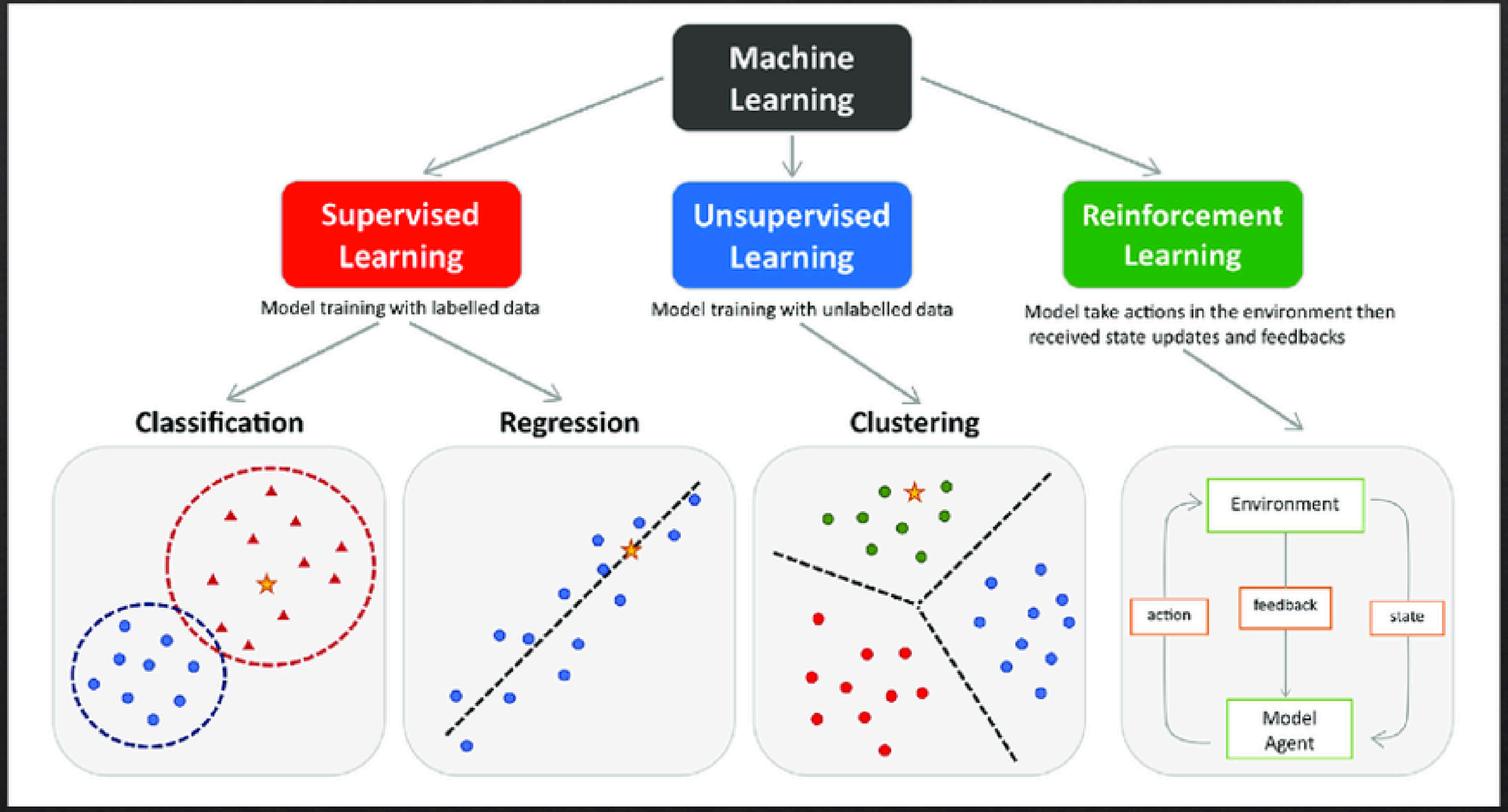
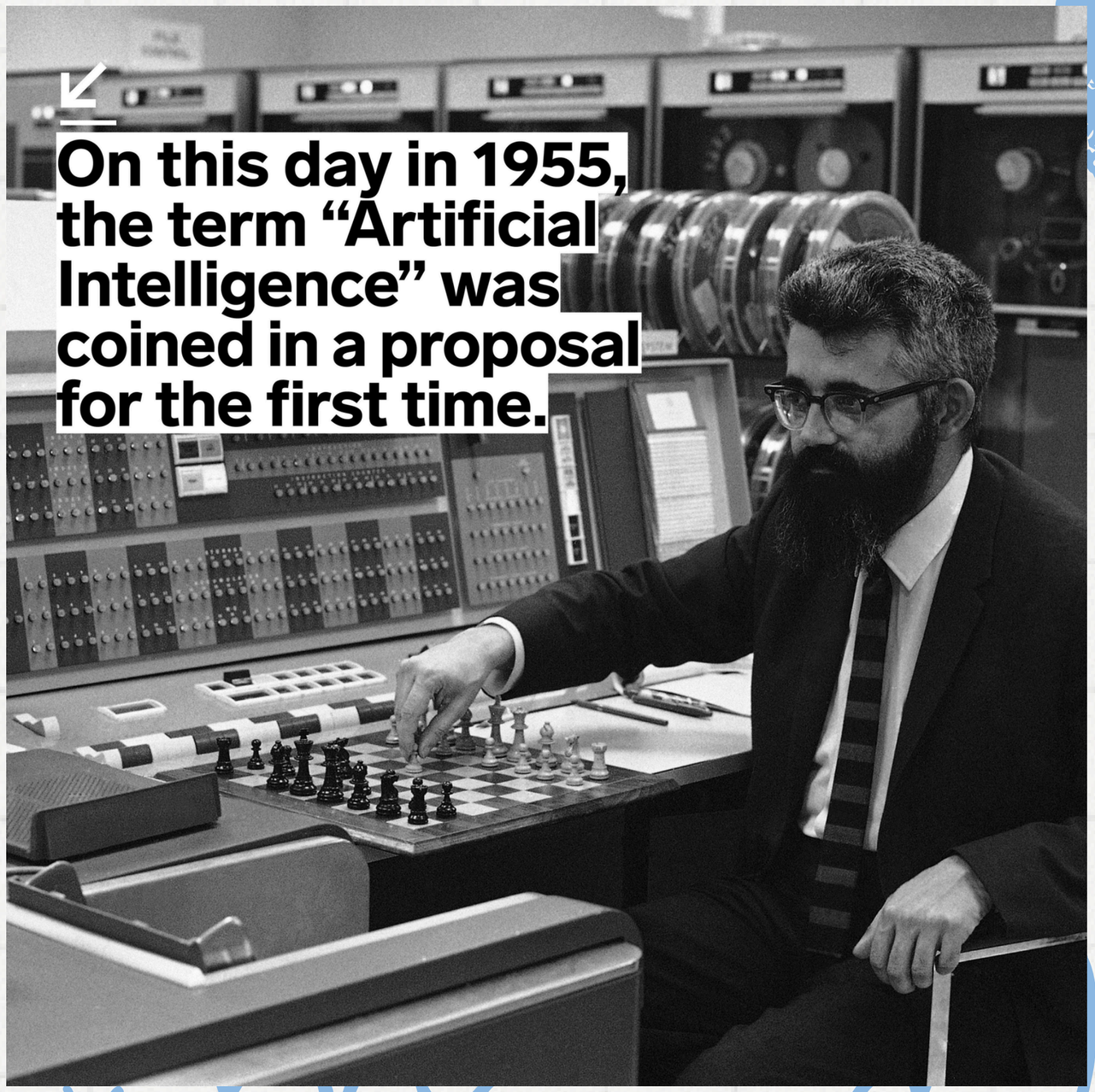
Types of machine learning



When was the term “AI” coined?

Term “AI was coined in 1955 (1956) by John McCarthy during the conference which they hosted together with Marvin Minsky”

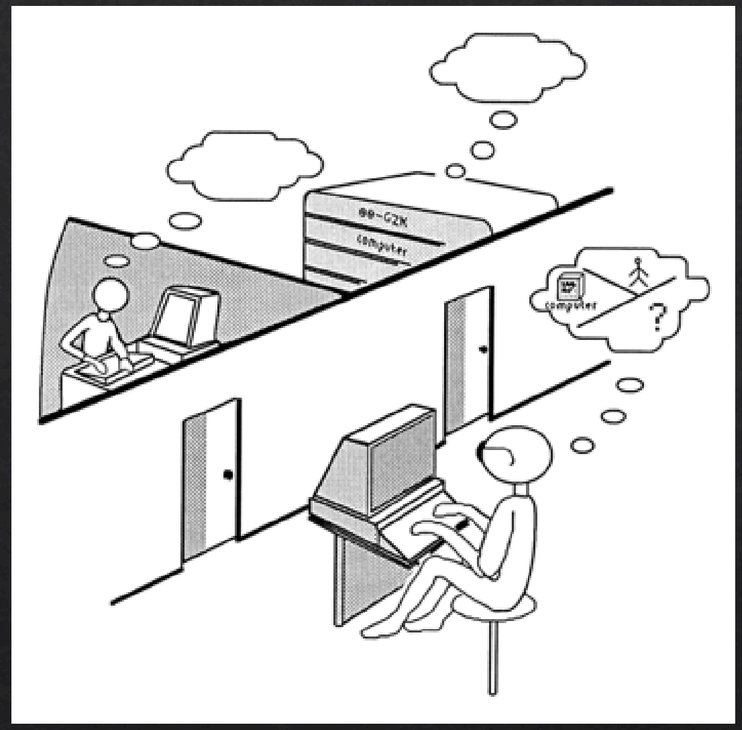


What is Turing test?

- The Turing Test is a measure of a machine's ability to exhibit intelligent behavior indistinguishable from that of a human. It was proposed by the British mathematician and computer scientist Alan Turing in his 1950 paper titled "Computing Machinery and Intelligence." The test is designed to evaluate a machine's capability to engage in natural language conversation.

- Here's how the Turing Test typically works:

- A human judge engages in a natural language conversation with both a human and a machine without knowing which is which.
- If the judge cannot reliably distinguish the machine from the human based on their responses, then the machine is said to have passed the Turing Test.



How to find distances

Euclidean distance

$$d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Manhattan distance

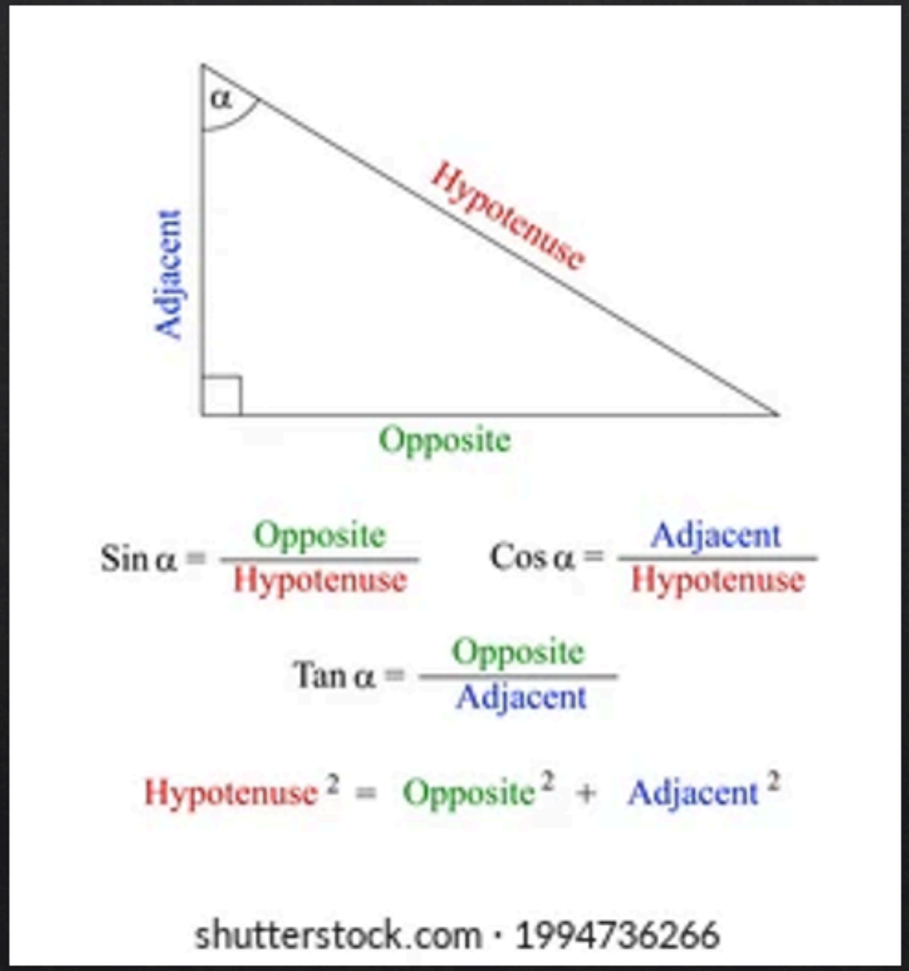
$$d_1((x_1, y_1), (x_2, y_2)) = |x_2 - x_1| + |y_2 - y_1|$$

What is the difference between Euclidean and Manhattan distances?

- Euclidean distance is the shortest path between source and destination which is a straight line.
- Manhattan distance is sum of all the real distances between source(s) and destination(d) and each distance are always the straight lines.

- Manhattan distance is usually preferred over the more common Euclidean distance when there is high dimensionality in the data.

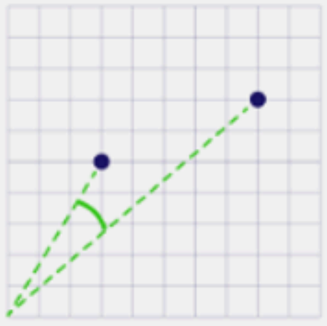
Cos/sin and how to find them



Cosine similarity and distance

- Cosine distance & Cosine Similarity metric is mainly used to find similarities between two data points. As the cosine distance between the data points increases, the cosine similarity, or the amount of similarity decreases, and vice versa.

$$similarity(A, B) = cos(\theta) = \frac{A \cdot B}{||A||\,||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sum_{i=1}^{n} B_i^2}.$$



Cosine Distance

$$1 - \frac{A \cdot B}{||A|| \quad ||B||}$$

Operations with vectors

Addition and Scalar Multiplication:

Addition: $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$

Scalar multiplication: $a(x, y) = (ax, ay)$

Dot product:

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2$$

$$\mathbf{a} \cdot \mathbf{b} = ||\mathbf{a}||\ ||\mathbf{b}|| \cos(v)$$

Length of a vector:

$$\textit{length of } \mathbf{v} = ||\mathbf{v}|| = d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$||\mathbf{v}|| = \sqrt{x^2 + y^2}$$

Exhaustive search

- brute-force search, is a straightforward method that systematically explores all possible solutions to find the optimal one
- Check every single possibility (all **permutations**), always keep track on the best one

- ***Pros*:** Guarantees finding the best solution, suitable for small problem spaces.
- ***Cons*:** Inefficient for large search spaces due to the exponential growth of possibilities.

Hill climbing

local search algorithm that continually moves towards the direction of increasing elevation in the search space.

- ***Process*:**
- Choose 1 random solution as currently best
- Replace with neighbor solution
- Run n times

(The algorithm iteratively improves the current solution by making small adjustments)

- *Pros:* Efficient for finding local optima in smooth landscapes.
- *Cons:* Prone to getting stuck in local optima and sensitive to the starting point.

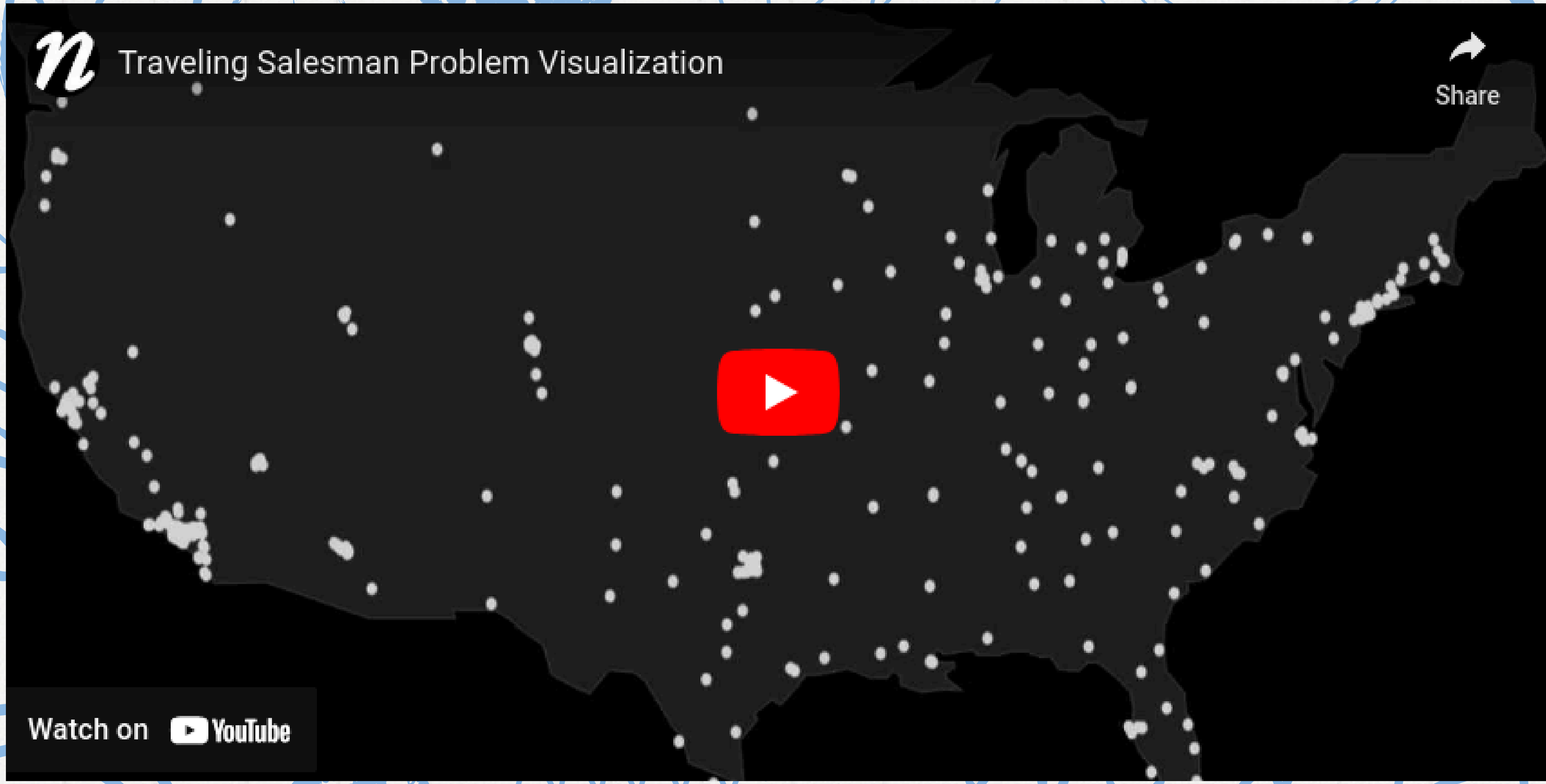
Possible code for Hillclimbing:

```
best_solution = GENERATE_RANDOM_SOLUTION()
best_distance = EVALUATE_DISTANCE(best_solution)
for(num_iterations in range(0,200)):
    new_solution=GENERATE_RANDOM_NEIGHBOR(best_solution)
    new_distance=EVALUATE_DISTANCE(new_solution)
    if(new_distance < best_distance):
        best_solution = new_solution
        best_distance = new_distance
#After the loop finalizes, the result of hillclimbing is in best_solution.
```

Greedy search

- makes locally optimal choices at each stage with the hope of finding a global optimum.
- ***Process*:** selects the best available option at each step without reconsidering previous choices.(1 solution at a time, several locally optimal choices in hopes to end up in global optima)

- *Pros*: Simple and computationally efficient for certain problems.
- *Cons*: May not always lead to the globally optimal solution as it lacks a comprehensive view of the entire search space. The algorithm never reverses the earlier decision even if the choice is wrong. It works in a top-down approach.


Traveling Salesman Problem Visualization
Share
Watch on YouTube


TSP - Hill Climbing
Share
ROUTE
Istanbul—Sofia
Saint Petersburg—Moscow
Brussels—Amsterdam
Dublin—Reykjavik
Reykjavik—London
Warsaw—Berlin
Bucharest—Istanbul
Milan—Munich
Budapest—Bucharest
Vienna—Budapest
Kiev—Warsaw
DISTANCE
23804.73
RANDOM
GREEDY
ALGORITHM
REMOVE OVERLAPS
HILL CLIMBING
SIMULATED ANNEALING
TEMP
Watch on YouTube

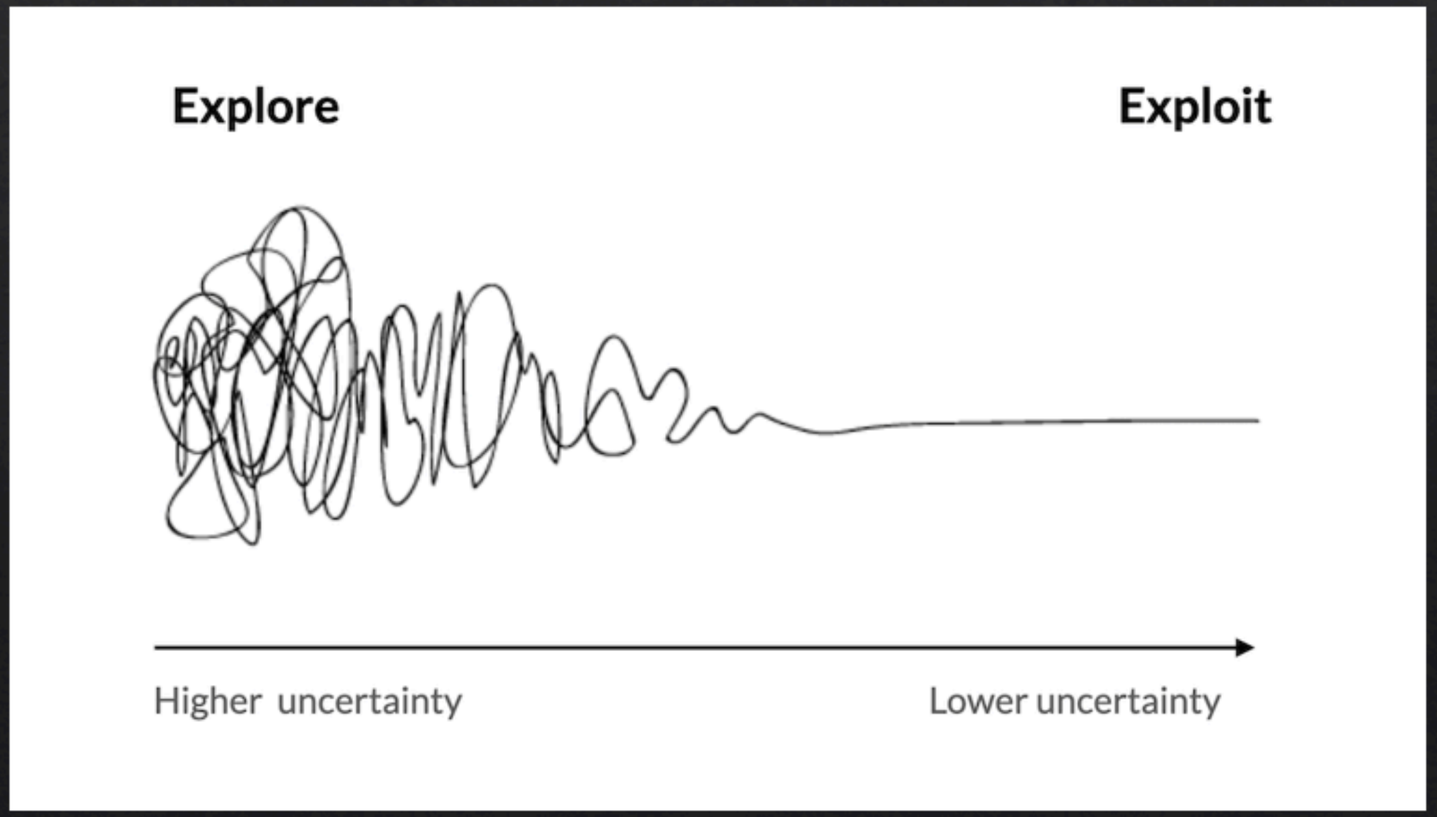
Simulated annealing

- probabilistic optimization algorithm inspired by the annealing process in metallurgy.
- ***Process***: The algorithm explores the solution space by allowing for occasional "bad" moves with decreasing probability(Temperature) over time.

- ***Pros***: Can escape local optima, suitable for complex and rugged search spaces.
- ***Cons***: Requires careful tuning of parameters and may have slower convergence compared to some deterministic algorithms.

Exploration vs exploitation

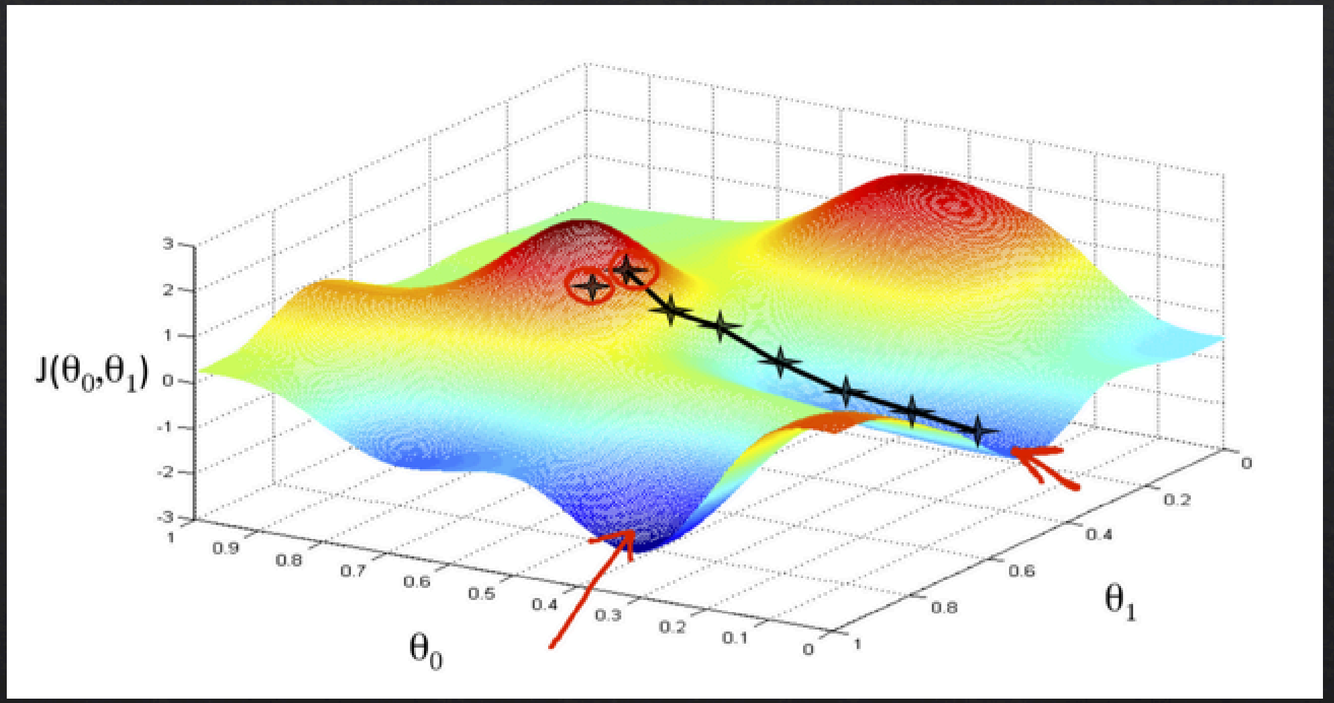
- Exploration is any action that lets the agent discover new features about the environment, while exploitation is capitalizing on knowledge already gained.



Gradient descent/ascent

- iterative optimization algorithm that minimizes/maximizes a function by **adjusting its parameters in the direction of steepest descent/ascent**.

- ***Process*:** The algorithm calculates the gradient of the function at each step and updates the parameters accordingly.
- *Pros*: Efficient for convex and smooth optimization problems.
- *Cons*: May converge to a local minimum, sensitivity to initial conditions, and challenges in handling non-convex functions.



Gradient ascent: $x^{(k+1)} = x^{(k)} + \gamma \nabla f(x^{(k)})$

$$x_{n+1} = x_n - \alpha \nabla f(x_n)$$

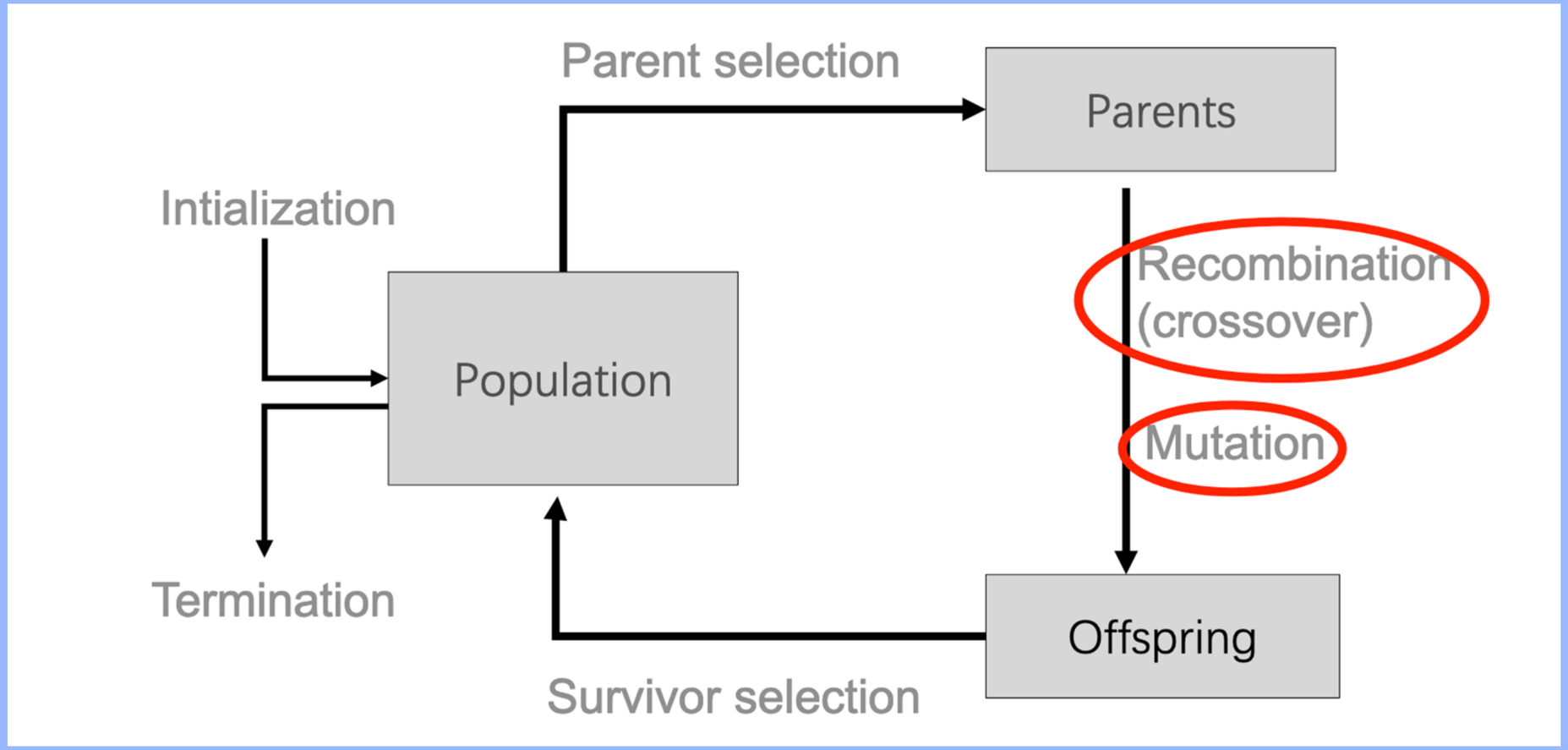
https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions/a/what-is-gradient-descent#:~:text=Gradient%20descent%20minimizes%20differentiable%20functions,direction%20of%20the%20negative%20gradient.

Which of the algorithms is the best?

- **No free lunch theorem!**

- There is no universal solution for all problems!

GENERAL SCHEME OF EA



EA IN PSEUDOCODE

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

ESSENTIAL PARTS OF EA REPRESENTATION

Population:

Representation: A collection of individuals or candidate solutions.
Structure: The population is often represented as an array or list, where each element corresponds to an individual genome.

Fitness Function:

Representation: A function that evaluates the quality of a solution.
Structure: The fitness function takes a genome as input and assigns a numerical value representing the solution's fitness or performance. It guides the selection process in the evolutionary cycle.

Genome(chromosomes):

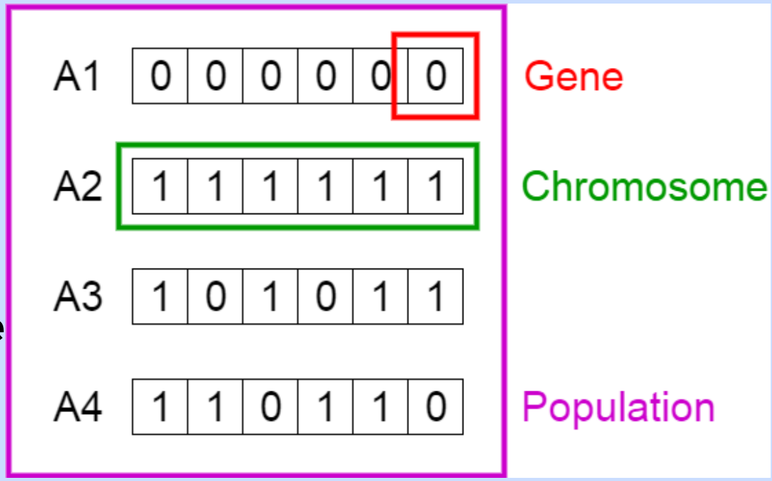
The encoded solution to the optimization problem.

The genome can be represented using various encoding schemes such as binary strings, real-valued vectors, integers, permutations, trees, or other structures depending on the nature of the problem.

Initiallisation and **termination** criteria are essential parts part of the algorithm indicating start and finish

Genotype structure

Locus: the position of a gene
Allele = 0 or 1 (What values a gene can have)
Gene: one element of the array
Genotype: a set of gene
valuesPhenotype: What could be built/developed based on the genotype



GENOTYPE VS PHENOTYPE

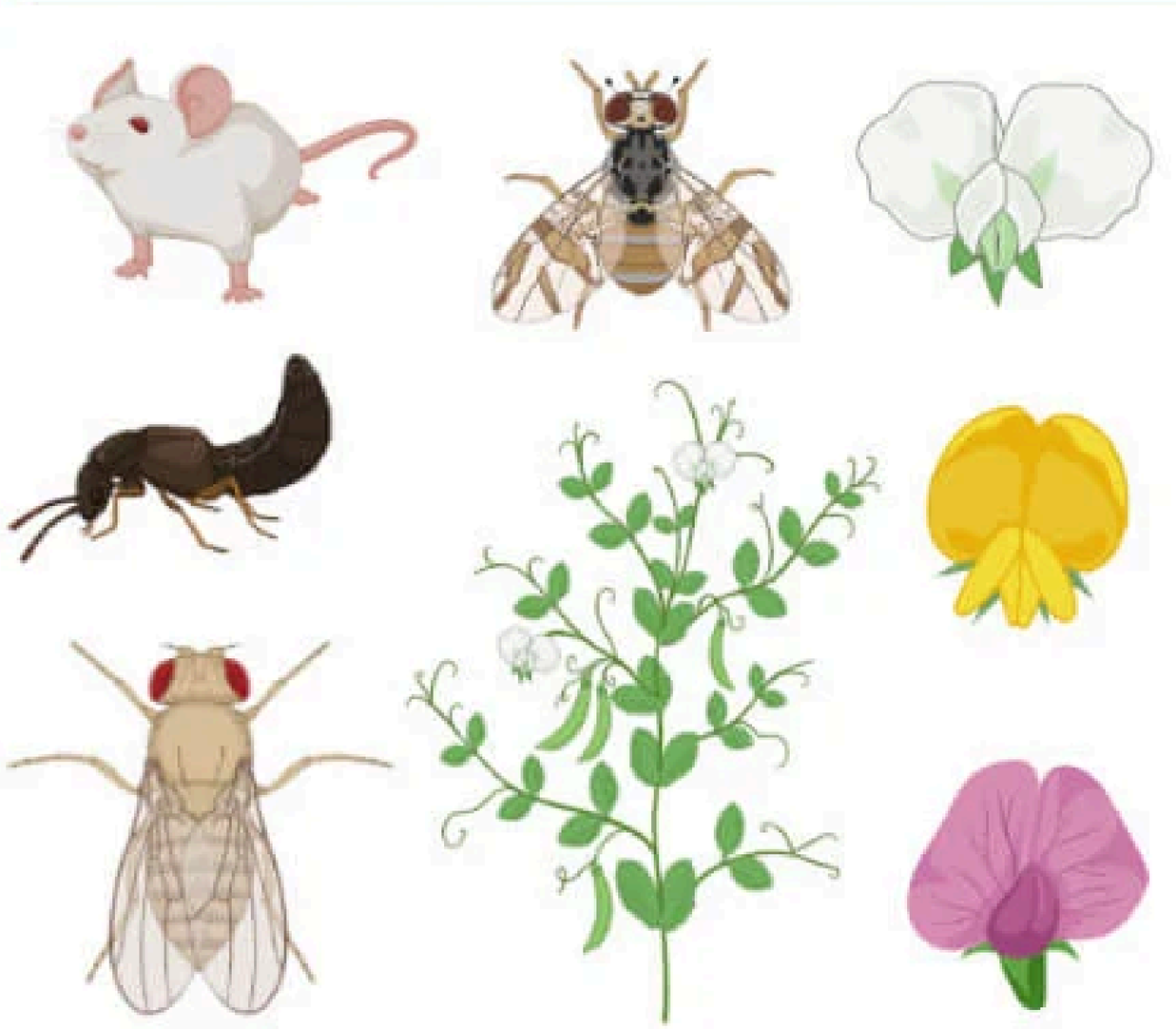
a set of gene values

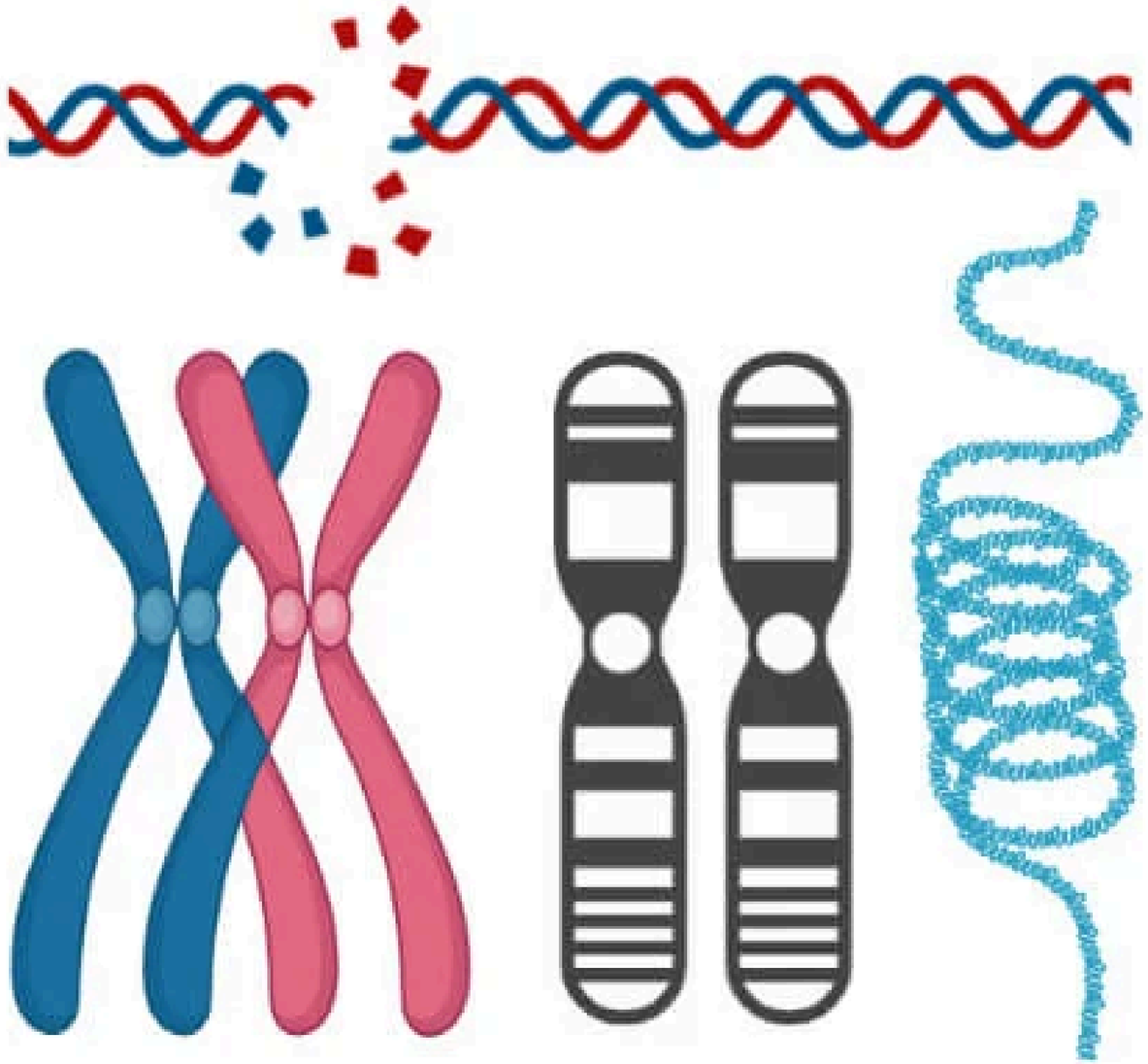
The genotype serves as the raw genetic material that undergoes genetic operations (crossover and mutation) during the evolutionary process. Example: For a binary string representation, the genotype might be a sequence of 0s and 1s: [1, 0, 1, 1, 0].

what could be built/developed based on the genotype

The phenotype is **what is evaluated by the fitness function** to determine the quality of an individual's solution to the problem. Example: If the genotype represents a binary string encoding, the decoded phenotype might be a corresponding integer or a real-valued number: 22.

Differences between Phenotype and Genotype





MOST COMMON REPRESENTATIONS OF GENOMES

Binary Representation:

strings of binary digits (0s and 1s).

Real-Valued Representation:

Genomes are represented as **vectors of real numbers**.
Suitable for optimization problems with continuous variables, like mathematical functions or parameter optimization.

Integer Representation:

Genomes are represented as **vectors of integers.**
Appropriate for problems where solutions must be whole numbers, such as certain combinatorial optimization tasks.

Permutation Representation:

Genomes are represented as permutations of a set of values.
Suitable for problems involving the ordering of elements, like the traveling salesman problem.

Tree-Based Representation:

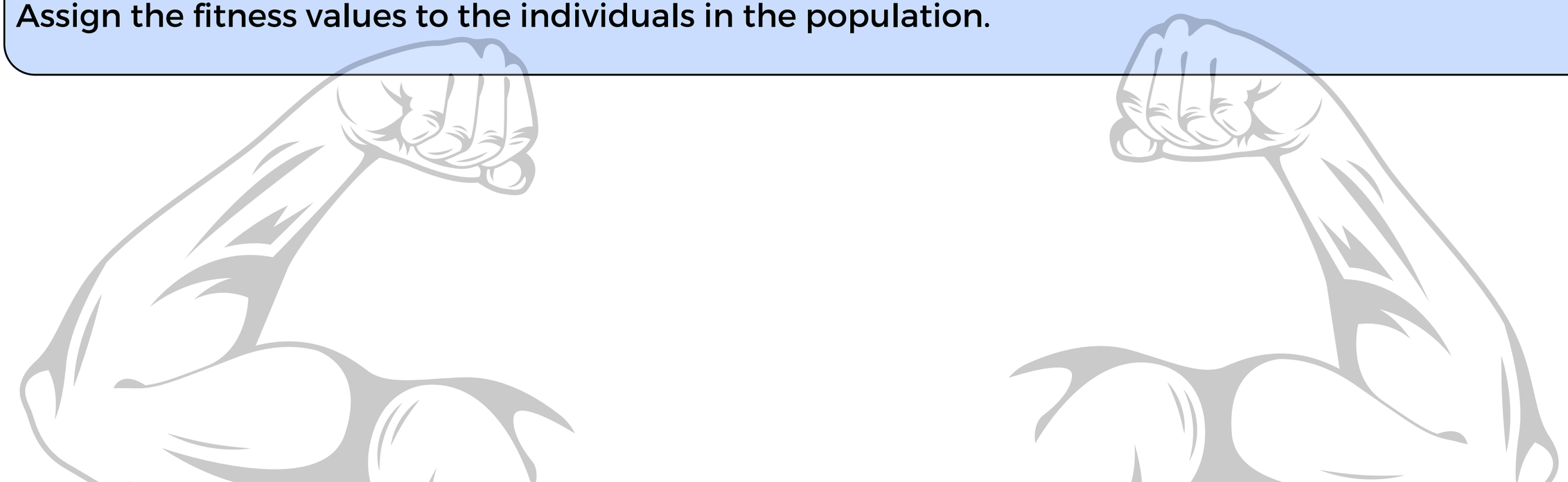
Genomes are represented as hierarchical tree structures.
Common in Genetic Programming (GP) and Genetic Algorithms (GA) for problems where the solution structure is essential.

SELECTION AND FITNESS FUNCTIONS

Fitness Function:

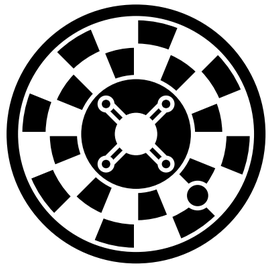
The fitness function evaluates the quality or performance of an individual, It assigns a numerical value indicating how well an individual satisfies the problem's objectives(no universal formula, depends on problem).

For each individual in the population, apply the fitness function to obtain a numerical fitness value.
Assign the fitness values to the individuals in the population.



SELECTION

the process of choosing individuals from the current population to act as parents for the creation of the next generation.

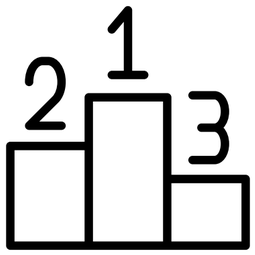


Roulette Wheel Selection:

Individuals are selected with a probability proportional to their fitness.
Fitness values are normalised to create a probability distribution.

Tournament Selection:

Randomly select a subset of individuals (a tournament) and choose the fittest individual from the subset.
Repeated until a sufficient number of parents are chosen.



Rank-Based Selection:

Individuals are ranked based on their fitness.
Selection is then based on the rank rather than absolute fitness.

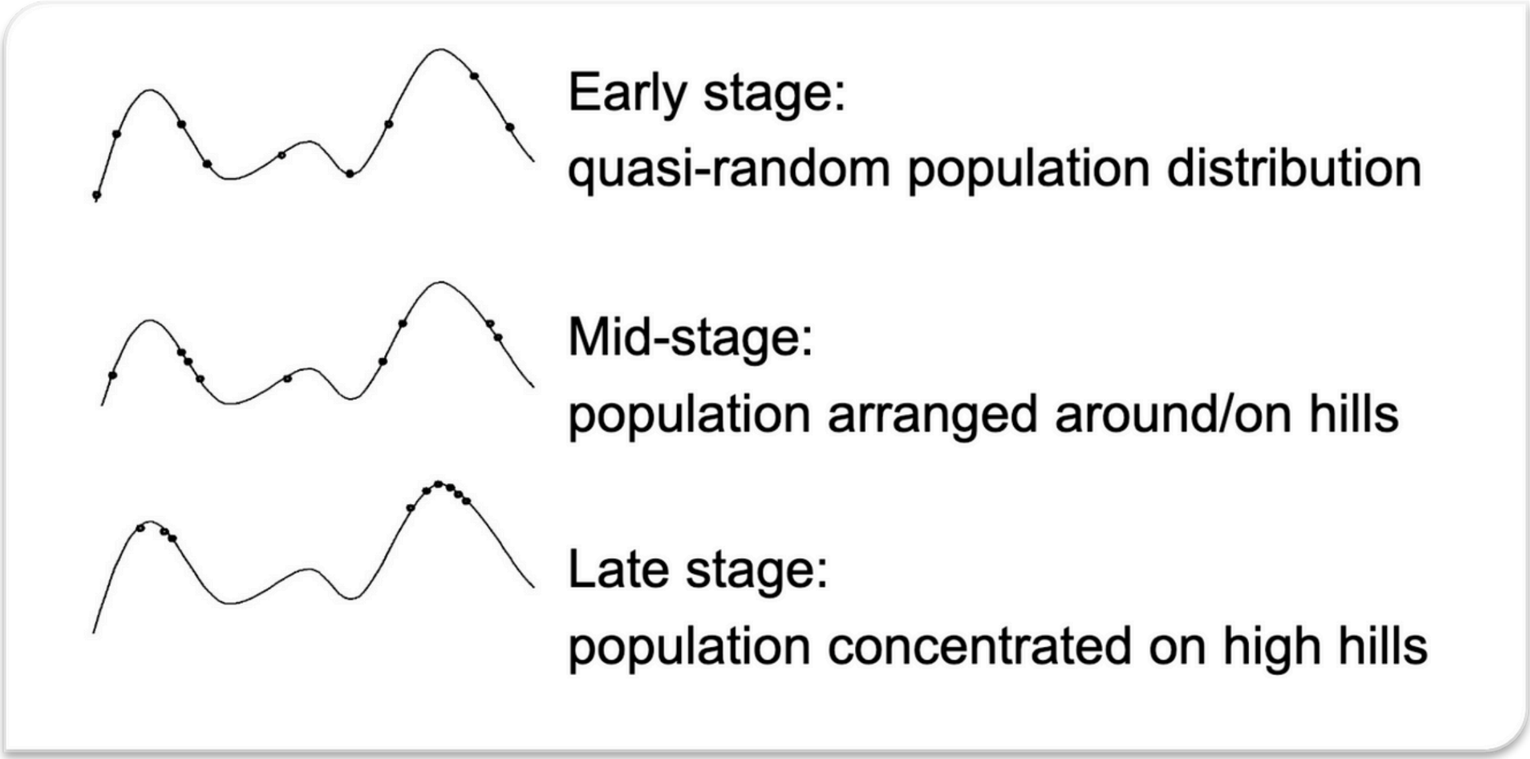
EVOLVING POPULATIONS

Diversity:

The population embodies a diverse array of potential solutions, fostering exploration across the solution space.

Variation - Unleashing Diversity:

Genetic operations inject diversity into the population, promoting the exploration of different solution pathways.



VARIATION OPERATIONS

Main rule

Arity - number of inputs

Arity = 1 is mutation

Arity > 1 is recombination

Crossover (Recombination):

Single-Point Crossover
Multi-Point Crossover
Uniform Crossover
Edge recombination
Partially Mapped Crossover (PMX)
Order Crossover (OX)

Mutation:

Bit Flip Mutation
Random Resetting
Gaussian Mutation
Swap Mutation
Inversion

MUTATION

introduces small random changes to the genetic material of individuals in the population

Mutation helps maintain diversity in the population and can lead to the exploration of new regions in the solution space

MUTATION FOR BINARY REPRESENTATIONS

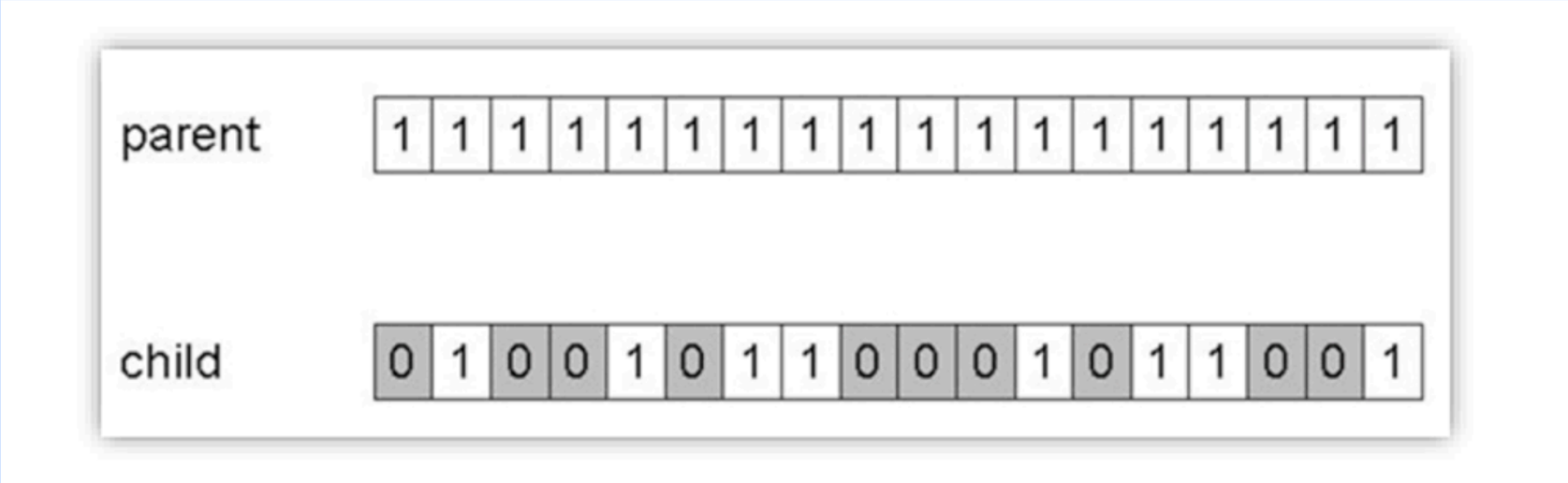
Bit-flip with probability p for each bit

Select Random Bits:

Randomly select a set of bits in the binary representation.

Flip the Selected Bits:

Flip (invert) the values of the selected bits.



INTEGER MUTATIONS

Creep Mutation:

small incremental changes to the values of randomly selected genes (integer variables) in the chromosome.

Increment (or decrement) the values of the selected genes by a small amount.

Example:

Original Chromosome: [4, 7, 12, 9, 5]
Randomly Selected: * *

Creep Mutated Chromosome: [4, 6, 12, 10, 5]

Random Resetting Mutation:

introduces more drastic changes compared to Creep Mutation.
Replaces the values of the selected genes with new random values.

Example:

Original Chromosome: [4, 7, 12, 9, 5]
Randomly Selected: * *

Random Resetting Chromosome: [4, 2, 12, 7, 5]

UNIFORM MUTATION FOR REAL VALUED AND FLOAT

draw random value from interval [min, max]

independently changing the value of each gene with a certain probability. It introduces uniform random variations to each gene.

Exaample:

Original Chromosome: [2.5, -1.8, 0.7, 3.2, -4.5]
Mutation Probability: 0.3
Mutation Range: [-1.0, 1.0]

Mutated Chromosome: [2.5, -0.5, 1.2, 3.2, -3.7]

NON-UNIFORM MUTATION FOR REAL VALUED AND FLOAT

adjusts the magnitude of mutation based on the generation number or other parameters. It allows the mutation to decrease over time, potentially converging to a more exploitative strategy.

Example:

Original Chromosome: [2.5, -1.8, 0.7, 3.2, -4.5]
Mutation Probability: 0.3
Initial Mutation Magnitude: 1.0
Generation Number: 5

Mutated Chromosome: [2.5, -1.6, 0.8, 3.0, -4.3]

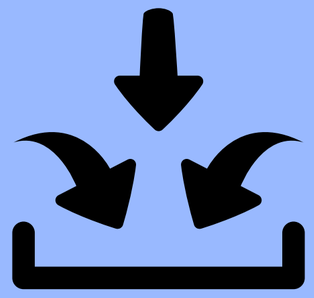
Add value drawn from Gaussian distribution parent value
$Xi\_child = Xi\_parent + N(0, \sigma)$

MUTATIONS FOR PERMUTATIONS

Swap Mutation:
Randomly selects two positions in the permutation and swaps the elements at those positions.

Example:

Original: [1, 2, 3, 4, 5]
After Swap Mutation: [1, 5, 3, 4, 2]



Insert Mutation:
Randomly selects a position in the permutation and moves the element at that position to another randomly chosen position.

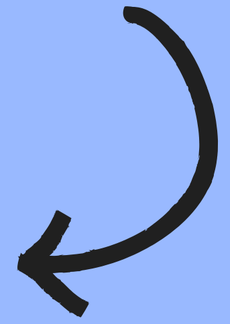
Example:

Original: [1, 2, 3, 4, 5]
After Insert Mutation: [1, 4, 2, 3, 5]

Scramble Mutation:
Description: Randomly selects a segment of the permutation and shuffles the elements within that segment.

Example

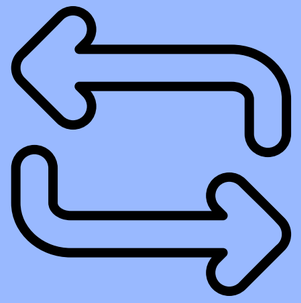
Original: [1, 2, 3, 4, 5]
After Scramble Mutation: [1, 3, 2, 4, 5]

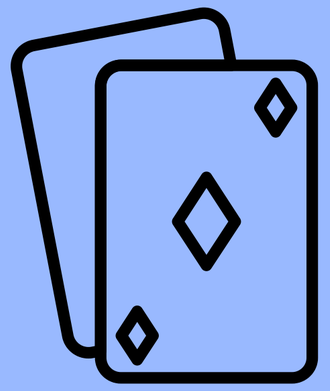


Inversion Mutation:
Inverts the order of a segment of the permutation.

Example:

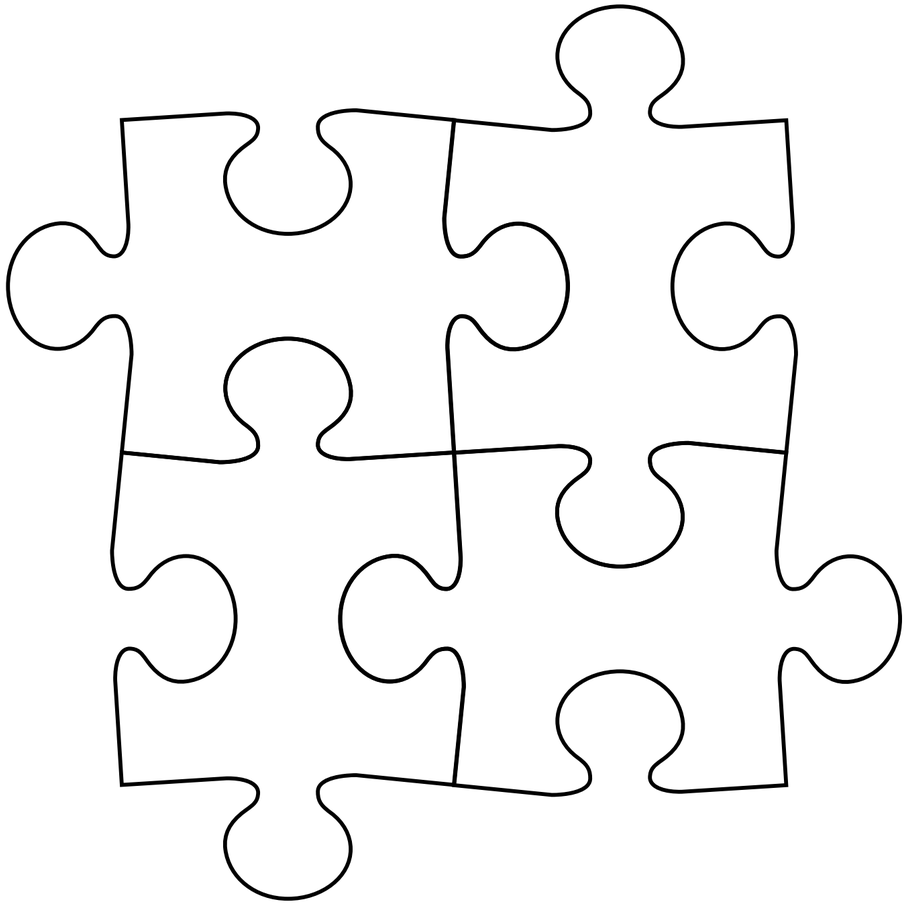
Original: [1, 2, 3, 4, 5]
After Inversion Mutation: [1, 4, 3, 2, 5]





RECOMBINATION (CROSSOVER)

It simulates the process of genetic recombination or mating in biological evolution by combining genetic material from two parent individuals to produce one or more offspring.



CROSSOVERS FOR BINARY

Single-Point Crossover:
a random crossover point is chosen, and the genetic material beyond that point is swapped between the two parents.

Parent 1: 01011011
Parent 2: 11001010
Crossover Point: ↑

Offspring 1: 0101|1010
Offspring 2: 1100|1011

Uniform Crossover:
treats each bit independently, and for each bit, there's a probability of it being inherited from Parent 1 or Parent 2.

Parent 1: 01011011
Parent 2: 11001010

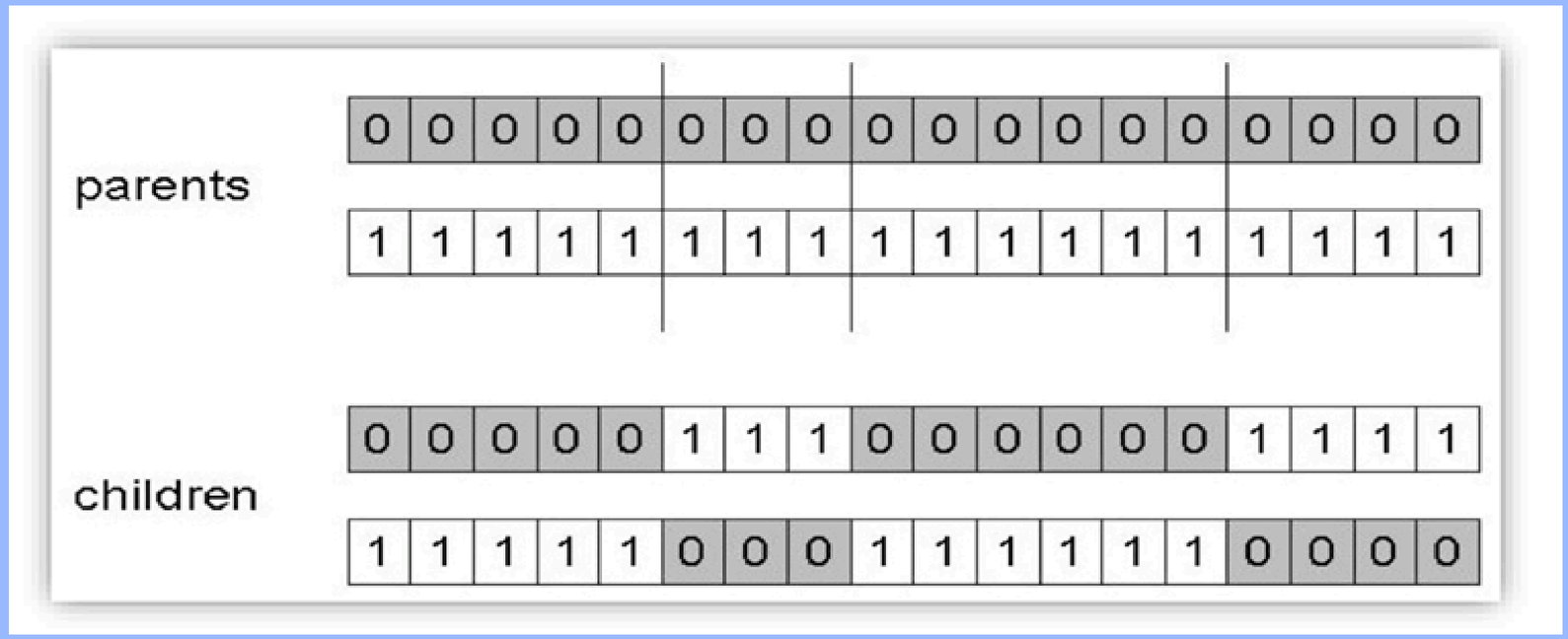
Mask: 11011001

Offspring 1: 1101|1001
Offspring 2: 0100|0100
In this example, the mask indicates which bits are inherited from Parent 1 (bit=1) or Parent 2 (bit=0).

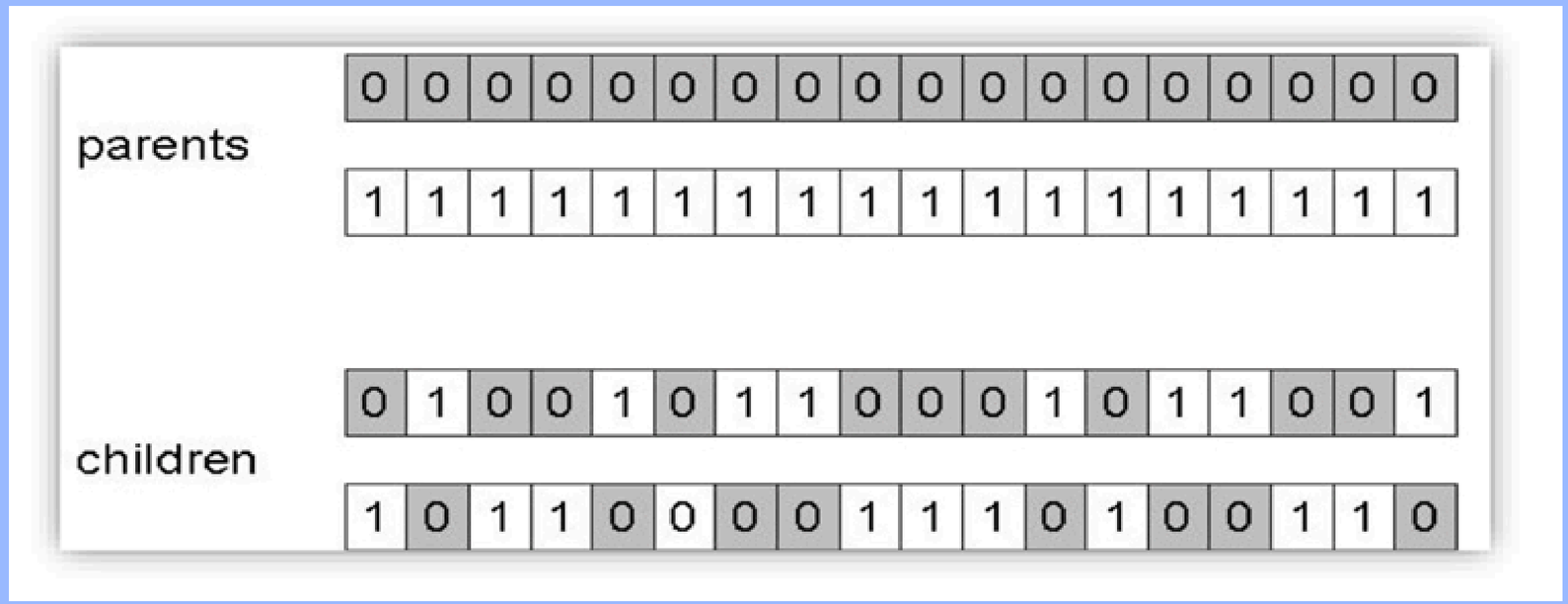
Multi-Point Crossover(n-point crossover):
Multi-Point Crossover involves selecting multiple crossover points, and the genetic material between these points is swapped between the parents.

Parent 1: 01011011
Parent 2: 11001010
Crossover Points: ↑ ↑

Offspring 1: 0101|0010
Offspring 2: 1100|1101



N-point
Choose n-points
split parents at these points



Uniform
For each index, choose parent at random
Other child gets Allele from the other parent

CROSSOVER FOR INTEGERS

N-Point Crossover:

Select N Random Crossover Points:

Choose N random positions along the chromosomes of both parents.
Alternate Crossover Segments

Example:

Parent 1: [1, 2, 3, 4, 5, 6, 7, 8]
Parent 2: [8, 7, 6, 5, 4, 3, 2, 1]
Crossover Points: ↑ ↑ ↑

Offspring 1: [1, 2, 3, 5, 4, 3, 2, 1]
Offspring 2: [8, 7, 6, 4, 5, 6, 7, 8]

.

Uniform Crossover:

Create a binary mask of the same length as the chromosome. Each bit represents whether the corresponding gene is inherited from Parent 1 (0) or Parent 2 (1).

Apply the binary mask to both parents to create offspring.

Example:

Parent 1: [1, 2, 3, 4, 5, 6, 7, 8]
Parent 2: [8, 7, 6, 5, 4, 3, 2, 1]
Binary Mask: [0, 1, 0, 1, 0, 1, 0, 1]

Offspring 1: [1, 7, 3, 5, 4, 3, 2, 1]
Offspring 2: [8, 2, 6, 4, 5, 6, 7, 8]

In this example, a binary mask is created, and each gene in the offspring is selected from either Parent 1 or Parent 2 based on the mask

CROSSOVER FOR REAL VALUED REPRESENTATIONS

Discrete Recombination:

(N-point crossover)
For Each Gene in the Chromosome:
Independently choose one of the parent genes with a probability of 0.5.
Create an offspring by selecting genes based on the choices made for each gene.

Example:

Parent 1: [2.5, -1.8, 0.7, 3.2, -4.5]
Parent 2: [1.0, 3.0, 2.0, 4.0, -2.0]

Offspring: [2.5, 3.0, 0.7, 4.0, -2.0]

In this example, each gene in the offspring is independently chosen from either Parent 1 or Parent 2 with equal probability.

Intermediate Recombination:
involves taking the weighted average of the corresponding genes from the parents to create the offspring. The weight of each parent's gene is determined by a mixing ratio (alpha), where alpha is a value between 0 and 1.

Calculate the weighted average of the corresponding genes from both parents using the mixing ratio (alpha). Use the weighted averages to create the offspring.

Example:

Parent 1: [2.5, -1.8, 0.7, 3.2, -4.5]
Parent 2: [1.0, 3.0, 2.0, 4.0, -2.0]
Mixing Ratio (alpha): 0.7

Offspring: [1.85, 0.6, 1.65, 3.52, -3.25]

$X_{child\_1} = \alpha X_{parent\_1} + (1 - \alpha) X_{parent\_2}$
$X_{child\_2} = (1 - \alpha) X_{parent\_2} + \alpha X_{parent\_1}$
α Average of parents, if $\alpha = 0.5$

CROSSOVER FOR PERMUTATIONS - PMX

Partially Mapped Crossover (PMX):
It creates offspring by **preserving the relative order of selected elements** from both parents while filling in the remaining positions based on the information from the parents.

- Two random crossover points are chosen.
- The segment between the crossover points is copied from one parent to the corresponding positions in the offspring.
- The remaining positions are filled by mapping the values from the other parent, ensuring no duplication.

Example:

Parent 1: 1 2 3 | 4 5 6 7 8 9
Parent 2: 5 7 2 | 1 8 3 6 4 9
Crossover Points: ↑ ↑

Offspring 1: 5 7 3 | 4 1 2 6 8 9
Offspring 2: 1 2 7 | 5 8 3 6 4 9

PMX is **designed to maintain the relative order of the selected elements**, making it particularly useful for problems where the order of elements is crucial.

EDGE RECOMBINATION CROSSOVER

is often applied to permutation problems such as the Traveling Salesman Problem (TSP). It creates offspring by combining the edges of both parents, attempting to preserve as many edges as possible.

- Create a list of edges for each parent based on the order of elements in the permutation.
- Start with an initial city (commonly chosen randomly).
- Select the next city based on the common edge with the current city in either parent.
- Repeat until all cities are included in the offspring.

Example:

Parent 1: 1 2 3 4 5
Parent 2: 2 4 5 1 3

Offspring: 1 2 4 5 3

Edge Recombination emphasizes preserving edges present in both parents, leading to diverse and potentially high-quality solutions in TSP-like problems.

PERMUTATION REPRESENTATIONS: CONSERVING ORDER

Order Crossover (OX):
creates offspring by preserving a subset of genetic material from the parent individuals while filling in the remaining positions based on the information from the parents.

- Two random crossover points are chosen.
- The segment between the crossover points is copied from one parent to the corresponding positions in the offspring.
- The remaining positions are filled by copying genetic material from the other parent, excluding elements already present in the copied segment.

Parent 1: 1 2 3 | 4 5 6 7 8 9
Parent 2: 9 3 7 | 8 2 6 5 1 4
Crossover Points: ↑ ↑

Offspring 1: 9 3 2 | 4 5 6 7 8 1
Offspring 2: 1 2 7 | 8 9 3 6 5 4

Order Crossover aims to maintain the relative order of elements while introducing diversity from both parents.

Cycle Crossover:
creates offspring by identifying cycles of genetic material between the parent individuals.

- Identify cycles of genetic material between the parents using alternating positions.
- Copy genetic material from one parent for odd cycles and from the other parent for even cycles.
- Fill in the remaining positions by copying genetic material from the other parent.

Parent 1: 1 2 3 4 5
Parent 2: 2 4 5 1 3

Cycles: (1 2 4) (3 5)

Offspring: 1 2 4 5 3

Cycle Crossover tends to preserve blocks of genetic material from both parents and is known for producing diverse offspring.

Brief repetition of previous week

	Mutation	Recombination
Binary	Bit flip	Single point, uniform, n-point
Integers	Creep mutation Random resetting	n-point, uniform
Real valued/float	Uniform non Uniform	Discrete Intermediate
Permutations	Swap, insert, scrumble inversion	PMX, Edge recombination, order crossover, cycle crossover

Genotype - set of genes or values
Phenotype - what could be developed based on the genotyoe

Locus: the position of a gene
Allele = 0 or 1 (What values a gene can have)
Gene: one element of the array
Genotype: a set of gene
valuesPhenotype: What could be built/developed based on the genotype

PARENT SELECTION

Fitness-Proportionate Selection (FPS)

• Probability for individual i to be selected for mating in a population size μ with FPS is

$$P_{FPS}(i) = f_i \Big/ \sum_{j=1}^{\mu} f_j$$

+

- faster convergence
- Intuitive
- Gives chance to weak individuals

−

- Exploitation > Exploration
- Premature convergence
- Loss of selection pressure#
- Loss of diversity

Tournament Selection

Idea for a procedure using only local fitness information:

• Pick k members at random then select the best of these

• Repeat to select more individuals

Probability of selecting i will depend on:

• Rank of i

• Size of sample k

• higher k increases selection pressure

- Picking without replacement increases selection pressure
- • Whether fittest contestant always wins (deterministic) or this happens with probability p

+

- Maintains Diversity
- easy-to-implement
- randomness can help avoid premature convergence and explore a diverse set of solutions

−

- Biased Towards Better Individuals
- Less Pressure for Diversity
- Biased Towards Better Individuals
- Lack of Global Information

SURVIVOR SELECTION

The process determines which individuals from the combined set of parents and offspring will make up the next generation.

ELITISM

Elitism involves preserving a certain number of the best individuals (based on fitness) from the current generation and allowing them to directly pass on to the next generation without undergoing genetic operations. Elitism helps maintain high-performing solutions across generations, contributing to convergence.

PRESERVE DIVERSITY

- **Explicit approaches**
- Make similar individuals compete for resources (fitness)
- Make similar individuals compete with each other for survival

Implicit approaches:
- Impose an equivalent of geographical separation
- Impose an equivalent of speciation

Fitness Sharing

- Restricts the number of individuals within a given niche by “sharing” their fitness
- Need to set the size of the niche σshare in either genotype or phenotype space
- run EA as normal but after each generation set

$$f'(i) = \frac{f(i)}{\sum_{j=1}^{\mu} sh(d(i,j))} \qquad sh(d) = \begin{cases} 1 - d/\sigma & d \le \sigma \\ 0 & otherwise \end{cases}$$

Crowding

New individuals replace similar individuals
- Randomly shuffle and pair parents, produce 2 offspring
- Each offspring competes with their nearest parent for survival (using a distance measure)
- Result: Even distribution among niches

Automatic Speciation

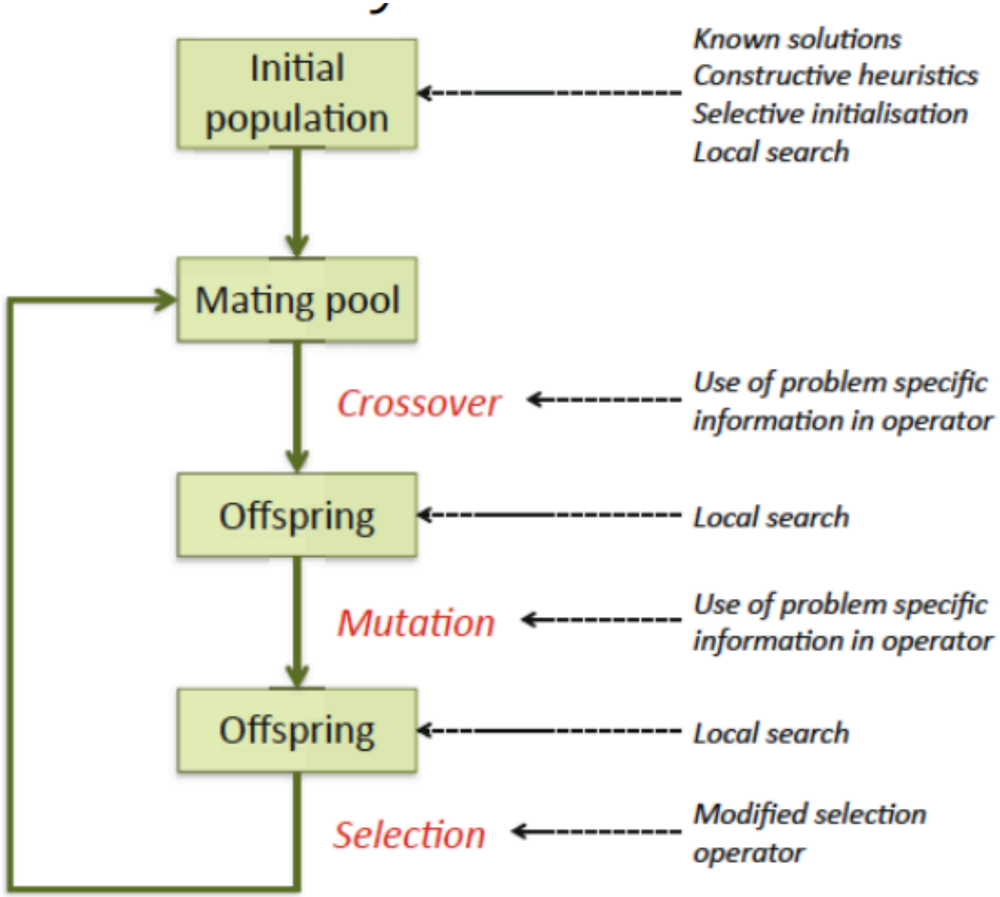
Either only mate with genotypically / phenotypically similar members or
- Add species-tags to genotype
- initially randomly set
- when selecting partner for recombination, only pick members with a good match

“Island” Model

Periodic migration of individual solutions between populations
- Run multiple populations in parallel
- After a (usually fixed) number of generations (an Epoch), exchange individuals with neighbours
- Repeat until ending criteria met

HYBRIDISATION

integration of different optimization techniques, algorithms, or problem-solving approaches within an evolutionary algorithm



LOCAL SEARCH

Local search is a optimization technique employed to fine-tune or improve individual solutions in the **neighborhood of the current candidate**. Unlike global search operators such as mutation and crossover that explore the entire solution space, **local search focuses on exploiting the local structure around a given solution**.

it is often applied as a post-processing step to individuals in the population, aiming to refine or optimize them further. It is particularly useful when the initial individuals generated by the evolutionary operators are close to optimal solutions but may still have room for improvement.

LAMARKIAN AND BALDWINIAN

Lamarckian Evolution:

Lamarck proposed the theory of the inheritance of acquired traits, suggesting that individuals could acquire new traits during their lifetime and pass them on to their offspring. In the context of evolutionary algorithms, Lamarckian evolution implies that changes made to an individual during its lifetime can be directly passed on to its descendants.

Lamarckian Evolution in EA:

allowing individuals to adapt and improve their solutions during their lifetime, and **these improvements would be directly reflected in the genetic material passed to the next generation.**

Baldwinian Evolution:

learned behaviors or adaptations acquired during an individual's lifetime could influence the direction of evolution. However, unlike Lamarck, Baldwin **did not propose direct inheritance of acquired traits**. Instead, he suggested that **learning could guide the exploration of the solution space and affect the evolutionary trajectory.**

Baldwinian Evolution in EA:

Baldwinian approach involves using learning or adaptation mechanisms to guide the evolutionary process. Individuals may learn during their lifetime, and this acquired knowledge influences their chances of survival and reproduction. However, the **acquired knowledge itself is not directly passed on to offspring**.

Pareto front and pareto optimality

Pareto Front:

The Pareto front is a set of solutions in multiobjective optimization where no solution is better than another in all objectives.
It represents the trade-offs between conflicting objectives, showcasing solutions that achieve the best compromise.

Pareto Optimality:

A solution is Pareto optimal if it belongs to the Pareto front; there is no other solution that dominates it.
Dominance implies being better than another solution in at least one objective without being worse in any other.

Finds the set of non-dominated solutions

Solution x dominates solution y, ($x \leqslant y$), if:
x is better than y in at least one objective,
x is not worse than y in all other objectives

3 EA Representations - efficiency

How can the choice of representation type for solutions make the search in an Evolutionary Algorithm more or less efficient? Give an example, with a specific optimization problem and two different solution representations where one is likely to give a more efficient search than the other.

Fill in your answer here

Maximum marks: 4

Suggested Solution

The representation allows us to encode some knowledge about the problem we are trying to solve and how to efficiently move from one solution to the next. We can also be able to choose a representation that only allows certain kinds of solution, allowing us to avoid searching for those solutions that are not admitted by the representation. (2p)

Example (2p):

Any representation that is likely to be more efficient than another could award points here. The most straightforward example is to use a permutation-type problem, like a travelling salesperson problem. A permutation representation here allows us to avoid searching for solutions that are invalid (because they are not permutations) thus saving time. A less efficient representation would be for instance an integer-style representation, that could represent solutions such as [1 1 2 1] which are not useful for the TSP search.

WHAT SUPERVISED LEARNING IS ALL ABOUT

Supervised learning is a fundamental approach in machine learning where algorithms learn by example, just like a student learns from a teacher. It involves feeding the algorithm **labeled data sets**, where **each data point has both an input (features) and a desired output (label)**. This allows the algorithm to learn the relationship between the inputs and outputs, enabling it to make predictions on new, unseen data.

MAIN COMPONENTS OF SUPERVISED MACHINE LEARNING

1 . Labeled Data

2. Features:
Characteristics or attributes of the data used by the algorithm to learn and make predictions.

3. Learning Algorithm:

Classification:
Categorizes data points (spam/not spam, cat/dog). Examples: Decision trees, support vector machines (SVMs), k-nearest neighbors (kNN).

Regression: Predicts continuous values (housing prices, weather forecasts). Examples: Linear regression, polynomial regression, random forests.

4. Model Training(Train):
iteratively adjusting the algorithm's internal parameters based on the labeled data to minimize the prediction error.

5. Model Evaluation(Dev):
Assessing the trained model's performance on unseen data to measure its ability to generalize to new situations.

6. Prediction(Test):
Using the trained model to make predictions on new, unseen data points based on their input features.

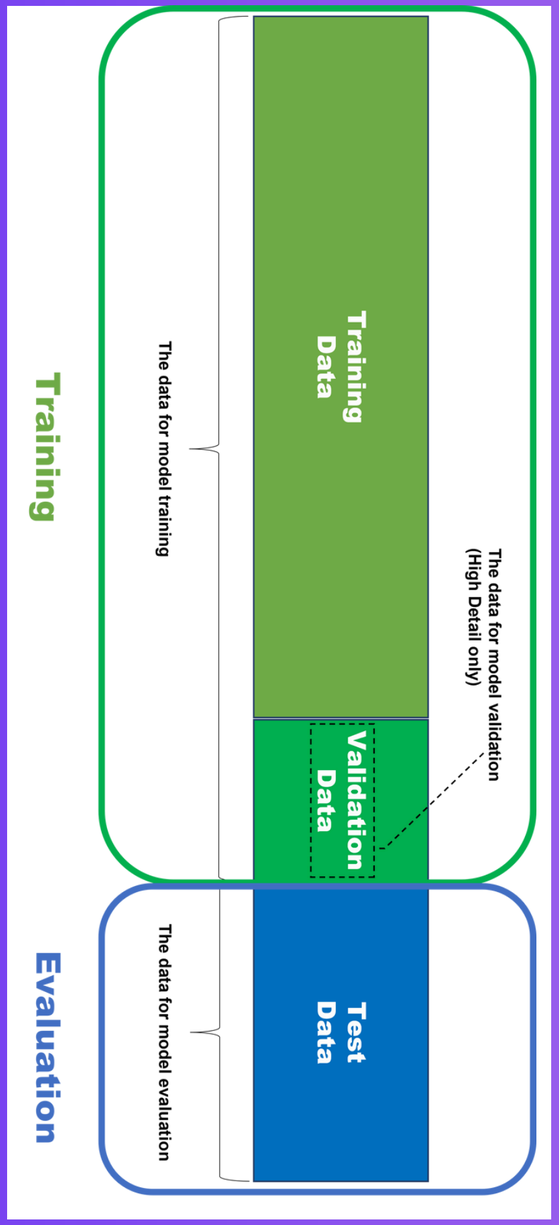
TRAIN DEV TEST

1. Train Set:
Largest portion of data (usually 70-80%), used to train the model and learn its parameters.

2. Development/Validation Set (Dev Set):
A smaller subset (10-20%) of data used to fine-tune the model hyperparameters (e.g., learning rate, number of layers). Used it to identify potential overfitting or underfitting issues

3. Test Set:
The smallest subset (10-20%) of your data held out completely unseen by the model during training and development. Performance on unseen data, simulating real-world scenarios.

Why are they necessary?
Train-Dev split: Prevents overfitting, where the model memorizes the training data and performs poorly on unseen data.
Test Set: Provides an unbiased estimate of the model's true performance on completely new data.



REGRESSION VS CLASSIFICATION

Feature	Regression	Classification
Output type	Continuous value	Discrete categories
Examples of tasks	Predicting prices, temperatures, population growth	Classifying emails, handwritten digits, images
Model goal	Predict a specific value	Assign a data point to a class

CONFUSION MATRIX AND IT'S FORMULAS

Predicted Class

Actual Class	**Positive**	**Negative**	
Positive	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP + FN)}$
Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN + FP)}$
	Precision $\frac{TP}{(TP + FP)}$	**Negative Predictive Value** $\frac{TN}{(TN + FN)}$	**Accuracy** $\frac{TP + TN}{(TP + TN + FP + FN)}$

Accuracy = (True Positives + True Negatives) / (Total Instances)	**Accuracy: Ratio of correctly classified instances to the total number of instances.**
Precision = TP / (TP + FP)	Precision: Ratio of correctly predicted positive instances to all predicted positive instances.
Recall = TP / (TP + FN)	Recall (Sensitivity): Ratio of correctly predicted positive instances to all actual positive instances.
Specificity = TN / (TN + FP)	Specificity: Ratio of correctly predicted negative instances to all actual negative instances.
F1 = 2 * (Precision * Recall) / (Precision + Recall)	F1-score: Harmonic mean of precision and recall, combining both metrics into a single score.

MACRO VS MICRO AVERAGE

Macro-averaging calculates each class's performance metric (e.g., precision, recall) and then takes the arithmetic mean across all classes. So, the macro-average gives equal weight to each class, regardless of the number of instances.

$$\text{Macro Average (Metric)} = (1 / N) * \Sigma (\text{Metric}_i)$$

Micro-averaging, on the other hand, aggregates the counts of true positives, false positives, and false negatives across all classes and then calculates the performance metric based on the total counts. So, the micro-average gives equal weight to each instance, regardless of the class label and the number of cases in the class.

$$\text{Micro Average (Metric)} = (\Sigma TP_i + \Sigma TN_i) / (\Sigma TP_i + \Sigma TN_i + \Sigma FP_i + \Sigma FN_i)$$

KNN ALGORITHM

Step 1: Selecting the optimal value of K

- K represents the number of nearest neighbors that needs to be considered while making prediction.

Step 2: Calculating distance

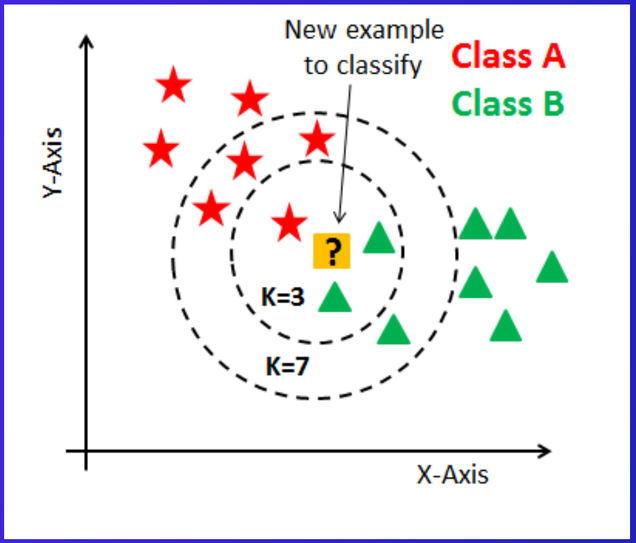
- To measure the similarity between target and training data points, Euclidean distance is used. Distance is calculated between each of the data points in the dataset and target point.

Step 3: Finding Nearest Neighbors

- The k data points with the smallest distances to the target point are the nearest neighbors.

Step 4: Voting for Classification or Taking Average for Regression

- The class with the most occurrences among the neighbors becomes the predicted class for the target data point.
- In the regression problem, the class label is calculated by taking average of the target values of K nearest neighbors.



PROS AND CONS OF KNN

Benefits of KNN:

Simple and easy to understand, Versatile: Can be used for both classification and regression tasks.

No explicit model training: No need to define a complex model, making it efficient for small datasets.

Drawbacks of KNN:

Sensitive to distance metric, High computational cost, Curse of dimensionality

Real-world applications of KNN:

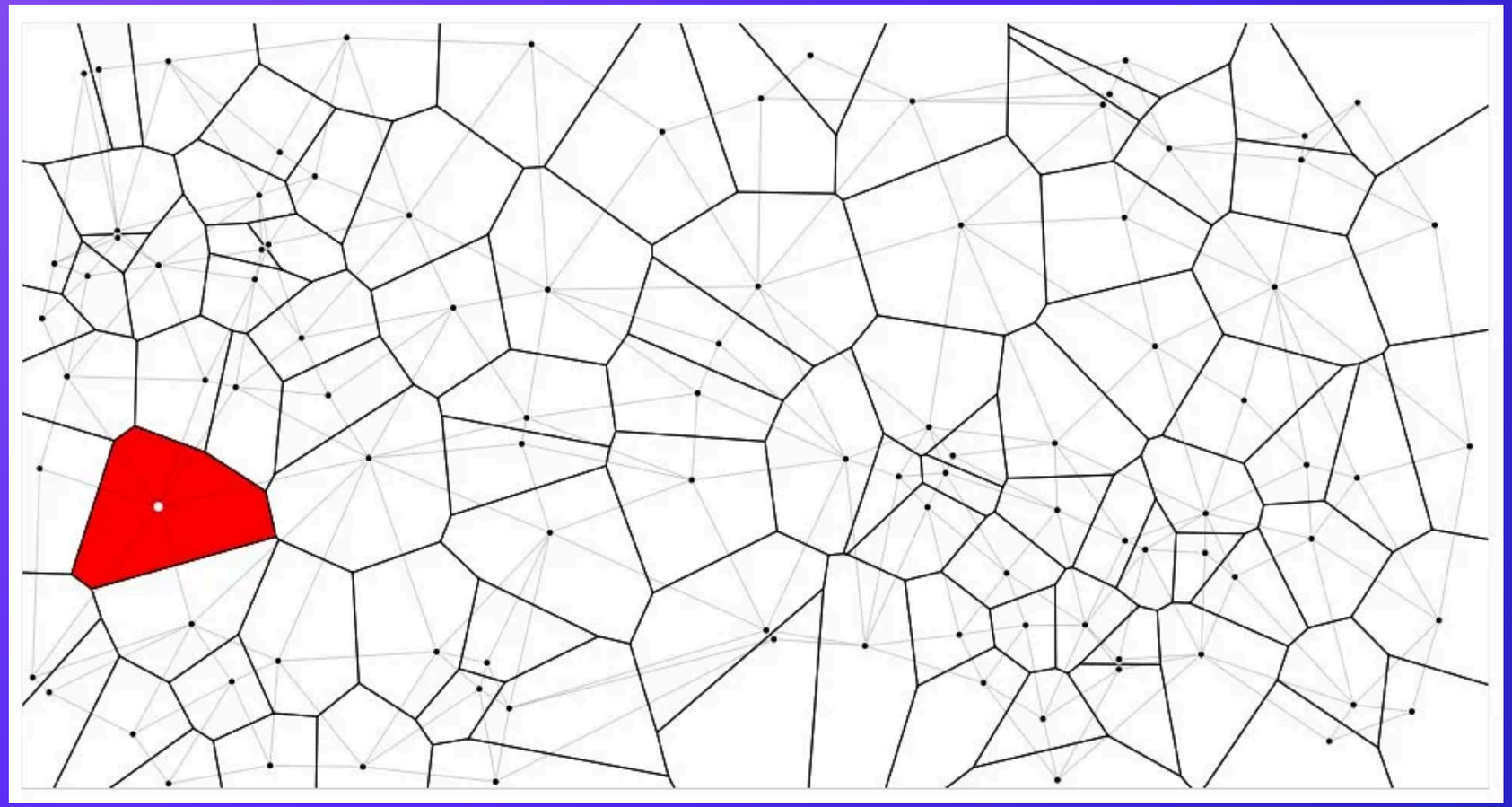
Recommending products based on past purchases (similar users bought these).

Identifying fraudulent transactions based on unusual patterns.

Image recognition by comparing features to labeled images.

VORONOI TESSALATION

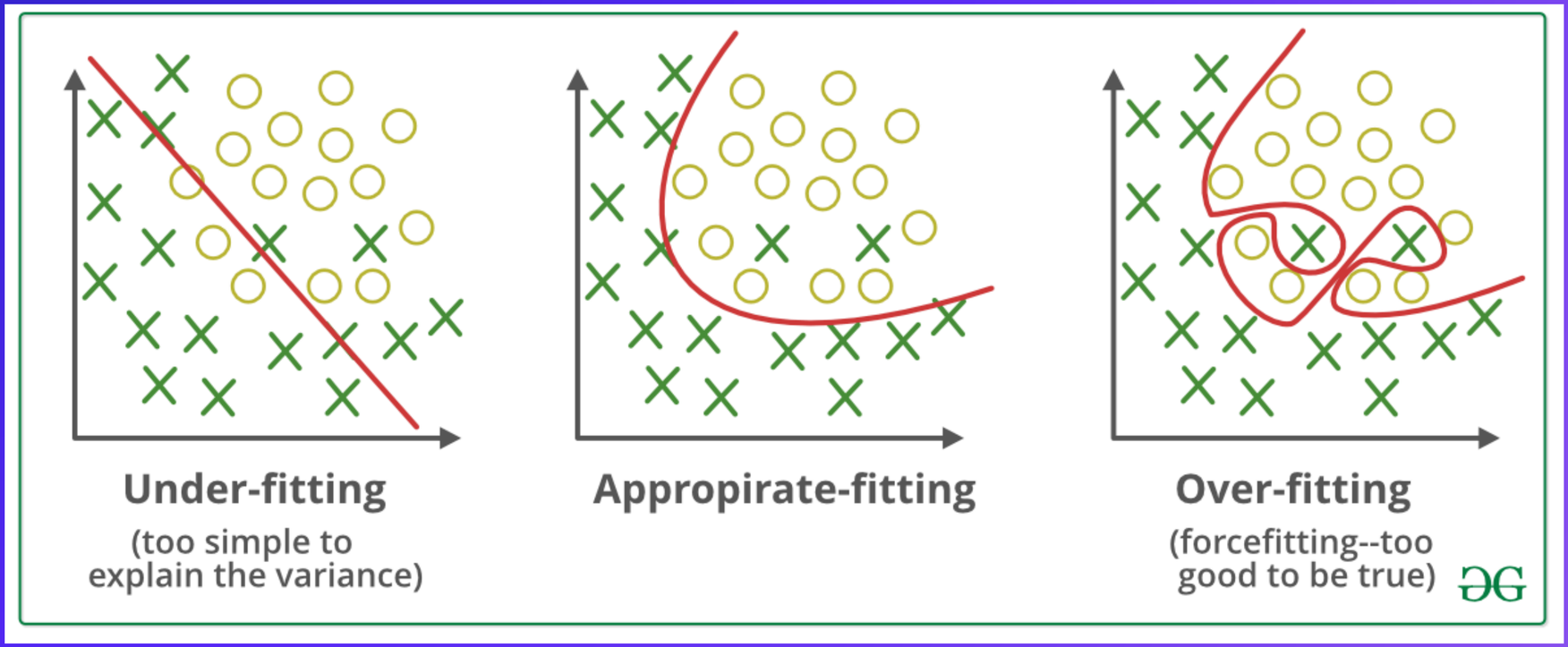
A CASE OF K-NN MACHINE LEARNING ALGORITHM WHEN K=1



https://medium.com/@malhotra.amit.236/love-thy-k-nearest-neighbors-58932fa66a08

OVERFITTING/UNDERFITTING OF KNN

The value of k in the KNN algorithm is related to the error rate of the model. A small value of k could lead to overfitting as well as a big value of k can lead to underfitting. Overfitting imply that the model is well on the training data but has poor performance when new data is coming.



FEATURE SCALING AND NORMALISATION OF VECTORS

Feature Scaling

Min-Max scaling: Scales features between a specified minimum and maximum value (usually 0 and 1).
Standard scaling (Z-score normalization): Subtracts the mean of each feature and then divides by the standard deviation.
Other scaling methods: Logarithmic scaling, power scaling, etc., depending on the data distribution.

Let

- $min_i = \min(\{x_{j,i} \mid j = 1,2,\dots,N\})$
- $max_i = \max(\{x_{j,i} \mid j = 1,2,\dots,N\})$

Define $scale_i(x_{j,i}) = \frac{x_{j,i} - min_i}{max_i - min_i}$

Benefits:

Improves convergence and stability of optimization algorithms.
Prevents features with large magnitudes from dominating the learning process.
Improves interpretability of some machine learning models.

Normalization:

L1 normalization (Manhattan distance): Sums the absolute values of each feature and divides each feature by the sum.
L2 normalization (Euclidean distance): Calculates the square root of the sum of squared values for each feature and divides each feature by the square root.
Benefits:

Improves performance in distance-based algorithms like k-Nearest Neighbors.
Can be useful for data with sparse features.

Find the

- mean $\mu_i = \text{mean}(\{x_{j,i} \mid j = 1,2,\dots,N\})$, and
- standard deviation $\sigma_i = \text{std}(\{x_{j,i} \mid j = 1,2,\dots,N\})$
- of your training set.

Define $scale_i(x_{j,i}) = \frac{x_{j,i} - \mu_i}{\sigma_i}$ for each feature $i = 1, \dots m$

LINEAR CLASSIFICATION

The core of linear classification is a linear equation that defines the boundary between categories. This equation takes the feature values of a data point and combines them with weights and a bias term. If the equation's result is greater than a certain threshold, it's classified as one category; otherwise, it's classified as the other.

$$f(\boldsymbol{x}) = f((x_1, x_2, \dots, x_m)) = w_0 + w_1 x_1 + w_2 x_2 + \dots w_m x_m$$

Strengths of Linear Classification:

Interpretable: The decision boundary is a straight line, making it easy to understand why a data point is classified a certain way.
Efficient: Relatively fast and computationally inexpensive, especially for large datasets.
Works well with high-dimensional data: Can handle many features, provided they are linearly separable.

Weaknesses:

Limited to linearly separable data: Cannot perfectly classify data that cannot be separated by a straight line.
May not capture complex relationships: Assumes a linear relationship between features and the target variable, which may not always hold true.

Real-world applications:

Spam filtering, Sentiment analysis, Image recognition

K-Fold Cross Validation

Let no of observations be 1000

Cross validation

For n-fold cross-validation, we can split the training set into n equally sized bins, S1, S2, …, Sn. We can run n many experiments. In experiment k, we use Sk as test set and the rest as training set. We can take the averages of the n many experiments as evaluation measures. The n algorithms will be partly different since they are trained on slightly different training sets. However, most of the training data are the same between two experiments; hence, the average gives a rough measure of what we may achieve by training on the whole training set. The main advantage of using cross-validation is we get a larger test set, the whole training set. Hence, the numbers are more reliable when we do not have many data.

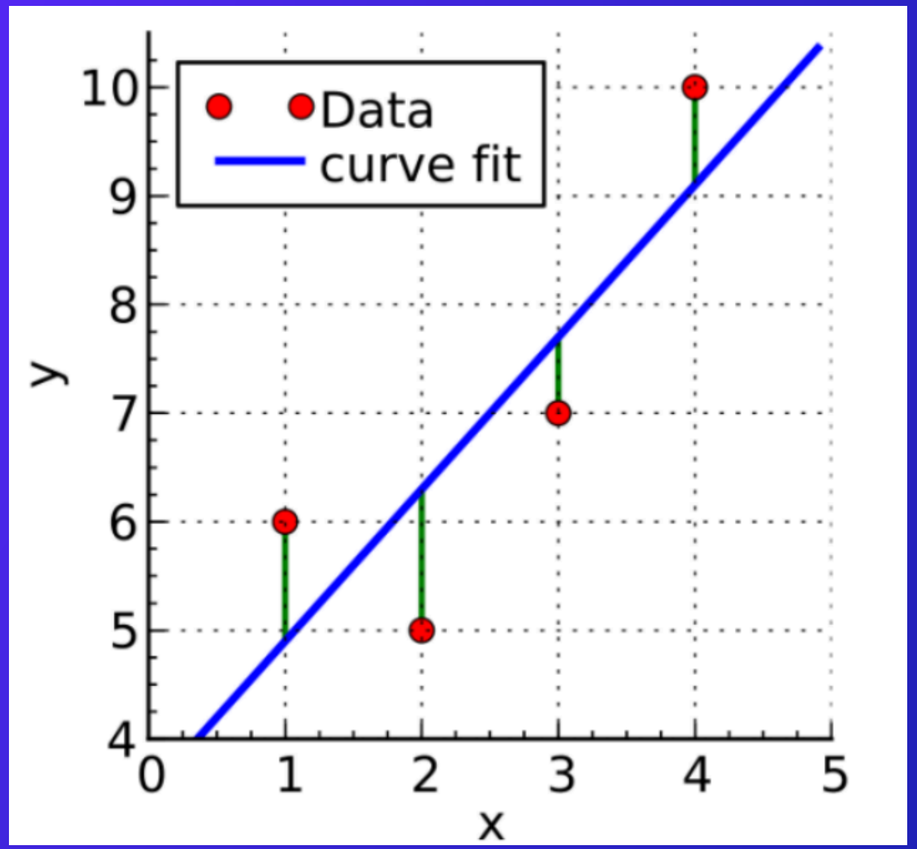
MSE

MSE is the type of loss function used for the linear regression/ classification.

We want to find the weights that minimise the mean square error

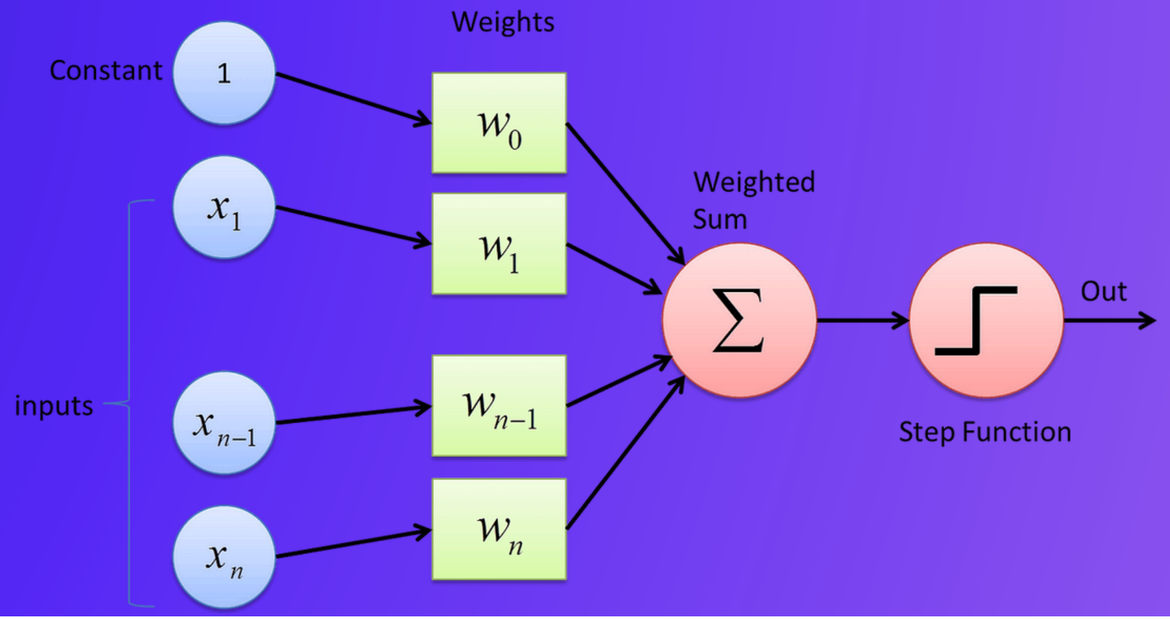
$$\frac{1}{N}\sum_{j=1}^{N}\left(t_j - \sum_{i=0}^{m} w_i x_{j,i}\right)^2$$

$$= \frac{1}{N}\sum_{j=1}^{N}\left(t_j - y_j\right)^2$$



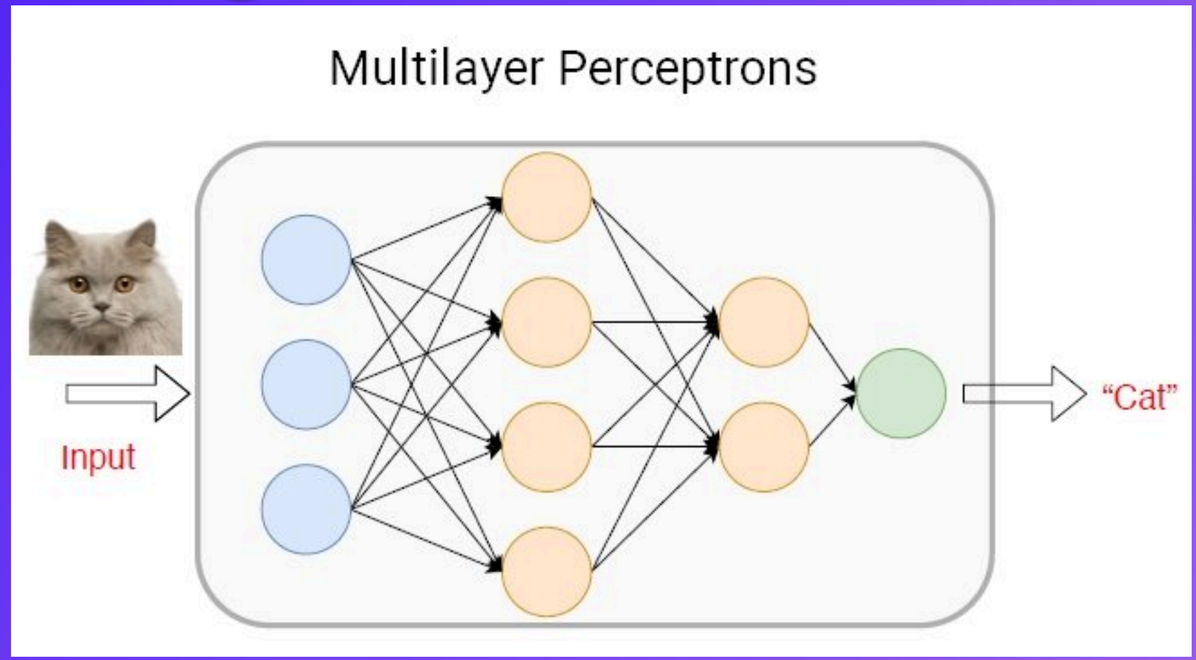
PERCEPTRON

A perceptron is a single-layer neural network that forms the building block of more complex neural networks. It acts as **a basic binary classifie**r, meaning it can only distinguish between two classes.



Structure:

Inputs: Represented by a vector of numerical values.
Weights: Each input has a corresponding weight, determining its influence on the output.
Bias: A constant value added to the weighted sum of inputs.
Activation function: A mathematical function that applies a threshold to the weighted sum and bias, producing the final output (0 or 1).



Functioning:

Each input is multiplied by its corresponding weight.
The weighted sums are added together and the bias is added.
The activation function is applied to the resulting sum.
If the sum is greater than or equal to a threshold (usually 0), the output is 1 (positive class). Otherwise, the output is 0 (negative class).

Learning:

A perceptron can learn by adjusting its weights through a process called the Perceptron Learning Rule. This rule iterates through the training data, adjusting weights to minimize the error between the predicted and desired outputs.

PERCEPTRON UPDATE RULE

- Adjust the threshold
  - Adjust bias term (w_0)

- add a new feature $x_0 = -1$ for all items
- Replace $(x_1, x_2, \ldots, x_m)$ with $(-1, x_1, x_2, \ldots, x_m)$
- Replace $\sum_{i=1}^{m} w_i x_i > \theta$ with $h = \sum_{i=0}^{m} w_i x_i > 0$

- Adjust weights

$$w_i = w_i + \eta(t - y)x_i = w_i - \eta(y - t)x_i$$

(covers all cases)

PERCEPTRON FORWARD VS UPDATE STEP

The forward step calculates the predicted output based on current parameters, while the update step uses the error to adjust those parameters for better future predictions.

The forward step doesn't involve any updating, only calculating a prediction.

7(a) # Perceptron

The perceptron was an early approach to machine learning. It was introduced and became popular around 1960. Describe the architecture of a perceptron and how it can be applied for classification. You do not have to describe how the perceptron is trained.

Fill in your answer here

Maximum marks: 6

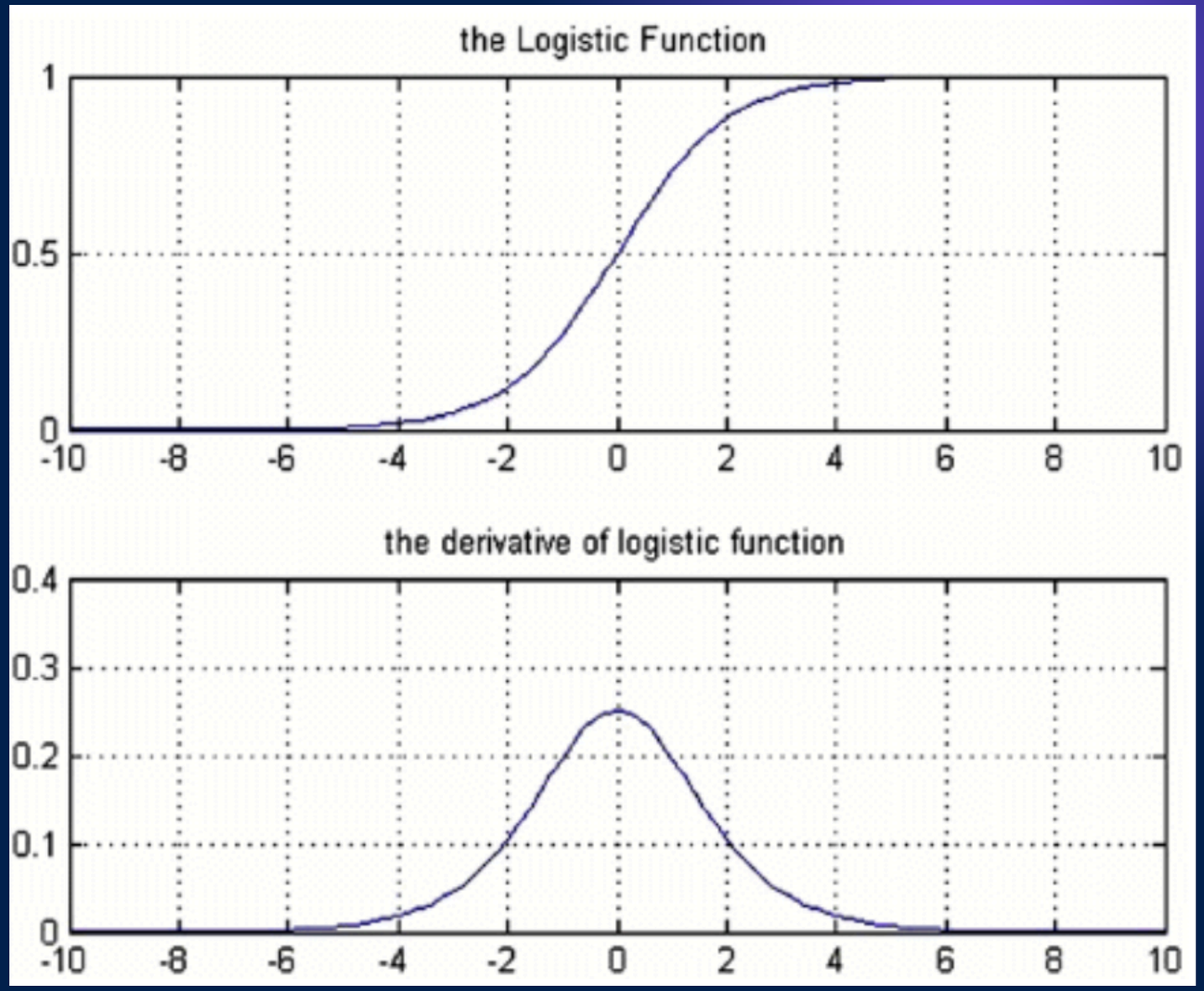
Solution proposals

- The perceptron is an algorithm for binary classification.
- The input should be a numerical vector of a certain length x = (x1, x2, …, xm).
- If the length is *m*, the percepton contains *m* many weights, w1, w2, wm, and a threshold w0.
- If the weighted sum, $s = \sum_{i=1}^{m} wi * xi$ is larger than the threshold w0, x is ascribed to the positive class. If s≤0, x is ascribed to the negative class.

This can also be described as adding one additional feature, x0 = -1 (or x0 = 1) to each input together wit an additional weight w0 and consider whether $\sum_{i=1}^{m} wi * xi > 0$ or not.

The Logistic Function

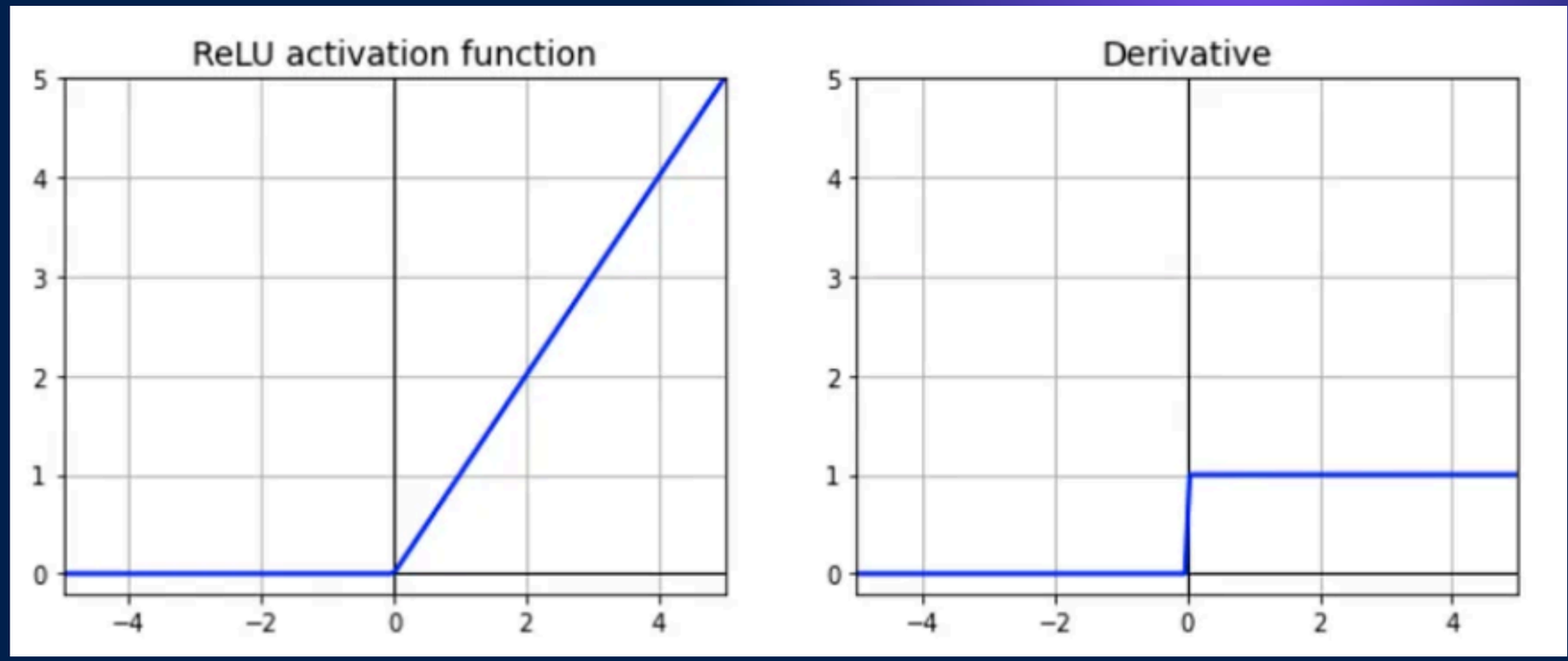
- $y = \sigma(z) = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$
- Differentiable
- $y' = y(1-y)$



Rectified Linear Activation (ReLU)

- returns 0 if the input is negative, but for any positive input, it returns that value back.

$$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$



7(g) # ReLU

Assume that you instead use ReLU (Rectified Linear Unit) as the activation function in the hidden layer. Give a formula for the output from the hidden node h1 given a general input (x1, x2) with ReLU as activation in the hidden layer.

Fill in your answer here

Maximum marks: 2

Solution proposal

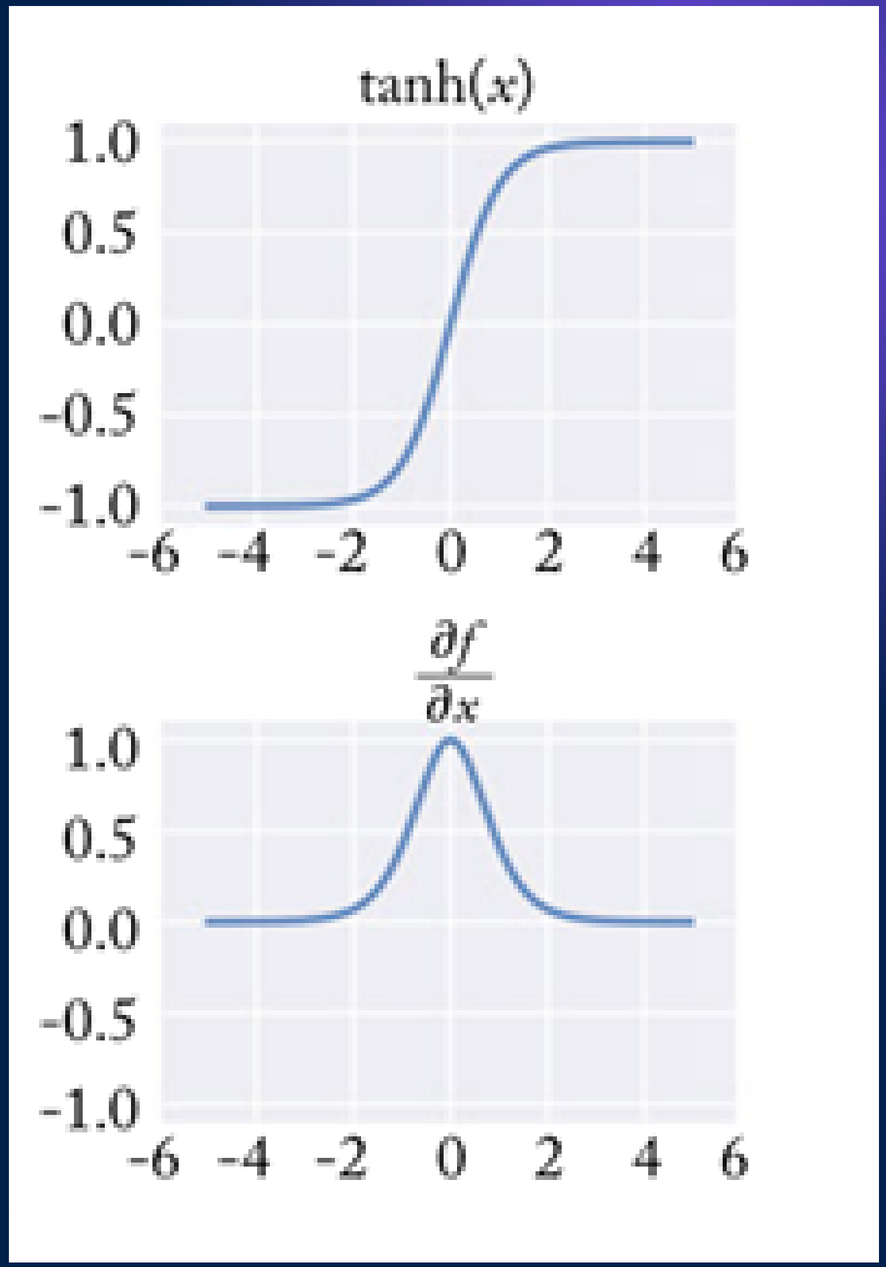
ReLU(h1) = h1 if h1>0 and ReLU(h1) = 0 if h1<0.

Hyperbolic Tangent (Tanh)

Hyperbolic Tangent function

$$\frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\frac{d}{dx}\tanh(x) = 1 - (\tanh(x))^2$$

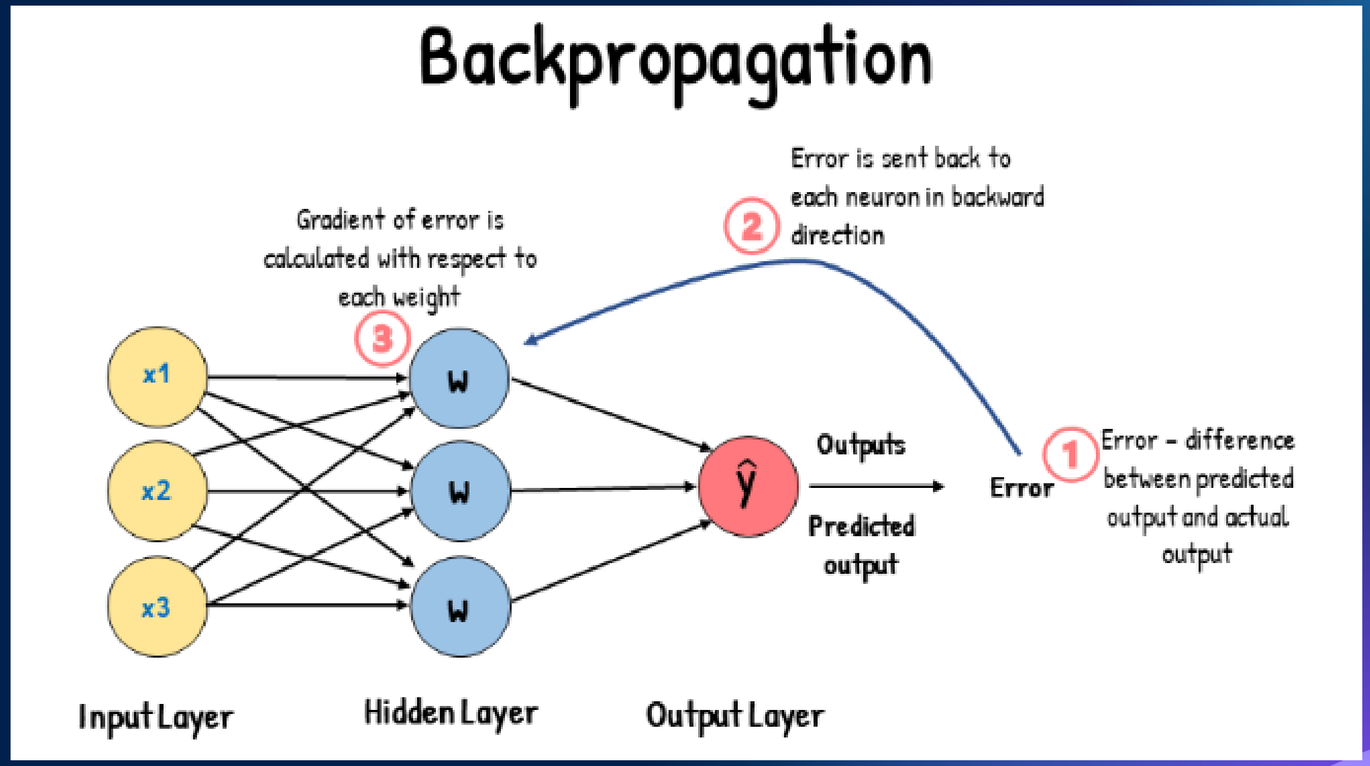


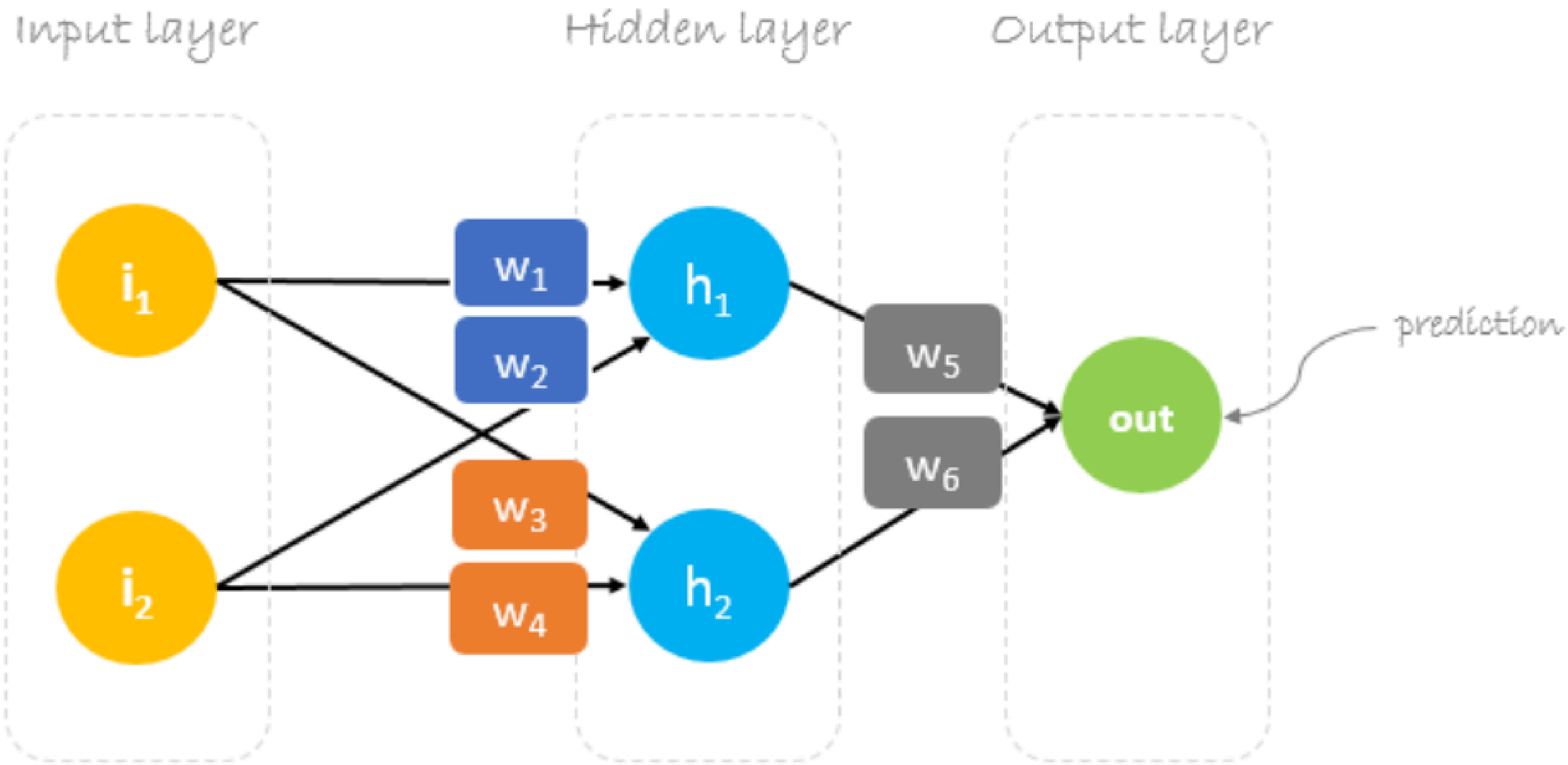
Backpropagation

What is Backpropogation

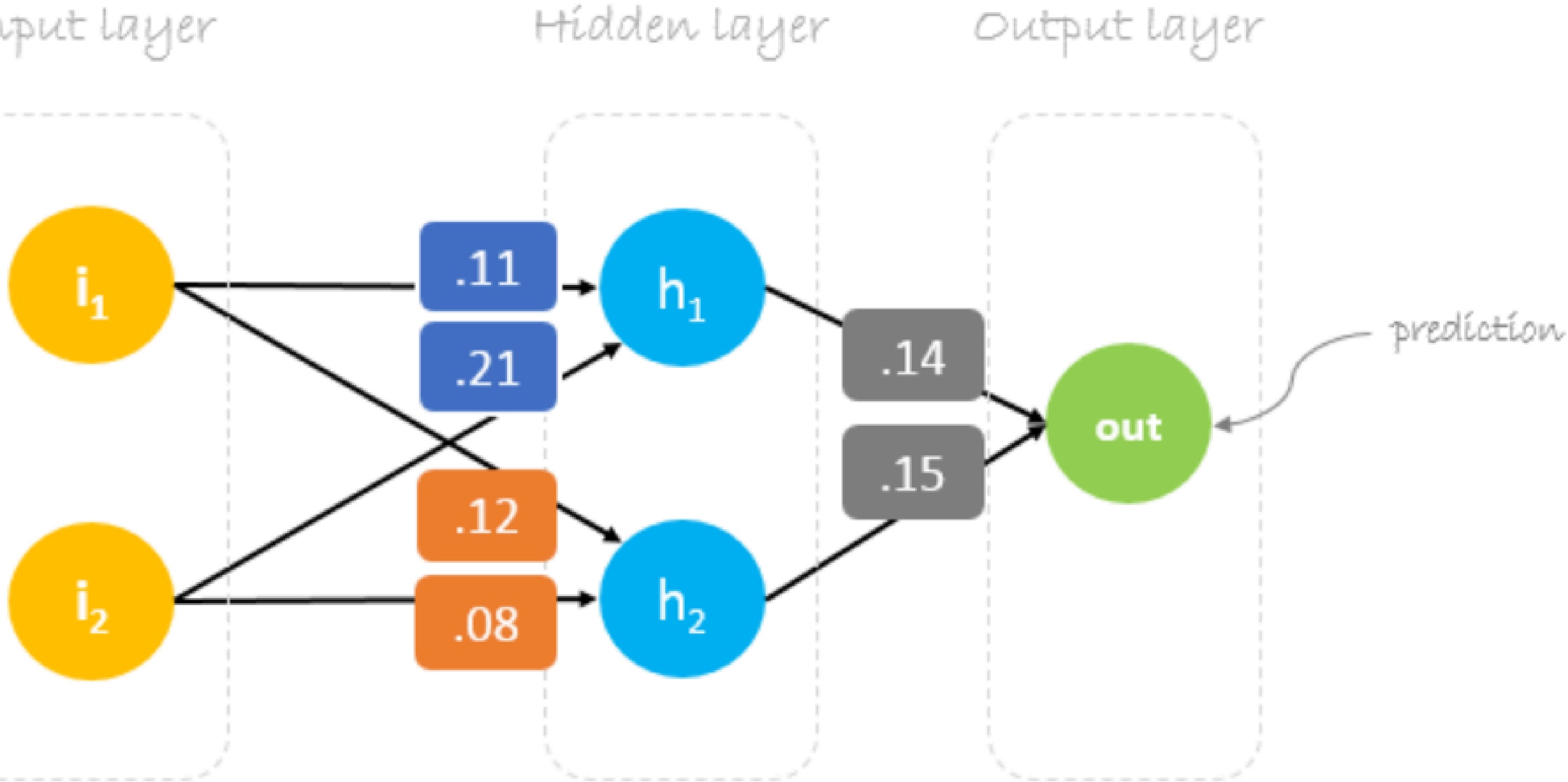
Backpropagation is an algorithm that is designed to test for errors working back from output nodes to input nodes.

So in general refers to the whole process of calculating gradient of the loss funcyion and its use in stochastic gradient descent in calculating the new weights



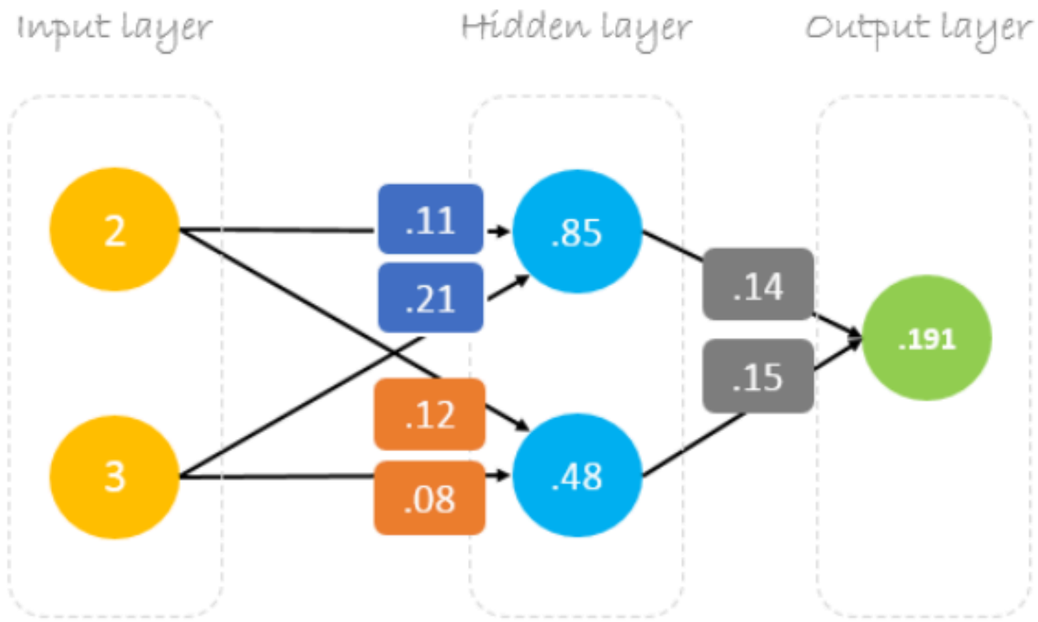


https://hmkcode.com/ai/backpropagation-step-by-step/


Input layer
Hidden layer
Output layer
i_1
i_2
.11
.21
.12
.08
h_1
h_2
.14
.15
out
prediction

Forward Pass

We will use given weights and inputs to predict the output. Inputs are multiplied by weights; the results are then passed forward to next layer.



Forward Pass

Matrix multiplication

$$[2 \quad 3] \cdot \begin{bmatrix} 0.11 & 0.12 \\ 0.21 & 0.08 \end{bmatrix} = [0.85 \quad 0.48] \cdot \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} = [0.191]$$

Details

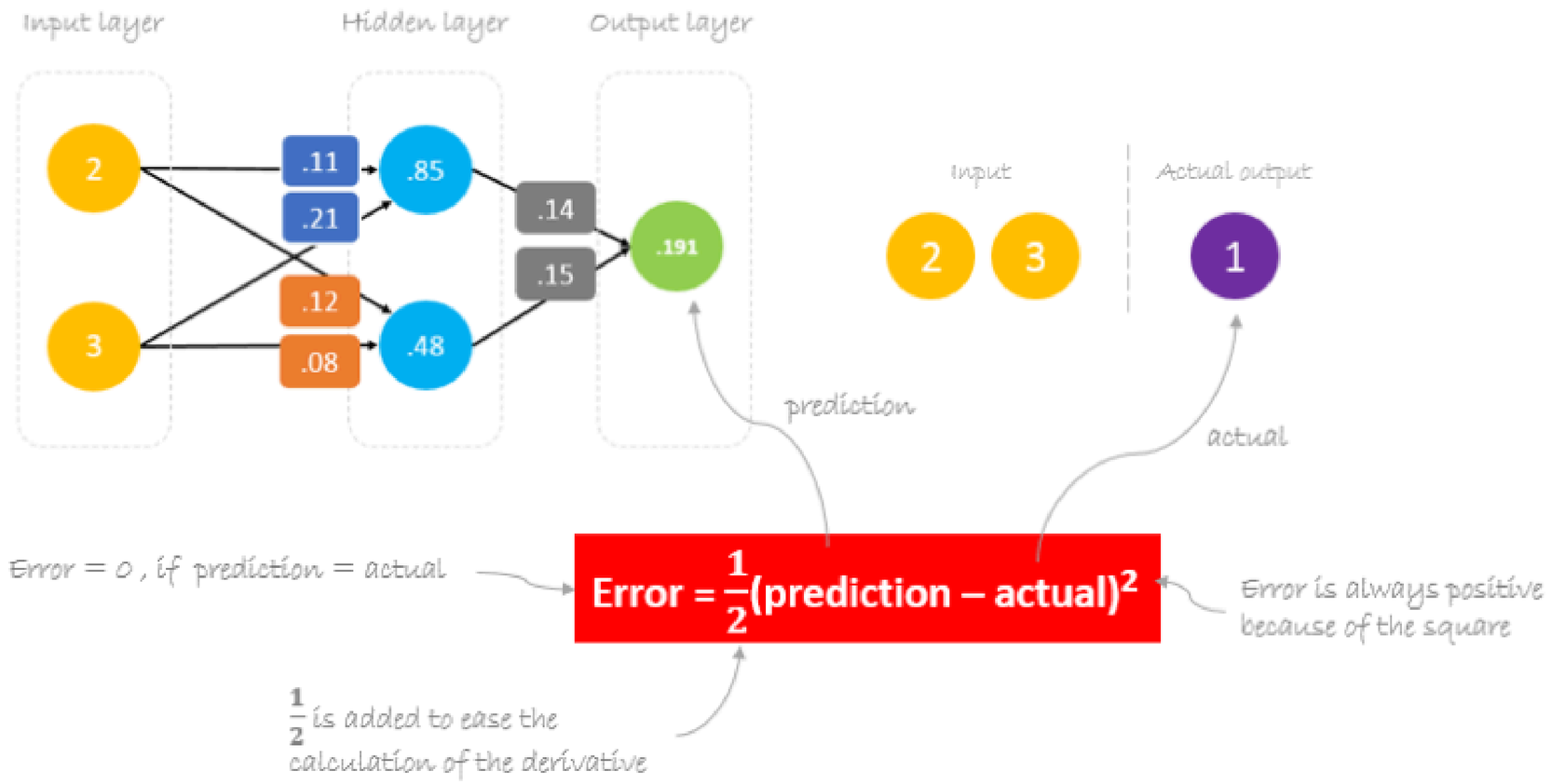
$2 \times .11 + 3 \times .21 = .85$

$2 \times .12 + 3 \times .08 = .48$

$.85 \times .14 + .48 \times .15 = .191$

Calculating Error

Now, it's time to find out how our network performed by calculating the difference between the actual output and predicted one. It's clear that our network output, or **prediction**, is not even close to **actual output**. We can calculate the difference or the error as following.



$$\text{Error} = \frac{1}{2}(0.191 - 1.0)^2 = 0.327$$

Reducing Error

Our main goal of the training is to reduce the **error** or the difference between **prediction** and **actual output**. Since **actual output** is constant, "not changing", the only way to reduce the error is to change **prediction** value. The question now is, how to change **prediction** value?

By decomposing **prediction** into its basic elements we can find that **weights** are the variable elements affecting **prediction** value. In other words, in order to change **prediction** value, we need to change **weights** values.

$$\textbf{prediction} = out$$

$$\textbf{prediction} = (h_1)\, w_5 + (h_2)\, w_6$$

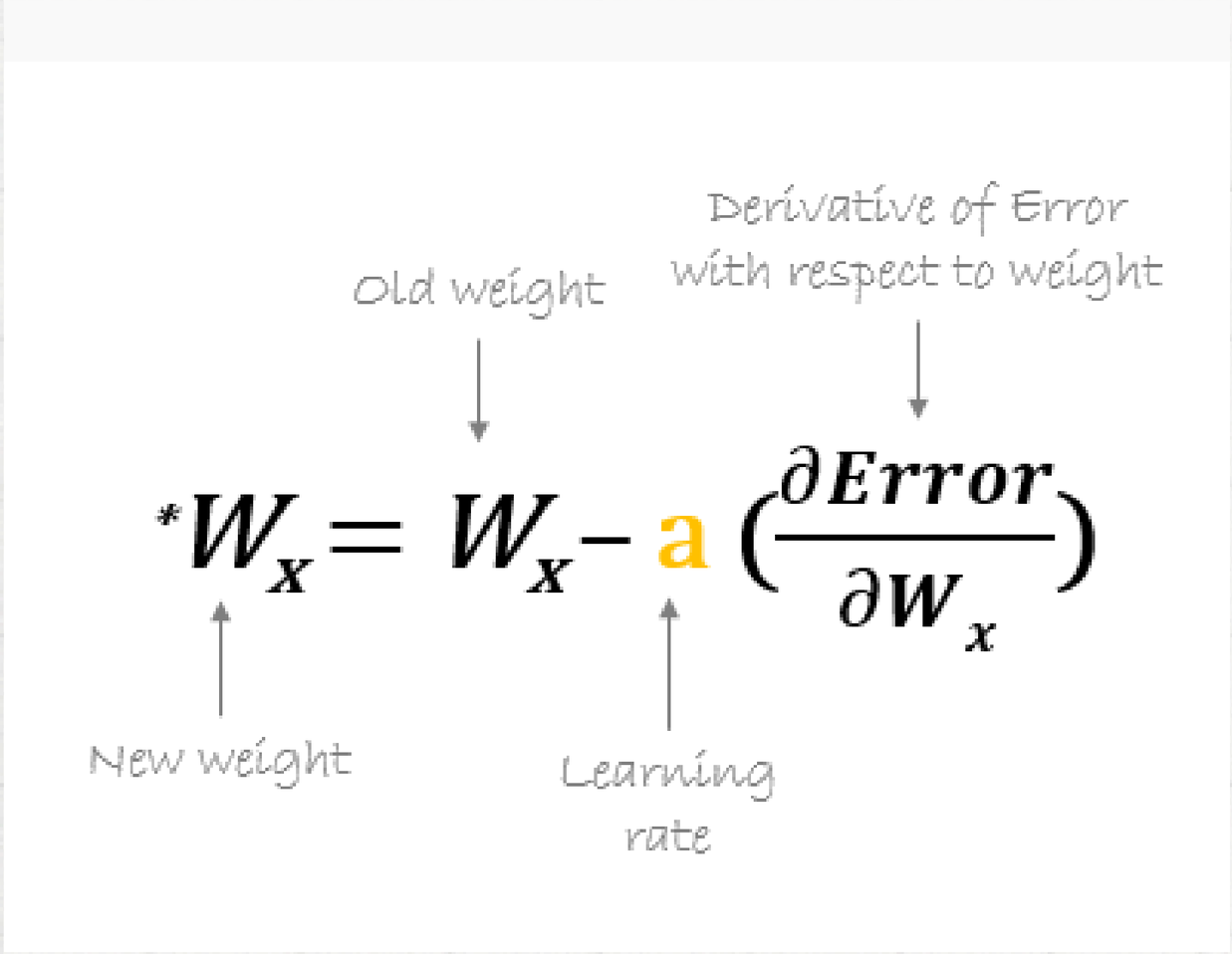
$$h_1 = i_1 w_1 + i_2 w_2$$

$$h_2 = i_1 w_3 + i_2 w_4$$

$$\textbf{prediction} = (i_1 w_1 + i_2 w_2)\, w_5 + (i_1 w_3 + i_2 w_4)\, w_6$$

to change ***prediction*** value,
we need to change ***weights***

General backpropagation formula



+ Apply as a chain rule

$$^{*}W_6 = W_6 - a\left(\frac{\partial Error}{\partial W_6}\right)$$

The derivation of the error function is evaluated by applying the chain rule as following

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$

chain rule

$$Error = \frac{1}{2}(prediction - actual)^2$$

$$prediction = (i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6$$

$$\frac{\partial Error}{\partial W_6} = \frac{\frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial \left(i_1 w_1 + i_2 w_2\right) w_5 + \left(i_1 w_3 + i_2 w_4\right) w_6}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2 * \frac{1}{2}(predictoin - actula)\frac{\partial (predictoin - actula)}{\partial prediciton} * (i_1 w_3 + i_2 w_4)$$

$h_2 = i_1 w_3 + i_2 w_4$

$$\frac{\partial Error}{\partial W_6} = (predictoin - actula) * (h_2)$$

Δ = prediction − actual

delta

$$\frac{\partial Error}{\partial W_6} = \Delta h_2$$

So to update `w6` we can apply the following formula

$$^{*}W_6 = W_6 - a\,\Delta h_2$$

$$\frac{\partial Error}{\partial W_1} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial h_1} * \frac{\partial h_1}{\partial W_1}$$ ← chain rule

$$Error = \frac{1}{2}(prediction - actual)^2$$

$$prediction = (h_1)\, w_5 + (h_2)\, w_6$$

$$h_1 = i_1 w_1 + i_2 w_2$$

$$\frac{\partial Error}{\partial W_1} = \frac{\partial \frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial (h_1)\, w_5 + (h_2)\, w_6}{\partial h_1} * \frac{\partial\, i_1 w_1 + i_2 w_2}{\partial w_1}$$

$$\frac{\partial Error}{\partial W_1} = 2 * \frac{1}{2}(predictoin - actula) \frac{\partial (predictoin - actula)}{\partial prediciton} * (w_5) * (i_1)$$

$$\frac{\partial Error}{\partial W_1} = (predictoin - actula) * (w_5 i_1)$$

$$\Delta = prediction - actual$$ ← delta

$$\frac{\partial Error}{\partial W_1} = \Delta\, w_5 i_1$$

updated weights →

$$^*w_6 = w_6 - a\,(h_2 \,.\, \Delta)$$

$$^*w_5 = w_5 - a\,(h_1 \,.\, \Delta)$$

$$^*w_4 = w_4 - a\,(i_2 \,.\, \Delta w_6)$$

$$^*w_3 = w_3 - a\,(i_1 \,.\, \Delta w_6)$$

$$^*w_2 = w_2 - a\,(i_2 \,.\, \Delta w_5)$$

$$^*w_1 = w_1 - a\,(i_1 \,.\, \Delta w_5)$$

Backward Pass

Using derived formulas we can find the new **weights**.

> **Learning rate:** is a hyperparameter which means that we need to manually guess its value.

$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

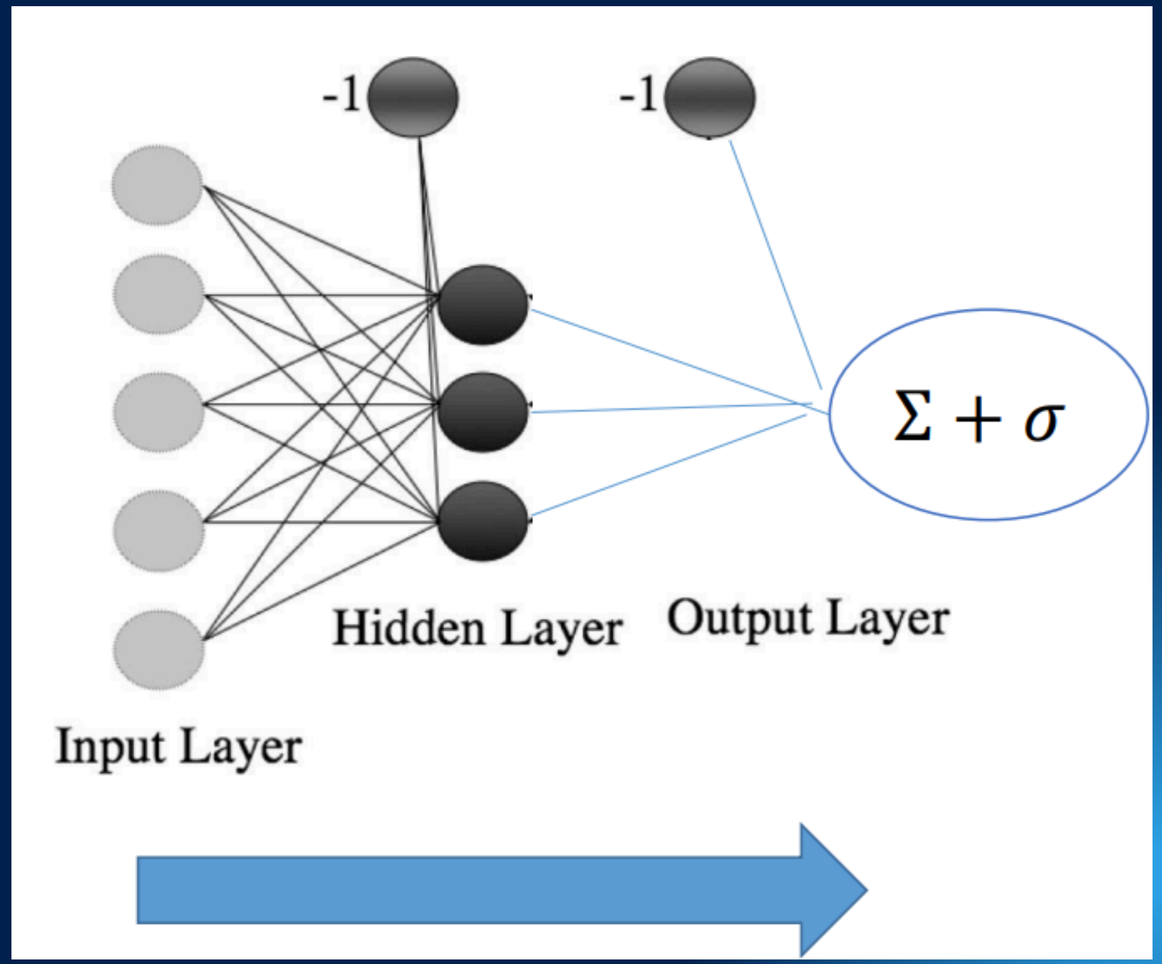
$a = 0.05$ ← **Learning rate,** we smartly guess this number

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 2 \\ 3 \end{bmatrix} . [0.14 \quad 0.15] = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

Binary classification

- The output layer contains a single neuron
- Logistic activation function in the output layer
- Similar to logistic regression



Backpropagation – Binary Classification

Backpropagation formulas:

$$w_{i,1} = w_{i,1} - \eta\delta_0(k_1)a_1$$

$$v_{i,j} = v_{i,j} - \eta\delta_0(hidden_j)x_i$$

We assume the logistic function in the hidden layer:

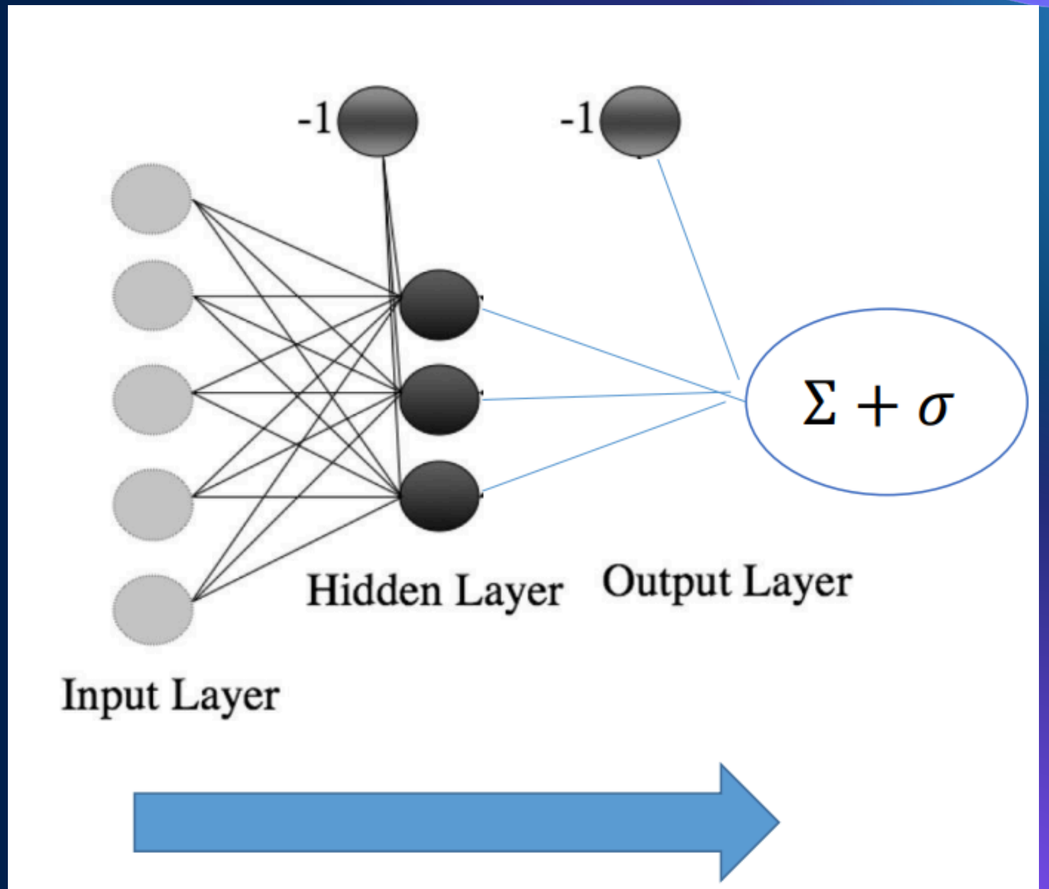
$$\delta_0(hidden_j) = \delta_0(k_1)(w_{j,1})a_j(1 - a_j)$$

Binary classification on final layer and using SE:

$$\delta_0(k_1) = (y - t)y(1 - y)$$

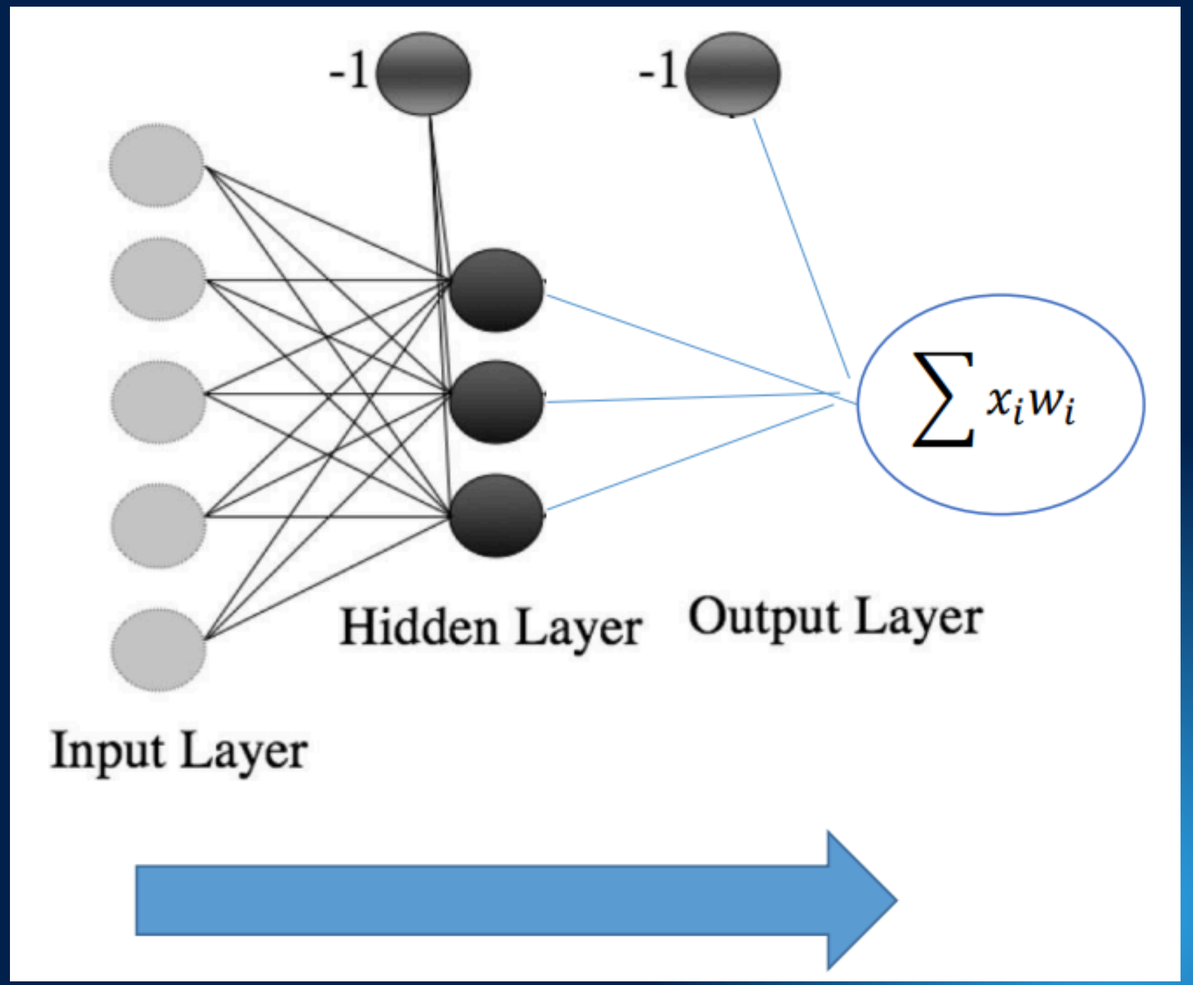
Binary classification on final layer and using cross entropy loss:

$$\delta_0(k_1) = (y - t)$$



Regression

- The output layer contains a single neuron, representing the continuous output value. No activation function is typically applied in the output layer for regression tasks.
- This can predict non-linear functions.



Backpropagation – Regression

Backpropagation formulas:

$$w_{i,1} = w_{i,1} - \eta\delta_0(k_1)a_1$$

$$v_{i,j} = v_{i,j} - \eta\delta_0(hidden_j)x_i$$

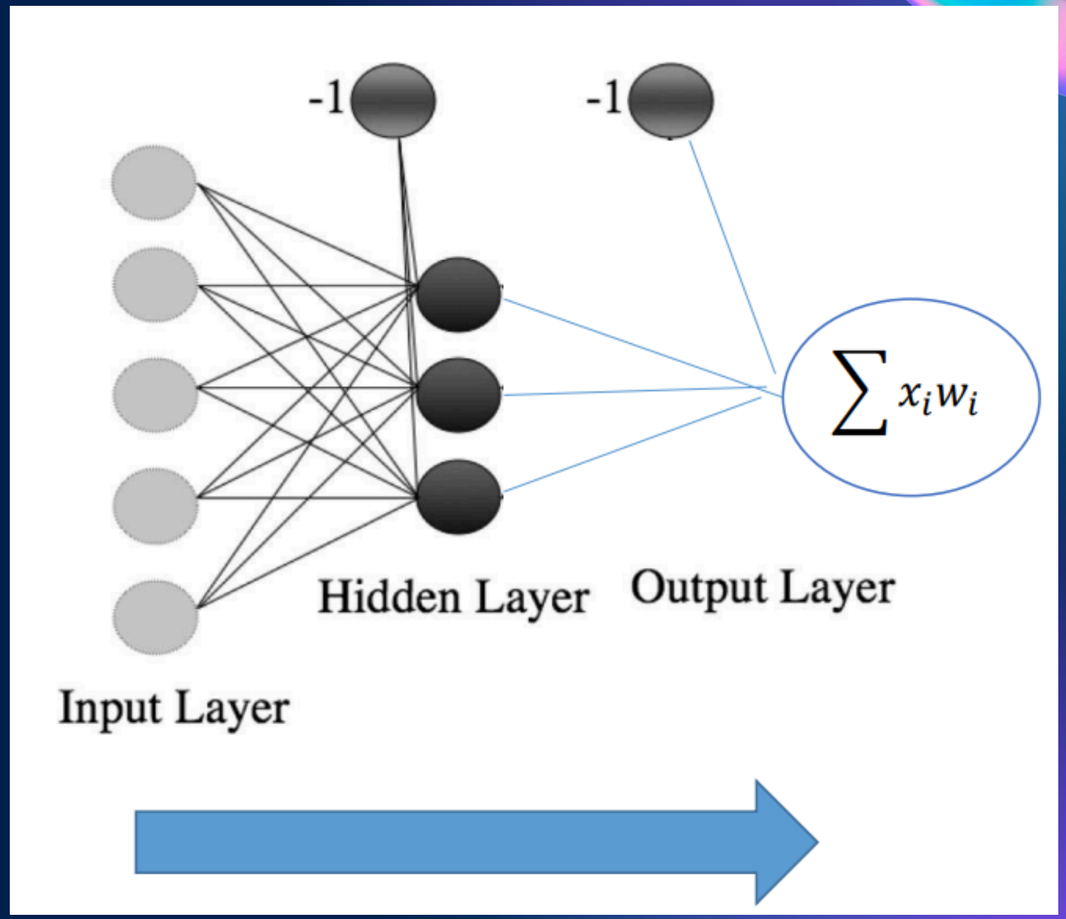
v- first set of weights (connections between input and hidden nodes)
w- second set of weights(connections between hidden and output nodes)
η is the learning rate
a_j is the is the result of the activation function, on the hidden layer
We assume the logistic function in the hidden layer:

$$\delta_0(hidden_j) = \delta_0(k_1)(w_{j,1})a_j(1 - a_j)$$

When calculating delta for hidden node we use the weight w before the update

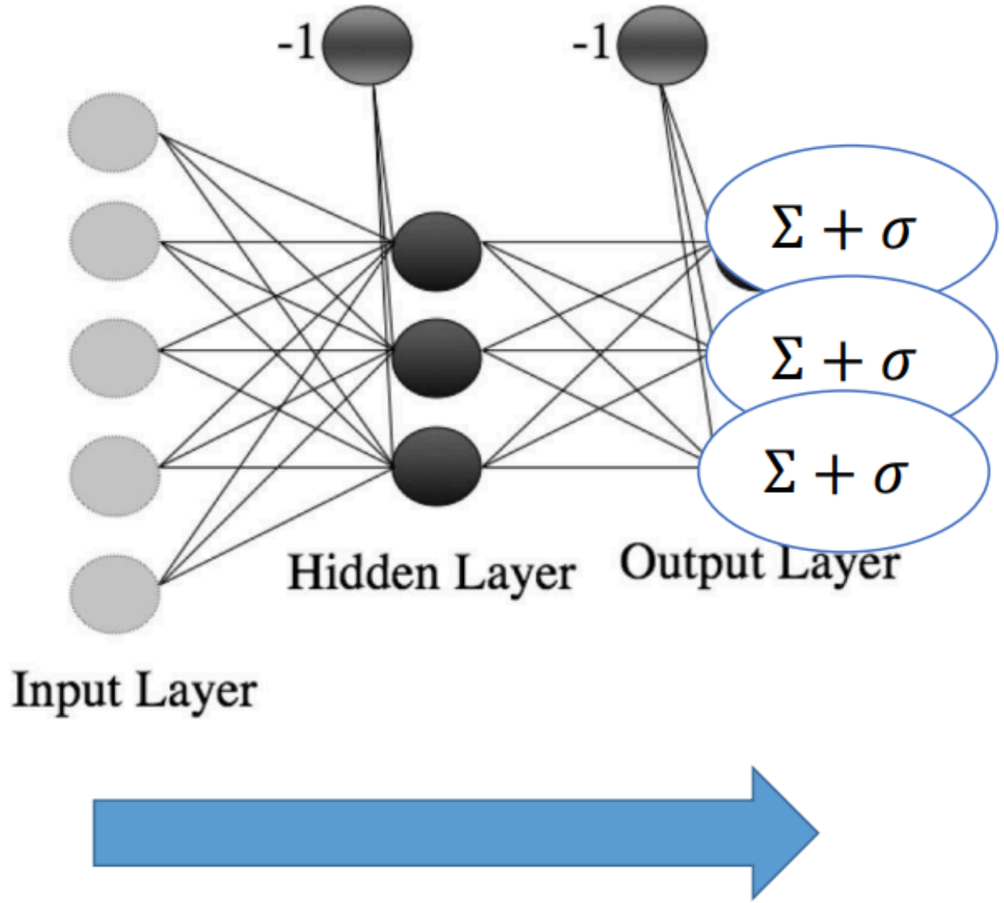
Regression on final layer:

$$\delta_0(k_1) = (y - t)$$



Multi label classification

- Several output nodes
- Logistic activation function
- Can be made multi-class classification by one vs. rest.



Backpropagation – Multi label Classification

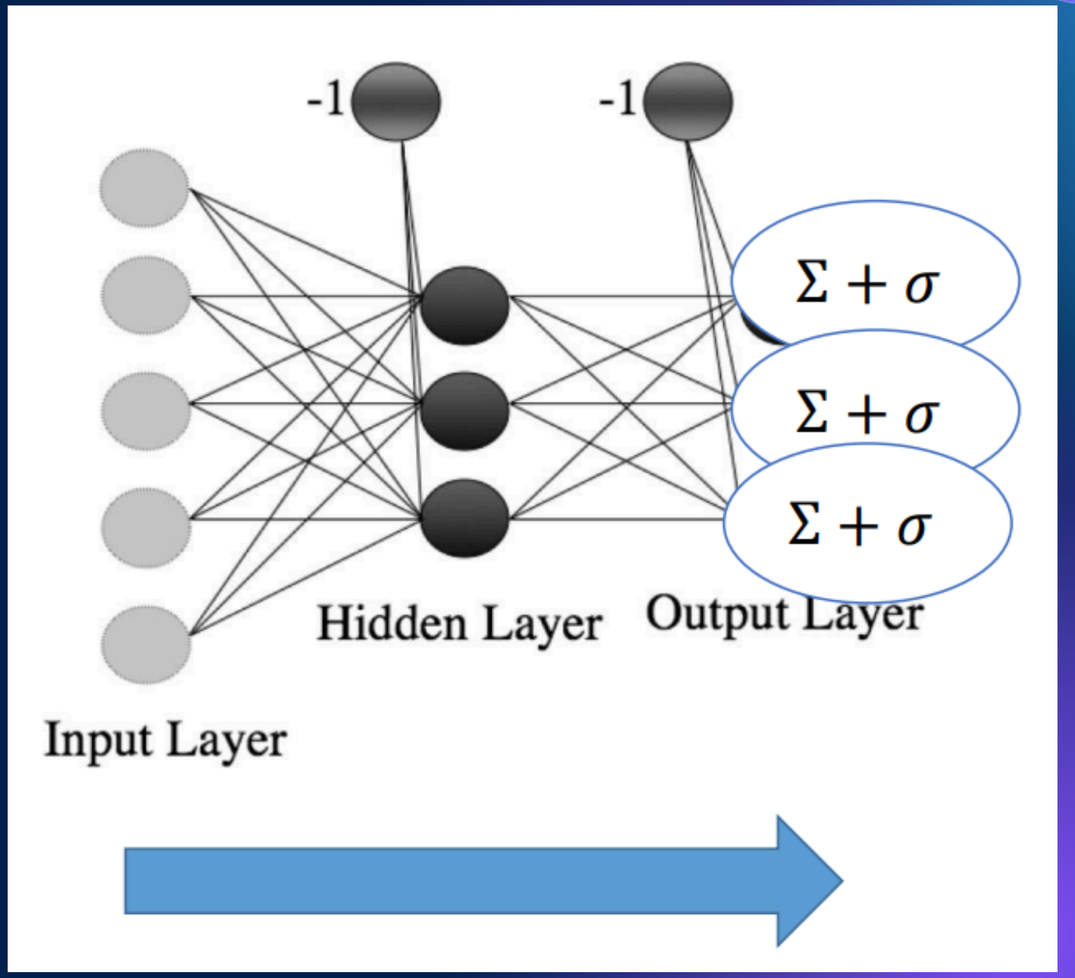
Multi-label classification:

$$w_{i,j} = w_{i,1} - \eta\delta_0(k_j)a_i$$

$$v_{i,j} = v_{i,j} - \eta\delta(hidden_j)x_i$$

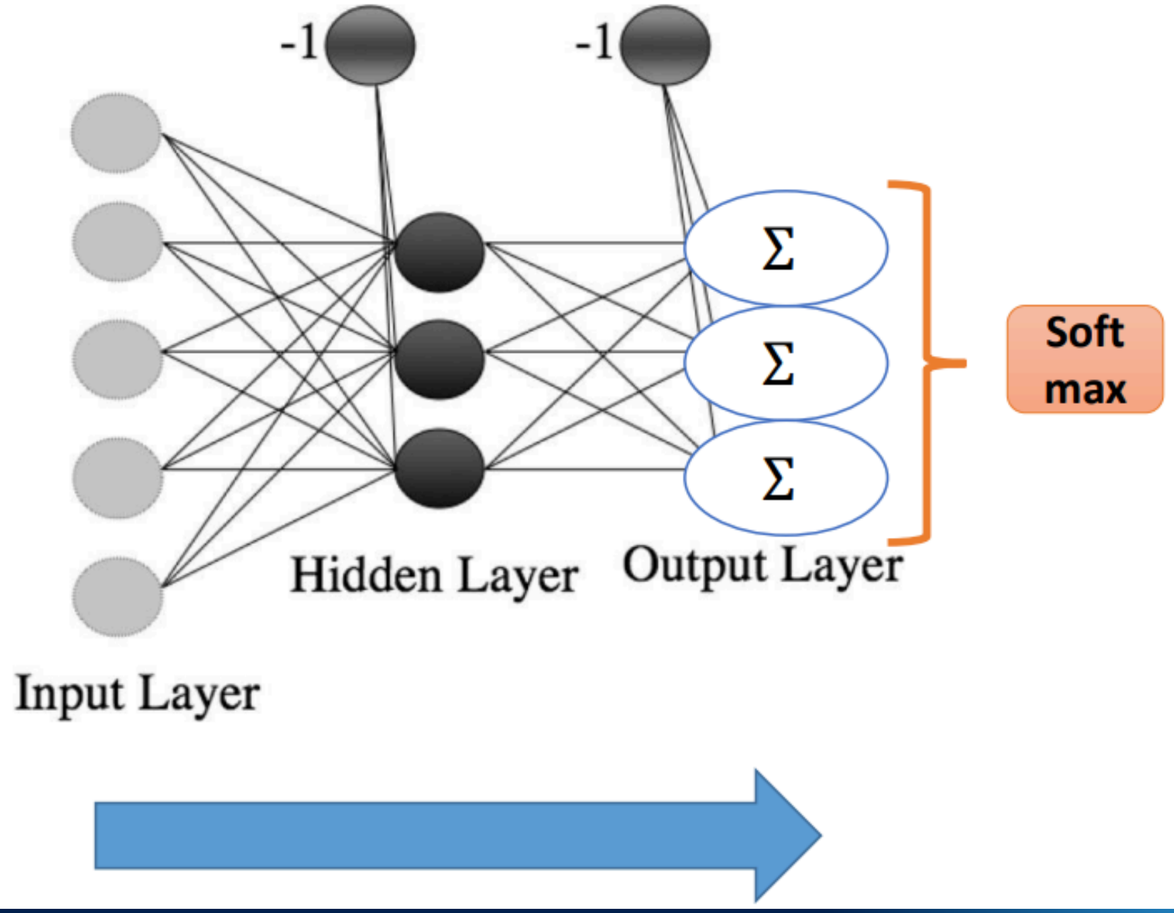
$$\delta_0(k_j) = (y_j - t_j)y_j(1 - y_j)$$

$$\delta(hidden_j) = a_j(1 - a_j)\sum \delta_0(k_i)(w_{j,i})$$



Multi class classification

- Several output nodes
- Sum the weighted inputs at each nodes
- The sums are brought together in the soft-max



Softmax reminder

The softmax function converts a vector of K real numbers into a probability distribution of K possible outcomes.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K.$$

Steps

1. Exponentiate each element $\exp(\mathbf{z}) = [e^{z_1}, e^{z_2}, \ldots, e^{z_K}]$
2. Sum the exponentiated values $\text{sum\_exp} = e^{z_1} + e^{z_2} + \ldots + e^{z_K}$
3. Compute probabilities $\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\text{sum\_exp}}$

The resulting vector contains values that range between 0 and 1 and sum to 1, transforming the original inputs into a probability distribution over the classes.

Softmax uses categorial cross entropy loss $CE = -\sum t_i log(y_i)$

Vanishing gradient

The vanishing gradient problem is a challenge that can occur during the training of deep neural networks, particularly in the context of gradient-based optimization algorithms like backpropagation. It refers to the phenomenon where the gradients of the loss function with respect to the weights become extremely small as the network's depth increases, making it difficult for the network to learn effectively.

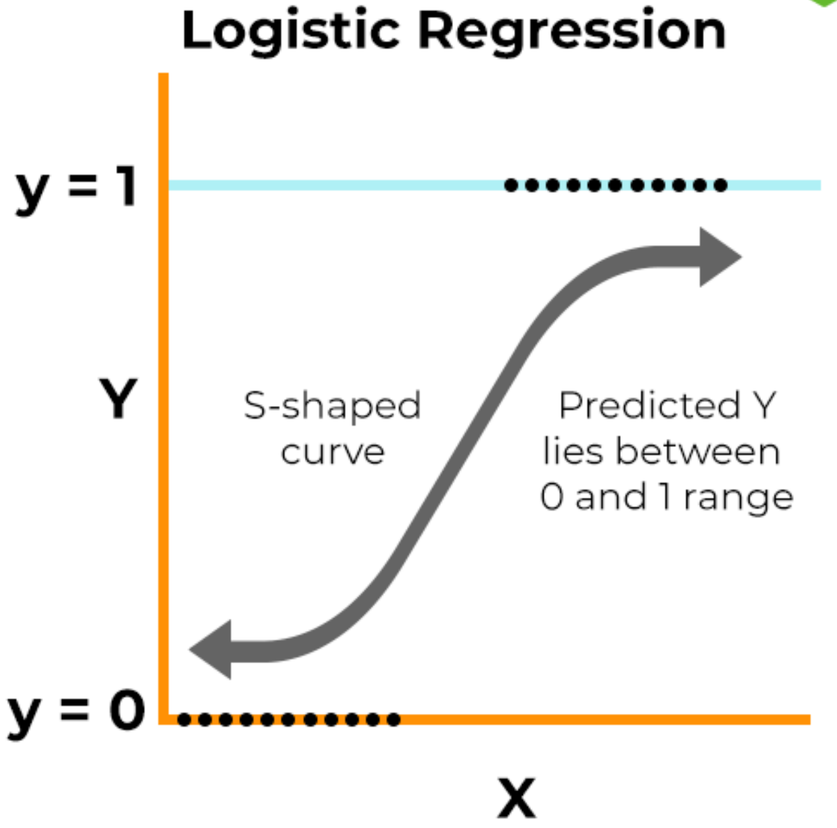
LOGISTIC REGRESSION

Binary classification, predicting the probability that an instance belongs to a particular class.

The sigmoid function

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

$$h\theta(X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}}$$

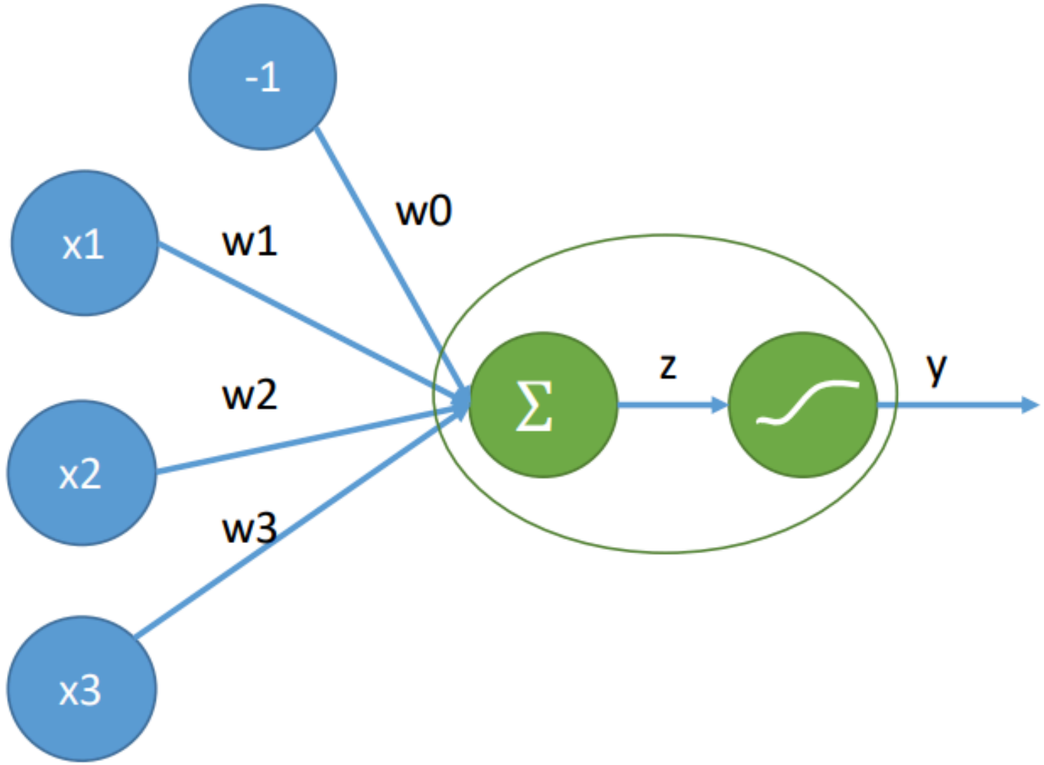


LOGISTIC REGRESSION

- First sum of weighted inputs :
- $z = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w} \cdot \boldsymbol{x}$
- Apply the logistic function σ to this sum

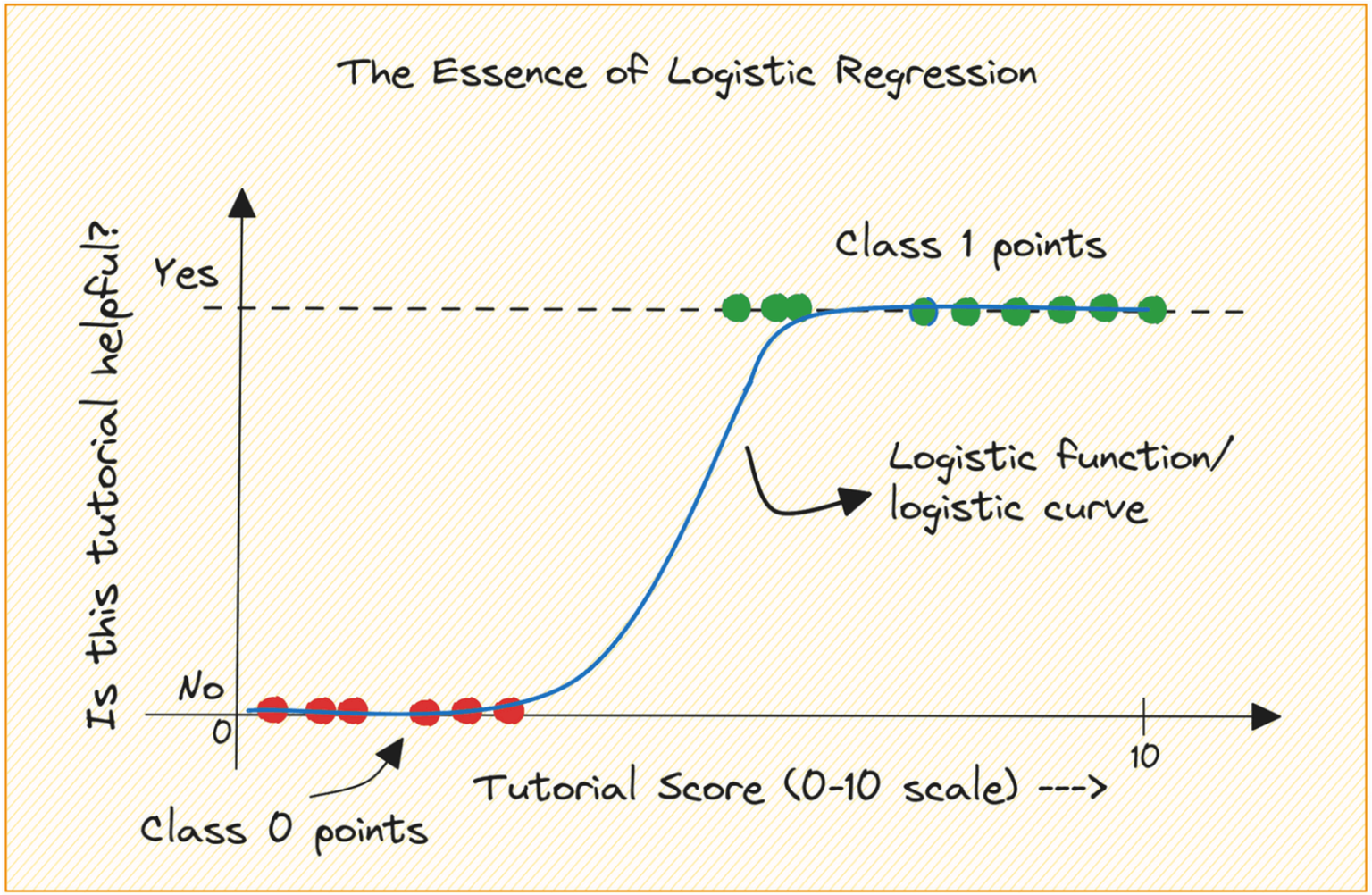
$$y = \sigma(z) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{x}}}$$

- For $\boldsymbol{x} = \vec{x}$ predict
  - as the positive class if $y > 0.5$,
  - otherwise, the negative class



LOGISTIC REGRESSION

The logistic function squeezes the output between 0 and 1, representing the probability. If the probability is greater than a threshold (often 0.5), the instance is predicted to belong to the positive class; otherwise, it is predicted to belong to the negative class.



CROSS ENTROPY LOSS

Error Function Formula:

$$\text{Cross-Entropy Loss} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)]$$

Where:

- n is the number of data points.
- y_i is the actual binary label (0 or 1).
- $\hat{p}_i$ is the predicted probability of belonging to the positive class.

WHAT IS THE DIFFERENCE BETWEEN THEM

Linear classification	Logistic regression
Used for predicting a continuous outcome. It models the relationship between the dependent variable and independent variables as a linear equation.	Used for predicting the probability of an instance belonging to a particular class in binary classification problems. It models the relationship using the logistic function to produce values between 0 and 1.
Outputs any real number, positive or negative. Predictions can be outside the range [0, 1].	Outputs values between 0 and 1 due to the logistic (sigmoid) function, representing probabilities.
Assumes a linear function: $f(\boldsymbol{x}) = f((x_1, x_2, \dots, x_m)) = w_0 + w_1x_1 + w_2x_2 + \dots w_mx_m$	Applies the logistic function to the sum of weighted inputs: $\frac{1}{1 + e^{-\vec{w}\cdot\vec{x}}}$
Type of problem: regression analysis. E.g.: predicting housing prices.	Type of problem: Binary classification. E.g.: predict an email is spam or not.

IMPLEMENTING THE GRADIENT DESCENT

- Input:
  - X, input, a $N \times m$ NumPy matrix:
    - N items, m features
  - T, corresponding target values,
    - $N \times 1$ column vector
    - (maybe it is given as a vector of length N and must be transformed)
  - Make a weight matrix, W,
    - $m \times 1$ column vector

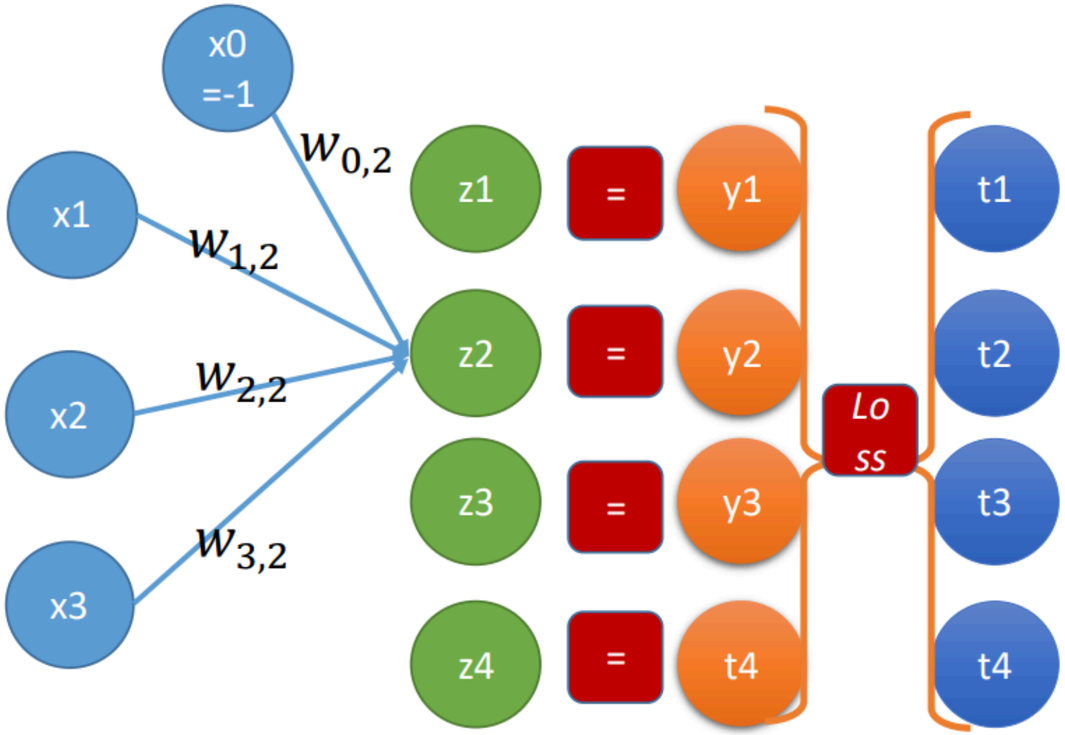
- Forward step:
  - $Y = X@W$
- Update step:
  - $W = W - \eta \nabla f$
  - $W -= \eta X.T(Y - T)$
  - (η, *eta*, is a learning rate)

ONE VS REST CLASSIFICATION APPROACH

Multi-Output Linear Classification

- $y_j = g(z_j) = z_j$
- MSE-loss:

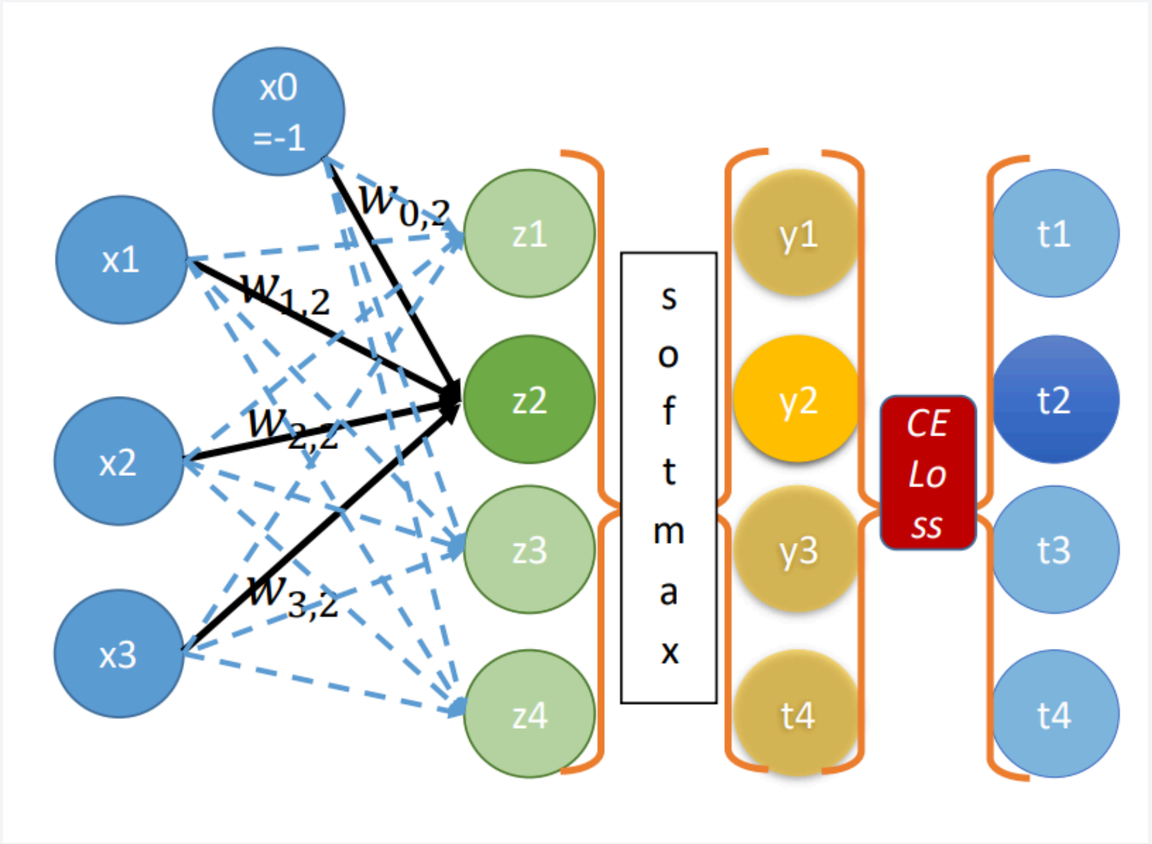
  $\sum_{k=1}^{N}(\sum_{j=1}^{n}(y_{k,j} - t_{k,j})^2)$
  - n output nodes
  - N input items
- y_j independent of $w_{h,k}$ $h \neq k$,
  - hence corresponds to n independent models
  - (Gets more interesting for multi-layer networks)



ONE VS REST CLASSIFICATION APPROACH

Multinomial Linear Classification

- $z_j = \sum_{i=0}^{n} w_{i,j} x_i$
- Apply the softmax-function, S, where
  - $y_j = (S(z_1, \ldots, z_n))_j = \frac{e^{z_j}}{\sum_{k=1}^{n} e^{z_k}}$
- Observe:
  - y_j depends on all the z_k
  - If $z_h > z_k$ then $y_h > y_k$
  - $0 < y_j < 1$
  - $\sum_{j=1}^{n} y_j = 1$
  - A probability distribution
  - $P(C_j|\vec{x}) = \frac{e^{\overrightarrow{w_j} \cdot \vec{x}}}{\sum_{k=1}^{n} e^{\overrightarrow{w_k} \cdot \vec{x}}}$



RL compared to supervised learning

Reinforcement learning	**Supervised learning**
Reinforcement learning is all about making decisions sequentially. In simple words, we can say that the output depends on the state of the current input and the next input depends on the output of the previous input	In Supervised learning, the decision is made on the initial input or the input given at the start
In Reinforcement learning decision is dependent, So we give labels to sequences of dependent decisions	In supervised learning the decisions are independent of each other so labels are given to each decision.
Example: Chess game,text summarization	**Example:** Object recognition,spam detetction

Types of reinforcement learning

On-Policy Learning

In on-policy learning, the agent learns about the optimal policy while following the same policy. It learns from experiences generated by its current policy and updates this policy directly.

- **more explorative**
- **often have simpler implementations compared to off-policy methods**
- **more stable**

Off-Policy Learning

In off-policy learning, the agent learns about the optimal policy while following a different policy. It can learn from data generated by any policy, not necessarily the one being evaluated or improved. This allows for more flexibility in exploration and exploitation strategies.

- **more exploitative**
- **can converge faster due to its ability to explore diverse experiences**
- **is often more stochastic**

DEFINITION UNSUPERVISED LEARNING

Unsupervised learning is a type of machine learning technique **where the algorithm explores the data and identifies hidden structures or relationships within it without being given specific instructions on what to look for.**

Commonly used for tasks such as clustering, dimensionality reduction, and anomaly detection, **where the goal is to understand the inherent structure of the data.**

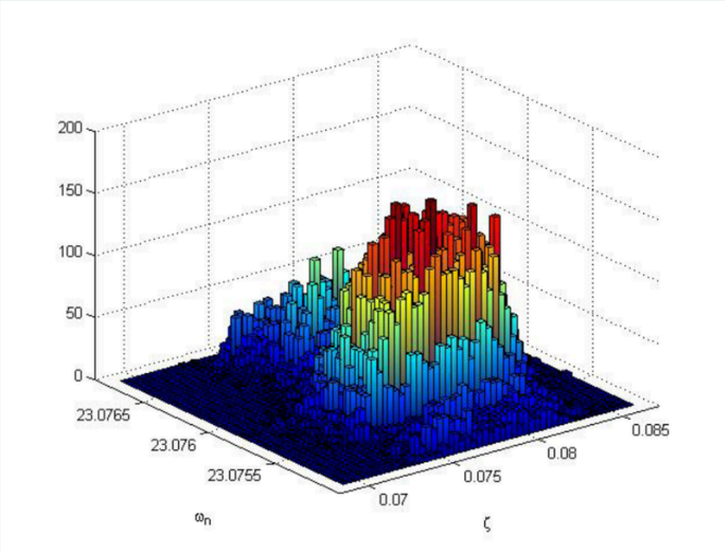
Unlike supervised learning, there are no predefined target variables, and the algorithm must infer the underlying structure of the data on its own.

DIFFERENCE BETWEEN SUPERVISED AND UNSUPERVISED LEARNING

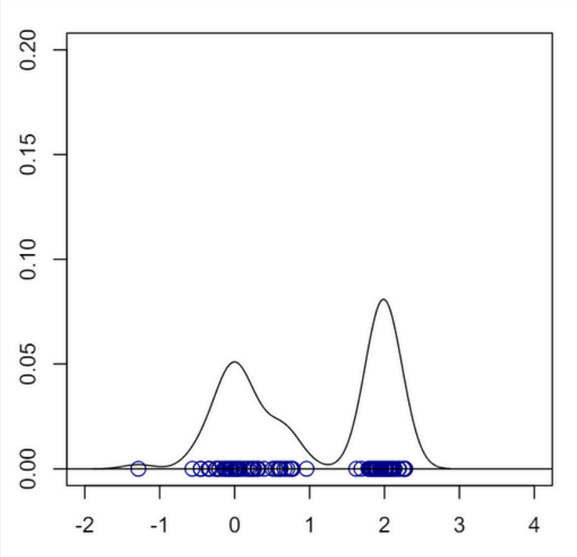
Aspect	Supervised Learning	Unsupervised Learning
Training Data	Labeled input-output pairs	Unlabeled data
Goal	Learn a mapping from inputs to outputs	Discover hidden patterns or structures
Example Applications	Image classification, regression, classification	Clustering, dimensionality reduction, anomaly detection
Evaluation	Performance metrics based on predicted outputs	Metrics based on discovered patterns or clusters
Feedback	Direct feedback based on labeled data	Indirect feedback based on data similarity or distribution

DENSITY FUNCTIONS

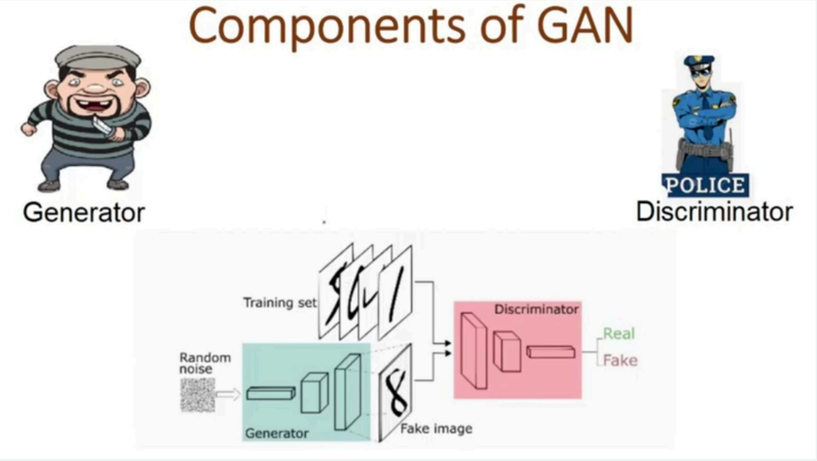
Histogram



Smooth density estimator



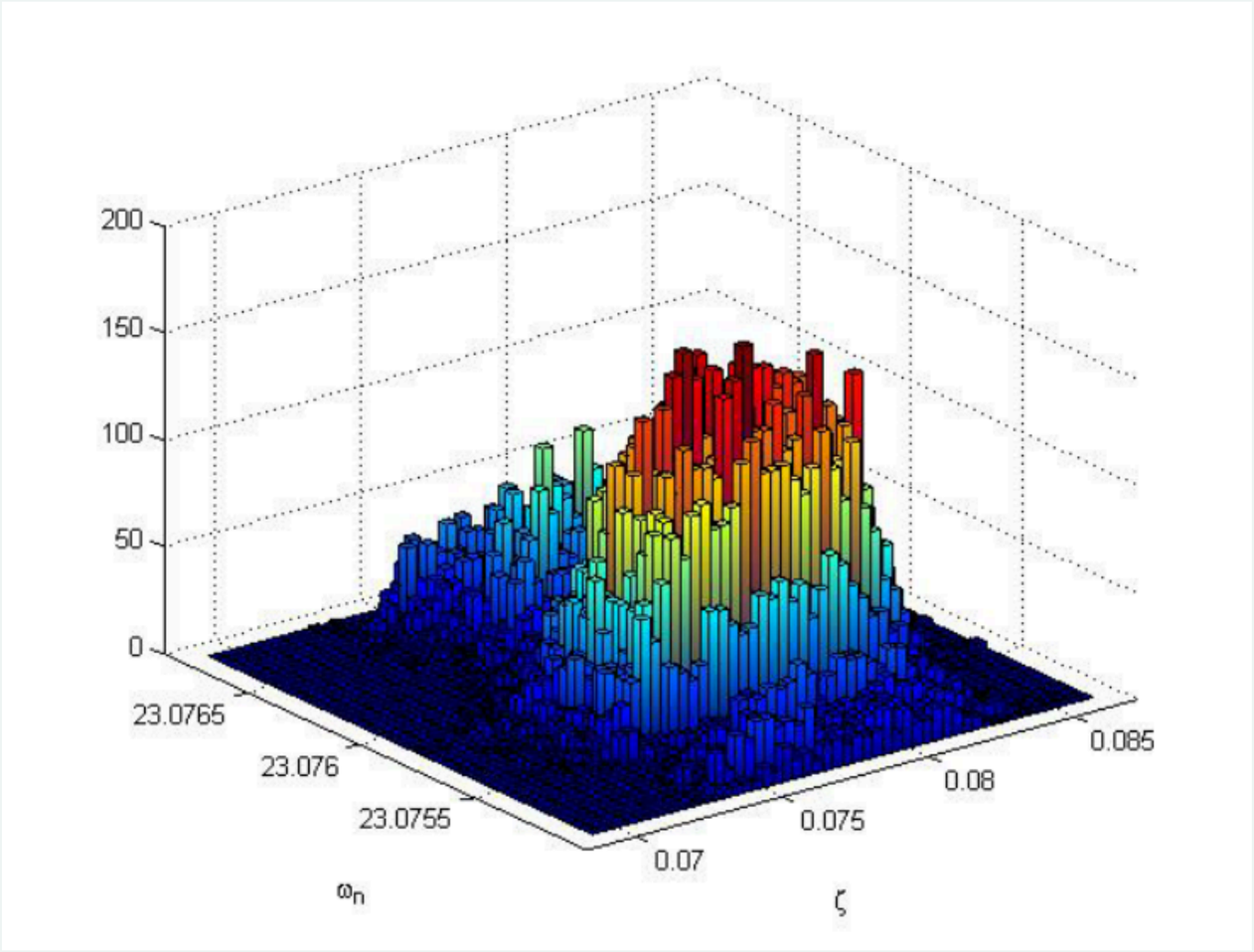
generative adversarial networks (GANs)



DENSITY FUNCTIONS

Histogram

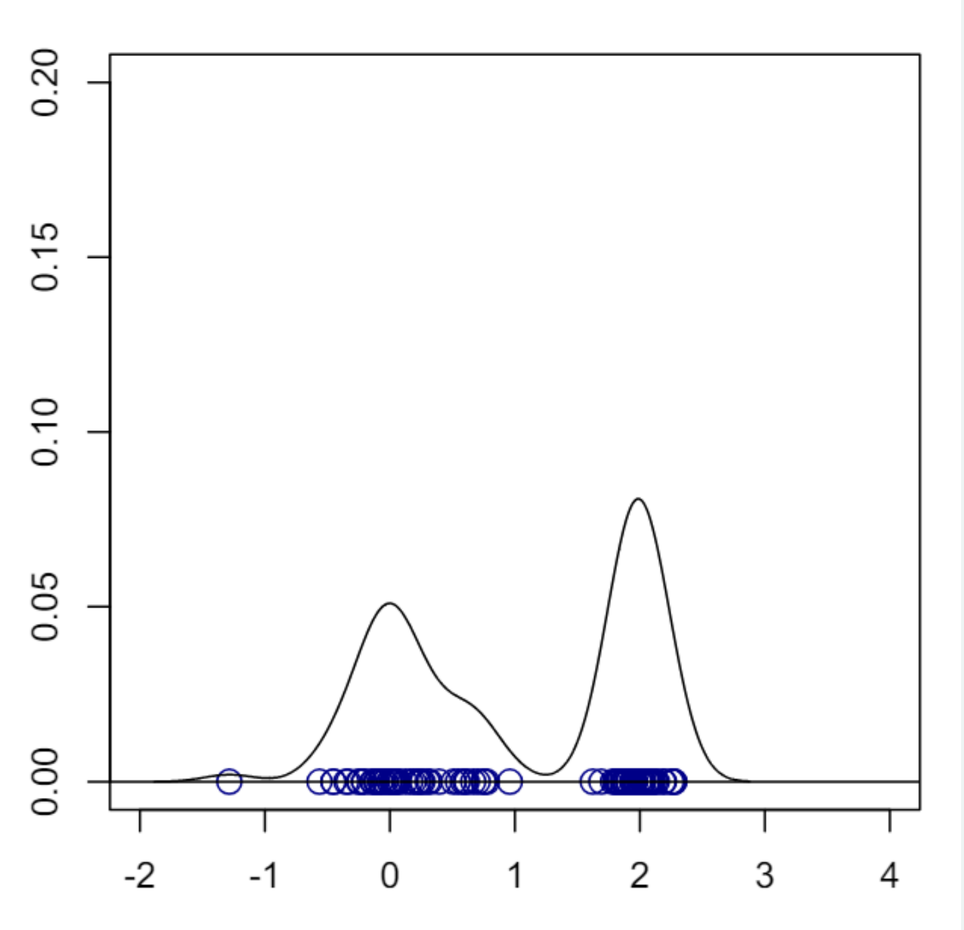
Consists of a series of adjacent bars, where the height of each bar represents the frequency or count of data points falling within a specified range or bin. Essentially, it shows how data is spread out over different intervals or categories.



DENSITY FUNCTIONS

Smooth density estimator

Kernel Density Estimation (KDE) is a statistical method used to estimate the probability density function (PDF) of a dataset by placing a kernel function at each data point and summing up their contributions to create a smooth density estimate.



HOW TO TRAIN KERNEL SMOOTH DENCITY ESTIMATOR

Training data: $x_1, \dots, x_n \in \mathbf{R}$ Test data: x

- Select a kernel function $K(x)$
  - A kernel is a "0-detector" with high positive value for x close to 0 and gradually lower values for x far from 0

- Compute $K(\frac{x-x_i}{h})$ for $i = 1,2, \dots, n.$
  - This detects if x is close to any of the points in the training data set, and h>0 controls the detection sensitivity

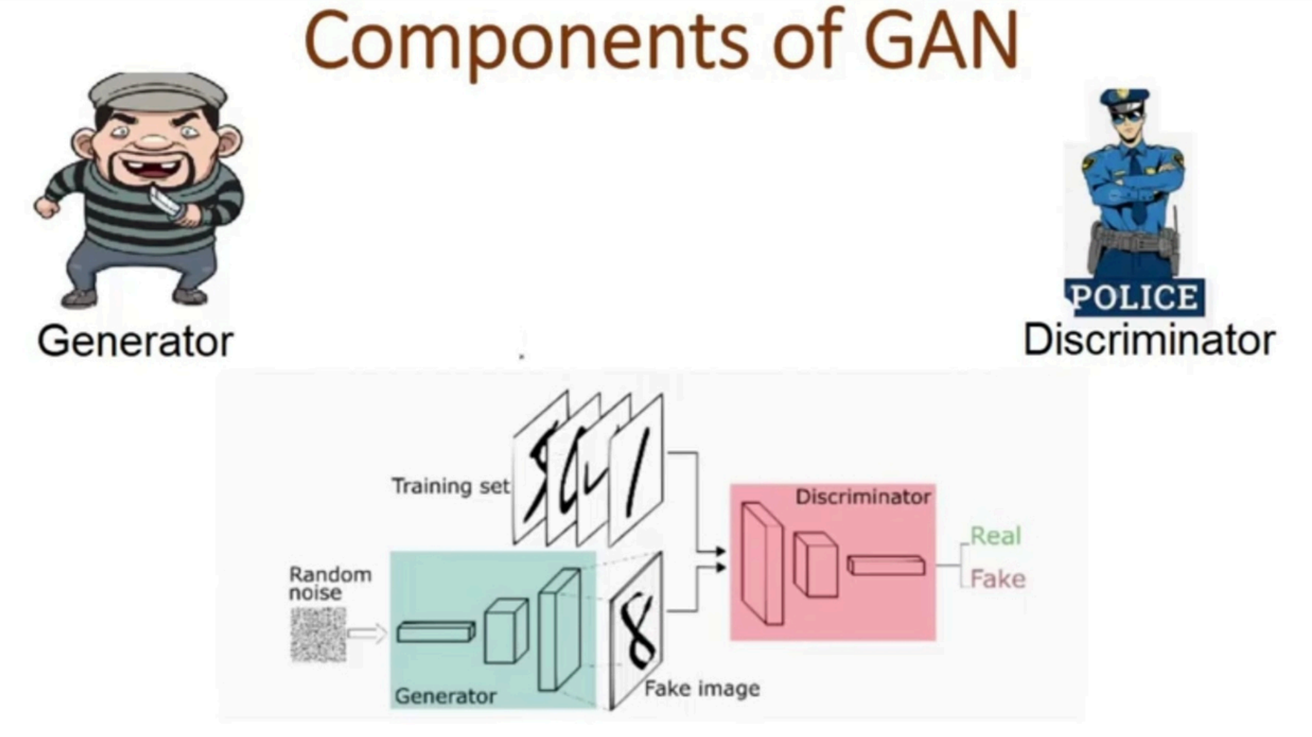
- Let $f(x) = \frac{1}{nh}\sum_i K(\frac{x-x_i}{h})$
  - This adds together all detector values, so $f(x)$ is higher the more training data points there are in neighborhood of x

(Page 25 of lecture 11, 2024)

DENSITY FUNCTIONS

generative adversarial networks (GANs)

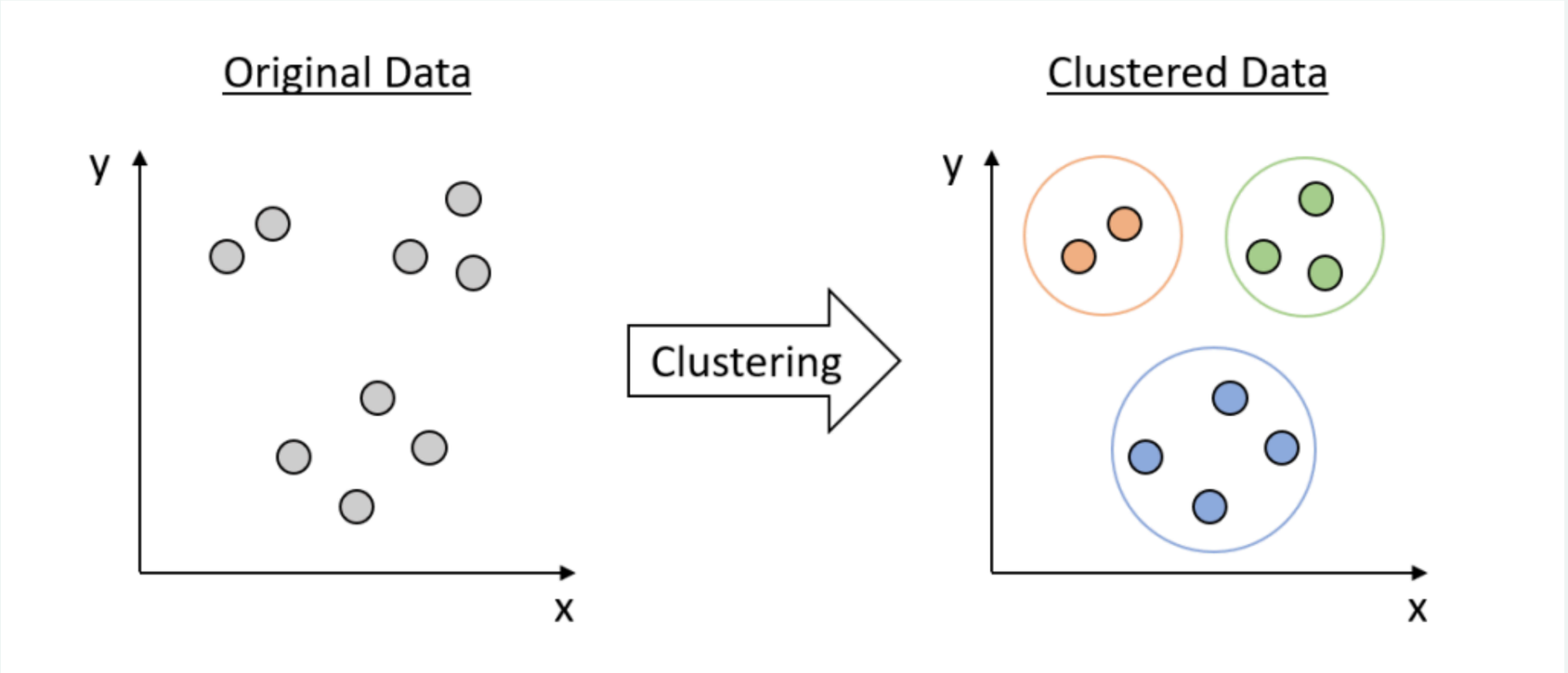
Consists of two neural networks: a generator and a discriminator. The generator learns to create realistic data samples, while the discriminator learns to distinguish between real and fake data. Through this adversarial process, both networks improve until the generator can produce data samples that are indistinguishable from real ones.



CLUSTERING

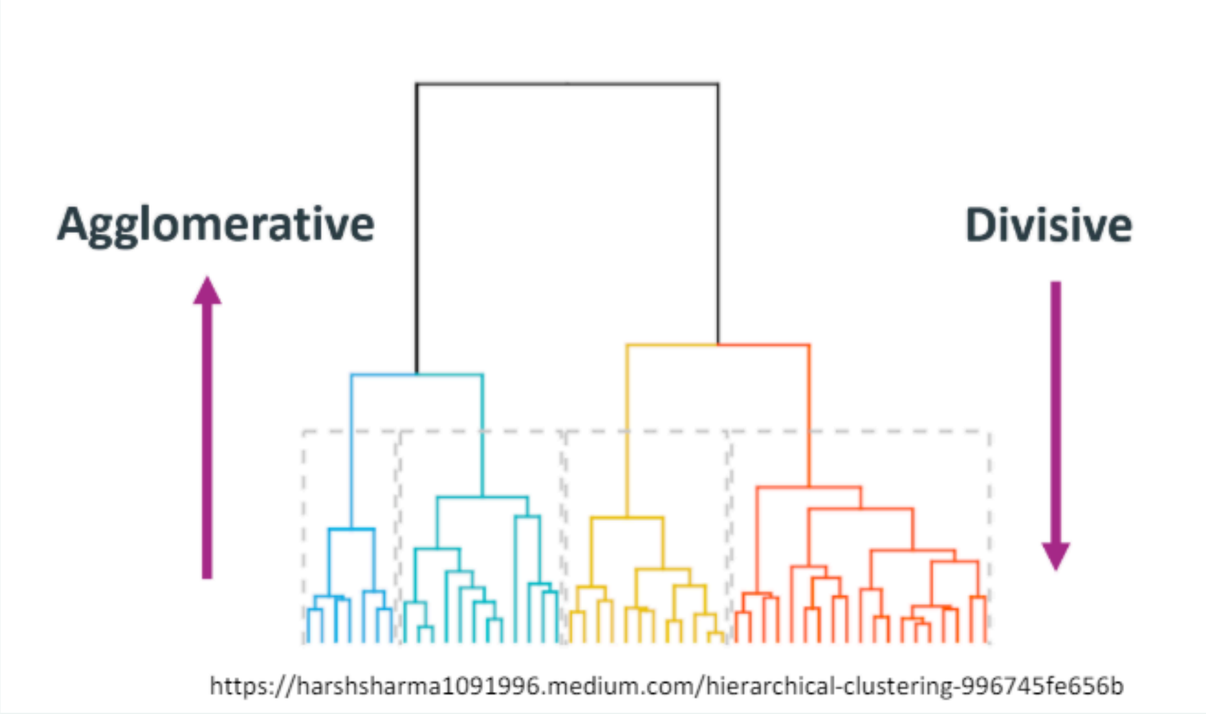
technique in unsupervised learning where the goal is to group similar objects or data points together into clusters, based on their intrinsic characteristics or features.

The main objective of clustering is to partition a dataset into subsets, or clusters, such that data points within the same cluster are more similar to each other than to those in other clusters.



HIERARCHICAL CLUSTERING

Hierarchical clustering is a clustering algorithm that creates a hierarchy of clusters. Unlike other clustering algorithms like k-means, hierarchical clustering does not require the number of clusters to be specified in advance. Instead, it creates a tree-like structure, called a dendrogram, where each node represents a cluster.



LINKAGE

Linkage refers to the criterion used to measure the distance or similarity between clusters when deciding which clusters to merge at each step of the algorithm.

TYPES OF LINKAGE IN CLUSTERING

SINGLE LINKAGE

In this method, the distance between two clusters corresponds to the minimum distance between two points in each cluster:

$$D(c_1, c_2) = \min D(x_1, x_2)$$

COMPLETE LINKAGE

In this second method, the distance between two clusters corresponds to the maximum distance between two points in each cluster:

$$D(c_1, c_2) = \max D(x_1, x_2)$$

AVERAGE LINKAGE

In this third method, the distance between two clusters corresponds to the mean distance of all pairs from each cluster:

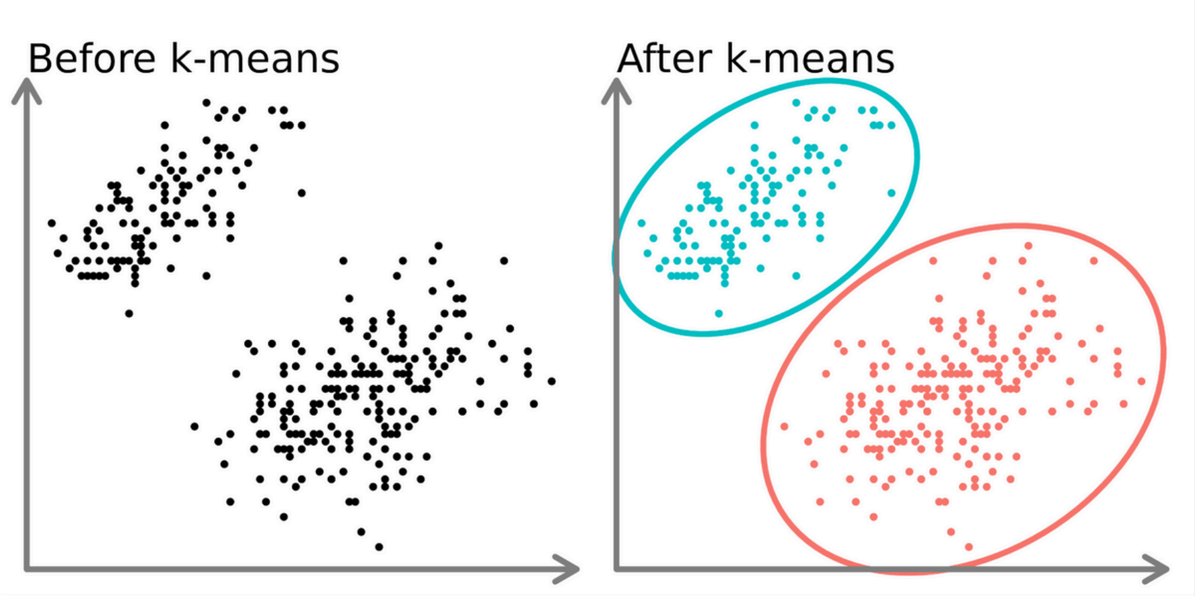
$$D(c_1, c_2) = \frac{1}{|c_1|}\frac{1}{|c_2|}\sum\sum D(x_1, x_2)$$

WARD'S LINKAGE

In this fourth method, the distance between two clusters corresponds to the increase in the sum of squares after the clusters have been merged. The goal is to minimize the total within cluster variance

K MEANS CLUSTEERING

The algorithm works by iteratively assigning data points to clusters and updating the cluster centroids until convergence. The goal of k-means clustering is to minimize the variance within clusters, meaning that data points within the same cluster are as similar to each other as possible, while data points in different clusters are as dissimilar as possible.



K MEANS CLUSTEERING ALGORITHM

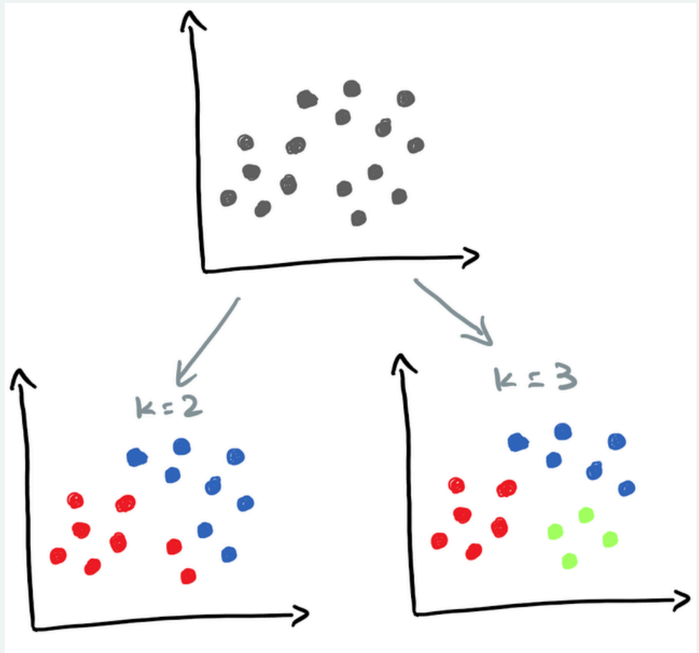
k-means clustering

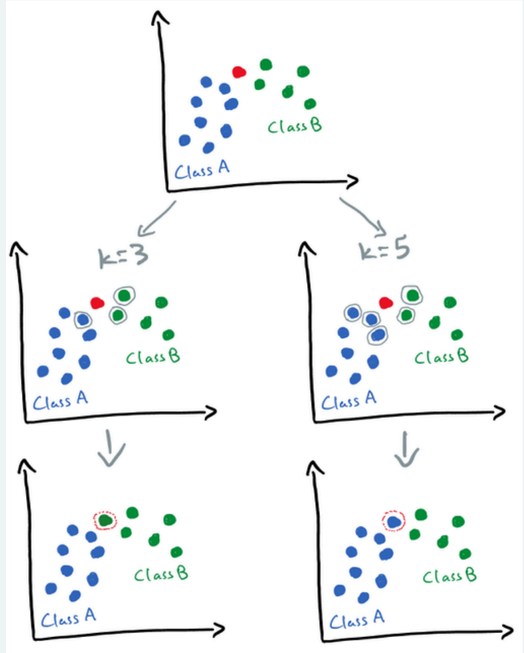
As before we wish to cluster samples $x_1, \ldots, x_n \in \mathbf{R}^p$

1) Randomly select k cluster centers $c_1, \ldots, c_k \in \mathbf{R}^p$
2) Assign each sample to the nearest cluster center
3) Update cluster centers to mean of samples in cluster
4) Repeat steps 2-3 until convergence

DIFFERENCE BETWEEN K MEAN AND K-NEAREST NEIGHBOURS

K-means Clustering	K-nearest Neighbors (KNN)
Unsupervised	Supervised
Partition data into k clusters	Predict the class of a new data point
Dataset with features only	Dataset with features and corresponding labels



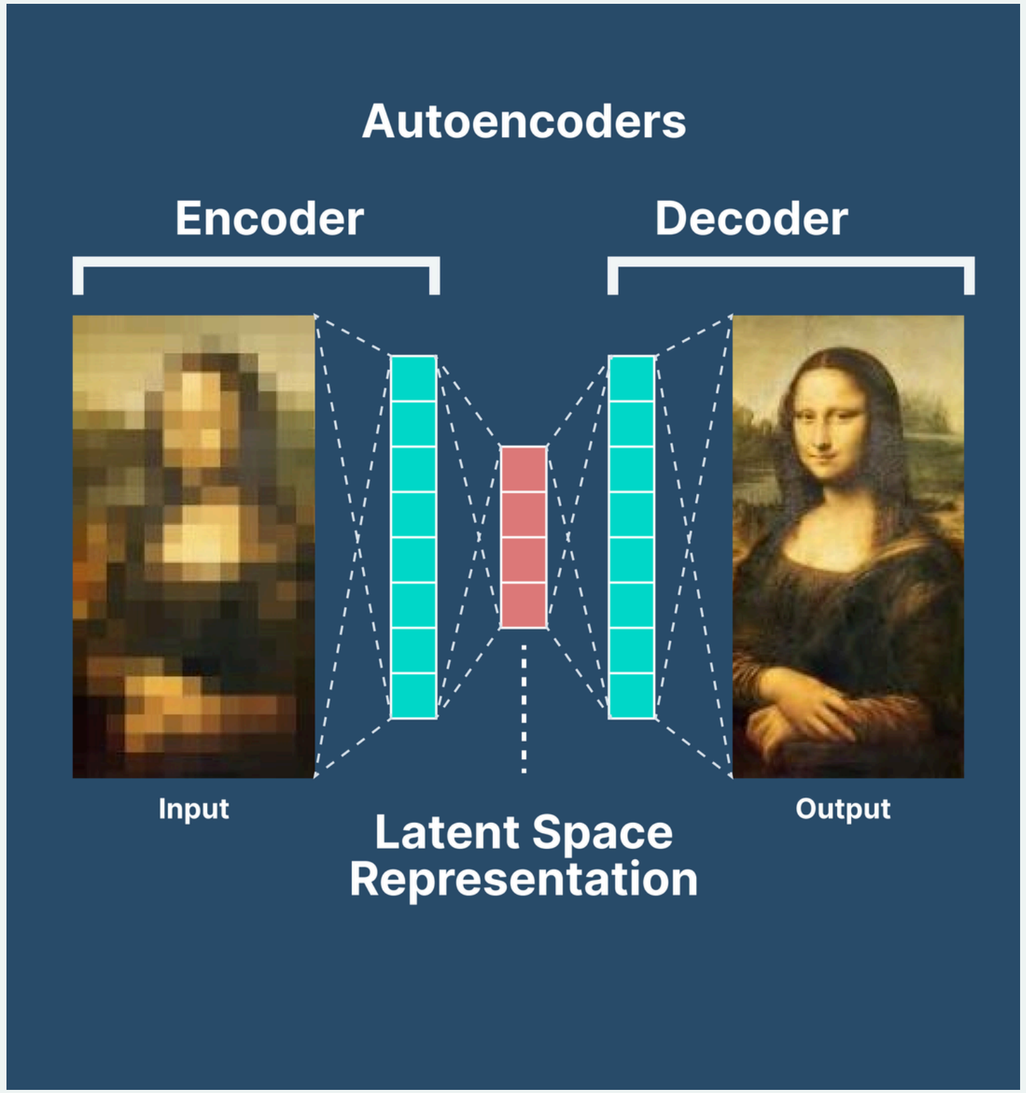


More about K means and KNN;

https://ealizadeh.com/blog/knn-and-kmeans/#:~:text=KNN%20is%20a%20supervised%20learning%20algorithm%20mainly%20used%20for%20classification,on%20the%20chosen%20distance%20metric).

AUTOENCODERS

Autoencoders are a type of artificial neural network used for unsupervised learning. They aim to learn a compressed, or encoded, representation of input data without supervision. The network learns to encode the input data into a lower-dimensional space and then decode it back to its original form.



TYPES OF AUTOENCODERS

Sparse autoencoders: trained to obtain a sparse hidden layer representation (= only a few hidden nodes active at a time)

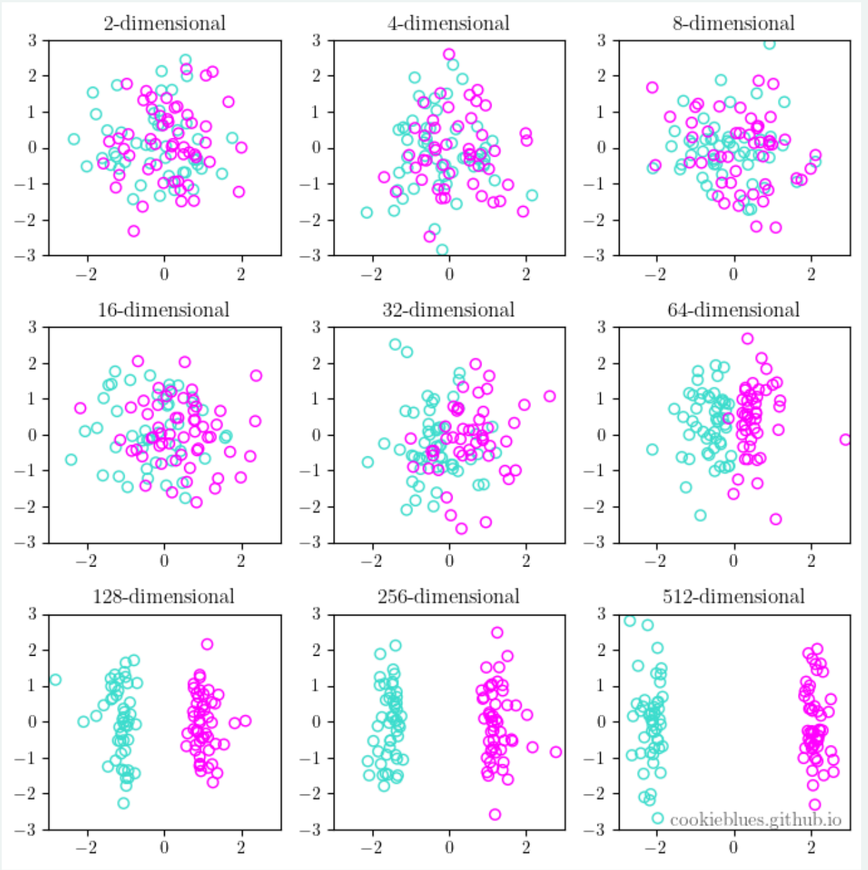
Denoising autoencoders: trained to recapture a denoised version of the input vector

Contractive autoencoders: trained to obtain a hidden layer representation that is robust to small changes in the input

Variational autoencoders: trained to obtain a representation useful also for generating new data from same distribution

CURSE OF DIMENSIONALITY

The curse of dimensionality refers to various problems that arise when working with high-dimensional data, such as increased computational complexity, sparse data distributions, and overfitting in machine learning models.



More about curse of dimensionality:

https://towardsdatascience.com/the-curse-of-dimensionality-5673118fe6d2

PCA

Principal Component Analysis (PCA) is a linear dimensionality reduction technique widely used for feature extraction and data visualization. It aims to transform high-dimensional data into a lower-dimensional space while preserving as much variance as possible.

1) Start with a data set:

$$X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ \vdots & \vdots \\ x_{n1} & x_{n2} \end{bmatrix} \begin{matrix} \text{sample 1} \\ \text{sample 2} \\ \text{.....} \\ \text{sample n} \end{matrix}$$

2) Center the data:

$$Y = \begin{bmatrix} x_{11} - m_1 & x_{12} - m_2 \\ x_{21} - m_1 & x_{22} - m_2 \\ \vdots & \vdots \\ x_{n1} - m_1 & x_{n2} - m_2 \end{bmatrix}$$

3) Find weights $\boldsymbol{w} = (w_1, w_2)$ maximizing length of $Y\boldsymbol{w}$:

$$\widehat{\boldsymbol{w}} = \underset{\|\boldsymbol{w}\|=1}{\operatorname{argmax}} \|Y\boldsymbol{w}\|$$

That's it! The vector $\widehat{\boldsymbol{w}}$ is called the 1st principal component of X.

EIGENVECTORS AND EIGENVALUES

Eigenvectors:

Imagine you have a dataset with multiple features, like height, weight, and age. An eigenvector represents a direction in this feature space. It's a special direction where, when you look at your data from that direction, it seems to change only in size, not in shape. In PCA, these directions are called principal components, and they help us understand the main patterns in our data.

Eigenvalues:

Now, imagine that each eigenvector has a number attached to it, called an eigenvalue. This number tells you how much the data spreads out along that direction (eigenvector). The bigger the eigenvalue, the more spread out the data is along that direction. So, eigenvalues help us decide which directions (or principal components) are the most important in describing our data's variability.

Detailed explanation of the concepts: https://www.youtube.com/watch?v=PFDu9oVAE-g

PCA DETAILED STEPS

1. Center the dataset

We can do so by substracting the means for each feature.

2. Compute the covariance matrix

Covariance indicates how much two variables change in relation to each other, meaning how much change in one variable causes change in another variable.

$$cov(x1, x2) = \frac{\sum_{i=1}^{n}(x1_i - \bar{x1})(x2_i - \bar{x2})}{n-1}$$

3. Compute the eigenvalues and eigenvectors

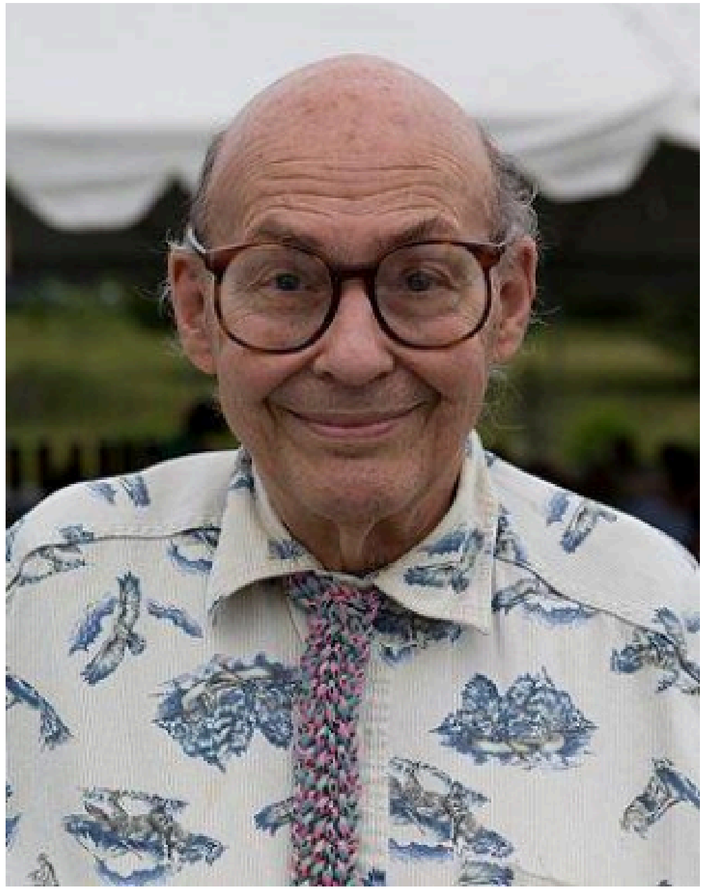
Let's say we have a matrix A, a non-zero vector X, and a scalar λ.
If the following is true:
then we say that X is an eigenvector and λ is an eigenvalue.

4. Sort the eigenvalues in descending order and the corresponding eigenvectors

We found the weights - the eigenvectors.
Now you can pick the number of principal components you want and project the data onto the principal components.

Minsky & Papert, *The perceptron* (1969):

- Showed:
  - Networks without hidden layers can only solve linearly separable problems
  - Many simple problems, like logical XOR, are not linearly separable
- Speculated
  - Networks with hidden layers are probably impossible to train
- Effect:
  - Halted the development of neural networks
- Why such an effect?
  - Minsky's position
  - A growing skepticism towards AI (funding)



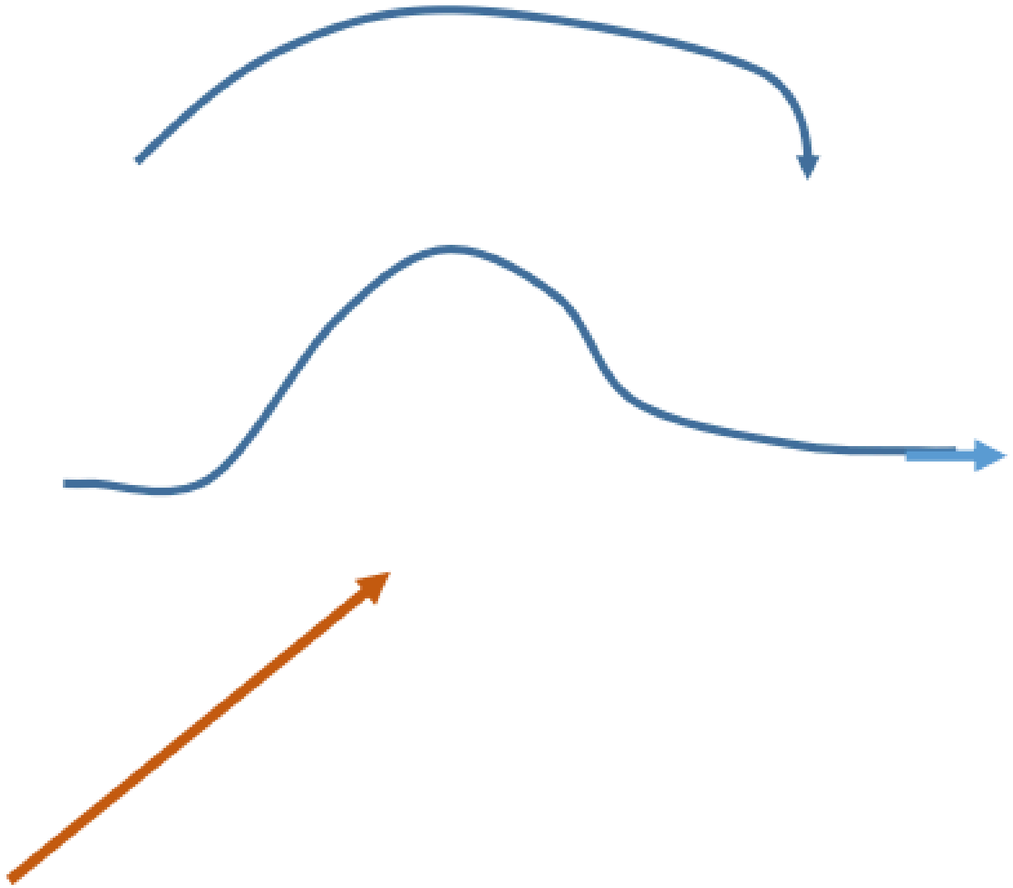
Marvin Minsky (1927-2016)
AI pioneer, MIT AI Lab

source

History of neural networks

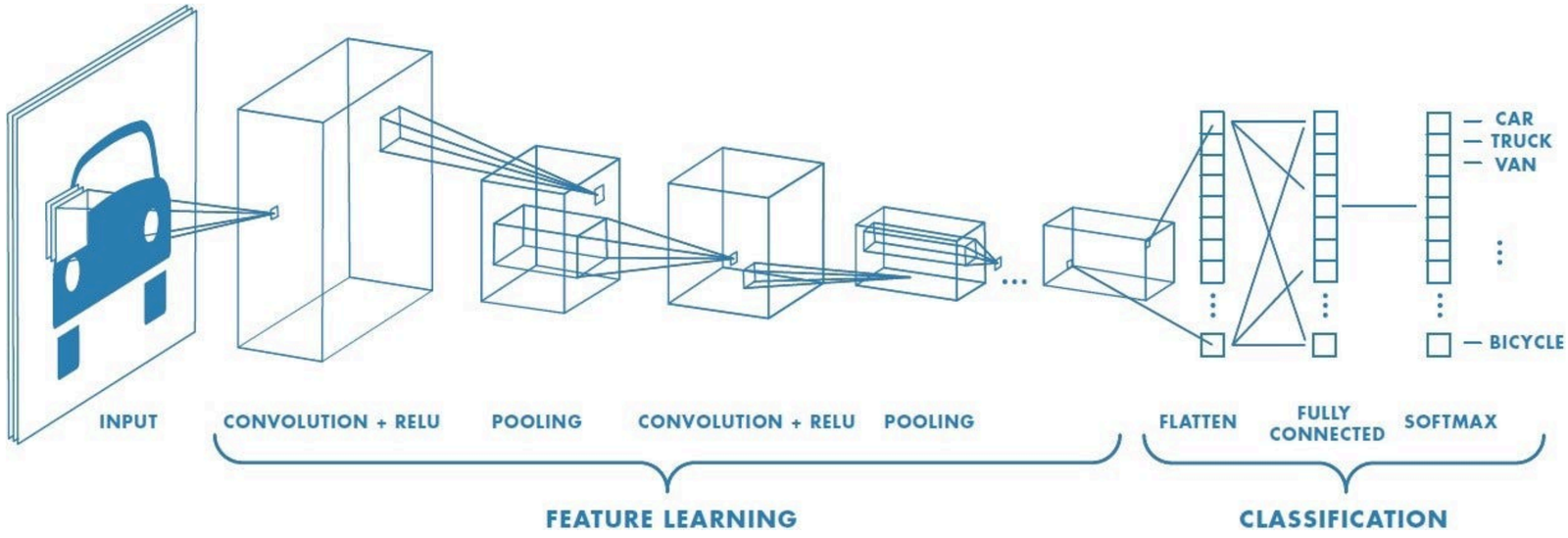
Three main epochs:

1. The beginning (→ 1969)
2. Backpropagation (1986-)
3. Deep learning (2011→)



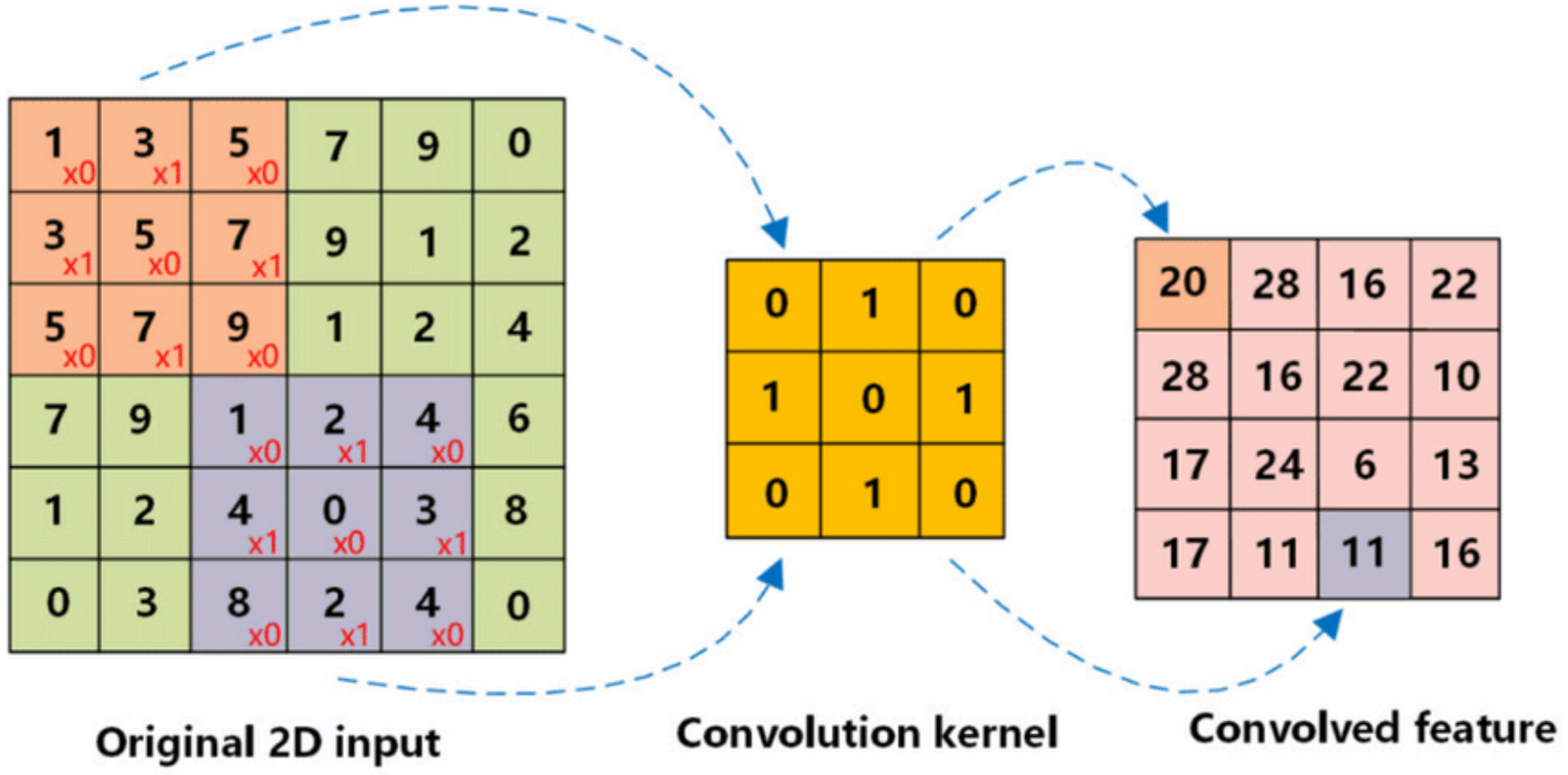
"What is a Convolutional Neural Network (CNN), and why is it widely used in image processing?"

CNN is a specialized type of neural network designed for processing structured grid data, such as images. It automatically detects patterns in images, like edges and textures, making it very effective for tasks like image classification, object detection, and facial recognition.



"What is a convolution operation in the context of CNNs?"

Convolution is a mathematical operation that combines two sets of information. In CNNs, it refers to the process of sliding a filter (or kernel) over an image to produce feature maps. This operation helps the model detect various features like edges and corners in the image.

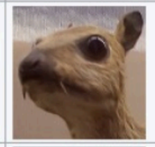
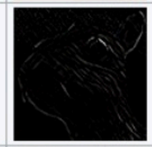
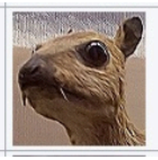
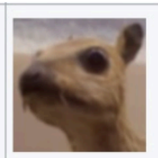
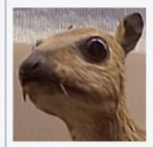


/06

"What role does a filter (or kernel) play in a CNN?"

A filter is a small matrix applied to an image to detect specific features, like edges. By sliding it across the image and performing element-wise multiplication, it helps extract critical features which are important for identifying objects in the image.

Traditional man-made filters

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

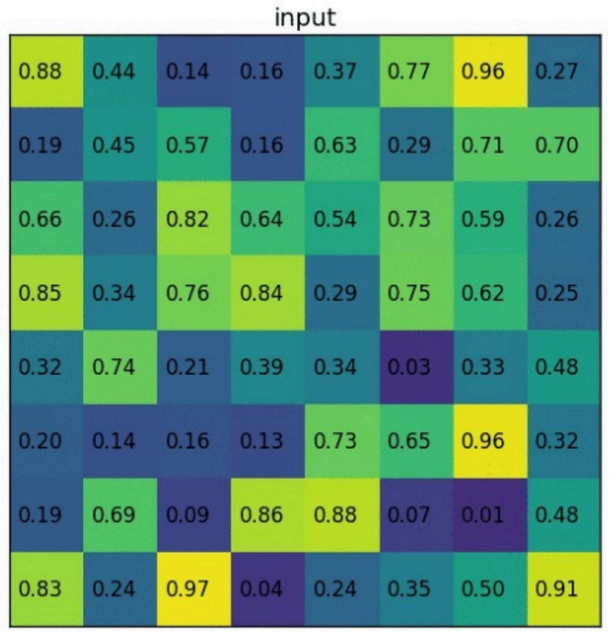
https://en.wikipedia.org/wiki/Kernel_(image_processing)

How do pooling layers contribute to CNNs, and what types are there?

Pooling layers reduce the spatial size of the feature maps, which decreases the number of parameters and computation. Common types are max pooling (selecting the maximum value in a window) and average pooling (calculating the average). This downsampling makes the model more robust to minor variations in the input.

Pooling

- Variants:
  - Max pooling
  - Average



https://en.wikipedia.org/wiki/File:RoI_pooling_animated.gif

What are residual connections, and how do they help in CNNs?

Residual connections, introduced in ResNet (Residual Networks), are shortcuts that skip one or more layers in a neural network. These connections add the input of a layer directly to its output. They help address the vanishing gradient problem in deep networks by allowing information to flow directly across layers. This makes training very deep networks more effective by enabling the network to "skip" certain layers when it’s beneficial, leading to better performance and faster convergence.