Artificial Intelligence

Created by	Ⓑ Borhan
Last edited time	@July 20, 2025 10:32 PM
Tag	

Resources

1. Atkia Apu's Note
2. Mahir Labib Dihan Sir's Note
    a. https://drive.google.com/file/d/1Uf8A_hCBnC2F1-QIez4U-ckrPEbmkYJe/view
    b. https://drive.google.com/file/d/1Ek05PYorrHT_xsqdB71c86dyKVjj58Ty/view

Introduction to AI, Turing test, Agents

Artificial Intelligence (AI) is a branch of computer science and engineering that deals with **intelligent behavior, learning, and adaptation** in machines.

Views of AI fall in four categories:

1. Thinking humanly
2. Thinking rationally
3. Acting humanly
4. Acting rationally

Turing Test: A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or a from a computer.

Capabilities of passing Turing test

1. **Natural language processing** to enable it to communicate successfully in English.
2. **Knowledge representation** to store what it knows or hears.
3. **Automated reasoning** to use the stored information to answer questions and draw new conclusions.
4. **Machine learning** to adopt to new circumstances and to detect and extrapolate patterns.

Agents: An agent is anything that **perceives its environment** through **sensors** and **acts** upon it through **actuators**.

Structure

- **Sensors**: Devices that collect environmental data
- **Percepts**: Data received from sensors
- **Actuators**: Mechanisms that allow the agent to act on the environment
- **Actions**: Tasks performed by actuators

Types of Agents

Agent Type	Description	Key Features	Limitations	Example
Simple Reflex Agent	Acts based on **current percepts, using condition-**action rules ("If X, then Y").	**No memory** of past percepts; **simple** to implement.	**Limited** intelligence; may enter **deadlocks** or **loops**.	**Thermostat** turning on/off based on temperature.

Agent Type	Description	Key Features	Limitations	Example
Model-Based Reflex Agent	Maintains an **internal state** to **track unobserved environment** aspects.	Uses **knowledge** of world evolution and action effects.	Requires **modeling** of environment dynamics.	Robot **navigating** a partially visible room.
Goal-Based Agent	Acts to **achieve specific goals** by evaluating action sequences.	**Flexible**; **explicit knowledge** representation.	Requires **search/planning** for goal achievement.	**Navigation system** planning a route.
Utility-Based Agent	Chooses actions **based on a utility function** measuring success. The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.	**Balances conflicting** goals and **success** likelihood.	**Complex** to compute utility for all states.	**Robot** prioritizing speed vs. safety.
Learning Agent	Learns from **experience to improve** performance.	Components: Learning Element, Performance Element, Critic, Problem Generator.	Needs **initial knowledge**; learning **takes time**.	**Self-driving car** adapting to traffic patterns.

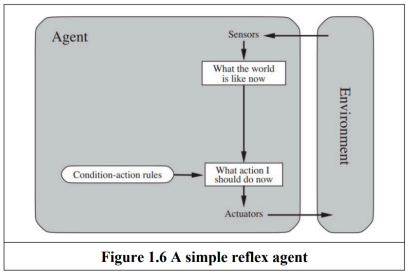


Figure 1.6 A simple reflex agent

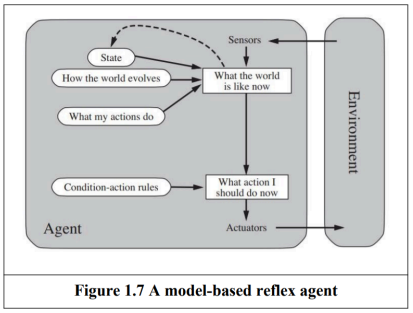


Figure 1.7 A model-based reflex agent

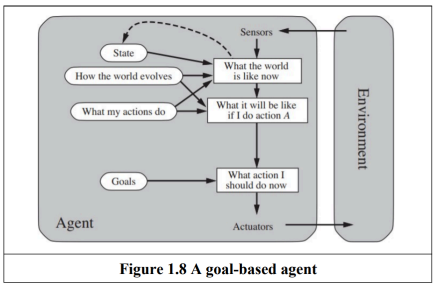
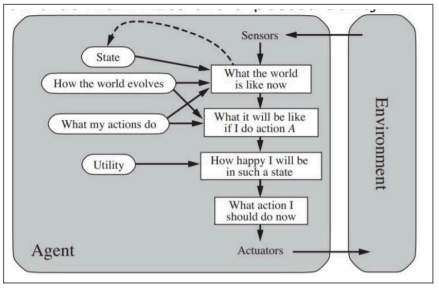


Figure 1.8 A goal-based agent



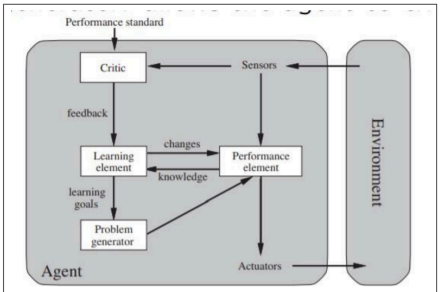


Figure 1.10 Learning Agents

1. **Reflexive Agent:** An agent whose action depends only on the current percepts.
2. **Model based :** An agent whose action is derived from an internal model of the current world state
    a. Partial observability
    b. Updating the internal state information as times goes by requires two kinds of knowledge to be encoded
        i. We need some information information about how t**he world evolves** independently of the agent

    ii. We need some information about how the agent's own **actions affect** the world

3. **Goal Based agent :** An agent that select actions that it believes will **achieve explicitly** represented goals.
    a. Expansion of Model based agent
    b. Desirable situation
    c. Searching and planning
4. **Utility Based Agent:** An agent that selects actions that it believes will **maximize the expected utility outcome state**
    a. Utility function
    b. Deals with happy and unhappy state
5. **Learning Agent**: An agent whose behavior improves over time based on its experience
    a. **Learning Element:** Responsible for making improvements.
    b. **Performance Element :** is what we have previously considered to be the entire agent: it takes in percepts and decides on actions
    c. **Critic:** The learning element uses **feedback** from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future.
    d. **Problem Generator:** suggesting actions that will lead to new and informative experiences.

- **Rational Agent:** A rational agent selects **actions that maximize its performance measure**, based on **percepts and build-in** knowledge.
    - act to gather information or explore for better outcomes
    - autonomous if their behavior is based on their own experience
    - Rationality does not imply **omniscience**
- **Autonomous Agents:** Agents whose behavior is **determined by their own experience** rather than pre-programmed decisions.
    - Sufficient initial knowledge to operate
    - Ability to learn and adapt to new situations
- **Omniscience:** An omniscient agent knows the **exact outcomes of its actions** and i**ts certain no other outcomes are possible**.
    - It is impossible in real-world

Different informed and uninformed search techniques

State space search: It is the **possible of all states** including start and goal state where a particular problem solution is going to be searched.

Data structure: Graph

Represented By

- A set of states
- An initial state
- A set of operators or actions
- A partial function/transition function
- A set of goal states
- An optional path cost function

Generalized state space search algorithm

```
Initialize:
   L : Linked List = {}
   Sc : Node = Null
   GOAL_FOUND : Boolean = FALSE
   GOAL_EXISTS : Boolean = TRUE

Add node si (intial) to L
WHILE GOAL_FOUND IS FALSE
AND GOAL_EXISTS IS TRUE
DO
   SET Sc : An unexpanded node from L
   IF Sc IS NOT NULL THEN
      FOREACH successor node Ss of Sc DO
         IF Ss EQUALS Sg THEN
            SET GOAL_FOUND = TRUE
         ELSE
            IF L not contains Ss THEN
               Add Ss to L
            END IF
         END IF
         MARK Sc as expanded
      END FOREACH
   ELSE
      SET GOAL_EXISTS = FALSE
   END IF
END WHILE
```

Search Tree:

- State space represented by Search Tree.
- A fundamental data structure that used to systematically represent and explore all possible search states of a problem to find a solution.
- Provides a framework for solving problems by simulating the decision making process through a sequence of actions.
- In search tree
  - A root node represents the initial state
  - Children of a node → successor
  - Fringe of a tree → States not yet expanded

Properties we use to evaluate an algorithm

1. Completeness : Guaranteed to find a solution if one exists
2. Optimal : If it always find the best solution
3. Time complexity: The amount of time an algorithm takes
4. Space Complexity : The amount of memory an algorithm requires

Informed vs Uninformed

Aspects	Uninformed	Informed
Definition	Search algorithm that explore the problem space without any domain-specific knowledge or heuristics, relying on problem	Search algorithms that use domain-specific heuristic or estimates of the cost to reach

Aspects	Uninformed	Informed
	structure and predefined rules. Also known as blind search algorithm.	the goal.
Knowledge Used	Only problem structure	Heuristics estimate cost to goal.
Time	Consuming	Quick Solution
Cost	Costly	Less
Time and Space Complexity	More	Less
Example	BFS, DFS	A*, Best first search

Uninformed Algorithm

1. **Breadth-First-Search Algorithm**
    a. Explores all nodes at the current depth before moving to the next level
    b. Uses a queue FIFO
    c. Shortest path but more memory
2. **Depth-First-Search**
    a. Explores as far as possible along each path before backtracking
    b. Uses a Stack LIFO
    c. Infinite loop, memory-efficient
    d. **How it detects cycle**
        i. When DFS is applied over a graph if DFS finds an edge that points to an already visited vertex
    e. **DFS is not optimal**
        i. Doesn't consider path cost
        ii. Many states keep reoccurring → No guarantee to find a solution
        iii. May go to infinite loop
        iv. Doesn't find the shortest path always
3. **Depth Limited Search**
    a. A variation of DFS that limits the depth of exploration to prevent infinite loops in large or infinite space spaces
    b. Useful when the goal depth is known but cannot find solution beyond the depth limit.
    c. Limitation
        a. Not optimal
        b. Incompleteness
        c. Effectiveness depend on the depth limit
4. **Iterative Deepening Depth First Search**
    a. combines BFS and DFS altogether
    b. Adv
        i. Ensure completeness and optimally → BFS
        ii. Memory Efficient → DFS
    c. Dis
        a. repeats all the work of the previous phase
5. **Uniform Cost Search**
    a. Extends BFS by considering path costs always expanding the last cost node first.
    b. Find the least cost, Slower than BFS
6. **Bidirectional Search**

a. Runs two simultaneous searching one from initial state (forward search) and another from goal (backward search)
b. It replaces one search graph with two small search subgraphs
c. The search stops when there two graphs intersect each other
d. When it doesn't work
    i. Implementation of tree is difficult
    ii. The goal state is unknown/unclear in advance
    iii. Finding an efficient way to check if a match exists is tricky which can increase the time.

Algorithm	TC	SC	Optimal	Complete
BFS	b^d	b^d	Yes	Yes
DFS	b^d	**bd**	**No**	**No**
DLS	b^l	**bl**	**No**	**if l ≥ d**
IDDFS	b^d	bd	Yes	Yes
Uniform	b^d	b^d	Yes	Yes
Bidirectional	**b^(d/2)**	**b^(d/2)**	Yes	Yes

Informed Search, Heuristics*, How heuristics help? A* search, Proof of optimality of A*

---

https://iipseries.org/assets/docupload/rsl2024AF233C7BF02A178.pdf

1. **Best-First Search**
    a. Greedy search, It picks such path that appears best at the moment.
    b. A blend of both DFS and BFS
    c. Choose the most promising node at each step
    d. It is applied by priority queue
    e. **Advantage :**
        i. It switch between BFS and DFS by gaining the advantage of both algorithm
        ii. More effective and capable than BFS, DFS
    f. **Disadvantage**
        i. Worst scenario : operate as an unguided DFS
        ii. **Get stuck in a loop**
        iii. **Not optimal**
        iv. **Not complete**
    g. **TC/SC :** b^m

$h(n) = g(n)$

$h(n) = \text{estimated cost from node n to the goal.}$

$g(n) = \text{cost from the start node to node n}$

2. **A* Algorithm**
    a. **Heuristic :** A heuristic is a technique designed to solve a problem faster than classic method or to find an approximate solution when the classic methods fail to find any exact solution.
        i. **How it helps in finding solution**
            1. It useful in reducing the time and resources required to find the solutions by f**ocusing the search on the most promising path**.
            2. It prioritizes which node to **explore based on their estimated cost t**o the goal.

3. It helps in state space search by guiding the exploration, r**educing the search space** and **improving efficiency.**
4. It allows algorithm to **focus on exploring path that are more likely to be solution**.

ii. **Admissible Heuristic:** A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal statement from n. An admissible heuristic never overestimates the cost to reach the goal, thus it is optimistic.

1. $h_2(n) \geq h_1(n)$ for all node n and both are admissible then h_2 dominates h_1. h_2 is better for search.

b. Evaluation Function, $f(n) = g(n) + h(n)$

c. Optimal, Complete

d. TC : Exponential.

e. A^* **is always optimal**

Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G_2.

$f(G_2) = g(G_2)$ since $h(G_2) = 0$
$f(G) = g(G)$ since $h(G) = 0$
$g(G_2) > g(G)$, since G_2 is suboptimal
$f(G_2) > f(G)$,
$h(n) \leq h^*(n)$
$g(n) + h(n) \leq g(n) + h^*(n)$
$f(n) \leq f(G) < f(G_2)$

A* will never select G_2 for expansion.

A heuristic is consistent if for every node n, every successor n′ of n generated by any action a,

$h(n) \leq c(n, a, n') + h(n')$

If h is consistent,

$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') = g(n) + h(n) \geq f(n)$

$f(n)$ is non-decreasing along any path. So, A* using graph search is optimal.

f. **Prove that the uniform-cost search is a special case of A* search.**

A* search uses the evaluation function:

$f(n) = g(n) + h(n)$

where:

- $g(n) =$ cost from the start node to node n
- $h(n) =$ heuristic estimate of the cost from n to the goal.

Uniform-Cost Search (UCS) does **not** use a heuristic, so:

$h(n) = 0$

Therefore, for UCS, the evaluation function becomes:

$f(n) = g(n) + 0 = g(n)$

This means UCS always expands the node with the lowest path cost *g*(*n*), exactly like A* with a zero heuristic.

Hence, Uniform-Cost Search is a special case of A Search where the heuristic function h(n)is zero for all nodes.

g. **Prove that, if heuristic function h never overestimates by more than cost c, A* using h returns a solution whose cost exceeds that of the optimal solution by no more that c.**

Now, suppose h(n) <= h(n)+c as given and let G2 be a goal that is sub-optimal by more than c, i.e. f(G2)=g(G2) > C* +c. Now consider any node n on a path to an optimal goal. We have*

f(n)=g(n)+h(n) <= g(n)+h(n)+c <= C*+c <= f(G2)*

so G2 will never be expanded before an optimal node is expanded

because f(n)<f(G2)

Local Search Algorithm: Local search algorithms are **essential tools** in artificial intelligence and optimization, employed to **find high-quality solutions** in **large** and **complex** problem spaces.

1. **Hill-Climbing Search:**
    a. It is a straightforward local search algorithm that iteratively moves towards better solution.
    b. Process : Start → Evaluate → Move → Repeat
    c. Pros: Easy to implement, works well in small search space
    d. **Cons**
        i. **Local Maxima:** Hill-climbing can get stuck at a local maximum.
        - **Plateaus:** On a **flat region** where neighboring states have the same heuristic value, hill-climbing may **wander aimlessly**, slowing progress or failing to find the goal.
        - **Ridges:** In state **spaces with ridges** (narrow paths of improvement), hill-climbing may **oscillate between states** without advancing toward the goal.
        - **No Backtracking:** Hill-climbing only considers the current state and its neighbors, without backtracking to explore alternative paths, missing better solutions elsewhere.
    - **Solution**
        - Introduce **Gradient descent** search which is a variation of all hill climbing that moves **downhill**
        - Introduce a small r**andom jump** to escape the **plateau**.
        - Use **stochastic** hill climbing where steps are probabilistic can help navigation **ridges**.
2. **Simulated Annealing**
3. **Local Beam Search**
4. **Genetic Algorithm**
5. **Tabu Search**

2 player zero sum games, mini-max algorithm, alpha-beta pruning

---

https://ocw.mit.edu/courses/15-053-optimization-methods-in-management-science-spring-2013/2e66a9d9a74dc5c11b620f70663400da_MIT15_053S13_tut08.pdf

https://www.arsdcollege.ac.in/wp-content/uploads/2020/03/Artificial_Intelligence-week3.pdf

The 2-person 0-sum game is a basic model in game theory. There are two players, each with associated set of strategies. While one player aims to maximize his payoff, the other player player attempts to take an action to minimize this payoff. The gain of a player is the loss of another.

Mini-Max Algorithm

Mini-max algorithm is a recursive or backtracking algorithm which is used in decision making and game theory.

- Both opponent plays optimally
- Both the players fight it as opponent player gets the minimum benefit while they get the maximum benefit.
- MAX will select maximized value
- MIN will select the minimized value
- A DFS algorithm
- Proceed all the way down to the terminal node then backtrack the tree as recursion.

The game is modeled as a game tree

1. Node → Game State
2. Edges → Legal Moves
3. Root → Current Position
4. Leaves → Game Outcomes with utility value (+1 → max, -1 → min)

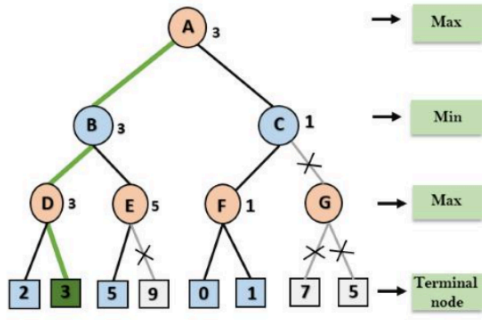
Process

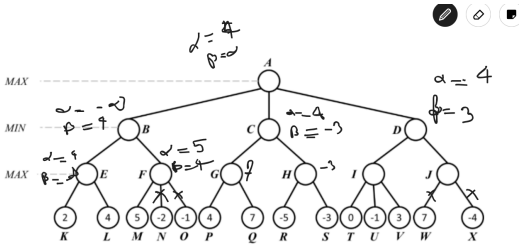
1. Start at the root (MAX turn)
2. Recursively, explore all possible moves down to terminal nodes or fixed depth
3. At leaf node assign values using a utility function or heuristic evaluation function for non-terminals states
4. Backpropagate values
    a. At max nodes → Select he child with highest value
    b. At min Nodes → Select the child with lowest value
5. At the root, choose the move that yields the highest value ensuring the best outcome against min optimal play.

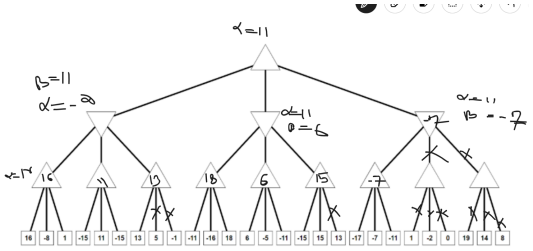
Analytics: $\mathrm{TC} : O(b^d), \mathrm{SC} : O(bd)$

Alpha-beta Pruning

- Modified version of minmax algorithm
- It uses an optimization technique for the minimax algorithm.
- The alpha-beta pruning to a standard minimax algorithm returns the same move as the standard does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow.
- Alpha : The best/highest-value choice we have found so far at any point along the path of maximizer. Initial value of a alpha is $-\infty$
- Beta: The best/lowest-value choice we have found so far at any point along the path of minimizer. The initial value is $+\infty$
- Condition for alpha-beta pruning
    - $\alpha \geq \beta$







Genetic algorithm, steps of genetic algorithm. (using MAXONE problem, see slides), Different Crossover and Selection techniques, GA for solving optimization problems

https://egyankosh.ac.in/bitstream/123456789/12697/1/Unit-11.pdf

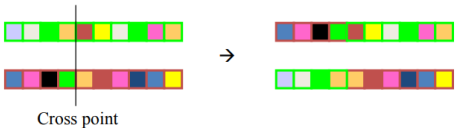
http://www.cs.cmu.edu/~02317/slides/lec_8.pdf

Genetic Algorithm: A genetic Algorithm is a search heuristic inspired by Charles Darwin's "Theory of Natural Evolution". IT is used to solve optimization problem by mimicking the process of natural selection where the fittest individual are selected for reproduction to produce offspring for the next generation.

Application of GA: Optimization, Automatic Programming, Machine and robot learning, Economic models, Ecological models, Population genetic models and Models of social systems.

Steps

1. Initialization: Start with randomly generated population
2. Evaluation: Evaluate each individual using a fittest function
3. **Selection:** Select the individuals for reproduction using different techniques
    a. **Types of selection techniques**
        i. **Roulette Wheel Selection:** Conceptually, this can be represented as a game of roulette - each individual gets a slice of the wheel, but more fit ones get larger slices than less fit ones.
        $P_i = \frac{F_i}{\sum F_i}$
        ii. **Rank based selection:** where the probability of an individual being selected for reproduction or survival is determined by its rank within the population, not its raw fitness score.
        iii. **Elitist Selection:** Chose only the most fit members of each generation.
        iv. **Cutoff Selection:** Select only those that are above a certain cutoff for the target function.
        v. **Scaling Selection :**
4. **Crossover:** Combine pair to produce offspring
    a. Types
        i. **Single Point:** Randomly select a single point for a crossover



2. **Two point crossover:** Avoids cases where genes at the beginning and end of a chromosome are always split

3. **Uniform**
    a. A random subset is chosen
    b. The subset is taken from parent 1 and the other bits from parent 2

Subset:	**BAABBAABBB**	**(Randomly generated)**
Parents:	1010001110	0011010010
Offspring:	0011001010	1010010110

5. **Mutation :** Random changes to some individual to maintain diversity
    a. Mutation prevents the algorithm to be trapped in a local minimum
6. **Termination:** Repeat the process until the criterion met

The basic algorithm

1. [Start] Genetic random population of n chromosomes
2. [Fitness] Evaluate the fitness $f(x)$ of each chromosome x in the population
3. [New Population] Create a new population by repeating following steps until the New Population is complete
    a. [Selection] Select two parent chromosomes from a population according to their fitness value
    b. [Crossover] With a crossover probability, cross over the parents to form a new offspring.
    c. [Mutation] With a mutation probability, mutate new offspring at each locus.
    d. [Accepting] Place new offspring in the new population
4. [Replace] Use new generated population for a further sum of the algorithm
5. [Test] If the condition is satisfied, stop and return the best solution in current population.
6. [Loop] Go to Step2 for fittest evaluation.

MaxOne problem: The *Maxone* problem is to find a binary string of length 'l' that contains the maximum number of one. The optimal solution is a string of all 1s.

Example:

We start with a population of n random string with L binary digits.

Let, $n = 4$ and $L = 6$

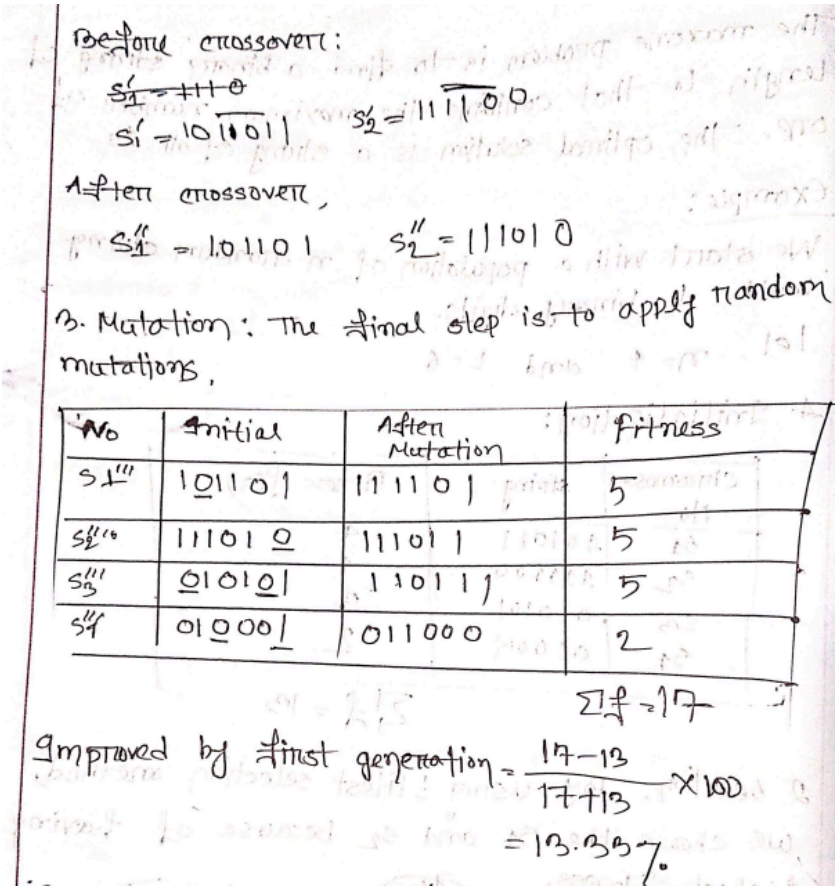
1. Initialization:

Chromosome No	string	Fitness $f(s)$
s_1	101011	4
s_2	111100	4
s_3	010101	3
s_4	010001	2

$\sum f = 13$

2. Selection: By using Elitest selection method, we choose the s_1 and s_2 because of having highest fitness.

3. Crossover: By applying two point crossover method at 3 and 5.



Bayes' rule, Belief update*, Naive Bayes Classifier, Formulation, Dealing with sparse data, Usage in document classification*, Gaussian Naive Bayes

Baye's Theorem: Baye's theorem is a fundamental principle in probability theory that allows for the computation of the conditional probability of a hypothesis H gives observed evidence E.

Derivation: $P(A|B) = \frac{P(A \cap B)}{P(B)}, P(B|A) = \frac{P(A \cap B)}{P(A)}$

$\Rightarrow P(A|B)P(B) = P(B|A)P(A)$

$\Rightarrow P(A|B) = \frac{P(B|A)P(A)}{P(A|B)}$

$\Rightarrow P(H|E) = \frac{P(E|H)P(H)}{P(E)} = \frac{P(E|H)P(H)}{P(E|H)P(H)+P(E|\neg H)P(\neg H)}$

Bayesian Network: It is a decision making tool . A BN is a powerful probabilistic graphical model used for decision making under uncertainty,

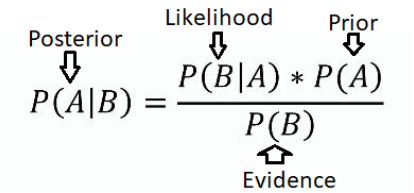
Principle:

1. **Structure:** It consists of DAG and CPT (Conditional Probability Tree)
    a. Node : Random Variable
    b. Edge : Conditional Dependency
    c. No Edge : Conditional Independence.

    Each node associated with a CPT that quantifies the probability of the node given its parent node. $P(X_i|Parent)$

    The network encodes the joint probability distribution of all variables as $P(X_1 ... X_n) = \prod P(X_i|Pa(X_i))$

2. **Probabilistic Inference :** BN update the probabilities of known variable using Baye's theorem, facilitating reasoning under uncertainity. For example, the network can be used to update knowledge of the state of a subset of variables when other variables (the evidence variables) are observed. This **process of computing the posterior distribution of variables given evidence** is called probabilistic inference.



3. Belief Update: A Bayesian update or belief update is a **change in probabilistic beliefs after gaining new knowledge**. For example, after observing a patient's test result, we might revise our probability that a patient has a certain disease. If this belief revision obeys Bayes's Rule, then it is called Bayesian. When the evidence is observed, the prior probabilities are update to posterior probabilities.

Application: Medical diagnosis, decision support, prediction and forecasting, anomaly detection.

Naive Bayes Classifier: Naive Bayes is a classification algorithm that uses probability to predict which category a data point belongs to, assuming that all features are unrelated.

Formulation

Bayes theorem, $P(C|X) = \frac{P(X|C)\cdot P(C)}{P(X)}$

C : Class, X : Feature

Naive assumption: All features are conditionally independent given the class.

$P(X|C) = \prod_{i=1}^{n} P(x_i|C)$

Hence, final classification becomes,

$\hat{C} = \arg\max_C P(C) \prod_{i=1}^{n} P(x_i|C)$

Dealing with Sparse Data (Zero Probabilistic)
In real datasets, some feature-class combinations may
not appear in training data, resulting in **zero probability**, which can nullify the whole product.

To handle this, we use
Laplace Smoothing (Additive Smoothing):

$P(x_i|C) = \frac{count(x_i,C)+\alpha}{\sum_j count(x_j,C)+\alpha\cdot|V|}$

Usage in document classification

Spam filtering, sentiment analysis, topic classification

Definitions

1. **Mass function:** The mass function, denoted m, assign a value range $[0, 1]$ to every subset frame of discernment.
2. **Belief (Bel):** The belief function, $Bel(A)$, for a subset A is the sum of the mass probabilities of all proper subset of A.

   $Bel(A) = \sum_{B\subseteq A} m(B)$
3. **Plausibility (Pls):** The plausibility function, $Pls(A)$, represents the maximum possible belief in A.

$Pls(A) = 1 - Bel(\neg A)$

4. **Belief Interval :** The belief interval for a subset A is the range $[Bel(A), Pls(A)]$, which express the range of belief in A.

Gaussian Naive Bayes Algorithm

Gaussian Naive Bayes is a type of Naive Bayes method working on continuous attributes and the data features that follows Gaussian distribution throughout the dataset.

1. Calculate Mean and Variance
   $\mu(X) = \frac{\sum}{n}$
   $\sigma^2 = \frac{\sum(xi-\mu)^2}{n-1}[\text{sample variance}]$
   $\sigma^2 = \frac{\sum(xi-\mu)^2}{n}[\text{population variance}]$
2. $P(X|Y) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp^{-\frac{(x-\mu_x)^2}{2\sigma^2}}$
3. $\text{Posterior}, P(Y) = \frac{P(Y)\prod P(X_i|Y)}{\text{evidence}}$

Decision Tree, Information Gain & Gini Index.

https://downloads.ctfassets.net/kdr3qnns3kvk/6nDiFgv0LRFz3ocCMvZGMR/c8b9acb313cae4f7ccc20a61058dbb80/We DecisionTrees.pdf

A decision tree is a machine learning model that uses a tree like structure to make decision based on a sequence of questions and conditions.

Entropy is a measure of impurity or disorder in a dataset

$Ent(S) = -\sum_{i=1}^{n} p_i \log_2 p_i$

Information Gain measures the reduction in entropy after a dataset is split on an attribute:
$IG(S, A) = Ent(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Ent(S_v)$

Gini Index :
$Gini(n) = 1 - \sum[p_i]^2$

Machine Learning, Steps of ML, Hyperparameter tuning, why needed?

https://www.cs.cmu.edu/~hn1/documents/machine-learning/notes.pdf

Machine learning is programming computers to **optimize a performance criterion** using example **data** or past **experience**.

Aspect	Traditional Programming	Machine Learning
Programming	Rules and logic are manually coded by the programmer.	Model learns patterns automatically from data.
Input	Data and explicit rules.	Data (often labeled for supervised learning).
Output	Deterministic output based on coded rules.	Predictions or decisions based on learned patterns.

Aspect	Traditional Programming	Machine Learning
Flexibility	Limited to predefined rules; changes require recoding.	Adapts to new data; retraining updates the model.
Use Cases	Well-defined, rule-based tasks (e.g., payroll, sorting).	Complex, pattern-based tasks (e.g., image recognition, predictions).
Examples (Lecture 1)	Calculating payroll	Speech recognition, personalized

Steps of ML

1. **Project Setup :** This is the first step to plan and set up the environment for machine learning project.
    a. **Understand Business Goal**
        i. Having conversation with stakeholders
    b. **Choose the solution to the problem**
        i. Which category of the models derive the highest impact
2. **Data Preparation:**
    a. Collection of data
        i. Clear about the goal with clear objective → Identify which data are vital for the model tuning
        ii. Collecting related data from various sources according to project requirement
    b. Cleaning of Data
        i. Identify and handle missing values, inconsistencies, removing duplicates
    c. Data transformation
        i. Convert cleaned data into a format suitable for machine learning
        ii. modifying or converting data, feature scaling, feature encoding
    d. Data Reduction
        i. Simplifying data without losing the essence
    e. Data Splitting
        i. Split data into different sets to ensure more reliable and actionable
        ii. Splitting of data
            1. **Training set:**
                a. actual dataset from which a model **trains**
                b. model sees and **learns** from this data
                c. **60%** of total data
            2. **Validation set**
                a. Used to training **hyperparameters**
                b. Model sees this data for **evaluation** but doesn't learn from this
                c. **15%** of total data
            3. **Testing set**
                a. **Evaluate** the model after training to complete
                b. Provides **unbiased evaluation** of the models
                c. **20-25%** of data
    f. Importance of data preparation
        i. Provide reliable prediction outcomes
        ii. Identify data issues or error

iii. Increase decision making capability
iv. Reduce cost
v. Increase model performance

3. **Deployment:**
   a. **Deploy the model:**
      i. integrating it into a production environment where it can process real world data
      ii. MLOPS
   b. **Monitor Model performance**
      i. Regularly test the performance of model as data
   c. **Improve Model**
      i. Continuously iterate and improve model
      ii. Replace model with an updated version

Feature	Random Split	Stratified Split	K-Fold Cross-Validation	Time-Based Split
How it works	Randomly divides data	Keeps class proportions same	Splits data into k parts, trains k times. in each iteration, k-1 folds are used for training, and the remaining 1 fold is used for validation	Splits data by time order. Earlier data points form the training set, and later data points form the test or validation set
When to use	Data has no order, balanced classes	Classification with imbalanced classes	When data is small, want reliable results	For time-series or sequential data
Pros	Simple and fast	Keeps classes balanced	Uses all data for training and testing	Keeps time order, avoids future data leaking
Cons	May not keep class balance	Only for classification problems	Takes more time to run	Not for random data, only ordered data
Example	Splitting customer data randomly	Heart disease data with rare cases	Small dataset with 5 folds	Stock prices split by date

Hyperparameter

Hyperparameter tuning is a fundamental process in machine learning that involves selecting the best set of hyperparameters to **maximize a model's performance and generalization ability**. **Hyperparameters are predefined configuration settings**, such as **learning rate, batch size, or the number of layers**, which are not learned from the data but **significantly influence the training process and final model quality**.

Some techniques

1. Grid Search : **exhaustively tests all** combinations of specified hyperparameter values
2. Random Search: which **samples** parameter combinations **randomly** to reduce **computation cost**

Importance of it

1. Improve model performance: capturing patterns without overfitting or underfitting
2. Balance bias and variance trade off: controls complexity that affects the bias-variance trade off.
3. Enhancing learning efficiency
4. Optimizes resource usage:

Supervised, Unsupervised and Reinforcement learning

Aspect	Supervised Learning	Unsupervised Learning	Reinforcement Learning
Definition	A model is trained on a labeled dataset with input-output pairs to predict outputs for new data	A model identifies patterns or structures in unlabeled data without predefined outputs	An agent learns by interacting with an environment, making decisions to maximize cumulative rewards .
Data Type	Labeled (input-output pairs)	Unlabeled (no predefined outputs)	No predefined input-output; uses rewards
Goal	Predict accurate outputs for new inputs	Find patterns or structures in data	Maximize cumulative reward
Feedback	Direct (correct labels)	None (inferred from data structure)	Delayed (reward signals from environment)
Examples	House price prediction, disease classification	Customer segmentation, data compression	Robot navigation, game playing
Algorithms	Linear regression, SVM, logistic regression	K-means, PCA	Q-learning, policy gradients
Human Involvement	Yes	No	Low

Gradient Descent Algorithm using a linear regression example

Gradient Descent is an optimization algorithm used to **find the optimal parameters of a machine learning** model by iteratively adjusting parameters to minimize.

In linear regression, the goal is to find the line that best fits a set of data points. The model point can be,

$y = wx + b$

$y : \text{predicted output}, w : \text{ weight}, x : \text{input}, b : \text{bias}$

We measure how well the model fits the data using a cost function. A common cost function for linear regression is the mean squared error (MSE)

$J(w, b) = \frac{1}{2m} \sum (\hat{y}_i - y_i)^2$

$\hat{y}_i = w_{xi} + b \text{ is predicted values}, y_i = \text{actual value}, m = \text{number of data point}$

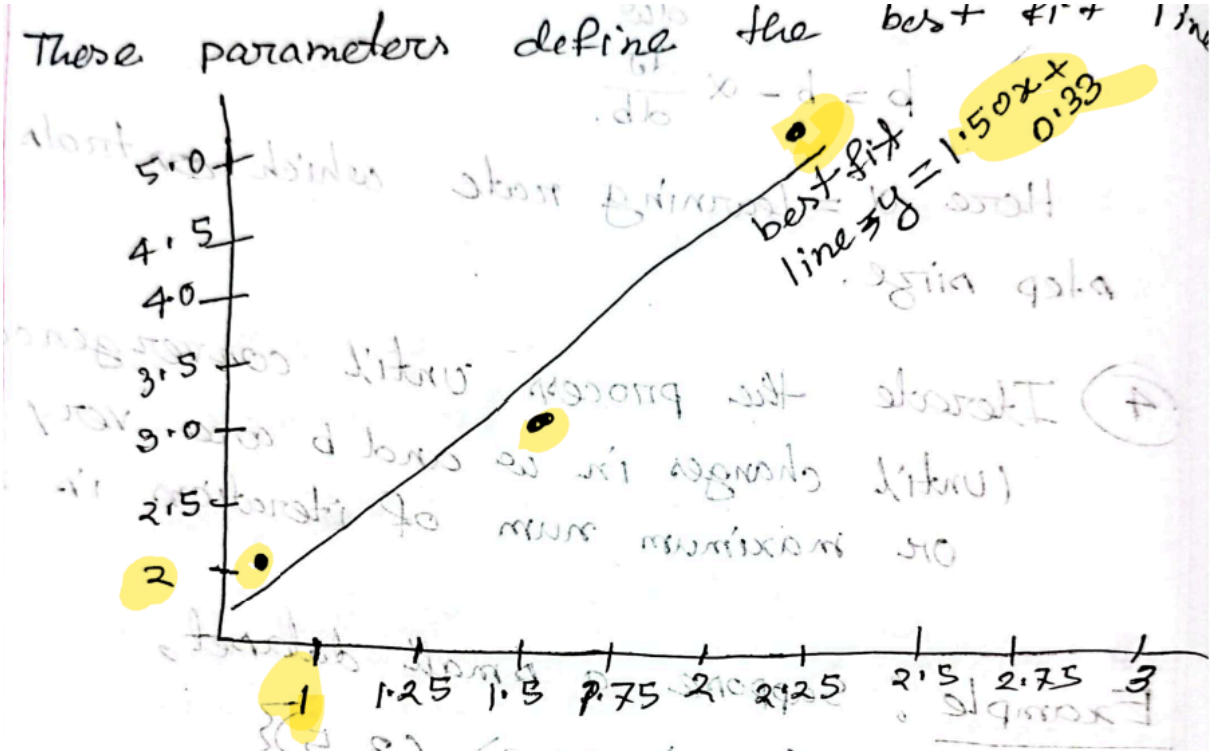
Algorithm

1. Lets, $w = b = 0$
2. Calculate the gradient descent
   a. $\frac{dJ}{dw} = \frac{1}{2m} \sum 2 \cdot (\hat{y}_i - y_i) \cdot (x_i + 0 - 0) = \frac{1}{m} \sum (\hat{y}_i - y_i) x_i$
   b. $\frac{dJ}{db} = \frac{1}{m} \sum (\hat{y}_i - y_i)$
3. Adjust w and b using gradient,
   a. $w = w - \alpha \frac{dJ}{dw}$
   b. $b = b - \alpha \frac{dJ}{db}$
4. Iterate the process until converges

Example: $w(x, y) = \{(1, 2), (2, 3), (3, 5)\}$

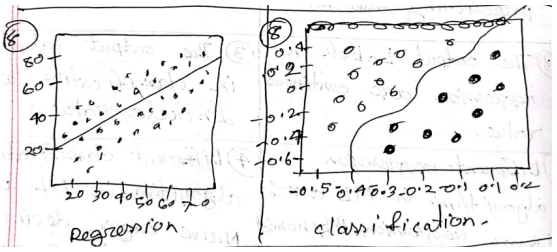
1. $w = b = 0$
2. Calculate gradients : For each iteration compute $\frac{dJ}{dw} \text{ and } \frac{dJ}{db}$
3. Learning rate, $\alpha = 0.001$

4. After multiple iteration, it finds values of $\mathrm{w\ and\ b}$ that minimizes $J(w,b)$



Regression and classification problems

Aspect	Regression	Classification
Definition	Regression is a type of supervised learning where the algorithm learns to **predict continuous values** based on input feature.	Classification is a type of supervised learning where the algorithm learns to **assign input to a specific category or class based** on input features.
Used for	Predicting the values	Predicting the class
Output Labels	Continuous values	Discrete
Algorithm	Linear, Polynomial, Decision tree	Naive Bayes, Decision tree, Logistic Regression
Evaluation Metrics	MSE, MAE. R^2 Score	Accuracy, Precision, ROC-AUC
Example	Predicting house price, forecasting sales, predicting temperature, stock price	Email Classification, Disease diagnosis, image recognition, fraud detection

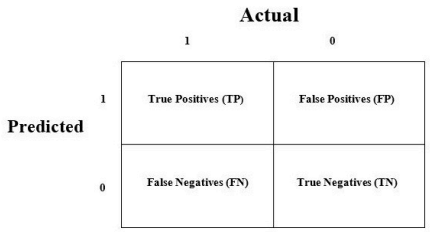


Performance metrics*, AUC - ROC

https://www.tutorialspoint.com/machine_learning/machine_learning_performance_metrics.htm

Performance metrics in machine learning are used to **evaluate the performance** of a machine learning **model**.

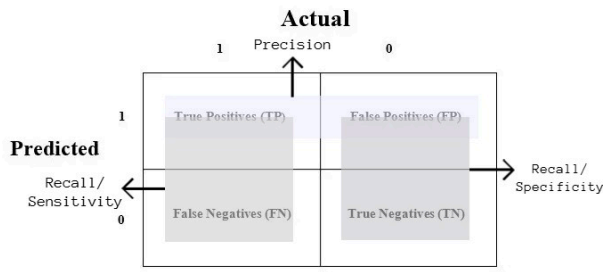
Confusion Matrix



Performance Metrics for Regression Problems

1. **Mean Absolute Error (MAE):**
   a. It is basically the sum of average of the absolute difference between the predicted and actual values.
   b. $\text{MAE} = \frac{1}{n}\sum|Y - \hat{Y}|$
2. **Mean Square Error (MSE):**
   a. average of the squared difference between target value and predicted value
   b. $\text{MAE} = \frac{1}{n}\sum(Y - \hat{Y})^2$
   c. Differentiable more optimized
3. **Root Mean Square Error (RMSE)**
   a. the square root of MSE
   b. $\text{MAE} = \sqrt{\frac{1}{n}\sum(Y - \hat{Y})^2}$
   c. Differentiable
   d. Handles small error done by MSE
4. R^2 Score:
   a. R Squared metric is generally used for **explanatory purpose** and provides an **indication of the goodness** or fit a set of predicted output values to the actual output values.
   b. $R^2 = 1 - \frac{\text{Sum of squared error}}{\text{Total Sum of squares}}$

Performance Metrics for Classification Metrics

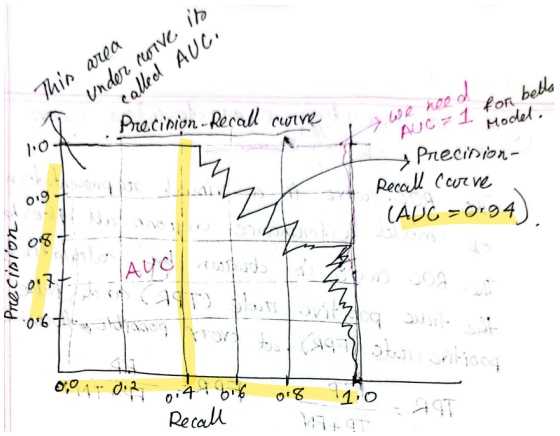


1. **Accuracy**
   a. It is the ratio correct prediction to total predictions
   b. $\frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{\text{Correct Prediction}}{\text{Total Observation}}$

2. **Precision**
    a. Proportion of true positive instances out of all predicted positive instances
    b. $\text{Precision} = \frac{TP}{TP+FP}$
    c. Defined as number of correct documents returned by our ML Model
    d. A precision score 1 → The model didn't miss any true positive and is able to classify well between correct and incorrect labelling
    e. A low precision (<0.5) → a high number of false positive
3. **Recall/Sensitivity**
    a. actual positive correctly identified
    b. $\text{Recall} = \frac{TP}{TP+FN}$
    c. Recall 1 → didn't miss any true positive and able to classify well
    d. Low recall (<0.5) → high number of false negative
4. **Specificity**
    a. proportion of actual negative correctly identified
    b. $\text{Specificity} = \frac{TN}{TN+FP}$
5. $F1$ Score:
    a. Harmonic mean of precision and recall
    b. $F1 = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

Precision-Recall Tradeoff

$\text{Precision} \propto \frac{1}{\text{Recall}}$, can improve one a a time, but not both. So we need precision-recall combination graph to observe both.



AUC

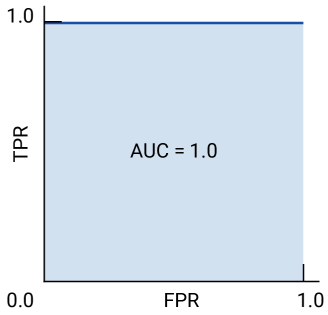
- Area Under curve
- For a better model, this should be 1

ROC AUC : Receiver Operating Characteristic Area Under Curve

https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

- ROC-AOC score is a measure of the ability of a classifier to distinguish between positive and negative instances
- The ROC curve is visual representation of model performance across all thresholds

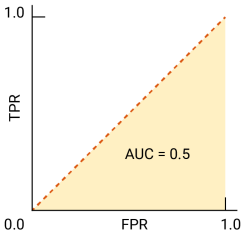
- Calculated by TPR and FPR
- $\text{TPR} = \frac{TP}{TP+FN}, \text{FPR} = \frac{TN}{TN+FP}$



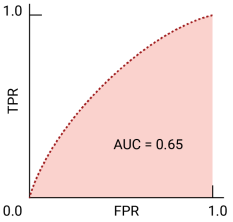
ROC and AUC of a hypothetical perfect model.

- AUC-ROC generates probability values instead of binary 0/1 values.
- It should be used where data set is roughly balanced
- ROC provides trade of between TPR And FPR
  - $\text{Value} < 0.5$ → Poor classifier
  - $\text{Value} = 0.5$ → Random
  - $\text{Value} > 0.7$ → Good
  - $\text{Value} = 0.8$ → Indicates strong classifier
  - $\text{Value} = 1$ → Perfectly predict everything

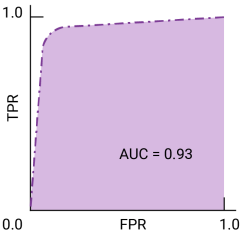
AUC-ROC Curve



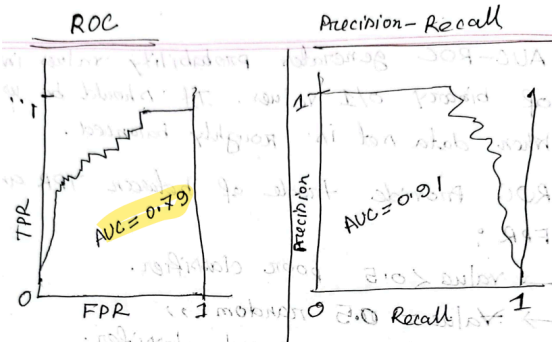
Random



Great AUC



Great AUC

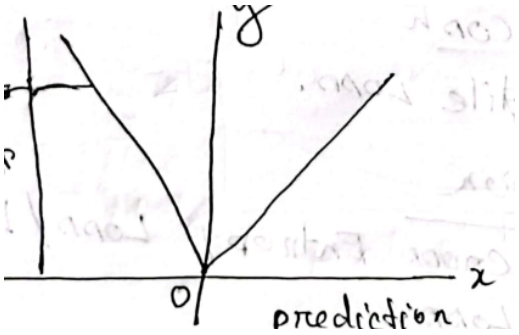


Loss functions, L1, L2, Huber loss, Binary Cross Entropy loss

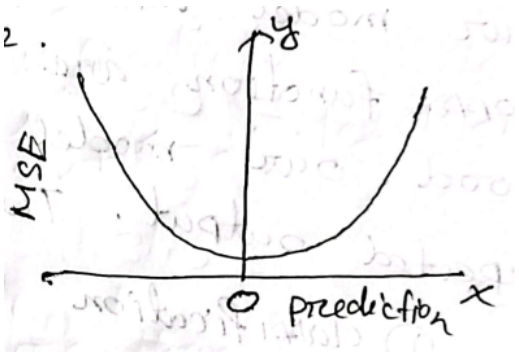
A loss function used in ML to measure the difference between the predicted output of a model and the actual target. Also known as cost function or error function.

Loss function for regression

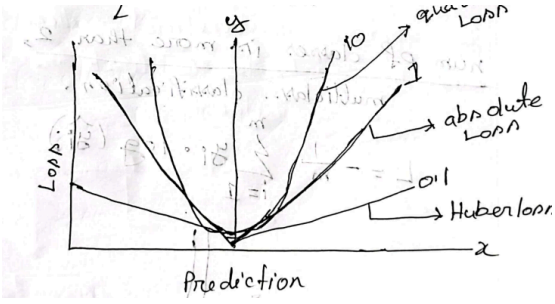
1. MAE/L_1 Loss
    a. average of absolute difference between the actual and the predicted value
    b. $MAE = \frac{1}{n}\sum|\hat{y}_i - y_i|$



2. MSE/L_2 Loss
    a. average of squared difference between actual and predicted value
    b. $MSE = \frac{1}{n}\sum(\hat{y}_i - y_i)^2$

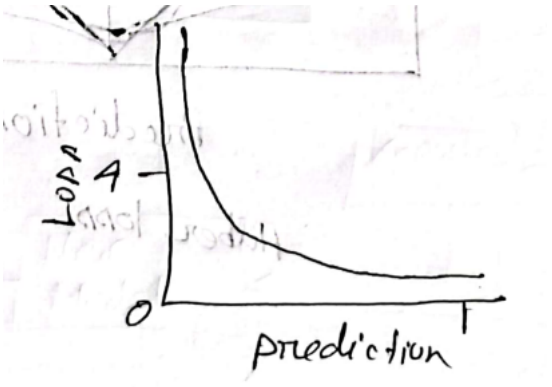


3. **Huber Loss**
    a. Defined as combination of MSE and MAE loss function
        i. MSE → when error is small (approx. 0)
        ii. MAE → when error is large (approx. α)
    b. Hyperparameter δ controls this error to make quadratic error
    c. $L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 & \text{Otherwise} \end{cases}$

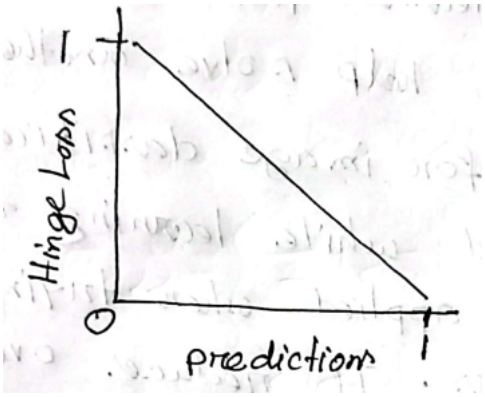


Loss functions for Classification

1. **Binary cross-entropy loss**
   a. Binary cross-entropy (log loss) is a loss function used in binary classification problems
   b. It measures the performance of a classified model whose predicted output is a probability value between 0 to 1
   c. When the number of classes is 2, its binary classification.
      i. $L = -\frac{1}{m}\sum y_i \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)$
   d. Binary cross-entropy for multiple classes (> 2)
      $L = \frac{1}{m}\sum y_i \log \hat{y}_i$



2. **Hinge Loss**
   a. It is developed for support vector machine model evaluation
      $L = max(0, 1 - y \times f(x))$



Overfitting and underfitting in terms of bias and variance

Bias : Gap between actual data of model and predicted value of data

- High Bias : Predicted value is more away than actual value (underfitting)
- Low Bias : Predicted value is near to actual value

Variance: Prediction value how much scatter with relation between each other

- Low Variance : Group of predicted does not scatter
- High Variance : Scatter with each other (Overfitting)

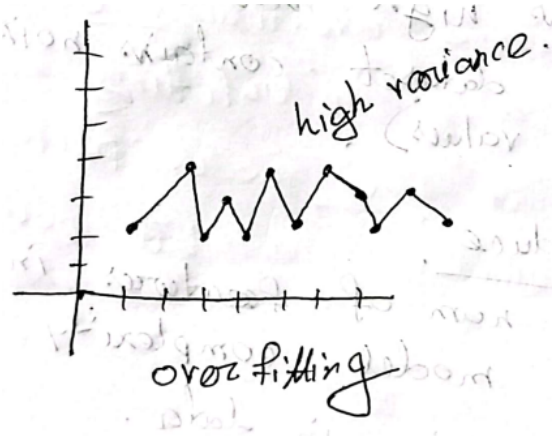
Overfitting: Overfitting is an undesirable condition in ML that occurs when the ML gives a**ccurate prediction for training data but** not for new data.

Reasons

1. High Variance and Low Bias
2. Model → too complex
3. Training data size → Less
4. Training data → irrelevant information, noise
5. Trains for too long on a single sample

Reductions

1. Removing Feature
2. Reduce Complexity
3. Increase the size of training data
4. Improve quality of training data
5. Early stopping during training phase



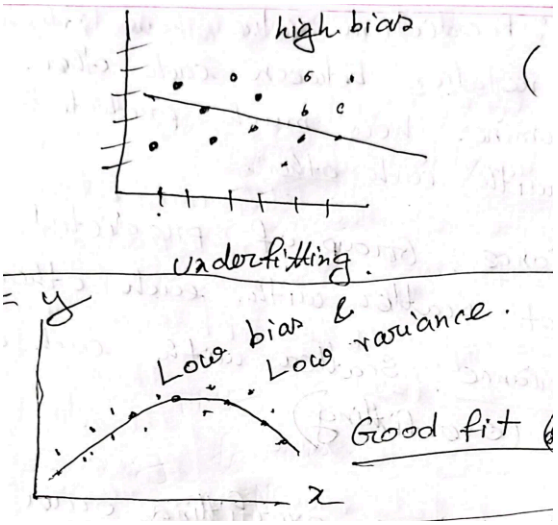
Underfitting: It represents **inability** of the model to learn the **training data effectively** and result in **poor performance both** on training and testing data.

Reasons

1. Too simple model
2. Model has no capability to represent complexity in data
3. Size of training data → less
4. High Bias
5. Training dataset → noise

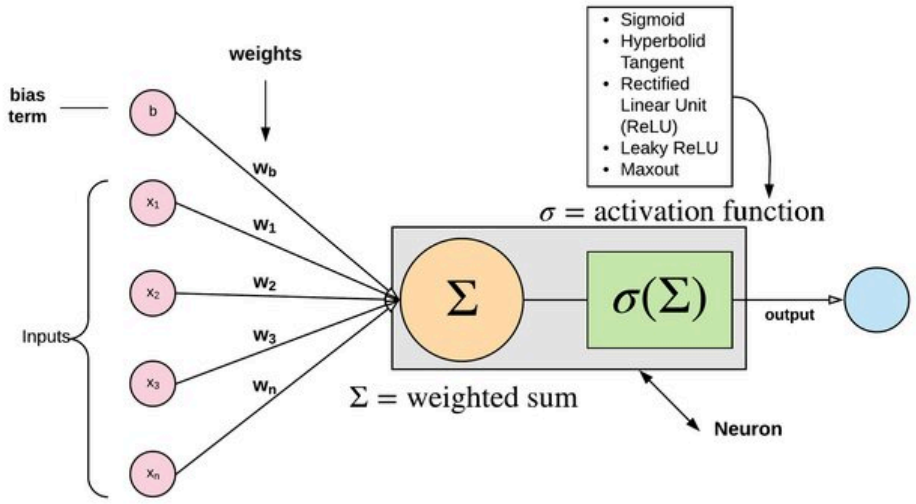
Reductions

1. Increase Complexity of model
2. Increase the size of training data
3. Increase number of features
4. Increase duration of dataset



Activation functions, linearity and differentiability of activation functions, ReLU activation functions, use of ReLU in deep learning

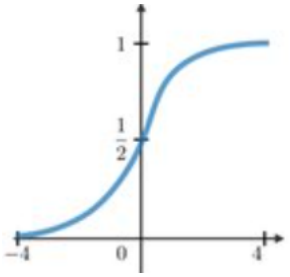
Activation function is a mathematical function used in artificial neural networks to determine the output of a neuron.



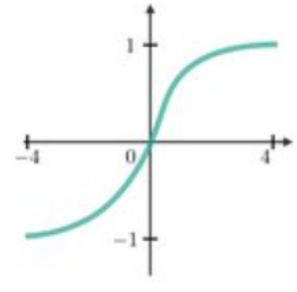
Common Activation functions

1. Linear: $f(x) = x$, $(-\inf, +\inf)$

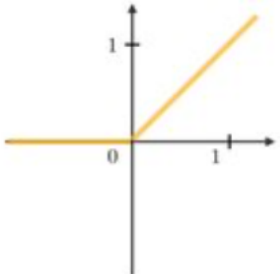
2. Sigmoid: $g(x) = \frac{1}{1+e^{-x}}$, $(0, 1)$



3. Tanh: $g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, (-1, 1)$

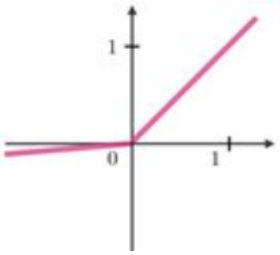


4. **ReLU : Reflected Linear Unit** $g(x) = max(0, x), [0, \inf)$

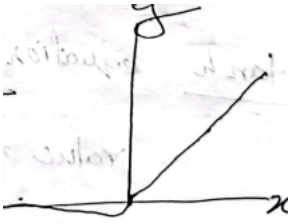


5. **Leaky ReLU:** $g(x) = \begin{cases} ax, & \text{for} x < 0 \\ x & x \geq 0 \end{cases}$

   $(-\inf, \inf)$



6. Swish : $g(x) = \frac{x}{1+e^{-x}}$



Activation function should be non-linear

- **WHY ?** To learn and represent complex and non-linear relationship between inputs and outputs
- If all activation functions are linear the NN would fail to mapping complex relation
  Output Layer 1 ：
  $f(w_1x + b_1)$

  Output Layer 2: $w_2f(w_1x + b_1) + b_2$

  If f is a linear function $f(x)$,

  $w_2(w_1x + b_1) + b_2 = w'x + b'$

  This is still a linear transformation, no matter how many layers are stacked.
- Linear activation makes deep network redundant where non-linear allows for **hierarchical feature learning, complex mapping**.

Activation function should be differentiable

1. **WHY?** It can be used in process of backpropagation neural network are trained using the backpropagation algorithm which involves forward and backward passing
   a. backward passing requires chain rule of calculus to compute derivatives layer by layer
      i. So, differentiability is must
2. **Example:** For a network output y with activation function f, the gradient with respect to w

   $\frac{dL}{dw} = \frac{dL}{df} \cdot \frac{df}{dz} \cdot \frac{dz}{dw}$
   where $z = wx + b$ and L is the loss function. If f is not differentiable, the chain rule can't propagate gradient through f.

ReLU activation function

ReLU is one of the most popular activation function used in NN especially in deep learning.
$f(x) = max(0, x), [0, \inf)$
It has become default choice

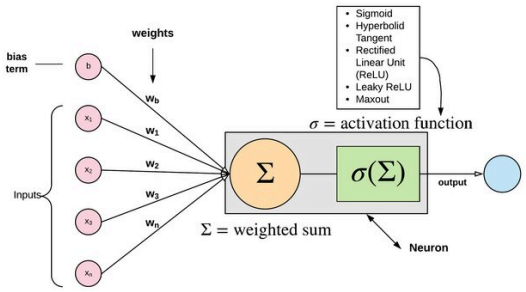
- simplicity and effectiveness
- positives values to pass through while setting all negative vales to zero.
- maintain necessary complex pattern

Use of ReLU

1. Non-linearity
2. It has constant gradient = 1 for $x > 0$ which to mitigate vanishing gradient problem.
3. Make training deep networks computationally efficient and effective
4. Gradient is constant and doesn't shrink, allowing effective backpropagate in deep learning
5. Non linearity and computational simplicity
6. Simple operation reduces computation

Basic structure of Artificial Neurons*
[Basic terminologies, Learning rate, momentum, threshold]

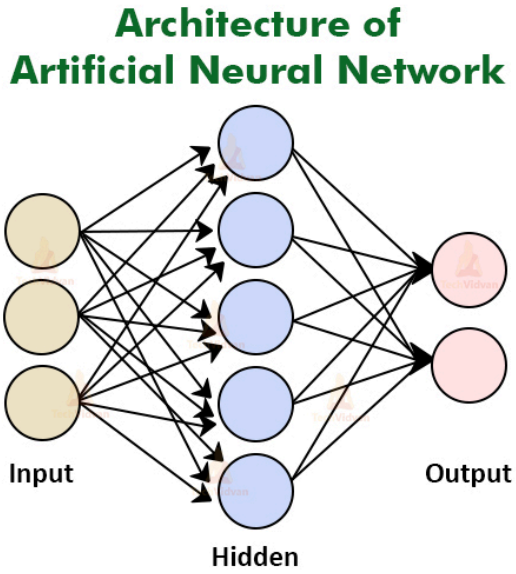
Artificial Neurons is a **mathematical model** to simulate how **neurons process information.** It is the fundamental building block of artificial neural network.



Artificial Neuron

Artificial Neural Network:

- A machine learning algorithm that uses a network of interconnected nodes to process data, inspired by the human brain.



Basic components of Perceptron:

Perceptron is type of ANN (Artificial Neural Network), which is a fundamental concept in ML.

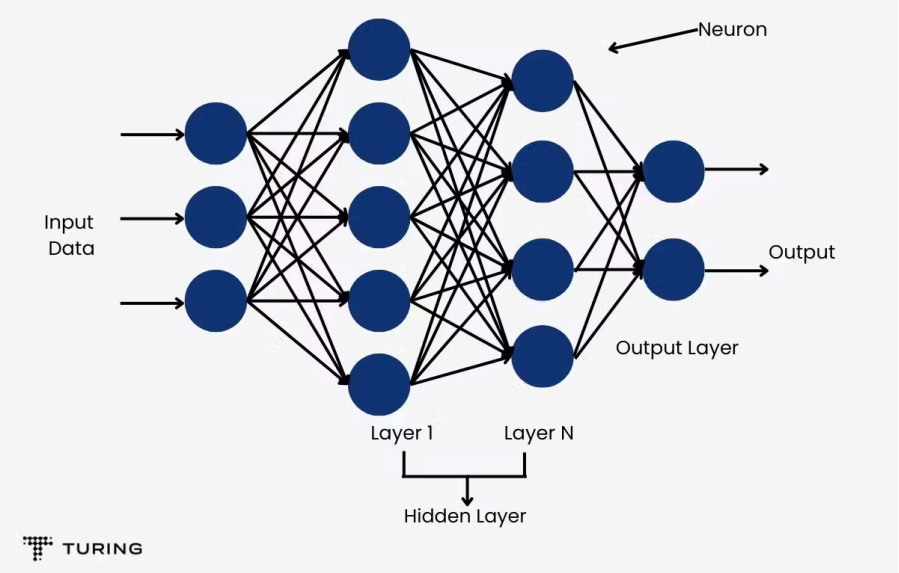
1. **Input Layer**
    a. Consists of one/more input neuros
    b. receive signal from external world
2. **Weight:**
    a. strength of the connection between input and output neuron
3. **Bias**
    a. added in input layer to provide the perceptron with additional flexibility
4. **Activation function:**
    a. Activation function is a mathematical function used in artificial neural networks to determine the output of a neuron.
5. **Output:**
    a. a single binary value either 0/1

Basic Terminologies

1. **Learning Rate:**
   a. a critical hyperparameter in ML and NN that determines the **size of the steps taken during optimization process to minimize loss function**.
   b. $W_{new} = W_{old} - LR \times G$
   c. Controls how much the model updates its weight wrt the gradient
   d. Small LR → 0.0001 → Slow convergence
   Large LR → 1.0 → Speed up Loss oscillates
   LR → 0.01 → weights are update at right space.
2. **Momentum**
   a. a parameter optimization technique that accelerates the gradient descent by adding a fraction of the previous update to the current update
   b. Reduce oscillations and improves convergence
   c. $V_t = \beta V_{t-1} - \eta \Delta L(w_t)$
3. **Threshold**
   a. refers to **a boundary value** used to determine **how a neuron behaves or how output are processed** particularly in binary classification or activation function.
   b. Decision making
   c. $y = \begin{cases} 1 & \text{if } \sum w_i x_i + b > threshold \\ 0 & \text{otherwise} \end{cases}$
   d. A NN predicts the probability that an email is spam and the threshold is 0.5. $P(0.7) = \text{SPAM}$

Structure of Multi-layer backpropagation network, derivation of backpropagation algorithm

Backpropagation is a fundamental algorithm for training artificial neural network. The goal of backpropagation is to reduce the difference between the models predicted output and the actual output by adjusting weights and biases.



Structure of Multi-layer backpropagation network

It typically consists of three main components.

1. Input layer
2. Hidden layer :
    a. intermediate layer that learn pattern from input data
    b. each neuron in hidden layer is connected to every in the previous and subsequent layers
    c. Hidden layers $\propto$ Network Capacity
3. Output layer
    a. apply a suitable activation function

Derivation of backpropagation

1 Backpropagation

1.1 Derivation

Introduction Backpropagation is a key algorithm for training artificial neural networks by minimizing the error between predicted and target outputs using gradient descent. This derivation outlines the process of adjusting weights layer by layer.

Define the Error The total error E in the network is the sum of squared differences between target values t_k and actual outputs a_k across all output nodes k:

$$E = \frac{1}{2}\sum_k (t_k - a_k)^2$$

The factor $\frac{1}{2}$ simplifies later derivative calculations. The goal is to adjust weights to minimize E.

Weight Change Rule (Gradient Descent) To minimize E, we update each weight w by a small step in the opposite direction of the gradient of E with respect to that weight:

$$\Delta w_{jk} \propto -\frac{\partial E}{\partial w_{jk}}$$

Incorporating the learning rate ε, the weight change is:

$$\Delta w_{jk} = -\varepsilon \frac{\partial E}{\partial w_{jk}}$$

Chain Rule Breakdown Since E is not directly a function of w_{jk}, we use the chain rule:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial a_k} \cdot \frac{\partial a_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{jk}}$$

Substitute into the weight change equation:

$$\Delta w_{jk} = -\varepsilon \frac{\partial E}{\partial a_k} \cdot \frac{\partial a_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{jk}}$$

Compute Each Partial Derivative Derivative of Error with Respect to Activation ($\frac{\partial E}{\partial a_k}$) For one output node:

$$E_k = \frac{1}{2}(t_k - a_k)^2$$

Derivative with respect to a_k:

$$\frac{\partial E_k}{\partial a_k} = \frac{\partial}{\partial a_k}\left[\frac{1}{2}(t_k - a_k)^2\right] = \frac{1}{2} \cdot 2(t_k - a_k) \cdot (-1) = -(t_k - a_k)$$

This shows the error's change depends on the difference between target and output.

Derivative of Activation with Respect to Net Input ($\frac{\partial a_k}{\partial net_k}$) Assuming a sigmoid activation function $a_k = \frac{1}{1+e^{-net_k}}$:

$$\frac{\partial a_k}{\partial net_k} = \frac{\partial}{\partial net_k}\left[(1 + e^{-net_k})^{-1}\right] = \frac{e^{-net_k}}{(1 + e^{-net_k})^2}$$

Rewrite in terms of a_k:

$$1 - a_k = \frac{e^{-net_k}}{1 + e^{-net_k}}, \quad \frac{e^{-net_k}}{(1 + e^{-net_k})^2} = a_k(1 - a_k)$$

So, $\frac{\partial a_k}{\partial net_k} = a_k(1 - a_k)$.

Derivative of Net Input with Respect to Weight ($\frac{\partial net_k}{\partial w_{jk}}$) The net input is $net_k = \sum_j w_{jk} a_j$. For w_{jk}:

$$\frac{\partial net_k}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}(w_{jk} a_j) = a_j$$

This depends on the activation a_j from the hidden node.

Weight Change Rule for Hidden-to-Output Weight Substitute the derivatives:

$$\Delta w_{jk} = -\varepsilon\left[-(t_k - a_k)\right] \cdot \left[a_k(1 - a_k)\right] \cdot \left[a_j\right]$$

Simplify:

$$\Delta w_{jk} = \varepsilon(t_k - a_k) a_k (1 - a_k) a_j$$

This adjusts w_{jk} based on error, output sensitivity, and hidden activation.

Weight Change for Input-to-Hidden Weight For w_{ji}, the error at the hidden node depends on all output nodes:

$$\frac{\partial E}{\partial w_{ji}} = \sum_k \frac{\partial E}{\partial a_k} \cdot \frac{\partial a_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial a_j} \cdot \frac{\partial a_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}}$$

$-\frac{\partial E}{\partial a_k} \cdot \frac{\partial a_k}{\partial net_k} = -(t_k - a_k)a_k(1 - a_k)$. $-\frac{\partial net_k}{\partial a_j} = w_{kj}$. $-\frac{\partial a_j}{\partial net_j} = a_j(1 - a_j)$. $-\frac{\partial net_j}{\partial w_{ji}} = a_i$. Combine:

$$\Delta w_{ji} = -\varepsilon \sum_k \left[-(t_k - a_k)a_k(1 - a_k)w_{kj}\right] \cdot \left[a_j(1 - a_j)a_i\right]$$

Define $\delta_k = (t_k - a_k)a_k(1 - a_k)$ and $\delta_j = \sum_k \delta_k w_{kj} a_j(1 - a_j)$:

$$\Delta w_{ji} = \varepsilon \delta_j a_i$$

Conclusion Backpropagation uses the chain rule to propagate errors backward, adjusting weights to minimize E. The derivation differs for hidden-to-output and input-to-hidden weights due to error propagation across layers.

Linearly separable and linearly non-separable problems

Convolution*, CNN*, usage of CNN*, different layers in CNN

Convolution

Convolution is a mathematical operation used in signal and image processing to extract features by applying a filter (or kernel) over an input. In the context of neural networks, this operation forms the foundation of Convolutional Neural Networks (CNNs), which are specialized for processing structured grid-like data such as images.

CNN

A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. CNNs are particularly effective for tasks like image classification, object detection, and facial recognition due to their ability to capture spatial hierarchies and patterns. Convolutional Neural Network consists of multiple layers.

Uses of Convolutional Neural Networks

- Image classification
- Object detection
- Facial recognition
- Image segmentation
- Medical Image Analysis
- Video Analysis

Different Layers in CNN

1. Input : Receives the **raw input data, such as pixel values of an image**, and passes it to subsequent layers for processing.
2. **Convolution:** Applies convolution operations using filters to extract features like edges, textures, or patterns from the input data.
3. **Activation Layer:** Introduces non-linearity (e.g., ReLU) to enable the network to learn complex patterns by transforming the convolved output.
4. **Pooling Layer:** Reduces the **spatial dimensions** (e.g., max pooling) of the feature maps, retaining important information while decreasing computational load.
5. **Fully Connected Layer: Connects all neurons from the previous layer** to **produce** high-level reasoning, consolidating features for final decision-making.
6. **Output Layer: Generates the final output**, such as class probabilities or a classification label, based on the processed features

RNN, uses of RNN, vanishing and exploding gradients problems, how solved?* Architectural difference between LSTM and GRU*

Recurrent Neural Networks (RNNs) are a class of neural networks designed to handle sequential data by maintaining a memory of previous inputs through recurrent connections.

Uses of Recurrent Neural Networks

- Natural Language Processing
- Time series prediction
- Speech recognition
- Machine translation

Vanishing and Exploding Gradients Problems

During backpropagation through time (BPTT) in RNNs, the following issues arise:

- **Vanishing Gradients:** Gradients become extremely small as they are **propagated back through many time steps**, causing early layers to **learn slowly** or not at all.
- **Exploding Gradients:** Gradients grow excessively large, leading to **unstable training and large weight** updates that **disrupt learning**.

Solutions to Vanishing and Exploding Gradients

- **Gradient Clipping:** Limits the gradient magnitude during backpropagation to prevent exploding gradients by capping values above a threshold.
- **Long Short-Term Memory (LSTM) Units:** Introduces memory cells and gates (input, forget, output) to preserve long-term dependencies, mitigating vanishing gradients
- **Gated Recurrent Units (GRU):** A simplified version of LSTM with update and reset gates, improving gradient flow and reducing vanishing gradient effects.
- **ReLU Activation:** Replaces traditional activations to avoid the saturation issues that contribute to vanishing gradients

Architectural Differences Between LSTM and GRU

Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) are advanced recurrent neural network architectures designed to mitigate the vanishing gradient problem. This document compares their architectural differences in a tabular format.

Aspects	LSTM	GRU
Gating Mechanisms	Uses three gates: input gate, forget gate, and output gate, along with a separate cell state and hidden state to control information flow.	Uses two gates: update gate and reset gate, with a single hidden state that handles both memory and update functions.
Number of Parameters	Higher due to the separate memory cell and three gates, increasing computational complexity.	Lower as it combines the forget and input functions into the update gate, reducing computational cost.
Memory Cell	Maintains a distinct cell state to preserve long-term memory, updated through the gates.	Lacks a separate cell state; the hidden state manages both short- and long-term memory.
Complexity and Training	More complex due to additional gates, potentially better for long sequences but slower to train.	Simpler and faster to train, often performing comparably on shorter sequences.

Transfer learning*, usage*

Transfer learning is a machine learning technique where a model trained on a large, general dataset is fine-tuned for a specific task with a smaller dataset.

Transfer learning is a machine learning technique where knowledge gained from solving one task is leveraged to improve a model's performance on a different but related task. Instead of training a model from scratch for every new problem, transfer learning uses a pre-trained model—typically trained on a large dataset for a source task—and adapts it (often by fine-tuning) for a target task, which may have less data or slightly different requirements

Usage

Application	Description
Image Classification	Adapts pre-trained models (e.g., ResNet) for new image categories with limited data.
Natural Language Processing	Fine-tunes models like BERT for tasks such as sentiment analysis or question answering.
Medical Imaging	Utilizes pre-trained models to detect diseases in X-rays or MRIs with small labeled datasets.
Speech Recognition	Adjusts pre-trained audio models for specific languages or accents.

Transformer model*, Attention mechanism in transformer architecture*, Transformer vs RNN*.

The Transformer model is a deep learning architecture introduced for sequence-to-sequence tasks, relying entirely on attention mechanisms and eliminating recurrent structures.

A transformer model is a type of neural network architecture that has revolutionized the field of artificial intelligence, especially in natural language processing (NLP) and other sequential data tasks. Introduced in the 2017 paper "Attention is All You Need," transformers have become the foundation for many modern AI systems, including large language models and generative AI.

Attention mechanism

Aspect	Description
Self-Attention	Allows the model to weigh the importance of different words in a sequence, capturing contextual relationships.
Multi-Head Attention	Uses multiple attention mechanisms in parallel to focus on different parts of the input simultaneously.
Scaled Dot-Product Attention	Computes attention scores by scaling the dot product of query and key vectors, improving gradient stability.

Transformer vs RNN

Aspects	Transformer	RNN
Architecture	Based on attention mechanisms, no recurrence.	Relies on sequential recurrence with hidden states.
Parallelization	Highly parallelizable, faster training on large data.	Sequential processing, slower due to time dependency.
Long-Term Dependencies	Effectively captures long-range dependencies via attention.	Struggles with vanishing/exploding gradients over long sequences.
Memory Usage	Fixed memory, independent of sequence length.	Memory grows with sequence length due to recurrence.