

```
1 0 0 1 1 0 0 0 1 0 1        1 0 0 1 1 0 0 0 1 0 1
0 1 1 1 1 0 1 0 1 1 0        0 1 1 1 1 0 1 0 1 1 0
-----------------------      -----------------------
1 0 0 1 1 0 1 0 1 1 0        0 1 1 1 1 0 0 0 1 0 1
```

fig: Single crossover

```
1 0 0 1 1 0 0 0 1 0 1        Random string - 0 0 1 1 0 1 1 0 1 1 0
0 1 1 1 1 0 1 0 1 1 0        -----------------------------------
-----------------------      1 0 0 1 1 0 0 0 1 0 1
1 0 0 1 1 0 1 0 1 0 1        0 1 1 1 1 0 1 0 1 1 0
                             -----------------------------------
                             1 0 1 1 1 0 1 0 1 1 1
```

fig: Multiple crossover

fig: Uniform Crossover

Single Crossover - A position in the string is choosen at random and the offspring is made up of the first Part of parent 1 + the second part of parent 2

Multipoint crossover - Multiple points are choosen with the offspring being made in the same way

Uniform Crossover - Random nos are used to select which parent to take each element from.

⇒ Crossover is not always useful depending upon the problem, the strings that are generated by crossover might not even be valid. The idea is that genetic alg works well on problems where the soln comes from putting together lots of little soln.

Mutation:

⇒ The another genetic operation, which effectively perform random search.

⇒ The value of any element of the string can be changed, governed by some probability 'p'.

⇒ For ourbinary alphabet in the knapsack, mutation causes a bit-flip.

Ex:

```
1 0 0 1 1 0 1 0 1 1 1
      ↓
1 0 0 1 0 0 1 0 1 1 1
```

fig: The effect of mutation on a string.

#Eltism: Tournaments: Niching;

⇒ This means that the max. fitness in the next gen can decreases atleast temporarily.

⇒ And since in the end we are only interested in the best soln. This seems a bit risky.

⇒ There is a variety of ways to avoid this
- Eltism
- Tournaments
- Niching.

* Eltism - which takes some members of the fittest string from 1 generation + puts them directly into the next population, Replacing string that are already there either at random (or) by choosing the least of it fit to replace.

* Tournaments - where the 2 parents + their 2 offspring completes with the 2 fittest out of the 4 being put into the new popn

⇒ Why Eltism + Tournaments ensures that good soln (or) not lost they both have the problem of Prematured convergency.

⇒ To overcome this problem through Niching also known as Island population.

* Niching - The popn is separated into several subpopn which all involves independently some period of time so that they are likely to have converged to diff local maxima + a few nos of some population are optionally injected as immigrants into another sub population.

The Basic Genetic Algorithm

- Initialization
  - Generate 'N' random strings of length 'l' with the choosen alphabet.
- Learning
  - Repeat
    - Create a new population
    - Repeat
      - * Select 2 string from current population preferably using fitness proportional selection.
      - * Recombine them in pairs to produce 2 new strings.
      - * Mutate offspring
      - * Either add 2 offspring to the population or use tournament to put to strings from the 4 of Parents & offspring into the population.
    - Untill 'N' strings for the new populations are generated.
    - optionally use Elitisim to take the fittest string from the parent generation & replace some other from the child generation.
    - keep track of the best string in the new population.
    - Replace the old popn in the new one.
  - Untill stopping criteria met.

2) Map colouring :

⇒ Graph colouring is a discrete optimization problem we want to colour a graph using only k colours & choose them in such a way that where the adjacent regions have diff colours.

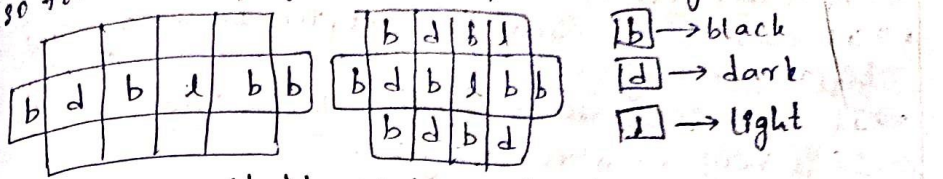
⇒ Even though it might not be possible we are going to try to solve the problem using a genetic algorithm.

⇒ There are 3 basic tasks that need to be perform.

- Encode possible soln as Strings
- choose a suitable fitness function
- Choose suitable genetic operators

Encode possible soln as strings

For this problem we choose our alphabets to consist of the 3 possible shades black is 'b', dark is 'd', light is 'l' so for the 6 region map a possible string is "bdblb"

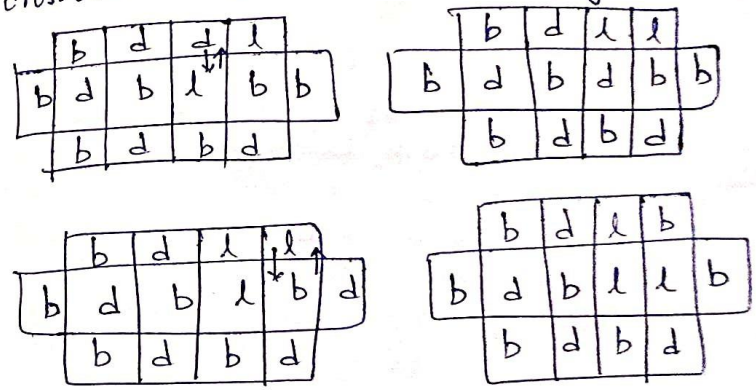


choose a suitable fitness function

The thing that we want to minimize is the no of times the 2 adjacent regions have the same colour.

choose suitable genetic operators

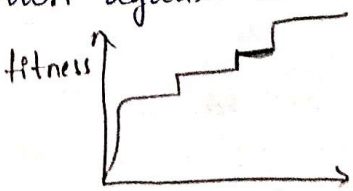
we will use the standard genetic operators for this crossover under mutation clearly.



Punctuated Equilibrium

⇒ The way that evaluation seems to work is known as Punctuated equilibrium.

⇒ There is basically a steady popn of some spices for a long time & then something changes over a very short time period there is a big change & everything settle down again.



Examples: knapsack problem, 4 peaks problem

Limitations of Genetic Algorithm

⇒ There are lots of good things about genetic they are very slow.
⇒ It takes lots of time.
⇒ It can be reached the o/p often be along before a string is produced.
⇒ It don't know about fitness landscale
⇒ It is very hard to analyize.
⇒ It makes the popn less divorce.
⇒ It reduces the mean fitness
⇒ We can't guarantee that the alg cover all, it not to give optional soln
⇒ It is risky to apply without knowledg the alg works.

Unit-5

Elements of Reinforcement Learning

There are four main elements of Reinforcement Learning, which are given below:

1. Policy

A policy can be defined as a way how an agent behaves at a given time. It maps the perceived states of the environment to the actions taken on those states. A policy is the core element of the RL as it alone can define the behavior of the agent. In some cases, it may be a simple function or a lookup table, whereas, for other cases, it may involve general computation as a search process. It could be deterministic or a stochastic policy:

For deterministic policy: a = π(s)

For stochastic policy: π(a | s) = P[At =a | St = s]

Elements of Reinforcement Learning

There are four main elements of Reinforcement Learning, which are given below:

2) Reward Signal: The goal of reinforcement learning is defined by the reward signal. At each state, the environment sends an immediate signal to the learning agent, and this signal is known as a **reward signal**. These rewards are given according to the good and bad actions taken by the agent. The agent's main objective is to maximize the total number of rewards for good actions. The reward signal can change the policy, such as if an action selected by the agent leads to low reward, then the policy may change to select other actions in the future.

3) Value Function: The value function gives information about how good the situation and action are and how much reward an agent can expect. A reward indicates the **immediate signal for each good and bad action**, whereas a value function specifies **the good state and action for the future**. The value function depends on the reward as, without reward, there could be no value. The goal of estimating values is to achieve more rewards.

Elements of Reinforcement Learning

4) Model: The last element of reinforcement learning is the model, which mimics the behavior of the environment. With the help of the model, one can make inferences about how the environment will behave. Such as, if a state and an action are given, then a model can predict the next state and reward.

The model is used for planning, which means it provides a way to take a course of action by considering all future situations before actually experiencing those situations. The approaches for solving the RL problems **with the help of the model** are termed as the **model-based approach**. Comparatively, an approach **without using a model** is called a **model-free approach**.

Key points of this example:

- **State:**
- At any point in time, the robot's "state" is its current location within the maze, including information like proximity to walls and potential pathways.
- **Action:**

- The robot can choose actions like "move forward", "turn left", "turn right" based on its current state.
- **Reward:**
- The robot receives a positive reward when it moves closer to the goal and a negative reward when it moves further away or hits a wall.
- **Learning process:**
- Through repeated trials, the robot learns which actions in each state lead to the highest cumulative reward, effectively building a strategy to navigate the maze without getting lost

How it works:

- **Exploration vs. Exploitation:**

  Initially, the robot might explore different paths randomly to understand the maze layout.

- **Policy Improvement:**

Over time, the robot gradually starts to prioritize actions that have led to positive rewards in the past, leading to a more efficient navigation strategy.

- **Q-Learning:**

A common reinforcement learning algorithm that can be used in this scenario involves creating a "Q-table" where each cell represents a state-action pair, storing the expected future reward for taking a particular action in a given state.

Real-world applications:

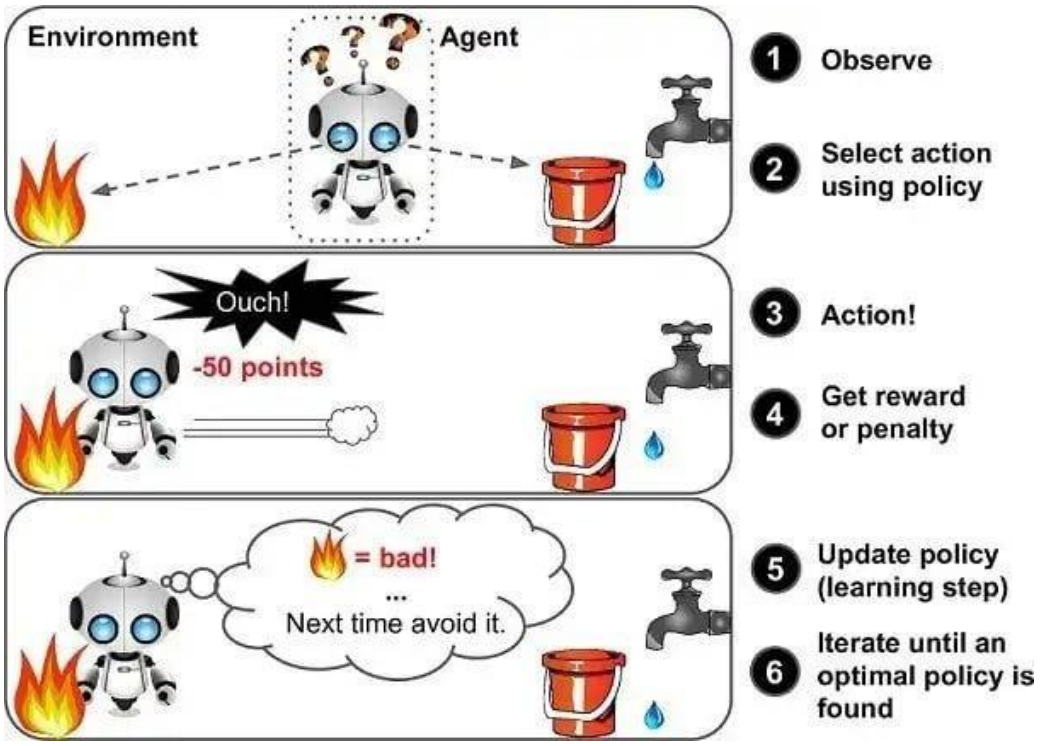
- **Robot navigation:**
- This concept is directly applicable to robots navigating complex environments like warehouses or disaster zones.
- **Autonomous vehicles:**
- Self-driving cars can use reinforcement learning to learn how to navigate roads, avoiding obstacles and making optimal route decisions.
- **Game playing:**
- AI agents in video games can learn to play effectively by receiving rewards for achieving goals within the game

Real-world applications:

- **Robot navigation:**

This concept is directly applicable to robots navigating complex environments like warehouses

or disaster zones.



Markov Chain Monte Carlo Methods – Sampling – Proposal Distribution – Markov Chain Monte Carlo – Graphical Models – Bayesian Networks – Markov Random Fields – Hidden Markov Models – Tracking Methods

Monte Carlo methods, or **Monte Carlo experiments**, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle. The name comes from the Monte Carlo Casino in Monaco, where the primary developer of the method.

Monte Carlo methods are mainly used in three distinct problem classes: **optimization, numerical integration,**

and generating draws from a probability distribution. They can also be used to model phenomena with significant uncertainty in inputs, such as calculating the risk of a nuclear power plant failure. Monte Carlo methods are often implemented using computer simulations, and they can provide approximate solutions to problems that are otherwise intractable or too complex to analyze mathematically.

Sampling distribution is essential in various aspects of real life. Sampling distributions are important for inferential statistics. A sampling distribution represents the distribution of a statistic, like the mean or standard deviation, which is calculated from multiple samples of a population. It shows how these statistics vary across different samples drawn from the same population.

In this article, we will discuss the Sampling Distribution in detail and its types along with examples and go through some practice questions too.

Sampling distribution is also known as a **finite-sample distribution**. Sampling distribution is the probability distribution of a statistic based on random samples of a given population. It represents the distribution of frequencies on how spread apart various outcomes will be for a specific population.

Some important terminologies related to sampling distribution are given below:

- **Statistic:** A numerical summary of a sample, such as mean, median, standard deviation, etc.
- **Parameter:** A numerical summary of a population is often estimated using sample statistics.
- **Sample**: A subset of individuals or observations selected from a population.
- **Population:** Entire group of individuals or observations that a study aims to describe or draw conclusions about.
- **Sampling Distribution:** Distribution of a statistic (e.g., mean, standard deviation) across multiple samples taken from the same population.

Central Limit Theorem(CLT): A fundamental theorem in statistics stating that the sampling distribution of the sample mean tends to be approximately normal as the sample size increases, regardless of the shape of the population distribution

Some important terminologies related to sampling distribution are given below:

- **Standard Error:** Standard deviation of a sampling distribution, representing the variability of sample statistics around the population parameter.
- **Bias**: Systematic error in estimation or inference, leading to a deviation of the estimated statistic from the true population parameter.
- **Confidence Interval:** A range of values calculated from sample data that is likely to contain the population parameter with a certain level of confidence.
- **Sampling Method:** Technique used to select a sample from a population, such as simple random sampling, stratified sampling, cluster sampling, etc.
- **Inferential Statistics:** Statistical methods and techniques used to draw conclusions or make inferences about a population based on sample data.
- **Hypothesis Testing:** A statistical method for making decisions or drawing conclusions about a population parameter based on sample data and assumptions about the population.

3 main factors influencing the variability of a sampling distribution are:

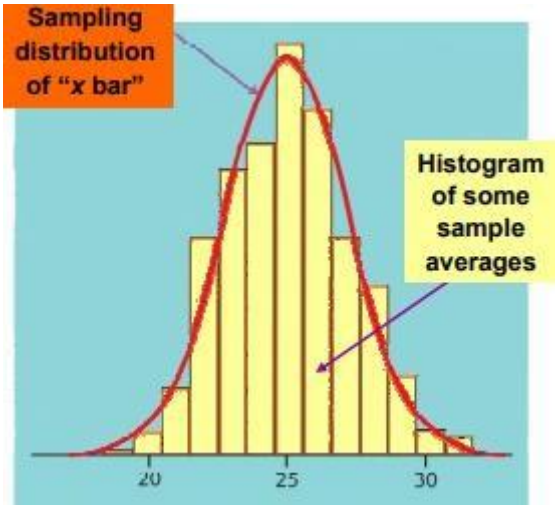
1. **Number Observed in a Population:** The symbol for this variable is "N." It is the measure of observed activity in a given group of data.
2. **Number Observed in Sample:** The symbol for this variable is "n." It is the measure of observed activity in a random sample of data that is part of the larger grouping.
3. **Method of Choosing Sample:** How you chose the samples can account for variability in some cases.

Sampling Distribution of Mean

it focuses on calculating the mean

or rather the average of every sample group

- chosen from the population and plotting the data points. The graph shows a normal distribution where the center is the mean of the sampling distribution, which represents the mean of the entire population.
- Mean, or center of the sampling distribution of $\bar{x}$, is equal to the population mean, μ.
- $\mu_{\bar{x}} = \mu$



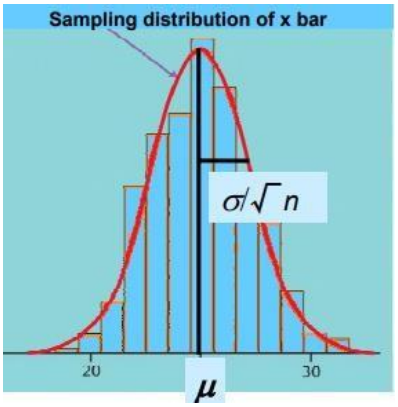
Standard deviation of the sampling distribution is σ/√n,

where n is the sample size.

$\sigma_{\bar{x}} = \sigma/\sqrt{n}$

Standard deviation of the sampling distribution measures how much the sample statistic varies from sample to sample. It is smaller than the standard deviation of the population by a factor of √n.

Averages are less variable than individual observations



T-Distribution

Sampling distribution involves a small population or a population about which you don't know much. It is used to estimate the mean of the population and other statistics such as confidence intervals, statistical differences and linear regression. T-distribution uses a t-score to evaluate data that wouldn't be appropriate for a normal distribution.

Formula for the t-score, denoted as t, is:

$$t = [x - \mu] / [s / \sqrt{(n)}]$$

where:

- **x** is Sample Mean

- **μ** is Population Mean (or an estimate of it)
- **s** is Sample Standard Deviation
- **n** is Sample Size

This formula calculates the difference between the sample mean and the population mean, scaled by the standard error of the sample mean. The t-score helps to assess whether the observed difference between the sample and population means is statistically significant.

Markov chain Monte Carlo methods create samples from a continuous random variable, with probability density proportional to a known function. These samples can be used to evaluate an integral over that variable, as its expected value or variance.

Practically, an ensemble of chains is generally developed, starting from a set of points arbitrarily chosen and sufficiently distant from each other. These chains are stochastic processes of "walkers" which move around randomly according to an algorithm that looks for places with a reasonably high contribution to the integral to move into next, assigning them higher probabilities.

Random walk Monte Carlo methods are a kind of random simulation or Monte Carlo method. However, whereas the random samples of the integrand used in a conventional Monte Carlo integration are statistically independent, those used in MCMC are autocorrelated. Correlations of samples introduces the need to use the Markov chain central limit theorem when estimating the error of mean values.

These algorithms create Markov chains such that they have an equilibrium distribution which is proportional to the function given.

In statistics, **Markov chain Monte Carlo** (**MCMC**) is a class of algorithms used to draw samples from a probability distribution. Given a probability distribution, one can construct a Markov chain whose elements' distribution approximates it – that is, the Markov chain's equilibrium distribution matches the target distribution. The more steps that are included, the more closely the distribution of the sample matches the actual desired distribution.

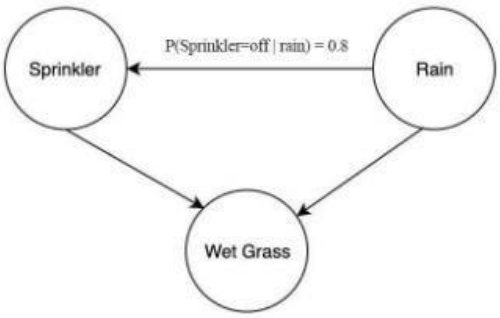
Markov chain Monte Carlo methods are used to study probability distributions that are too complex or too highly dimensional to study with analytic techniques alone. Various algorithms exist for constructing such Markov chains, including the Metropolis–Hastings algorithm.

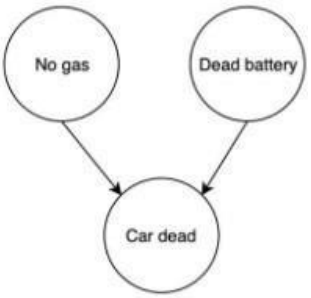
Graphical Model

One major difference between Machine Learning (ML) and Deep Learning (DL) is the
amount of domain knowledge sought to solve a problem. ML algorithms regularly exploit domain knowledge. However, the solution can be biased if the knowledge is incomplete. However, if it is done correctly, we can solve problems more efficiently. The Graphical model (GM) is a branch of ML which uses a graph to represent a domain problem. Many ML & DL algorithms, including Naive Bayes' algorithm, the Hidden Markov Model, Restricted Boltzmann machine and Neural Networks, belong to the GM. Studying it allows us a bird's eye view on many ML algorithms. In this article, we focus on the basic in representing a problem using a graphical model. Later
in this series, we will discuss how inference is made and how the model is trained.

Probabilistic graphical modeling combines both probability and graph theory. The probabilistic part reason under uncertainty. So we can use probability theory to model and argue the real-world problems better. The graph part models the dependency or correlation.

In GM, we model a domain problem with a collection of random variables ($X_1, \ldots,$ Xn) as a joint distribution $p(X_1, \ldots, Xn)$. The model is represented by a graph. Each node in the graph represents a variable with each edge representing the dependency or correlation between two variables.





For example, the sprinkler problem above is modeled by three variables:
1. whether the sprinkler is automatically on or off,
2. whether it is raining, and
3. whether the grass is wet.
And we model the problem with the joint probability $p(X_1, \ldots, X_n)$

The likelihood of the observations $P(E)$ — the chance of the lab results (Evidence E).

The marginal probability $P(X_1)$, $P(X_1, X_3)$, etc … — the chance of having Alzheimer when you are 70.

· Conditional probability (or a posterior belief based on evidence), $P(Y|E)$ — the chance of having Alzheimer given your parent has it.

· Maximum a Posterior (MAP), arg max $P(X, E)$ — the most likely disease given the lab results.

For example, the marginal probability of having the grass wet is computed by summing over other variables. For the conditional probability, we can first apply Bayes' Theorem followed by the corresponding marginal probabilities.

$$p(X_1) = \sum_{X_2} \sum_{X_3} \ldots \sum_{X_n} p(X_1, \ldots, X_n) \qquad p(X_1 \mid X_2) = \frac{\sum_{X_3} \ldots \sum_{X_n} p(X_1, \ldots, X_n)}{\sum_{X_1} \sum_{X_3} \ldots \sum_{X_n} p(X_1, \ldots, X_n)}$$

Joint probability

Which probability distributions below is more complex? Let's give some serious thoughts here because it helps us to understand machine learning (ML) better.

$$p(d, i, g, s, l), \quad p(d)\, p(i)\, p(g \mid d, i)\, p(s \mid i)\, p(l \mid g)$$

For the L.H.S. distribution, we can expand it using the chain rule.

$$p(d, i, g, s, l) = \frac{p(d)\, p(i \mid d)}{p(d, i)}\, p(g \mid d, i)\, p(s \mid d, i, g)\, p(l \mid d, i, g, s)$$

Let's assume that after further analysis of the domain problem, we discover the independence claims below. For example, the first claim is variable i is independent of d.

$i \perp d,\ s \perp \{d, g\},\ l \perp \{d, i, s\}$ ($a \perp b$ means a is independent of b)

Therefore, we can simplify the R.H.S. expression as:

Simplified to

$$p(d, i, g, s, l) = p(d)\, p(i)\, p(g \mid d, i)\, p(s \mid i)\, p(l \mid g)$$

Original: $p(d, i, g, s, l) = p(d)\, p(i \mid \cancel{d})\, p(g \mid d, i)\, p(s \mid \cancel{d}, i, \cancel{g})\, p(l \mid \cancel{d}, \cancel{i}, g, \cancel{s})$

s is independent of d or g

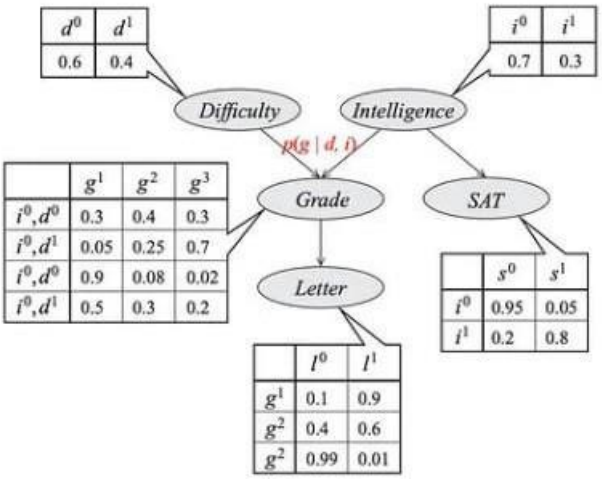
We can draw this dependency using a directed Graphical model. Each node represents a variable and to quantify the dependency, we associate a conditional probability for each node.

$$P(x_i \mid x_{A_i})$$

A_i: the parents of x_i

These model parameters are presented as the table entries below.

These model parameters are presented as the table entries below.



d^0	d^1
0.6	0.4

i^0	i^1
0.7	0.3

	g^1	g^2	g^3
i^0, d^0	0.3	0.4	0.3
i^0, d^1	0.05	0.25	0.7
i^1, d^0	0.9	0.08	0.02
i^1, d^1	0.5	0.3	0.2

	s^0	s^1
i^0	0.95	0.05
i^1	0.2	0.8

	l^0	l^1
g^1	0.1	0.9
g^2	0.4	0.6
g^2	0.99	0.01

So what is the complexity for these models? $p(d, i, g, s, l)$ have 5 variables with the possible combinations of 32 (2^5). To model p, we sample data for 32 possibilities. On the contrary, the graph above has 4 tables with only 26 parameters. Don't underestimate the curse of exponential. 64 variables will have a combination of

18,446,744,073,709,551,616 (20 digits)

i.e. a simple 8 × 8 grayscale image will contain enough variables (pixels) that make an ML problem too complex to solve. In ML, it is important to discover independence to break the curse of this exponential complexity, for example, Naive Bayes classifier, Hidden Markov Model, Maximum likelihood estimation with i.i.d., etc... The distribution of a variable can be calculated given all its neighbors are know. In ML, the original grand design is missing. We spot global insight through localized findings.

Bayesian Belief Network

Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as: "A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**.

Bayesian networks are probabilistic, because these networks are built from a *probability*

distribution, and also use probability theory for prediction and anomaly detection. Real world applications are probabilistic in nature, and to represent the relationship between

multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time**

series prediction, and decision making under uncertainty.

Bayesian Network can be used for building models from data and experts opinions, and it

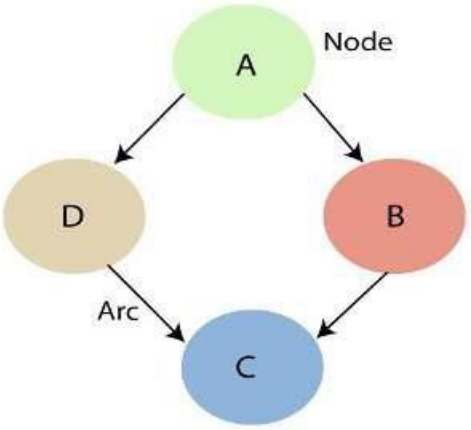
consists of two parts:

- *Directed Acyclic Graph*
- **Table of conditional probabilities.**

The generalized form of Bayesian network that represents and solve decision problems under

uncertain knowledge is known as an **Influence diagram**.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:



Each **node** corresponds to the random variables, and a variable can

be **continuous** or **discrete**.

- **Arc or directed arrows** represent the causal relationship or conditional probabilities

between random variables. These directed links or arrows connect the pair of nodes in

the graph.

These links represent that one node directly influence the other node, and if there is

no directed link that means that nodes are independent with each other

- **In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.**
- **If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.**
- **Node C is independent of node A.**

The Bayesian network has mainly two components:

- *Causal Component*
- **Actual numbers**

Each node in the Bayesian network has condition probability distribution $\mathbf{P(X_i}$ $\mathit{|Parent(X_i)\ ),}$

which determines the effect of the parent on that node.

Bayesian network is based on Joint probability distribution and conditional probability. So

let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables x1, x2, x3,. … , xn, then the probabilities of a different combination of x1,

x2, x3.. xn, are known as Joint probability distribution.

$\mathbf{P[x_1, x_2, x_3,\ldots, x_n]}$, it can be written as the following way in terms of the joint probability

distribution.

$\mathbf{= P[x_1|\ x_2, x_3,\ldots, x_n]P[x_2, x_3,\ldots, x_n]}$

$\mathbf{= P[x_1|\ x_2, x_3,\ldots, x_n]P[x_2|x_3,\ldots, x_n]\ldots P[x_{n-1}|x_n]P[x_n].}$

In general for each variable Xi, we can write the equation as:

$\mathbf{P(X_i|X_{i-1},\ldots\ldots, X_1) = P(X_i|Parents(X_i))}$

Explanation of Bayesian network:

Let's understand the Bayesian network through an example by creating a directed acyclic

graph:

Problem:

Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.

Solution:

- The Bayesian network for the above problem is given below. The network structure is

showing that burglary and earthquake is the parent node of the alarm and directly

affecting the probability of alarm's going off, but David and Sophia's calls depend on

alarm probability.

- The network is representing that our assumptions do not directly perceive the burglary

and also do not notice the minor earthquake, and they also not confer before calling.

- The conditional distributions for each node are given as conditional probabilities table

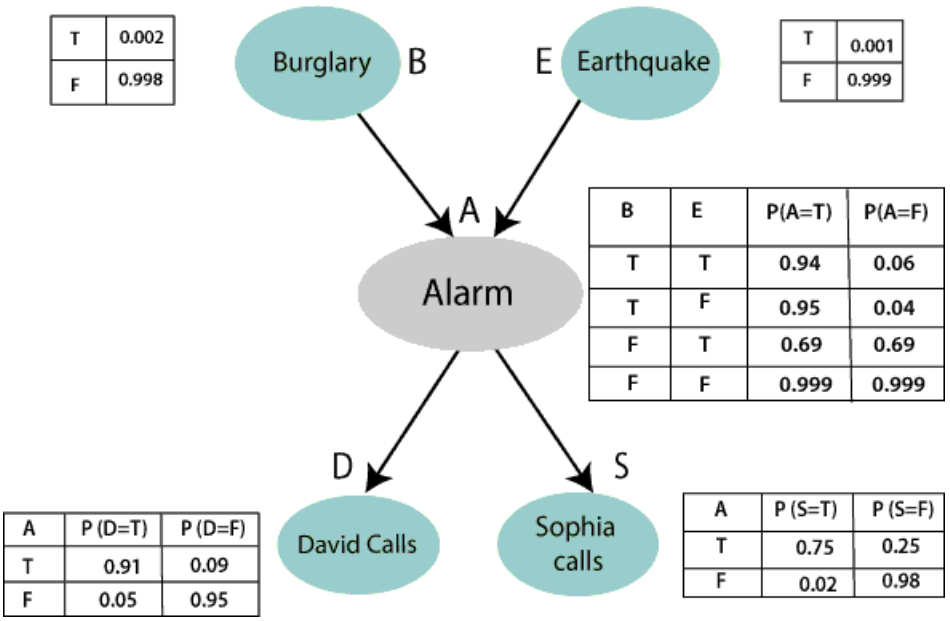
or CPT.

- Each row in the CPT must be sum to 1 because all the entries in the table represent an

exhaustive set of cases for the variable.

can write the events of problem statement in the form of probability: **P[D, S, A, B, E]**,

can rewrite the above probability statement using joint probability distribution:
P[D, S, A, B, E]= P[D | S, A, B, E]. P[S, A, B, E]
=P[D | S, A, B, E]. P[S | A, B, E]. P[A, B, E]
= P [D| A]. P [S| A, B, E]. P[A, B, E]
= P[D | A]. P[S | A]. P[A| B, E]. P[B, E]
= P[D | A]. P[S | A]. P[A| B, E]. P[B |E]. P[E]



T	0.002
F	0.998

T	0.001
F	0.999

B	E	P(A=T)	P(A=F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P (D=T)	P (D=F)
T	0.91	0.09
F	0.05	0.95

A	P (S=T)	P (S=F)
T	0.75	0.25
F	0.02	0.98

Let's take

the observed probability for the Burglary and earthquake component:

P(B= True) = 0.002, which is the probability of burglary.

P(B= False)= 0.998, which is the probability of no burglary.

P(E= True)= 0.001, which is the probability of a minor earthquake

P(E= False)= 0.999, Which is the probability that an earthquake not occurred. We can provide the conditional probabilities as per the below tables: **Conditional probability table for Alarm A:**

The Conditional probability of Alarm A depends on Burglar and earthquake

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	E	P(A= True)	P(A= False)
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

P(S, D, A, ¬B, ¬E) = P (S|A) *P (D|A)*P (A|¬B ^ ¬E) *P (¬B) *P (¬E).

= 0.75* 0.91* 0.001* 0.998*0.999

= 0.00068045.

Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

Gibbs Sampling

- Fix the values of observed variables
- Set the values of all non observed variables randomly
- Perform a random walk through the space of complete variable assignments. On each move
- 1. pick a variable X
- 2. calculate Pr (X=true all other variables)
- 3. set X to true with that probability

Repeat many times. Frequency with which any variable X is true is its posterior probability

Converges to true posterior when frequencies stop changing significantly

- Stationary distribution, mixing

calculate Pr (X=true all other variables)

Recall: a variable is independent of all others given its Markov Blanket

- **parents - children - other parents of children**

So problem becomes calculating Pr(X=true MB(x))

- **we solve this sub-problem exactly**

- **fortunately, it is easy to solve**

P(X)= α P(X| Parents(X)) π P(Y| Parents(Y))

P(X|A,B,C) = $\frac{P(X,A,B,C)}{P(A,B,C)}$

$= \frac{P(A)P(X|A)P(C)P(B|X,C)}{P(A,B,C)}$

= P(X|A)P(B|X,C)

= αP(X|A)P(B|X,C)

P(X|E) = $\frac{number\ of\ samples\ with\ X=x}{total\ number\ of\ samples}$

Advantages

- No samples are discarded

- No problem with samples of low weight

Can be implemented very efficiently

- **Disadvantages**
- **Can get stuck if relationship between two variables**
- **Many variations have been devised to make MCMCM**

Markov Random fields

A Markov random Field is an undirected graph

where each node captures the (discrete or Gaussian) probability distribution of a variable and

the edges represent dependencies between those variables and are weighted to represent the relative strengths of the dependencies.

The defining difference between a Bayesian network and a Markov random field is that a Bayesian network is directed while a Markov random field is undirected.

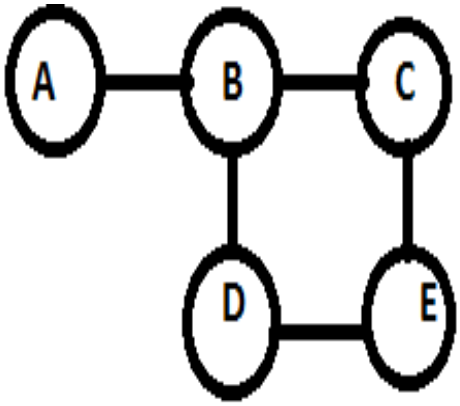
a hidden Markov model is directed and thus a sub-type of Bayesian network rather than a sub-type of Markov random field.

A **conditional random field** is a Markov random field that has specifically been set up to allow the values of a specific variable or variables to be predicted based on the values of the other variables

Undirected graphical models edge represents the potential between two variables, syntactically, Factorization distribution probabilities between variable.

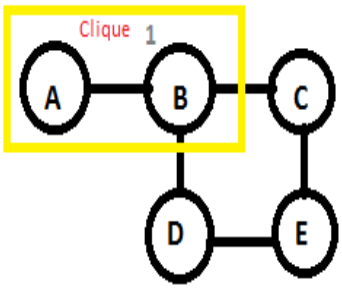
CPDs for Bayesian networks, we have tables to incorporate relations between nodes in **Markov networks**.

However, there are two crucial differences between these tables and CPDs.



Clique in graph theory.it is a subset of vertices of an undirected graph.

P(A, B, C, D, E) α Φ(A,B) Φ (B,C) Φ (B,D) Φ (C,E) Φ (D,E)



P(X) = $\frac{1}{z}\bar{\Pi}_{Ceclique}\phi_C(x_C)$ **(Potential functions)**

Such that: It induces sub graph is complete in every vertices in a clique is adjacent. So, clique in this graph adjust adjacently one by one.