Instance Based Learning
Genetic Algorithm and
Reinforcement Learning

UNIT-5

Uzma Sulthana

UNIT-5

Instance Based Learning

- Introduction
- k-nearest neighbor learning
- Locally Weighted Regression
- radial basis function
- Case-based reasoning.

Genetic Algorithms

- Representing hypotheses
- Genetic Operators
- Fitness Function and Selection
- An Illustrative Example.

Reinforcement Learning

- Introduction
- The Learning Task
- Q Learning.

Explain the principles of Instance-Based Learning, specifically k-nearest neighbor (k-NN) learning.

What is instance-based learning? Explain the key features and disadvantages of these methods.

INSTANCE BASED LEARNING

- Introduction
- k-nearest neighbor learning
- Locally Weighted Regression
- radial basis function
- Case-based reasoning.

1. INTRODUCTION-INSTANCE BASED LEARNING

Key Idea:

- In contract to learning methods that construct a general, explicit description of the target function when training examples are provided, instance – based learning constructs the target function only when a new instance must be classified.

- Each time a new query instance is encountered, its relationship to the previously stored example is examined in order to assign a target function value of the new instance.

EXAMPLE:

Sl. No.	Height	Weight	Target
1	150	50	Medium
2	155	55	Medium
3	160	60	Large
4	161	59	Large
5	158	65	Large
6	157	54	?

- Training will not happen
- Based on relation attributes target assigned

- Used distance measure to find the relation- based on the nearest neighbor we will find the target function.

1. INTRODUCTION-INSTANCE BASED LEARNING

- Instance based learning includes **nearest neighbor** and locally weighted regression methods that assumes instance can be represented as points in Euclidean space.
- It also includes **case-based reasoning** methods that use more **complex, symbolic representation for instances.**
- Instance based methods are sometimes referred to as "**lazy**" learning methods bcz they **delay** processing until a new instance must be classified.

SUMMARY

- Instance based learning methods such as **nearest neighbor and locally weighted regression** are conceptually straightforward approaches to approximating **real-valued or discrete-valued target functions.**
- Learning in these algorithms consists of **simply storing** the presented training data. When a new query instance is encountered, a set of similar related instances are retrieved from memory and used to classify the new query instance.

ADVANTAGE & DISADVANTAGE

Advantage

- Training is very fast.
- Learn complex target function
- Don't lose information

Disadvantage

- Cost of classifying new instances can be high
- Slow at query time
- Easily fooled by irrelevant attribute

Describe K-nearest Neighbour learning Algorithm for the continuous (real) valued target function. Also, explain the distance weighted KNN algorithm for the real-valued target function. Discuss the importance of choosing the appropriate value of k. Provide a real-world example where k-NN is applicable.

2. K-NEAREST NEIGHBOR LEARNING

- The most basic instance-based method is the k-NEAREST NEIGHBOR algorithm.
- This algorithm assumes all instances correspond to points in the n-dimensional space R^n.
- The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.
- The arbitrary instance $\boldsymbol{x}$ be described by the feature vector

$$\langle a_1(x), a_2(x), \ldots a_n(x) \rangle$$

where $\boldsymbol{a_r(x)}$ denotes the value of the r^{th} attribute of instance $\boldsymbol{x}$.

- Then the distance between two instances $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ is defined to be $\boldsymbol{d(x_i, x_j)}$, where

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

- In nearest-neighbor learning the target function may be either discrete-valued or real-valued.

2. K-NEAREST NEIGHBOR LEARNING

- Discrete-valued Target Function
- Real Valued Target Function

2. K-NEAREST NEIGHBOR LEARNING

- Let us first consider learning ***discrete-valued target functions*** of the form

$$f : \Re^n \rightarrow V.$$

- Where, V is the finite set $\{v_1, \ldots v_s\}$

- The k- Nearest Neighbor algorithm for approximation a **discrete-valued target function** is given below:

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

EXAMPLE-KNN

Sl. No.	Height	Weight	Target
1	150	50	Medium
2	155	55	Medium
3	160	60	Large
4	161	59	Large
5	158	65	Large
6	157	54	?

We need to find nearest neighbor based on that assign the target label.
We will take value of K=3 for understanding purpose.(meaning find 3 nearest neighbor)

1. **First calculate** the **distance** : using formula

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

SQRT((150-157)*2+(50-54)*2)=8.06

Sl. No.	Height	Weight	Target	Distance
1	150	50	Medium	8.06
2	155	55	Medium	2.24
3	160	60	Large	6.71
4	161	59	Large	6.40
5	158	65	Large	11.05
6	157	54	?	

EXAMPLE-KNN

Sl. No.	Height	Weight	Target	Distance
1	150	50	Medium	8.06
2	155	55	Medium	2.24
3	160	60	Large	6.71
4	161	59	Large	6.40
5	158	65	Large	11.05
6	157	54	?	

2. Based on the Minimum Distance identify the k-nearest --- where k=3 given

Sl. No.	Height	Weight	Target	Distance	Nearest Points
1	150	50	Medium	8.06	
2	155	55	Medium	2.24	1
3	160	60	Large	6.71	3
4	161	59	Large	6.40	2
5	158	65	Large	11.05	
6	157	54	?		

EXAMPLE-KNN

Sl. No.	Height	Weight	Target	Distance	Nearest Points
1	150	50	Medium	8.06	
2	155	55	Medium	2.24	1
3	160	60	Large	6.71	3
4	161	59	Large	6.40	2
5	158	65	Large	11.05	
6	157	54	?		

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

values = medium.
(v)

$\hat{f}(x_q) = \delta(m, f(x_1)) + \delta(m, f(x_2)) + \delta(m, f(x_3))$

$= \delta(m, m) + \delta(m, L) + \delta(m, L)$

here x_1, x_2, x_3 are nearest points.

$= 1 + 0 + 0$

$\hat{f}(x_q) = 1$

values = Large.
(v)

$\hat{f}(x_q) = \delta(L, f(x_1)) + \delta(L, f(x_2)) + \delta(L, f(x_3))$

$= \delta(L, m) + \delta(L, L) + \delta(L, L)$

$\hat{f}(x_q) = 0 + 1 + 1$

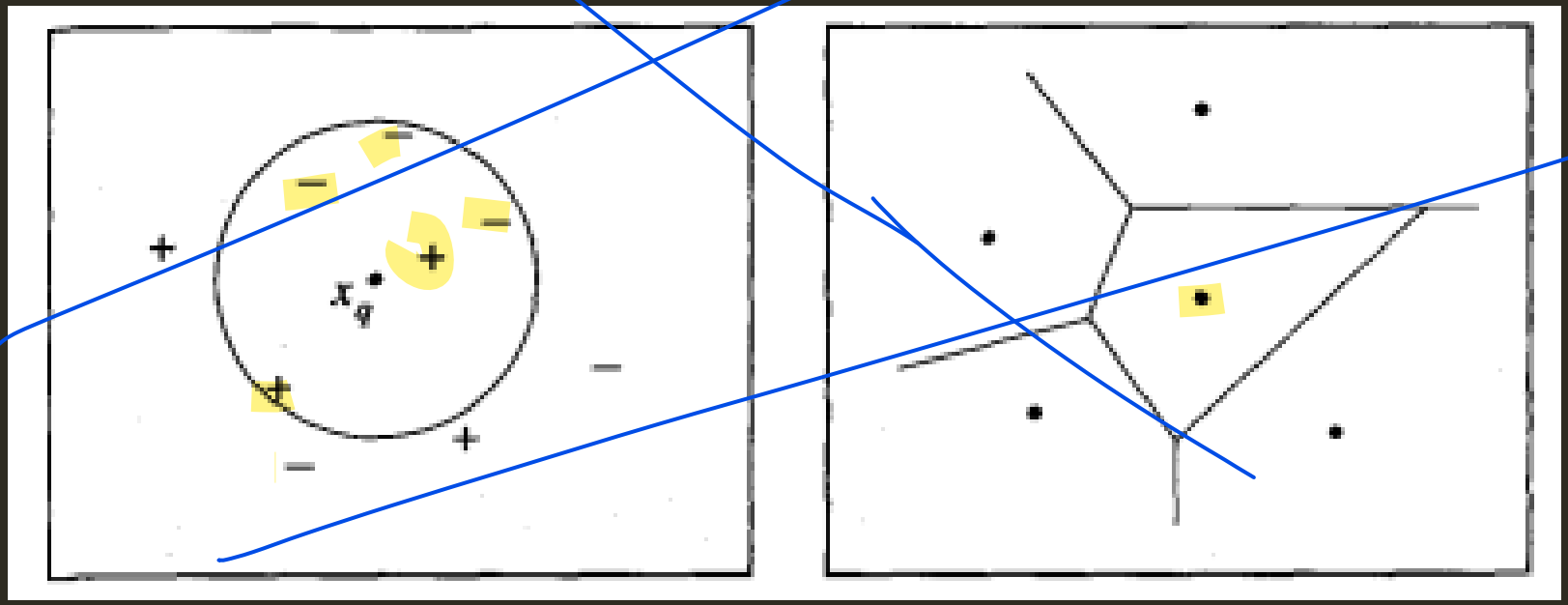
$\hat{f}(x_q) = 2$

When value = Large $\hat{f}(x_q) = 2$ > 1

So

new instance is classified as Large.

EXAMPLE-VORONOI DIAGRAM



2. K-NEAREST NEIGHBOR LEARNING

- The K- Nearest Neighbor algorithm for approximation a **real-valued target function** is given below $f : \Re^n \rightarrow \Re$

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

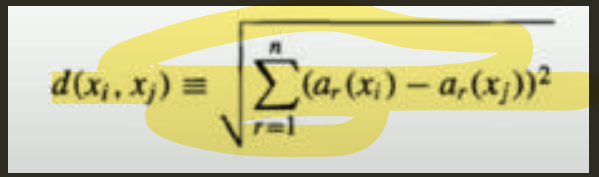
$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

EXAMPLE-KNN

Sl. No.	Height	Weight	Target
1	150	50	1.5
2	155	55	1.2
3	160	60	1.8
4	161	59	2.1
5	158	65	1.7
6	157	54	?

We need to find nearest neighbor based on that assign the target label.
We will take value of K=3 for understanding purpose.(meaning find 3 nearest neighbor)

1. **First calculate** the **distance** : using formula



SQRT((150-157)*2+(50-54)*2)=8.06

Sl. No.	Height	Weight	Target	Distance
1	150	50	1.5	8.06
2	155	55	1.2	2.24
3	160	60	1.8	6.71
4	161	59	2.1	6.40
5	158	65	1.7	11.05
6	157	54	?	

EXAMPLE-KNN

Sl. No.	Height	Weight	Target	Distance
1	150	50	1.5	8.06
2	155	55	1.2	2.24
3	160	60	1.8	6.71
4	161	59	2.1	6.40
5	158	65	1.7	11.05
6	157	54	?	

2. Based on the Minimum Distance identify the k-nearest --- where k=3 given

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Sl. No.	Height	Weight	Target	Distance	Nearest Points
1	150	50	1.5	8.06	
2	155	55	1.2	2.24	1
3	160	60	1.8	6.71	3
4	161	59	2.1	6.40	2
5	158	65	1.7	11.05	
6	157	54	?		

$$\hat{f}(x_q) = \frac{1.2 + 1.8 + 2.1}{3} = 1.7$$

DISTANCE WEIGHTED KNN ALGORITHM

- Discrete-valued Target Function
- Real Valued Target Function

EXAMPLE-CONSIDER DISTANCE AS WELL

Sl. No.	Height	Weight	Target	Distance	Nearest Points
1	150	50	Medium	8.06	
2	155	55	Medium	2.24	1
3	160	60	Large	6.71	3
4	161	59	Large	6.40	2
5	158	65	Large	11.05	
6	157	54	?		

Distance weighted kNN algorithm

Wrongly classified : It should be **Medium** but **KNN** gave us **Large**

Discrete-valued Target Function

Distance-Weighted Nearest Neighbor Algorithm

- The refinement to the k-NEAREST NEIGHBOR Algorithm is to weight the contribution of each of the k neighbors according to their distance to the query point xq, giving greater weight to closer neighbors.
- For example, in the k-Nearest Neighbor algorithm, which approximates discrete-valued target functions, we might weight the vote of each neighbor according to the inverse square of its distance from xq.
- Distance-Weighted Nearest Neighbor Algorithm for approximation a discrete-valued target functions

Discrete-valued Target Function

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

EXAMPLE

Sl. No.	Height	Weight	Target	Distance	Nearest Points
1	150	50	Medium	8.06	
2	155	55	Medium	2.24	1
3	160	60	Large	6.71	3
4	161	59	Large	6.40	2
5	158	65	Large	11.05	
6	157	54	?		

Calculate wi

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

Sl. No.	Height	Weight	Target	Distance	1/distance^2	Nearest Points
1	150	50	Medium	8.06		
2	155	55	Medium	2.24	0.45	1
3	160	60	Large	6.71	0.15	3
4	161	59	Large	6.40	0.16	2
5	158	65	Large	11.05		
6	157	54	?			

EXAMPLE

Sl. No.	Height	Weight	Target	Distance	1/distance^2	Nearest Points
1	150	50	Medium	8.06		
2	155	55	Medium	2.24	0.45	1
3	160	60	Large	6.71	0.15	3
4	161	59	Large	6.40	0.16	2
5	158	65	Large	11.05		
6	157	54	?			

Value = medium

$f'(nq) = w_1 * \delta(m_1, f/n_1)) + w_2 * \delta(m, f/n_2)) + w_3 * \delta(m, f/n_3))$

$= 0.45 * \delta(m, m) + 0.16 * \delta(m, L) + 0.15 * \delta(m, L)$

$= 0.45 * 1 + 0 + 0$

$f'(nq) = 0.45$

Value = Large

$f'(nq) = w_1 * \delta(L, f/n_1)) + w_2 * \delta(L, f/n_2)) + w_3 * (L, f/n_3))$

$= 0.45 * \{\delta(L, m) + 0.16 * (\delta(L, L + 0.15 * \delta(L, L)$

$= 0 + 0.15 + 0.16$

$f'(nq = 0.31$

medium is highest so labeled as medium.

Real Valued Target Function

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

EXAMPLE

Sl. No.	Height	Weight	Target	Distance	Nearest Points
1	150	50	1.5	8.06	
2	155	55	1.2	2.24	1
3	160	60	1.8	6.71	3
4	161	59	2.1	6.40	2
5	158	65	1.7	11.05	
6	157	54	?		

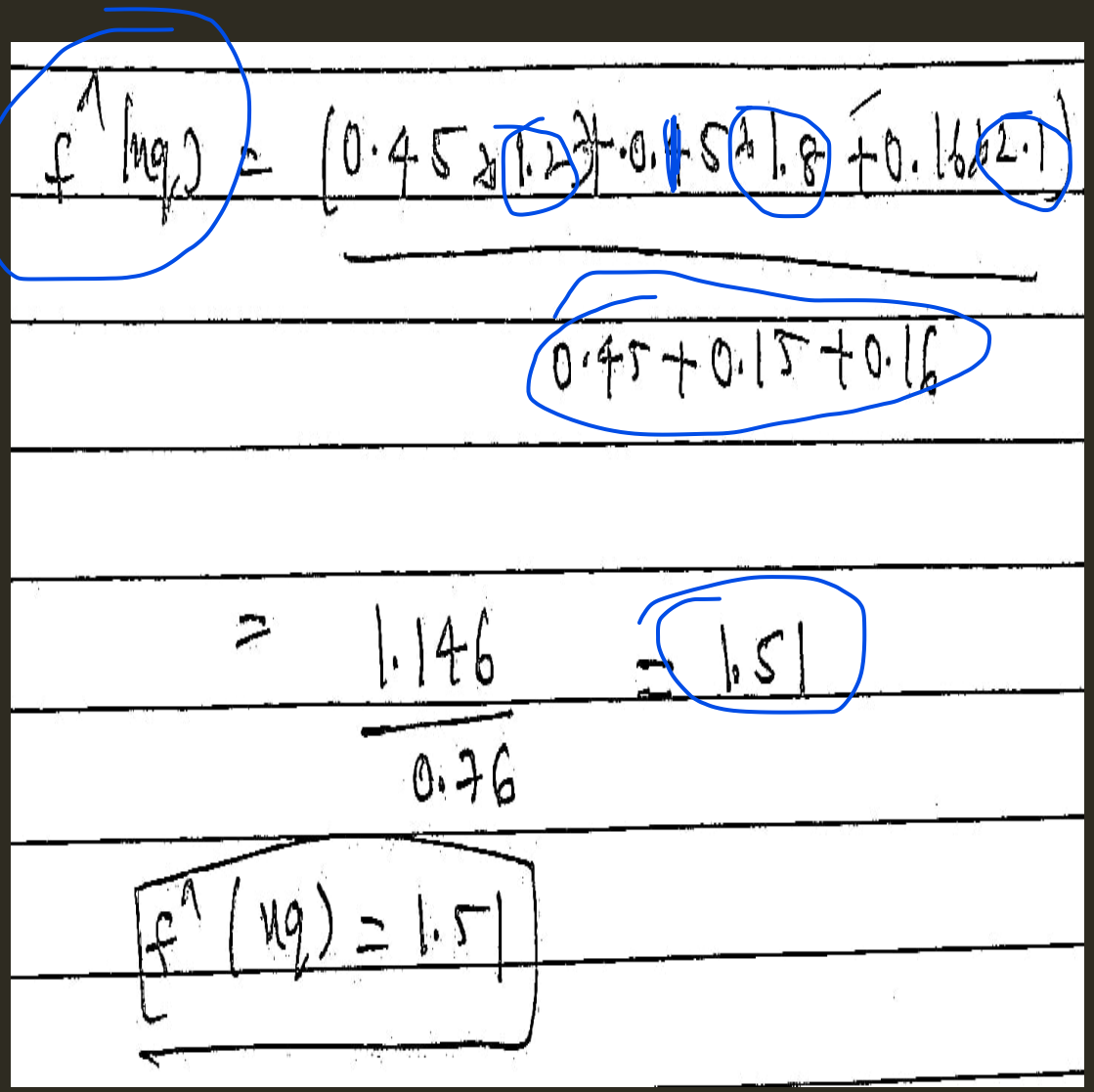
Calculate wi

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

Sl. No.	Height	Weight	Target	Distance	1/distance^2	Nearest Points
1	150	50	1.5	8.06		
2	155	55	1.2	2.24	0.45	1
3	160	60	1.8	6.71	0.15	3
4	161	59	2.1	6.40	0.16	2
5	158	65	1.7	11.05		
6	157	54	?			

EXAMPLE

Sl. No.	Height	Weight	Target	Distance	1/distance^2	Nearest Points
1	150	50	1.5	8.06		
2	155	55	1.2	2.24	0.45	1
3	160	60	1.8	6.71	0.15	3
4	161	59	2.1	6.40	0.16	2
5	158	65	1.7	11.05		
6	157	54	?			



3. LOCALLY WEIGHTED REGRESSION

LOCALLY WEIGHTED REGRESSION

- Locally weighted regression is instance based learning algorithm.
- The phrase "**locally weighted regression**" is called
  - **local** because the function is approximated based only on data near the query point,
  - **weighted** because the contribution of each training example is weighted by its distance from the query point, and
  - **regression** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

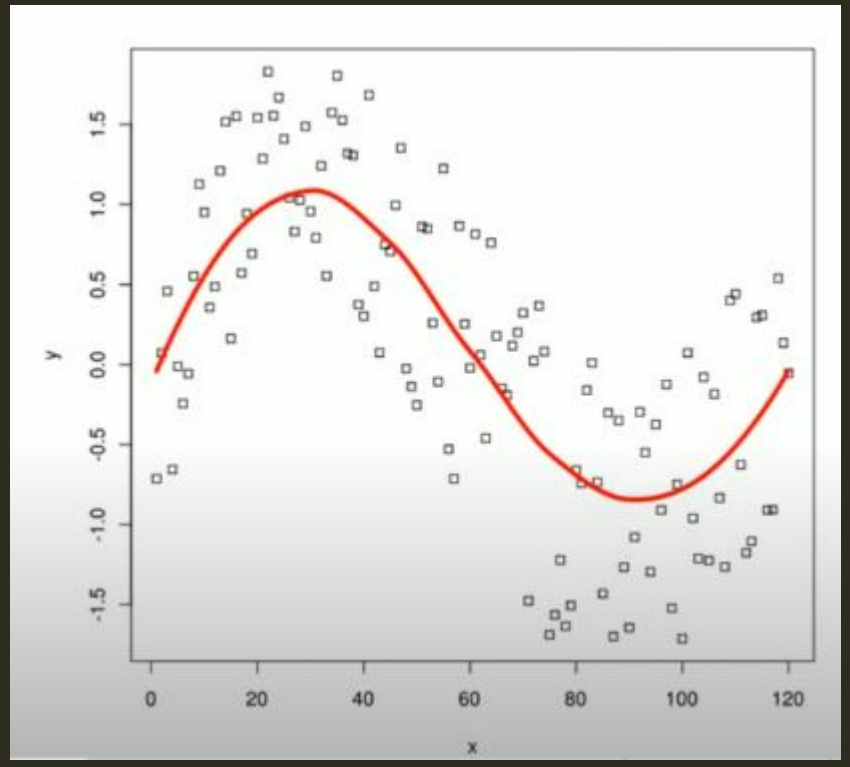
LOCALLY WEIGHTED REGRESSION

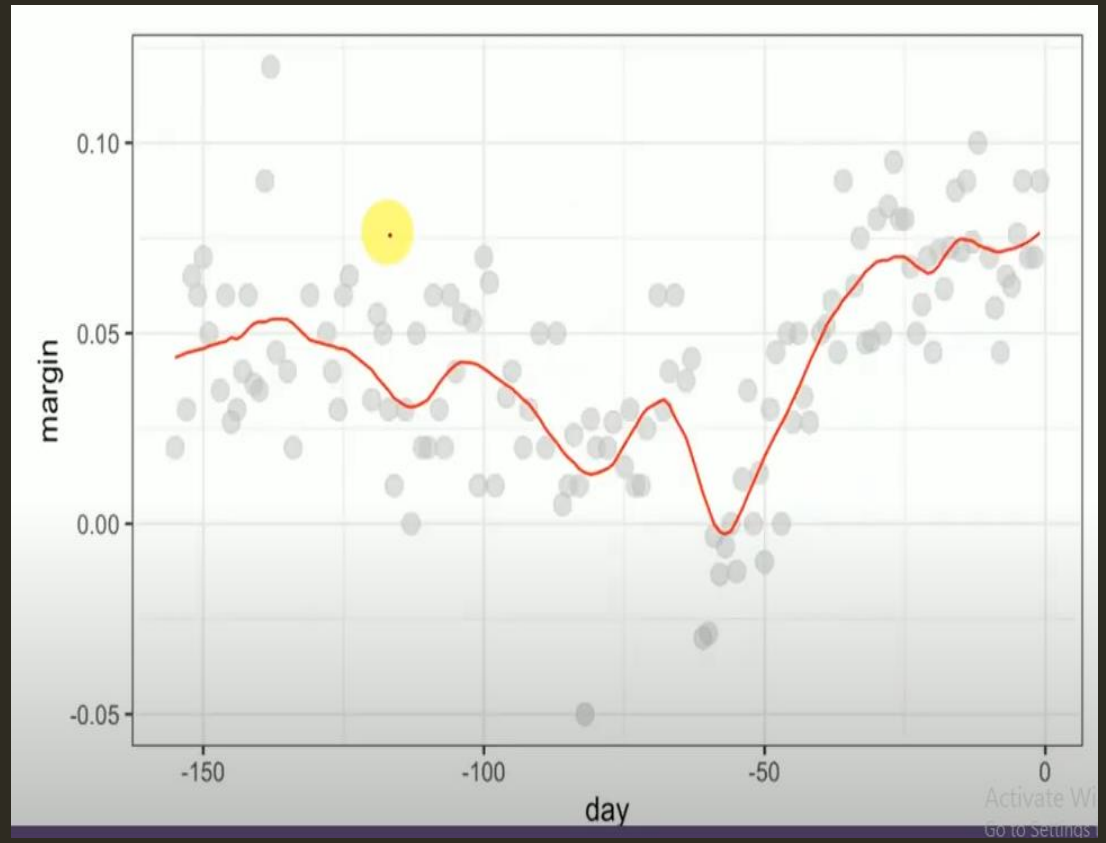
- Given a new query instance x_q, the general approach in locally weighted regression is to construct an approximation $\hat{f}$ that fits the training examples in the neighborhood surrounding x_q.

- This approximation is then used to calculate the value $\hat{f}(x_q)$, which is output as the estimated target value for the query instance.

EXAMPLE

Find a line (Straight line or curve) which will fit to this particular data.





WORKING OF THE ALGORITHM

- Consider locally weighted regression in which the target function ***f*** is approximated near x_q using a linear function of the form

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

- Where, $a_i(x)$ denotes the value of the i^{th} attribute of the instance x

- Gradient descent of ANN

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\, x_{id}$$

WORKING OF THE ALGORITHM

- Derived methods are used to choose weights that minimize the squared error summed over the set D of training examples using gradient descent

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

- Which led us to the gradient descent training rule

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x)$$

WORKING OF THE ALGORITHM

Need to modify this procedure to derive a local approximation rather than a global one. The simple way is to redefine the error criterion E to emphasize fitting the local training examples. Three possible criteria are given below.

1. Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from x_q:

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

3. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

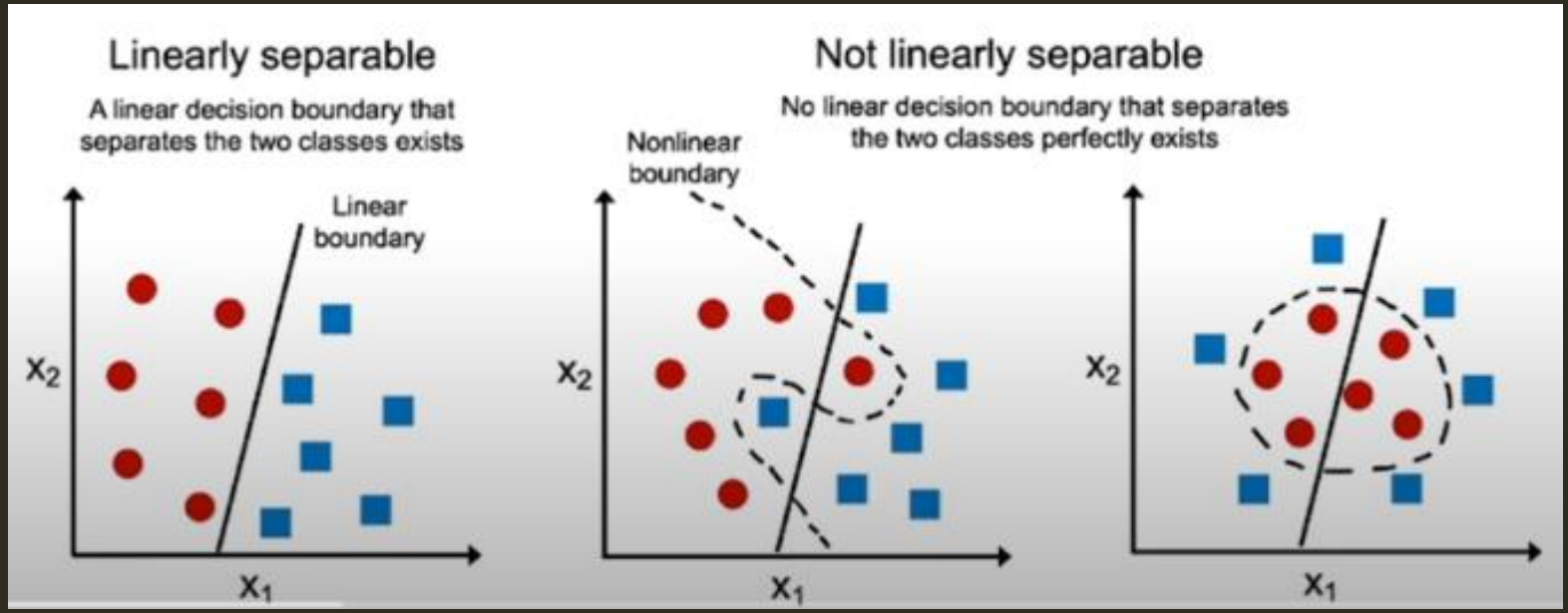
WORKING OF THE ALGORITHM

- If we choose criterion three and re-derive the gradient descent rule, we obtain the following training rule

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x))\ (f(x) - \hat{f}(x))\ a_j(x) \qquad (8.7)$$

4. RADIAL BASIS FUNCTION

- Data can be either linearly Separable or Non-Linearly Separable

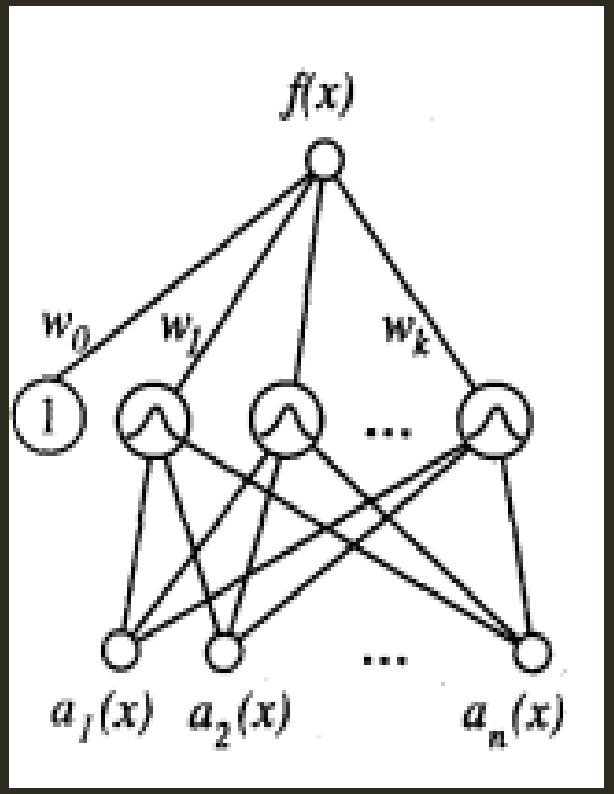


- Single layer perceptron can be used for classifying linearly Separable data.
- Multi layer perceptron is required for classifying Non-linearly Separable data.

4. RADIAL BASIS FUNCTION

- Radial Basis Function is a type of Multi Layer Perceptron which has one input layer, one output layer and with strictly one hidden layer.

- The hidden layer uses a non-linear radial basis function as the activation function, which converts the input parameters into high dimension space which is then fed into the network to linearly separate the problem.



4. RADIAL BASIS FUNCTION

This neural network is useful for

- interpolation,
- function approximation,
- time series prediction,
- classification and
- system control.

4. RADIAL BASIS FUNCTION

- Typical Radial Basis Functions (RBF) are:

- The **Gaussian RBF** which monotonically decreases with distance from the center.

$$H(x) = e^{\frac{-(x-c)^2}{r^2}}$$

- where, c is the center and r is the radius.

- A **Multiquadric RBF** which monotonically increases with distance from the center.

$$H(x) = \sqrt{\frac{r^2 + (x-c)^2}{r}}$$

4. RADIAL BASIS FUNCTION

In this approach, the learned hypothesis is a function of the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

- where **xu** is an instance from X and
- where the kernel function K,(d(x,, x)) is defined so that it decreases as the distance d(x,, x) increases.
- Here k is a user provided constant that specifies the number of kernel functions to be included.

$$K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

5. CASE-BASED REASONING.

- **Instance-based methods** such as k-NEAREST NEIGHBOR and locally weighted regression share three key properties.
- **First**, they are lazy learning methods in that they defer the decision of how to generalize beyond the training data until a new query instance is observed.
- **Second**, they classify new query instances by analyzing similar instances while ignoring instances that are very different from the query.
- **Third**, they represent instances **as** real-valued points in an n-dimensional Euclidean space.

5. CASE-BASED REASONING.

- In CBR the instances are not represented as real-valued points, but instead, they use a ***rich symbolic*** representation and the methods used to retrieve similar instances are correspondingly more elaborate.

- CBR has been applied to problems such as
  - conceptual design of mechanical devices based on a stored library of previous designs,
  - reasoning about new legal cases based on previous rulings, and

5. CASE-BASED REASONING.

Case-based reasoning consists of a cycle of the following four steps:

1. **Retrieve -** Given a new case, retrieve similar cases from the case base.
2. **Reuse -** Adapt the retrieved cases to fit to the new case.
3. **Revise -** Evaluate the solution and revise it based on how well it works.
4. **Retain -** Decide whether to retain this new case in the case base.

5. EXAMPLE

Example:

- Help desk that users call with problems to be solved.
- When users give a description of a problem, the closest cases in the case base are retrieved.
- The diagnostic assistant could recommend some of these to the user, adapting each case to the user's particular situation.
- If one of the adapted cases works, that case is added to the case base, to be used when another user asks a similar question.
- If none of the cases found works, some other method is attempted to solve the problem, perhaps by adapting other cases or having a human help diagnose the problem.
- When the problem is finally solved, the solution is added to the case base.

5. PROTOTYPICAL EXAMPLE OF A CASE-BASED

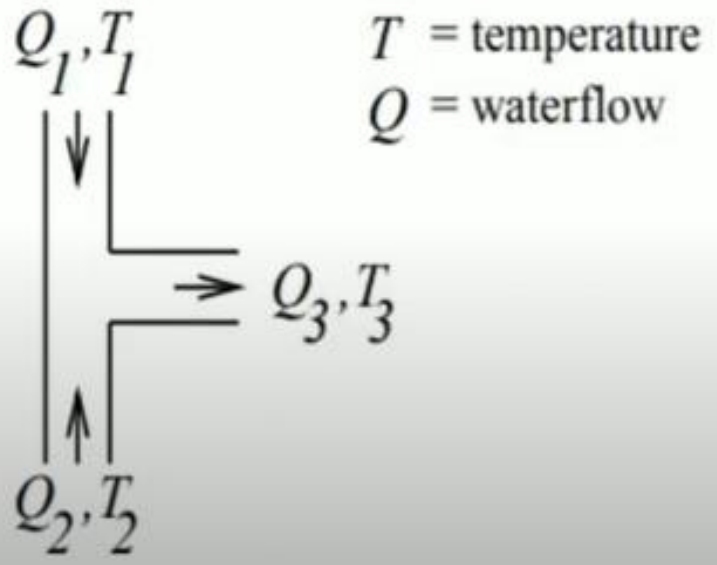
- The CADET system employs case-based reasoning to assist in the conceptual design of simple mechanical devices such as **water faucets.**
- It uses a library containing approximately **75 previous designs and design fragments** to suggest conceptual designs to meet the specifications of new design problems.
- Each instance stored in memory (e.g., a water pipe) is represented by describing both its structure and its qualitative function.
- New design problems are then presented by specifying the desired function and requesting the corresponding structure.

5. PROTOTYPICAL EXAMPLE OF A CASE-BASED

- The problem setting is illustrated in below figure

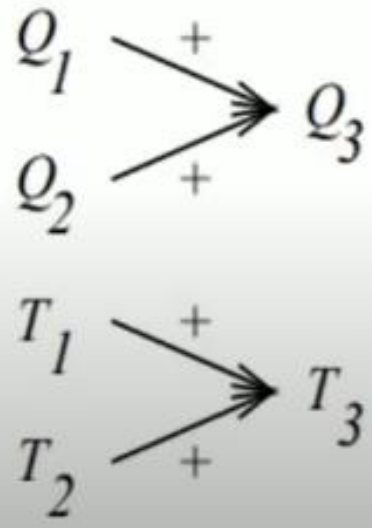
A stored case: T−junction pipe

Structure:



T = temperature
Q = waterflow

Function:



6. REMARKS ON LAZY AND EAGER ALGORITHM

Lazy Learning:

- Simple stores training data and waits until it gets test tuple
- Less training time, more prediction time
- Ex: All instances based learning algorithms

Eager Learning:

- When we give a training set, it constructs a model for classification before getting new example
- More training time, less prediction time
- Ex: Decision tree, Naïve Bayes, ANN etc.

GENETIC ALGORITHMS

Genetic Algorithms

- Representing hypotheses
- Genetic Operators
- Fitness Function and Selection
- An Illustrative Example.

GENETIC ALGORITHM

- It is a adaptive heuristic search algorithm
- Genetic algorithms provide an approach to learning that is based loosely on simulated evolution.
- The algorithm reflects the process of natural selection where the fittest individuals are selected for reduction in order to produce offspring of the next generation.
- Application : Image processing, Designing electronic circuits, code-breaking, and artificial creativity.

GENETIC ALGORITHM-MOTIVATION

- Genetic algorithms (GAS) provide a learning method motivated by an analogy to biological evolution.
- Rather than search from general-to-specific hypotheses, or from simple-to-complex, GAS generate successor hypotheses by repeatedly mutating and recombining parts of the best currently known hypotheses.
- At each step, a collection of hypotheses called the current population is updated by replacing some fraction of the population by offspring of the most fit current hypotheses.
- The process forms a generate-and-test beam-search of hypotheses, in which variants of the best current hypotheses are most likely to be considered next.

GENETIC ALGORITHM-MOTIVATION

- **The popularity of GAS is motivated by a number of factors including:**

➢ **Evolution** is known to be a successful, robust method for adaptation within biological systems.

➢ GAS can search spaces of hypotheses containing complex interacting parts, where the impact of each part on overall hypothesis fitness may be difficult to model.

➢ Genetic algorithms are easily parallelized and can take advantage of the decreasing costs of powerful computer hardware.

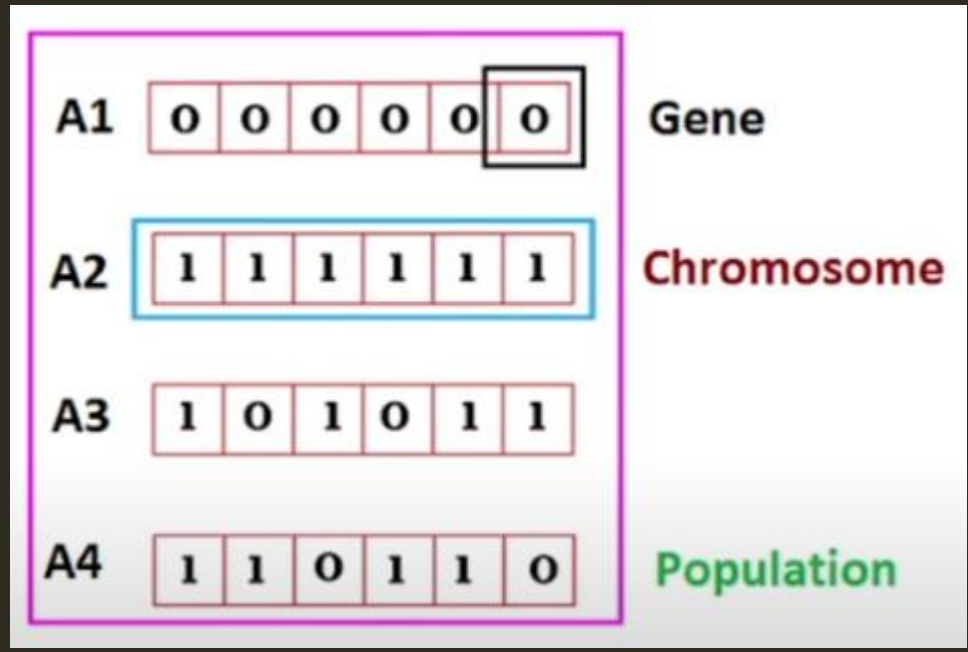
GENETIC ALGORITHM

There are five phases in Genetic Algorithm:

1. Initialization
2. Fitness evaluation
3. Selection
4. Crossover
5. Termination

GENETIC ALGORITHM

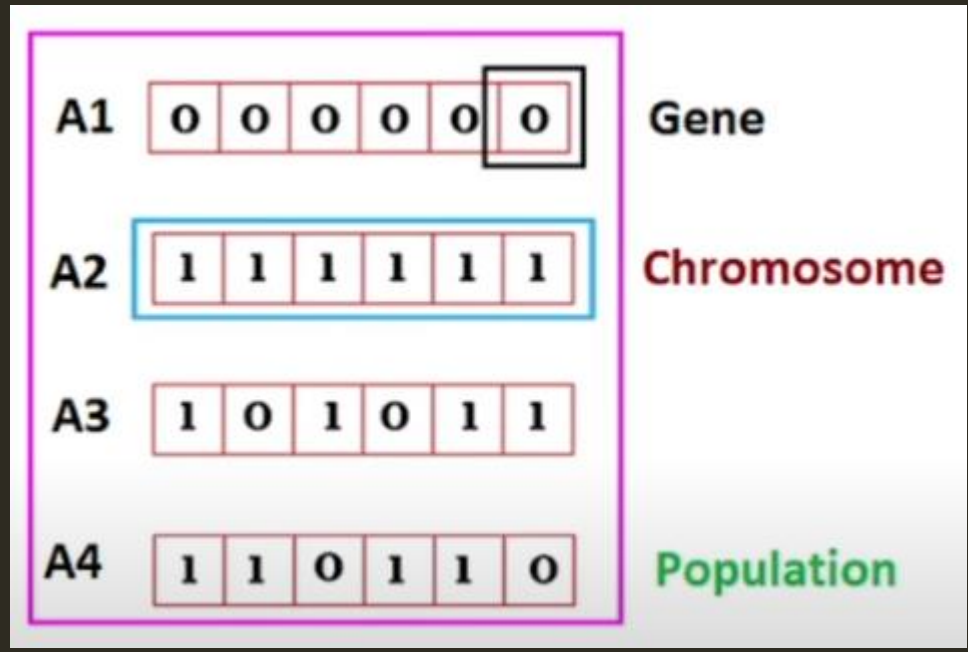
1. **Initialization:**
   - The process begins with a set of Individuals which is called population
   - Each individual is a solution to the problem you want to solve known as chromosome.
   - An individual is characterized by a set of parameters(variables) known as genes
   - Genes are joined into a string to form a chromosome(solution).



GENETIC ALGORITHM

2. Fitness evaluation:

- The fitness function determines how fit an individual is ? (the ability of an individual to compete with other individuals).
- It gives a fitness score to each individual
- The probability that an individual will be selected for reproduction is based on its fitness score



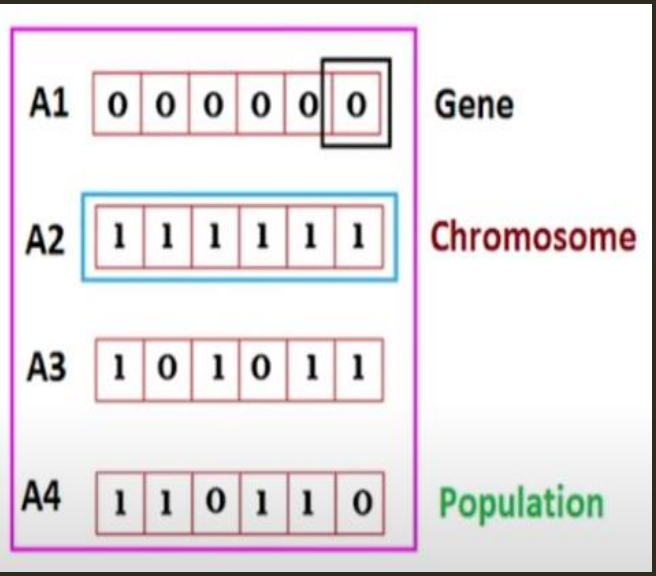
GENETIC ALGORITHM

3. Selection:

- **The idea of selection phase is to select the fittest individuals and let them pass their genes to the next generation.**
- **Two pairs of individuals (parents) are selected based on their fitness scores.**
- **Individuals with high fitness have more chance to be selected for reproduction.**
- Probabilistically select (1 - ***r)p*** members of ***P*** to add to *Ps*.
- The probability Pr(hi) of selecting hypothesis ***hi*** from ***P*** is given by

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

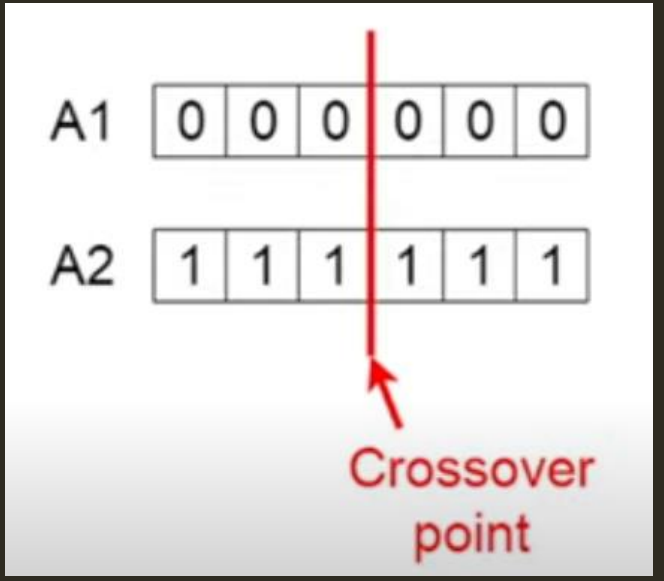
- **Roulette wheel selection, Tournament selection and Rank-based selection.**



GENETIC ALGORITHM

4. Crossover:

- Crossover is the most significant phase in a genetic algorithm
- For each pair of parents to be made, a crossover point is chosen at random from within the genes.
- For example, consider the crossover point to be 3 as shown

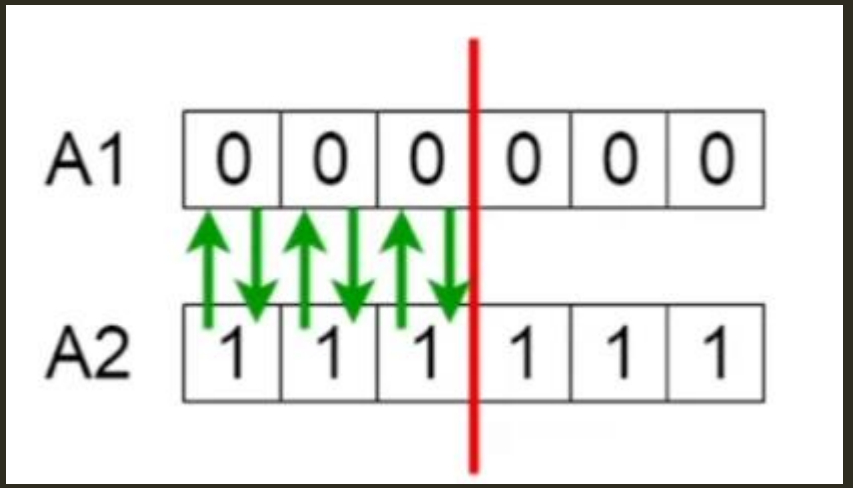


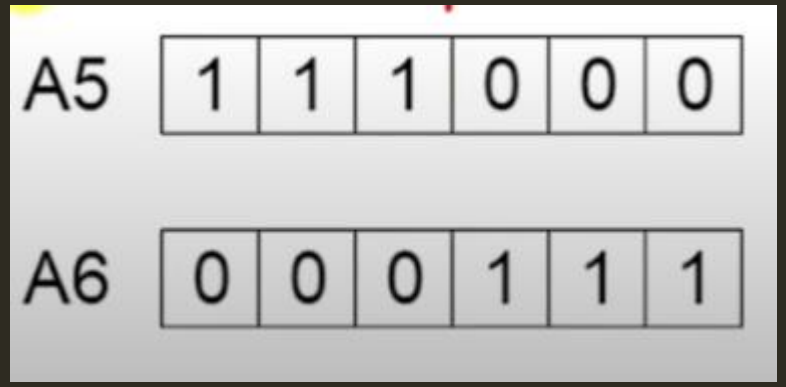
GENETIC ALGORITHM

Offspring:

- Offspring are created by exchanging the genes of parents among themselves until the crossover point is reached.

- New offspring are added to the population.



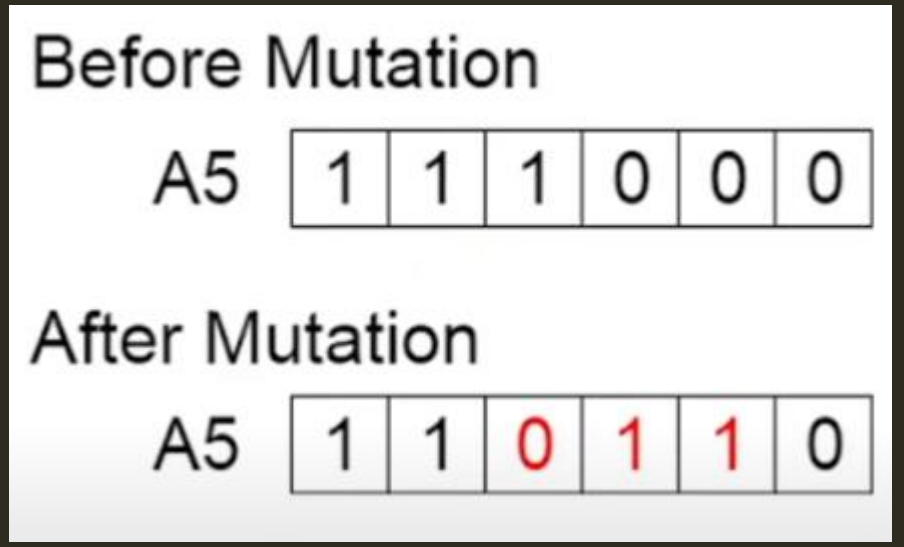


GENETIC ALGORITHM

Mutation:

- In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability
- This implies that some of the bits in the bit string can be flipped.

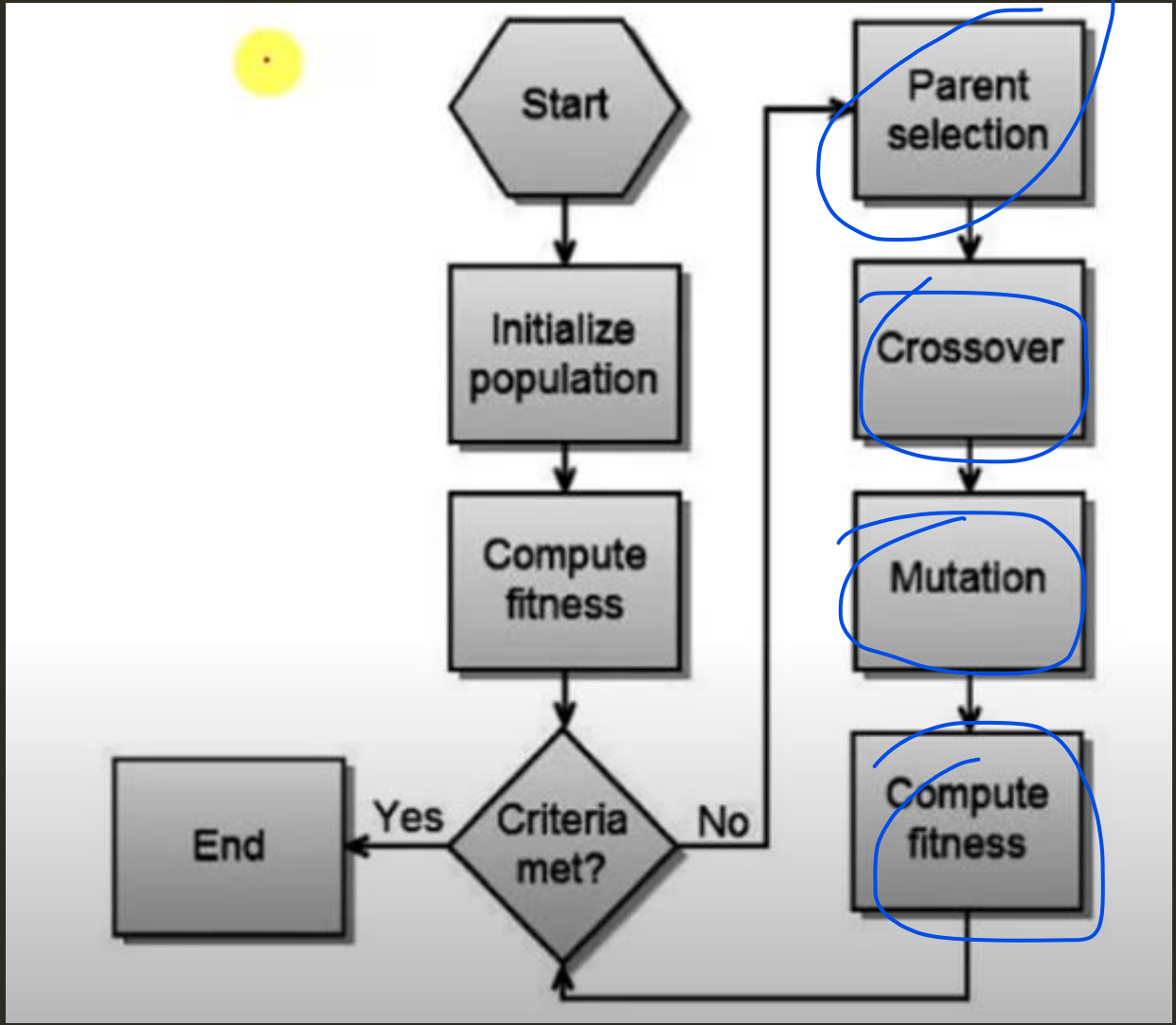
- Flip bit mutation
- Gaussian mutation
- Exchange/Swap mutation



GENETIC ALGORITHM

5. Termination:

- **The algorithm terminates if the population has converged (Does not produce offspring which are significantly different from the previous generation.)**


Start
Initialize population
Compute fitness
Criteria met?
Yes
End
No
Parent selection
Crossover
Mutation
Compute fitness

GENETIC ALGORITHM

- The problem addressed by generic algorithms is to search a space of candidate hypothesis to identify the best hypothesis
- Hypothesis fitness:

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

GENETIC ALGORITHM

- Working of genetic algorithm:

i) Operates iteratively by updating a pool of hypotheses (Population).

ii) On each iteration, all members of the population are evaluated according to the fitness function.

iii) A new population is then generated by probabilistically selecting the most fit individuals from the current population.

iv) Selected hypothesis are carried out to next population.

v) Others are used as the basis for creating new offspring individuals by applying genetic operations such as crossover and mutation.

GA($Fitness$, $Fitness\_threshold$, p, r, m)

Fitness: A function that assigns an evaluation score, given a hypothesis.

$Fitness\_threshold$: A threshold specifying the termination criterion.

p: The number of hypotheses to be included in the population.

r: The fraction of the population to be replaced by Crossover at each step.

m: The mutation rate.

- *Initialize population:* $P \leftarrow$ Generate p hypotheses at random
- *Evaluate:* For each h in P, compute $Fitness(h)$
- While $[\max_h Fitness(h)] < Fitness\_threshold$ do

  Create a new generation, P_S:

  1. *Select:* Probabilistically select $(1 - r)p$ members of P to add to P_S. The probability $\Pr(h_i)$ of selecting hypothesis h_i from P is given by

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

  2. *Crossover:* Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from P, according to $\Pr(h_i)$ given above. For each pair, $\langle h_1, h_2 \rangle$, produce two offspring by applying the Crossover operator. Add all offspring to P_s.
  3. *Mutate:* Choose m percent of the members of P_s with uniform probability. For each, invert one randomly selected bit in its representation.
  4. *Update:* $P \leftarrow P_s$.
  5. *Evaluate:* for each h in P, compute $Fitness(h)$

- Return the hypothesis from P that has the highest fitness.

REPRESENTING HYPOTHESES

- Hypotheses in Genetic Algorithms are often represented by bit strings.

Ex: Attribute Outlook – sunny, overcast, rain

outlook=sunny bitstring – 100

outlook=overcast bitstring – 010

outlook=(overcast v rain) bitstring – 011

If bitstring is 111 then most general case.

REPRESENTING HYPOTHESES

A rule precondition such as $(Outlook = Overcast \vee Rain) \wedge (Wind = Strong)$ can then be represented by the following bit string of length five:

Outlook	*Wind*
011	10

- Attribute wind – strong, weak

  wind=strong bit string=10

IF-THEN rule:

Assume only two attributes in playtennis table

IF wind=strong THEN Playtennis=YES bitstring = 111 10 10

if bitstring is 111 10 11 then ambiguity is present.

To overcome above use only one bit for output.

REPRESENTING HYPOTHESES

Rule postconditions (such as $PlayTennis = yes$) can be represented in a similar fashion. Thus, an entire rule can be described by concatenating the bit strings describing the rule preconditions, together with the bit string describing the rule postcondition. For example, the rule

IF $Wind = Strong$ THEN $PlayTennis = yes$

would be represented by the string

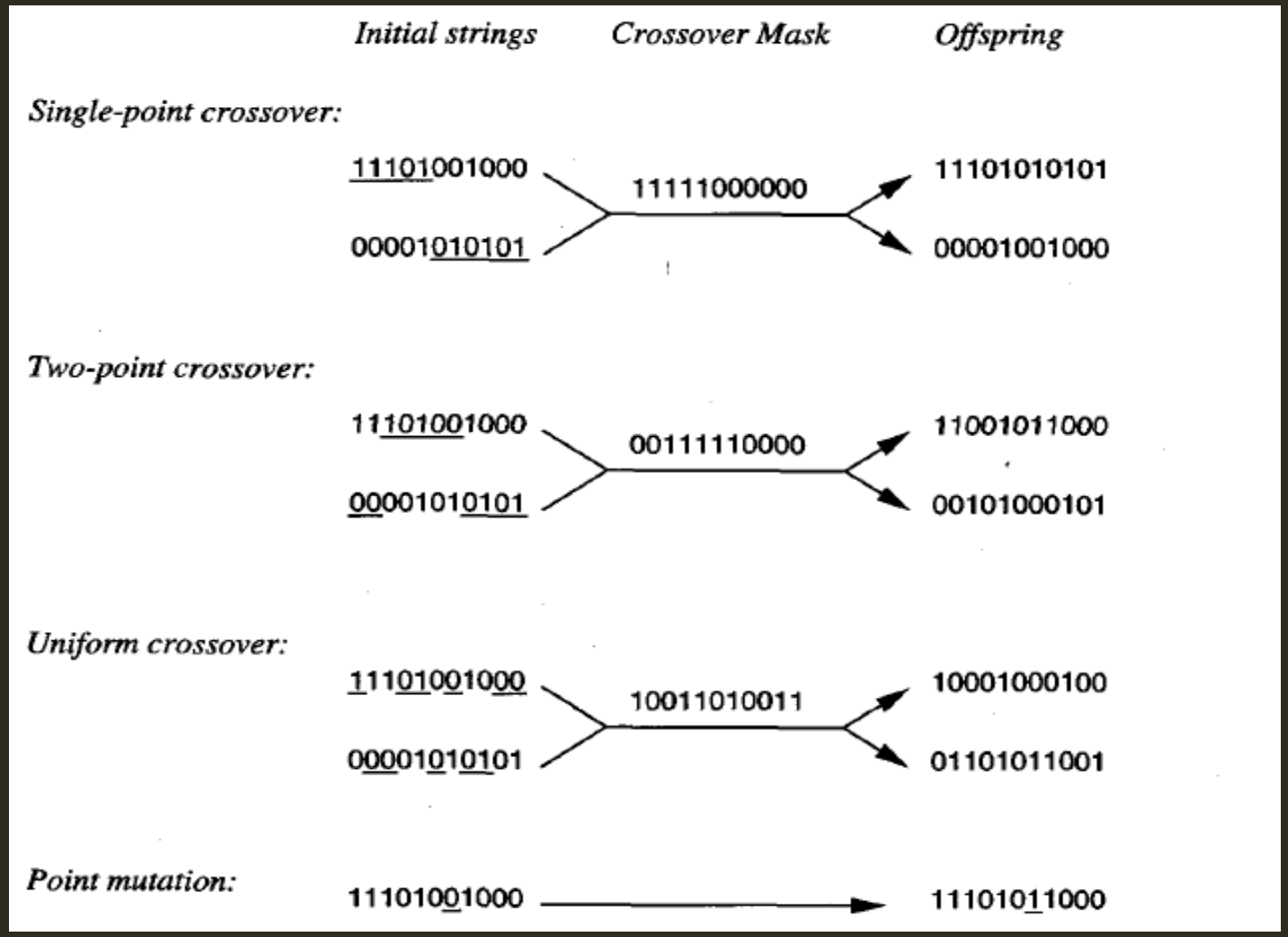
Outlook	*Wind*	*PlayTennis*
111	10	10

GENETIC OPERATORS

Genetic Operators

- The generation of successors in a GA is determined by a set of operators that recombine and mutate selected members of the current population.
- Crossover and Mutation
- **crossover operator** produces two new offspring from two parent strings, by copying selected bits from each parent.
- The bit at position i in each offspring is copied from the bit at position i in one of the two parents.(Crossover Mask)

GENETIC OPERATORS



GENETIC OPERATORS

- **Mutation** operator produces small random changes to the bit string by choosing a single bit at random, then changing its value.

FITNESS FUNCTION AND SELECTION

- The fitness function defines the criterion for ranking potential hypotheses and for probabilistically selecting them for inclusion in the next generation population.
- Fitness function may measure the overall performance of the resulting procedure rather than performance of individual rules.
- **Fitness proportionate selection or roulette wheel selection** :The probability that a hypothesis will be selected is given by the ratio of its fitness to the fitness of other members of the current population.

FITNESS FUNCTION AND SELECTION

- Tournament selection : Two hypotheses are first chosen at random from the current population.
- With some predefined probability p the more fit of these two is then selected, and with probability (1 - p) the less fit hypothesis is selected.
- Rank Selection: Hypothesis is selected from sorted list. Hypotheses are selected depending on their fitness value.

ILLUSTRATE AND EXAMPLE

GABIL

- GABIL uses a GA to learn boolean concepts represented by a disjunctive set of propositional rules.
- Representation

If $a_1 = T \wedge a_2 = F$ THEN $C = T$ If $a_2 = T$ Then $C = F$

	a_1	a_2	C	a_1	a_2	C
h_1 =	10	01	1	11	10	0
h_2:	01	11	0	10	01	0

ILLUSTRATE AND EXAMPLE

- Let **d1 (d2)** denote the distance from the leftmost (rightmost) of these two crossover points to the rule boundary immediately to its left.
- The crossover points in the second parent are now randomly chosen, subject to the constraint that they must have the same d1 and ***d2*** value.
- For example, if the two parent strings are

	a_1	a_2	c	a_1	a_2	c
h_1 :	10	01	1	11	10	0

and

	a_1	a_2	c	a_1	a_2	c
h_2 :	01	11	0	10	01	0

and the crossover points chosen for the first parent are the points following bit positions 1 and 8,

	a_1	a_2	c	a_1	a_2	c
h_1 :	1[0	01	1	11	1]0	0

where "[" and "]" indicate crossover points, then $d_1 = 1$ and $d_2 = 3$. Hence the allowed pairs of crossover points for the second parent include the pairs of bit positions $\langle 1, 3\rangle$, $\langle 1, 8\rangle$, and $\langle 6, 8\rangle$. If the pair $\langle 1, 3\rangle$ happens to be chosen,

ILLUSTRATE AND EXAMPLE

	a_1	a_2	c	a_1	a_2	c
h_2 :	0[1	1]1	0	10	01	0

then the two resulting offspring will be

	a_1	a_2	c
h_3 :	11	10	0

and

	a_1	a_2	c	a_1	a_2	c	a_1	a_2	c
h_4 :	00	01	1	11	11	0	10	01	0

As this example illustrates, this crossover operation enables offspring to contain a different number of rules than their parents, while assuring that all bit strings generated in this fashion represent well-defined rule sets.

- **Fitness function.** The fitness of each hypothesized rule set is based on its classification accuracy over the training data. In particular, the function used to measure fitness is

$$Fitness(h) = (correct(h))^2$$

where $correct(h)$ is the percent of all training examples correctly classified by hypothesis h.

Extensions

- Add Alternative : Constraint on a specific attribute by changing a 0 to a 1 in the substring corresponding to the attribute.

- Drop Condition: This extension performs a more drastic generalization step, by replacing all bits for a particular attribute by a 1.
- This operator corresponds to generalizing the rule by completely dropping the constraint on the attribute, and was applied on each generation with probability .60.

UNIT-5

REINFORCEMENT LEARNING

Reinforcement Learning

- Introduction
- The Learning Task
- Q Learning.

REINFORCEMENT

- Reinforcement learning addresses the question of how an **autonomous agent** that senses and acts in its **environment** can **learn** to choose optimal actions to achieve its goals

- This very generic problem covers tasks such as **learning to control a mobile robot, learning to optimize operations in factories, and learning to play board games.**

INTRODUCTION

- Each time the agent performs an action in its environment, a trainer may provide a **reward or penalty** to indicate the desirability of the resulting state.
- **Reinforcement Learning is an aspect of Machine learning** where an agent learns to **behave in an environment**, by performing certain **actions.**

- Lets starts the explanation with an example – say there is a small baby who starts learning how to walk.

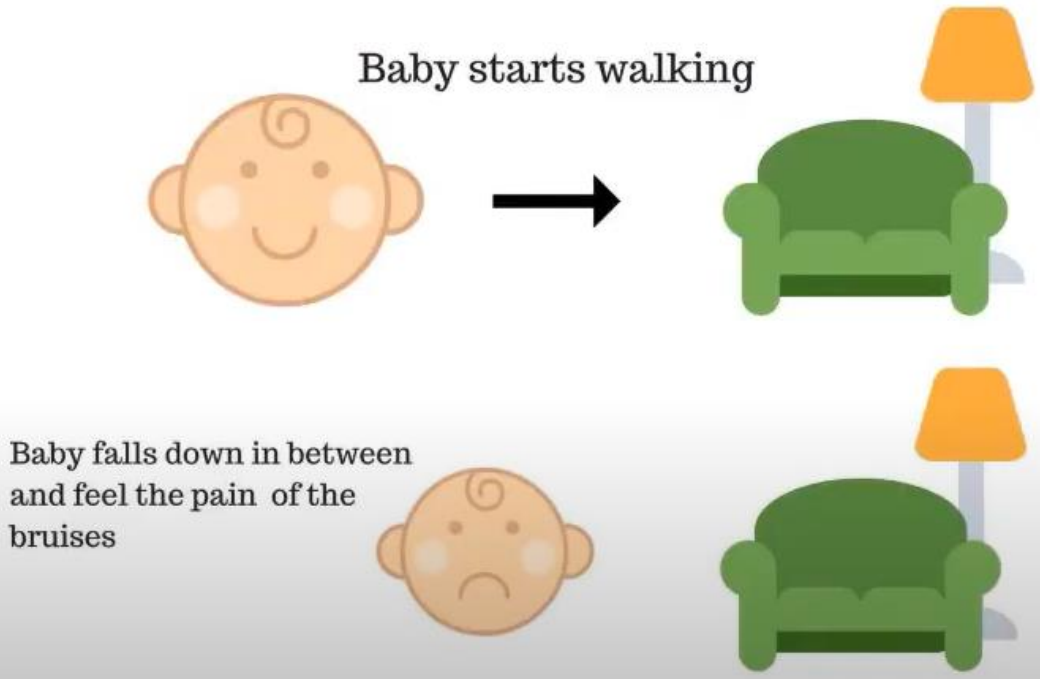
- Let's divide this example into two parts:

1. Baby starts walking and successfully reaches the couch

   Since the couch is the end goal, the baby and the parents are happy.

- So, the baby is happy and receives appreciation from her parents. It's positive — the baby feels good (Positive Reward +n)

2. Baby starts walking and falls due to some obstacle in between and gets bruised.



- That's how we humans learn — by trail and error. Reinforcement learning is conceptually the same, but is a computational approach to learn by actions.

INTRODUCTION

- For example, when training an agent to play a game the trainer might provide a positive reward when the game is won, negative reward when it is lost, and zero reward in all other states.
- The task of the agent is to learn from this indirect, delayed reward, to choose sequences of actions that produce the greatest cumulative reward.

INTRODUCTION

- Consider building a learning robot.
- The robot, or ***agent,*** has a set of sensors to observe the ***state*** of its environment, and a set of ***actions*** it can perform to alter this state.

- For example, a mobile robot may have sensors such as a camera and sonars, and actions such as "move forward" and "turn."

- Its task is to learn a control strategy, or policy, for choosing actions that achieve its goals.

- For example, the robot may have a goal of docking onto its battery charger whenever its battery level is low.

INTRODUCTION

- Let's suppose that our reinforcement learning agent is learning to play Mario as a example. The reinforcement learning process can be modeled as an iterative loop that works as below:

INTRODUCTION

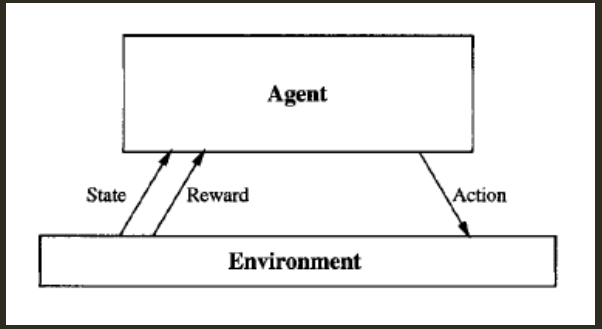
- The RL Agent receives state S^0 from the environment i.e. Mario
- Based on that state S^0, the RL agent takes an action A^0, say — our RL agent moves right. Initially, this is random.
- Now, the environment is in a new state S^1 (new frame from Mario or the game engine)
- Environment gives some reward R^1 to the RL agent. It probably gives a +1 because the agent is not dead yet.

This RL loop continues until we are dead or we reach our destination, and it continuously outputs a sequence of state, action and reward.

- The basic aim of our RL agent is to maximize the reward.

INTRODUCTION

- For example, the goal of docking to the battery charger can be captured by assigning a positive reward (e.g., +loo) to state-action transitions that immediately result in a connection to the charger and a reward of zero to every other state-action transition.
- This **reward function** may be built into the robot, or known only to an external teacher who provides the reward value for each action performed by the robot.
- The task of the robot is to perform sequences of actions, observe their consequences, and learn a control policy.
- The control policy we desire is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agent.
- This general setting for robot learning is summarized in Figure 13.1



INTRODUCTION

- It includes sequential scheduling problems such as choosing which taxis to send for passengers in a large city, where the reward to be maximized is a function of the wait time of the passengers and the total fuel costs of the **taxi fleet.**

In general, we are interested in any type of agent that must learn to choose actions that alter the state of its environment and where a cumulative reward function is used to define the quality of any given action sequence.

Reward Maximization

- The RL agent basically works on a hypothesis of reward maximization. That's why reinforcement learning should have best possible action in order to maximize the reward.
- The cumulative rewards at each time step with the respective action is written as:

$$G_t = \sum_{k=0}^{T} R_{t+k+1}$$

- However, things don't work in this way when summing up all the rewards.

Discounting of rewards works

- We define a discount rate called **gamma**. It should be between 0 and 1. The larger the gamma, the smaller the discount and vice versa.
- So, our cumulative expected (discounted) rewards is:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \, where \gamma \in [0, 1)$$

$$R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots$$

- Sigma (Sum up)
- Discount rate
- Rewards received at each state
- Expanded form of the Equation

Tasks and their types in reinforcement learning

- **Continuous tasks :** These are the types of tasks that continue forever.
- **Episodic task:** In this case, we have a starting point and an ending point called the terminal state. This creates an episode: a list of States (S), Actions (A), Rewards (R).

Exploration and exploitation trade off

- There is an important concept of the exploration and exploitation trade off in reinforcement learning. Exploration is all about finding more information about an environment, whereas exploitation is exploiting already known information to maximize the rewards.
- Real Life Example: Say you go to the same restaurant every day. You are basically exploiting. But on the other hand, if you search for new restaurant every time before going to any one of them, then it's exploration. Exploration is very important for the search of future rewards which might be higher than the near rewards.

Approaches to Reinforcement Learning

- we will only take 2 major approaches
  - Policy-based approach
  - Value Based

Policy-based approach

- In policy-based reinforcement learning, we have a policy which we need to optimize. The policy basically defines how the agent behaves:

$$a = \pi(s)$$

a : Actions

s : State

π: Policy Func.

- We learn a policy function which helps us in mapping each state to the best action.

Value Based

- In value-based RL, the goal of the agent is to optimize the value function *V(s)* which is defined as a function that tells us the maximum expected future reward the agent shall get at each state.
- The value of each state is the total amount of the reward an RL agent can expect to collect over the future, from a particular state.

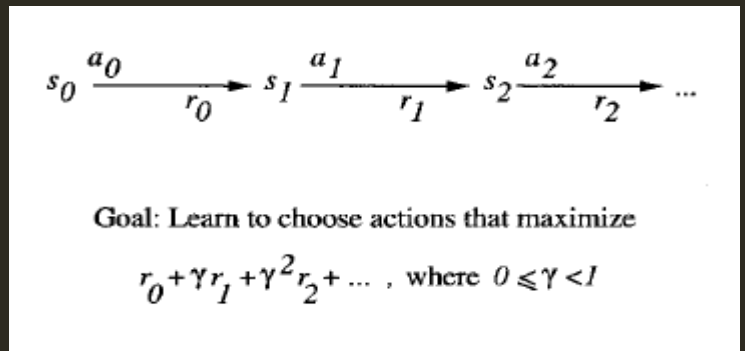
$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s \right]$$

Expected — Reward discounted — Given that state

- The agent will use the above value function to select which state to choose at each step. The agent will always take the state with the biggest value.

INTRODUCTION

- An agent interacting with its environment. The agent exists in an environment described by some set of possible states S.
- It can perform any of a set of possible actions A.
- Each time it performs an action a,
- In some state **st** the agent receives a real-valued reward r, that indicates the immediate value of this state-action transition.
- This produces a sequence of states si, actions ai, and immediate rewards ri as shown in the figure.
- The agent's task is to learn a control policy,
- $\pi : S \rightarrow A$, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.



INTRODUCTION

- The problem of learning a **control policy** to choose actions is similar in some respects to the function approximation problems.
- The target function to be learned in this case is a control policy, pie : S + A, that outputs an appropriate action a from the set A, given the current state s from the set S.

1. Delayed reward. The task of the agent is to learn a target function ***n*** that maps from the current state ***s*** to the optimal action ***a = pie(s).***

2. Exploration. In reinforcement learning, the agent influences the distribution of training examples by the action sequence it chooses.

INTRODUCTION

3. Partially observable states. Although it is convenient to assume that the agent's sensors can perceive the entire state of the environment at each time step, in many practical situations sensors provide only partial information.

For example, a robot with a forward-pointing camera cannot see what is behind it.

4. ***Life-long learning.*** Unlike isolated function approximation tasks, robot learning often requires that the robot learn several related tasks within the same environment, using the same sensors.

For example, a mobile robot may need to learn how to dock on its battery charger, how to navigate through narrow corridors, and how to pick up output from laser printers

- ***Delayed reward:*** The task of the agent is to learn a target function ***n*** that maps from the current state ***s*** to the optimal action ***a = π(s)***
- We have always assumed that when learning some target function such as ***n***, each training example would be a pair of the form ***(s, π(s))***
- In reinforcement learning, however, training information is not available in this form
- Instead, the trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions
- The agent, therefore, faces the problem of ***temporal credit assignment:*** determining which of the actions in its sequence are to be credited with producing the eventual rewards

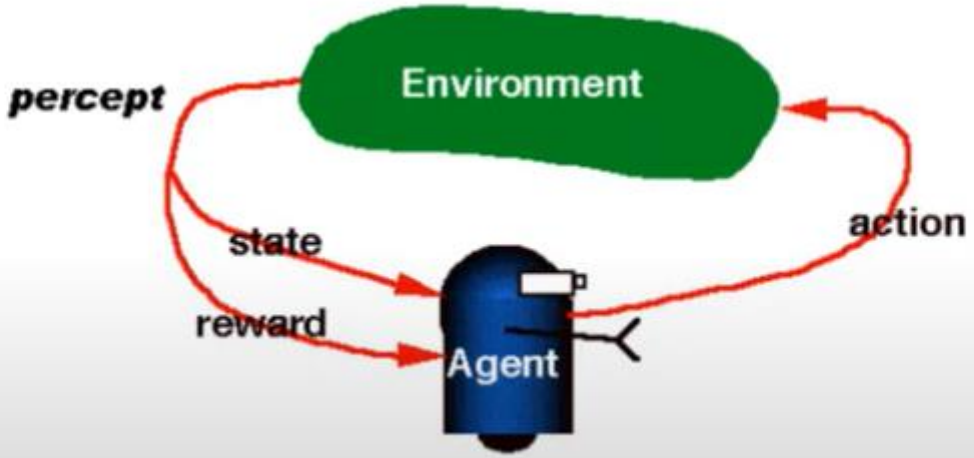
- ***Exploration:*** In reinforcement learning, the agent influences the distribution of training examples by the action sequence it chooses
- This raises the question of which experimentation strategy produces most effective learning
- The learner faces a tradeoff in choosing whether to favor **exploration** of unknown states and actions (to gather new information), or **exploitation** of states and actions that it has already learned will yield high reward (to maximize its cumulative reward)

- ***Partially observable states:*** Although it is convenient to assume that the agent's sensors can perceive the entire state of the environment at each time step, in many practical situations sensors provide only partial information.
- For example, a robot with a forward-pointing camera cannot see what is behind it
- In such cases, it may be necessary for the agent to consider its previous observations together with its current sensor data when choosing actions, and the best policy may be one that chooses actions specifically to improve the observability of the environment

- ***Life-long learning:*** Unlike isolated function approximation tasks, robot learning often requires that the robot learn several related tasks within the same environment, using the same sensors
- For example, a mobile robot may need to learn how to dock on its battery charger, how to navigate through narrow corridors, and how to pick up output from laser printers
- This setting raises the possibility of using previously obtained experience or knowledge to reduce sample complexity when learning new tasks

RL another view

- RL is learning from interaction



- Each percept(e) is enough to determine the State(the state is accessible)
- The agent can decompose the Reward component from a percept.
- The agent task: to find a optimal policy, mapping states to actions, that maximize long-run measure of the reinforcement
- Think of reinforcement as reward
- Can be modeled as MDP model!

Review of MDP(Markov Decision Process) model

- MDP model <S,T,A,R>

- S– set of states
- A– set of actions
- T(s,a,s') = P(s'|s,a)– the probability of transition from s to s' given action a
- R(s,a)– the expected reward for taking action a in state s

$$R(s,a) = \sum_{s'} P(s'|s,a) r(s,a,s')$$

$$R(s,a) = \sum_{s'} T(s,a,s') r(s,a,s')$$

MDP(more detailed view)

A Markov decision process is a 5-tuple $(S, A, P_\cdot(\cdot,\cdot), R_\cdot(\cdot,\cdot), \gamma)$, where

- S is a finite set of states,
- A is a finite set of actions (alternatively, A_s is the finite set of actions available from state s),
- $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t+1$,
- $R_a(s, s')$ is the immediate reward (or expected immediate reward) received after transitioning from state s to state s', due to action a
- $\gamma \in [0, 1]$ is the discount factor, which represents the difference in importance between future rewards and present rewards.

THE LEARNING TASK

- formulate the problem of learning sequential control strategies more precisely

we might assume the agent's:

1. actions are deterministic or that they are nondeterministic
2. can predict the next state that will result from each action
3. trained by an expert
4. train itself

LEARNING TASK

- *For example, we might assume*
- *the agent's actions are deterministic or that they are nondeterministic.*
- *We might assume that the agent can predict the next state that will result from each action, or that it cannot.*
- *We might assume that the agent is trained by an expert who shows it examples of optimal action sequences, or that it must train itself by performing actions of its own choice.*
- *Here we define one quite general formulation of the problem, based on Markov decision processes. This formulation of the problem follows the problem illustrated in Figure 13.1.*

LEARNING TASK

- In a Markov decision process (**MDP**) the agent can perceive a set ***S*** of distinct states of its environment and has a set ***A*** of actions that it can perform.
- At each discrete time step *t*, the agent senses the current state ***st,*** chooses a current action ***a t,*** and performs it.
- The environment responds by giving the agent a reward ***rt*** = ***r (st, a t)*** and by producing the succeeding state $s_{t+1} = \delta(s_t, a_t)$.
- Here the functions δ and r are part of the environment and are not necessarily known to the agent.
- In an MDP, the functions $\delta(s_t, a_t)$ and $r(s_t, a_t)$ depend only on the current state and action, and not on earlier states or actions.
- In general, δ and r may be nondeterministic functions, but we begin by considering only the deterministic case.

LEARNING TASK

- The task of the agent is to learn a ***policy,*** $\pi : S \rightarrow A$, for selecting its next action a_t based on the current observed state ***st;*** that is, $\pi(s_t) = a_t$.
- How shall we specify precisely which policy π we would like the agent to learn?
- Approach is to require the policy that produces the greatest possible cumulative reward for the robot over time.
- To state this requirement more precisely, we define the cumulative value $V^{\pi}(s_t)$ achieved by following an arbitrary policy π from an arbitrary initial state s_t as follows:

$$V^{\pi}(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$
$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i} \qquad (13.1)$$

Q Learning

- Q-learning is a values-based learning algorithm in reinforcement learning

Q Learning Algorithm

Q learning algorithm

For each s, a initialize the table entry $\hat{Q}(s, a)$ to zero.
Observe the current state s
Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$

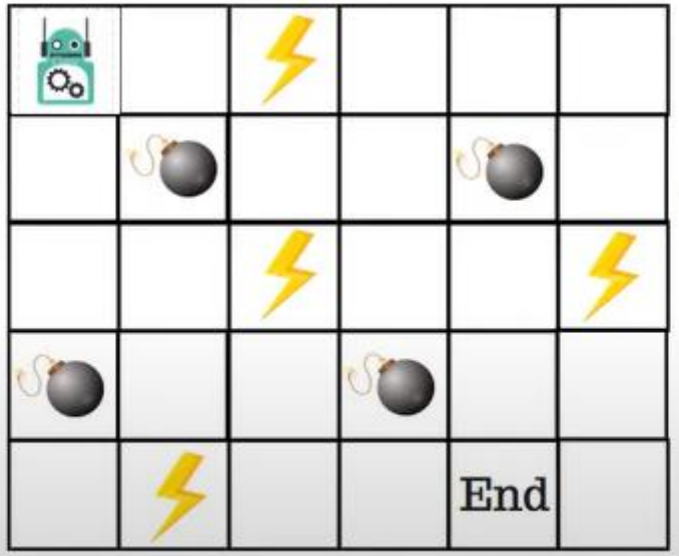
Q learning algorithm, assuming deterministic rewards and actions. The discount factor γ may be any constant such that $0 \leq \gamma < 1$.

Q Learning Illustration

- Let's say that a robot has to cross a maze and reach the end point. There are mines, and the robot can only move one tile at a time. If the robot steps onto a mine, the robot is dead. The robot has to reach the end point in the shortest time possible

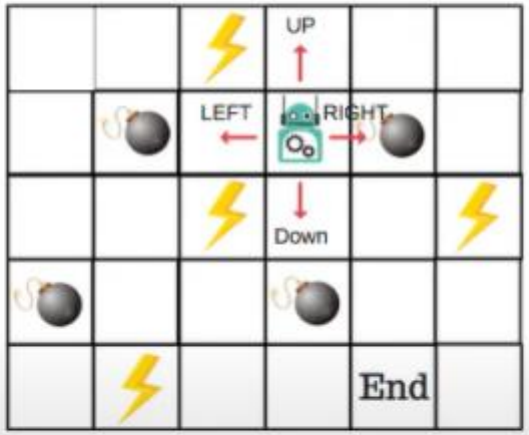
- The scoring/reward system is as below:
- The robot loses 1 point at each step. This is done so that the robot takes the shortest path and reaches the goal as fast as possible.
- If the robot steps on a mine, the point loss is 100 and the game ends.
- If the robot gets power ⚡, it gains 1 point.
- If the robot reaches the end goal, the robot gets 100 points.

Now, the obvious question is: How do we train a robot to reach the end goal with the shortest path without stepping on a mine?



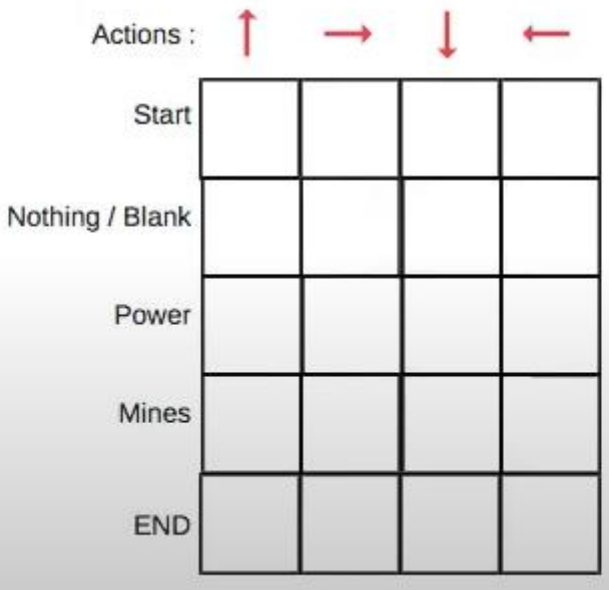
Introducing the Q-Table

- Q-Table is just a fancy name for a simple lookup table where we calculate the maximum expected future rewards for action at each state. Basically, this table will guide us to the best action at each state.



- There will be four numbers of actions at each non-edge tile. When a robot is at a state it can either move up or down or right or left.

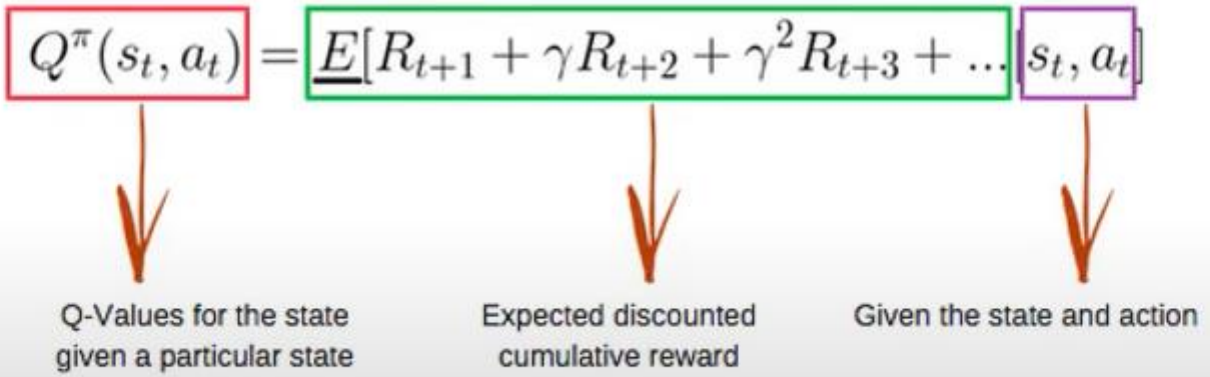
- In the Q-Table, the columns are the actions and the rows are the states.



- Each Q-table score will be the maximum expected future reward that the robot will get if it takes that action at that state. This is an iterative process, as we need to improve the Q-Table at each iteration.
- But the questions are:
  - How do we calculate the values of the Q-table?
  - Are the values available or predefined?
  - To learn each value of the Q-table, we use the Q-Learning algorithm.

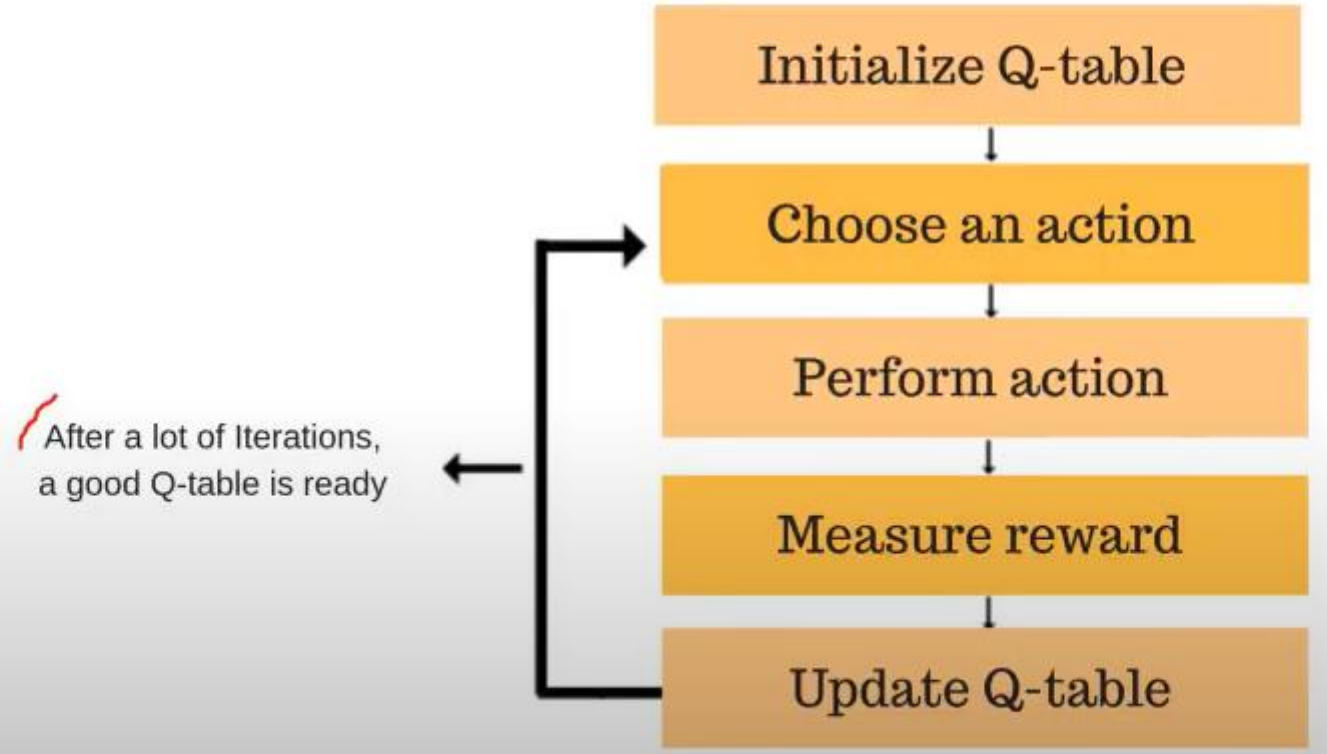
Mathematics: the Q-Learning algorithm

- **Q-function:** The Q-function uses the Bellman equation and takes two inputs: state (s) and action (a).



- Using the above function, we get the values of Q for the cells in the table.
- When we start, all the values in the Q-table are zeros.
- There is an iterative process of updating the values. As we start to explore the environment, the Q-function gives us better and better approximations by continuously updating the Q-values in the table.
- Now, let's understand how the updating takes place.

Introducing the Q-learning algorithm process

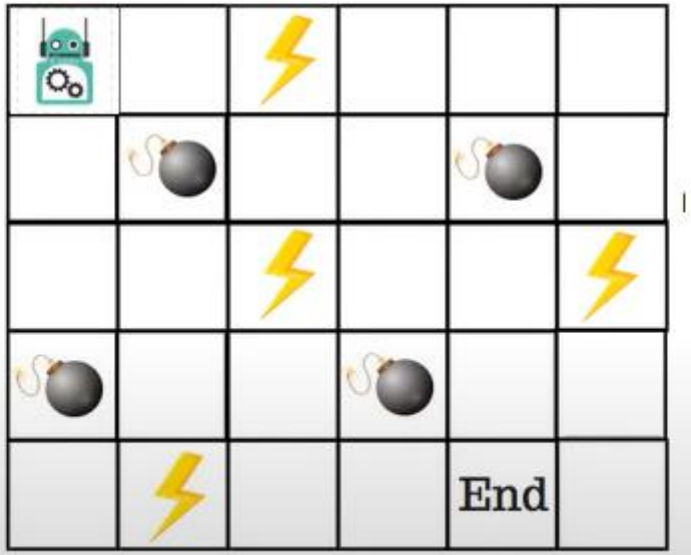


Step 1: initialize the Q-Table

- We will first build a Q-table. There are n columns, where n= number of actions. There are m rows, where m= number of states. We will initialise the values at 0.

Actions :	↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

In our robot example, we have four actions (a=4) and five states (s=5). So we will build a table with four columns and five rows.



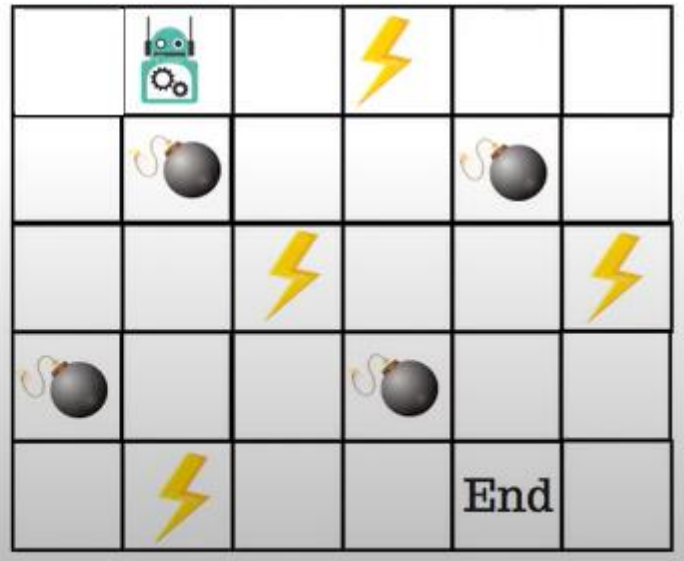
Steps 2 and 3: choose and perform an action

- This combination of steps is done for an undefined amount of time. This means that this step runs until the time we stop the training, or the training loop stops as defined in the code.

- We will choose an action (a) in the state (s) based on the Q-Table. But, as mentioned earlier, when the episode initially starts, every Q-value is 0.

- We'll use something called the epsilon greedy strategy.
- In the beginning, the epsilon rates will be higher. The robot will explore the environment and randomly choose actions. The logic behind this is that the robot does not know anything about the environment.
- As the robot explores the environment, the epsilon rate decreases and the robot starts to exploit the environment.
- During the process of exploration, the robot progressively becomes more confident in estimating the Q-values.

- **For the robot example, there are four actions to choose from**: up, down, left, and right. We are starting the training now — our robot knows nothing about the environment. So the robot chooses a random action, say right.

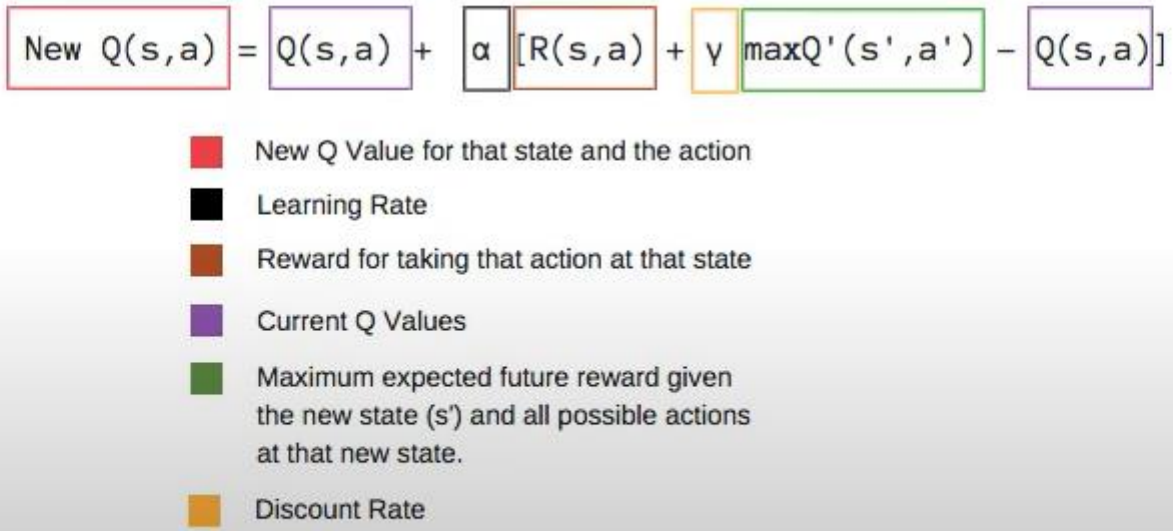


Actions :	↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

- We can now update the Q-values for being at the start and moving right using the Bellman equation.

Steps 4 and 5: evaluate

- Now we have taken an action and observed an outcome and reward. We need to update the function Q(s,a).



- In the case of the robot game, to reiterate the scoring/reward structure is:
  - **power** = +1
  - **mine** = -100
  - **end** = +100

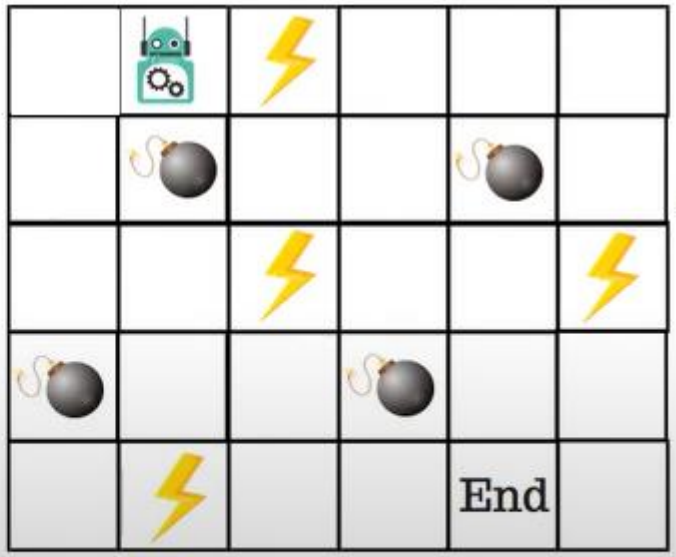
New Q(start,right) = Q(start,right) + α[some ... Delta value]

Some ... Delta value = R(start,right) + max(Q`(nothing,down),Q`(nothing,left),Q`(nothing,right)) - Q(start,right)

Some ... Delta value = 0 + 0.9 * 0 - 0 = 0

New Q(start,right) = 0 + 0.1* 0 = 0

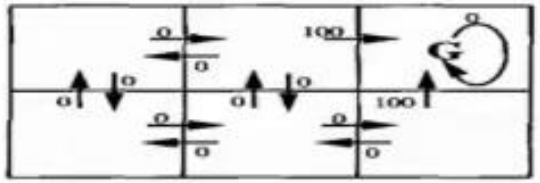
We will repeat this again and again until the learning is stopped. In this way the Q-Table will be updated.



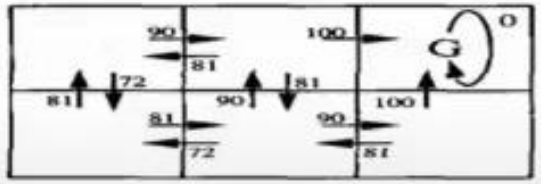
Actions :	↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

The Learning Task

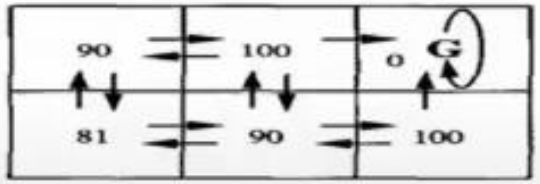
- To illustrate these concepts, a simple grid-world environment is depicted in the topmost diagram of the Figure



$r(s, a)$ (immediate reward) values



$Q(s, a)$ values



$V^*(s)$ values

One optimal policy

The Learning Task

- The six grid squares in the diagram represent six possible states, or locations, for the agent
- Each arrow in the diagram represents a possible action the agent can take to move from one state to another
- The number associated with each arrow represents the immediate reward ***r(s,a)*** the agent receives if it executes the corresponding state-action transition
- Note the immediate reward in this particular environment is defined to be zero for all state-action transitions except for those leading into the state labeled ***G***
- It is convenient to think of the state ***G*** as the goal state, because the only way the agent can receive reward, in this case, is by entering this state
- Note in this particular environment, the only action available to the agent once it enters the state ***G*** is to remain in this state
- For this reason, we call ***G*** an ***absorbing state***

The Learning Task

- Once the states, actions, and immediate rewards are defined, and once we choose a value for the discount factor γ, we can determine the optimal policy π^* and its value function $V^*(s)$
- In this case, let us choose $\gamma = 0.9$
- The diagram at the bottom of the figure shows one optimal policy for this setting
- Like any policy, this policy specifies exactly one action that the agent will select in any given state
- The optimal policy directs the agent along the shortest path toward the state ***G***

The Learning Task

- The diagram at the right of the Figure shows the values of V^* for each state
- For example, consider the bottom right state in this diagram
- The value of V^* for this state is 100 because the optimal policy in this state selects the "move up" action that receives immediate reward 100
- Thereafter, the agent will remain in the absorbing state and receive no further rewards
- Similarly, the value of V^* for the bottom center state is 90

The Learning Task

- This is because the optimal policy will move the agent from this state to the right (generating an immediate reward of zero), then upward (generating an immediate reward of 100)
- Thus, the discounted future reward from the bottom center state is

$$0 + \gamma 100 + \gamma^2 0 + \gamma^3 0 + \cdots = 90$$

- Recall that ***V**** is defined to be the sum of discounted future rewards over the infinite future
- In this particular environment, once the agent reaches the absorbing state ***G,*** its infinite future will consist of remaining in this state and receiving rewards of zero

Quick Review: Q-Learning

- Q-Learning is a value-based reinforcement learning algorithm which is used to find the optimal action-selection policy using a Q function.
- Our goal is to maximize the value function Q.
- The Q table helps us to find the best action for each state.
- It helps to maximize the expected reward by selecting the best of all possible actions.
- Q(state, action) returns the expected future reward of that action at that state.
- This function can be estimated using Q-Learning, which iteratively updates Q(s,a) using the **Bellman equation.**
- Initially we explore the environment and update the Q-Table. When the Q-Table is ready, the agent will start to exploit the environment and start taking better actions.

THANK YOU

PREPARE WELL FOR EXAMS