

## . UNIT-4

### Genetic Algorithms :

#### Motivation:

Motivation is one of the key factors driving us towards achieving something. Without motivation, we will do nothing. Therefore, motivation is one of the key aspects when it comes to corporate management. In order to achieve the best business results, the organization needs to keep employees motivated.

In order to motivate the employees, organizations do various activities. The activities the companies do basically the results and findings of certain motivational theories.

#### Genetic Algorithms:

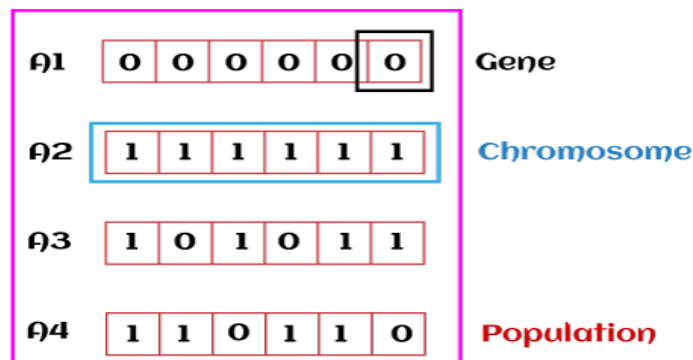
The genetic algorithm works on the evolutionary generational cycle to generate high-quality solutions. These algorithms use different operations that either enhance or replace the population to give an improved fit solution.

It basically involves five phases to solve the complex optimization problems, which are given as below:

- **Initialization**
- **Fitness Assignment**
- **Selection**
- **Reproduction**
- **Termination**

##### 1. Initialization

The process of a genetic algorithm starts by generating the set of individuals, which is called population. Here each individual is the solution for the given problem. An individual contains or is characterized by a set of parameters called Genes. Genes are combined into a string and generate chromosomes, which is the solution to the problem. One of the most popular techniques for initialization is the use of random binary strings.



##### 2. Fitness Assignment

Fitness function is used to determine how fit an individual is? It means the ability of an individual to compete with other individuals. In every iteration, individuals are evaluated based on their fitness function. The fitness function provides a fitness score to each individual. This score further determines the probability of being selected for reproduction. The high the fitness score, the more chances of getting selected for reproduction.

### 3. Selection

The selection phase involves the selection of individuals for the reproduction of offspring. All the selected individuals are then arranged in a pair of two to increase reproduction. Then these individuals transfer their genes to the next generation.

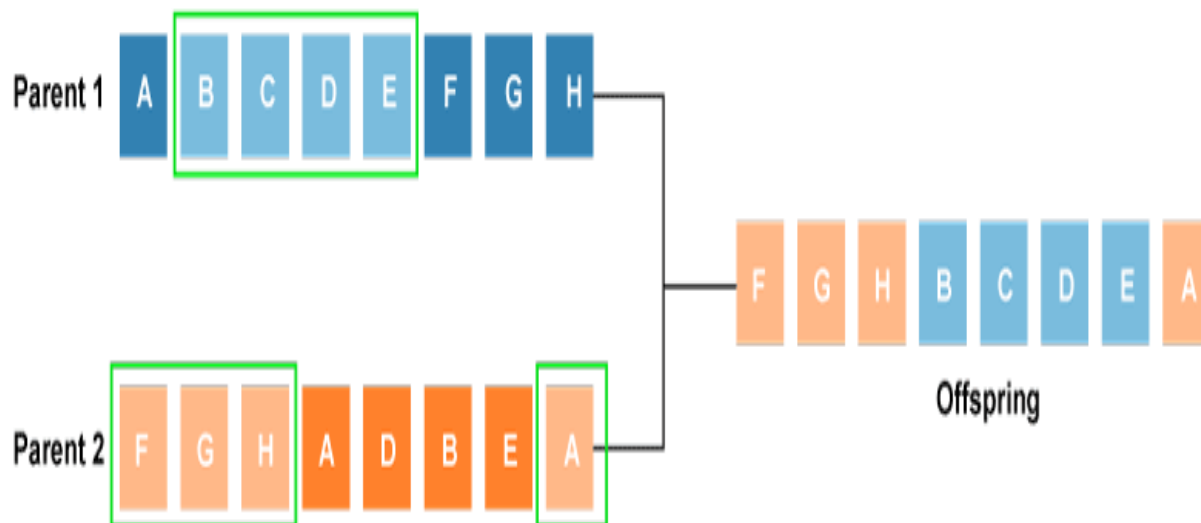
There are three types of Selection methods available, which are:

- Roulette wheel selection
- Tournament selection
- Rank-based selection

### 4. Reproduction

After the selection process, the creation of a child occurs in the reproduction step. In this step, the genetic algorithm uses two variation operators that are applied to the parent population. The two operators involved in the reproduction phase are given below:

- **Crossover:** The crossover plays a most significant role in the reproduction phase of the genetic algorithm. In this process, a crossover point is selected at random within the genes. Then the crossover operator swaps genetic information of two parents from the current generation to produce a new individual representing the offspring.



The genes of parents are exchanged among themselves until the crossover point is met. These newly generated offspring are added to the population. This process is also called or crossover. Types of crossover styles available:

- One point crossover
- Two-point crossover
- Livery crossover
- Inheritable Algorithms crossover

- **Mutation**

The mutation operator inserts random genes in the offspring (new child) to maintain the diversity in the population. It can be done by flipping some bits in the chromosomes. Mutation helps in solving the issue of premature convergence and enhances diversification. The below image shows the mutation process:

Types of mutation styles available,

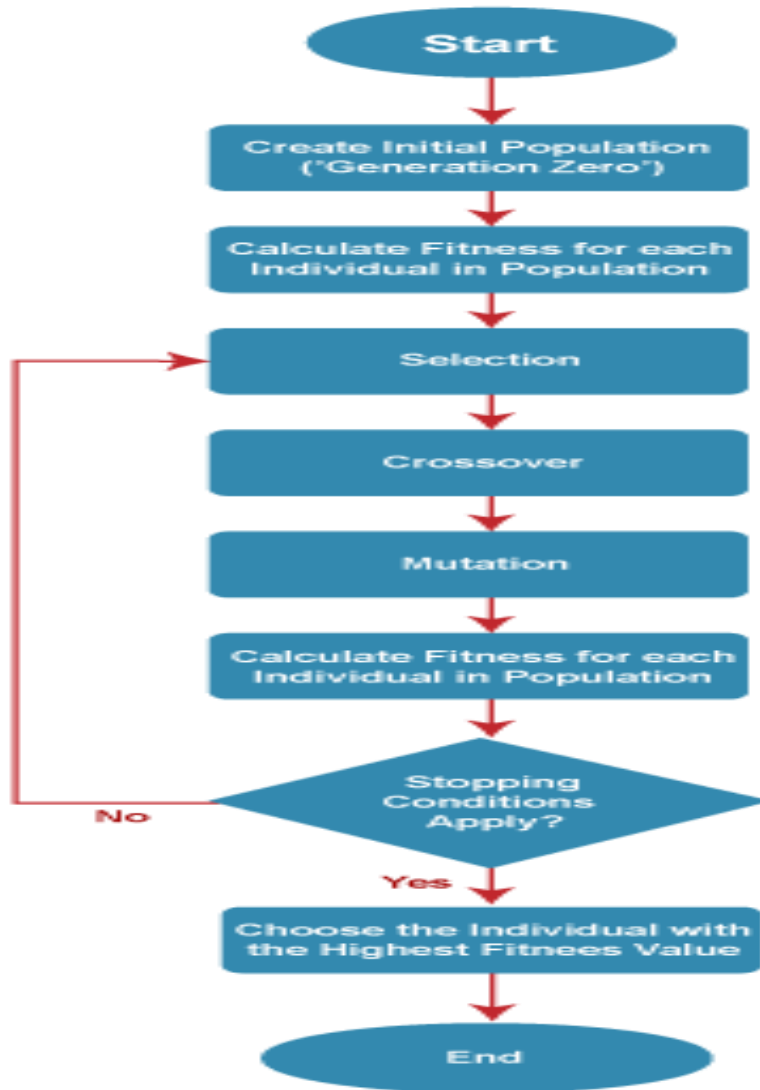
- **Flip bit mutation**
- **Gaussian mutation**
- **Exchange/Swap mutation**



## 5. Termination

After the reproduction phase, a stopping criterion is applied as a base for termination. The algorithm terminates after the threshold fitness solution is reached. It will identify the final solution as the best solution in the population.

General Workflow of a Simple Genetic Algorithm



## Genetic Programming

**Genetic Programming** is a new method to generate computer programs. It was derived from the model of biological evolution. Programs are 'bred' through continuous improvement of an initially random population of programs. Improvements are made possible by stochastic variation of programs and selection according to prespecified criteria for judging the quality of a solution. Programs of Genetic Programming systems evolve to solve predescribed automatic programming and machine learning problems

### Hypothesis Space Search (I)

- GA search can move very abruptly (as compared to Backpropagation, for example), replacing a parent hypothesis by an offspring that may be radically different from the parent.
- The problem of *Crowding*: when one individual is more fit than others, this individual and closely related ones will take up a large fraction of the population.
- Solutions:
- Use tournament or rank selection instead of roulette selection.

- Fitness sharing
- restriction on the kinds of individuals allowed to recombine to form offsprings.

### **Models of Evolution and Learning:**

- Proposition: Experiences of a single organism directly affect the genetic makeup of their offsprings.
- Assessment: This proposition is wrong: the genetic makeup of an individual is unaffected by the lifetime experience of one's biological parents.
- However: Lamarckian processes can sometimes improve the effectiveness of computerized genetic algorithms.
- If a species is evolving in a changing environment, there will be evolutionary pressure to favor individuals with the capability to learn during their lifetime.
- Those individuals who are able to learn many traits will rely less strongly on their genetic code to “hard-wire” traits. As a result, these individuals can support a more diverse gene pool, relying on individual learning of the “missing” or “sub-optimized” traits in the genetic code. This more diverse gene pool can, in turn, support more rapid evolutionary adaptation. Thus the capability of learning can accelerate the rate of evolutionary adaptation of a population.

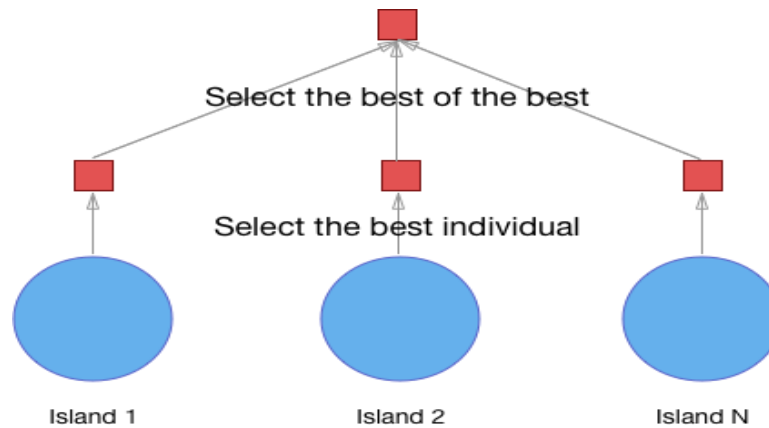
Models of evolution and learning:

### **Abstract**

We study a model of evolving populations of self-learning agents and analyze the interaction between learning and evolution. We consider an agent-broker that predicts stock price changes and uses its predictions for selecting actions. Each agent is equipped with a neural network adaptive critic design for behavioral adaptation. We discuss three cases in which either evolution or learning, or both, are active in our model. We show that the Baldwin effect can be observed in our model, viz. originally acquired adaptive policy of best agent-brokers becomes inherited over the course of the evolution. We also compare the behavioral tactics of our agents to the searching behavior of simple animals.

### **Parallel genetic algorithm**

Parallel genetic algorithm is such an algorithm that uses multiple genetic algorithms to solve a single task [1]. All these algorithms try to solve the same task and after they've completed their job, the best individual of every algorithm is selected, then the best of them is selected, and this is the solution to a problem. This is one of the most popular approach to parallel genetic algorithms, even though there are others. This approach is often called ‘island model’ because populations are isolated from each other, like real-life creature populations may be isolated living on different islands. Image 1 illustrates that.



## Learning Sets of Rules

Introduction:

### 1 Learning Rules

- We can learn sets of rules by using ID3 and then converting the tree to rules.
- We can also use a genetic algorithm that encodes the rules as bit strings.
- But, these only work with predicate rules (no variables).
- They also consider the set of rules as a whole, not one rule at a time.

### 2 Rules

- First-order predicate logic (calculus) [3] formalizes statements using predicates (boolean functions) and functions. Both can have variables.
- A rule set can look like

*IF Parent(x,y) THEN Ancestor(x,y)*

*IF Parent(x,z)  $\wedge$  Ancestor(z,y) THEN Ancestor(x,y)*

- Here, Parent(x,y) is a predicate that indicates that y is the parent of x.
- These two rules form a recursive function which would be very hard to represent using a decision tree or propositional representation.
- In **Prolog** [4], programs are set of first-order rules with the form as above (known as Horn clauses).
- We can view the learning of rules as the learning of Prolog programs.

### 3 Sequential Covering

- The idea in a *sequential covering* algorithm is to learn one rule, remove the data it covers, then repeat.

## Sequential Covering Algorithm

### Prerequisites: Learn-One-Rule Algorithm

Sequential Covering is a popular algorithm based on Rule-Based Classification used for learning a disjunctive set of rules. The basic idea here is to learn one rule, remove the data that it covers, then repeat the same process. In this process, In this way, it covers all the rules involved with it in a sequential manner during the training phase.

#### Algorithm Involved:

Sequential\_covering (Target\_attribute, Attributes, Examples, Threshold):

    Learned\_rules = { }

    Rule = Learn-One-Rule(Target\_attribute, Attributes, Examples)

    while Performance(Rule, Examples) > Threshold :

        Learned\_rules = Learned\_rules + Rule

        Examples = Examples - {examples correctly classified by Rule}

        Rule = Learn-One-Rule(Target\_attribute, Attributes, Examples)

    Learned\_rules = sort Learned\_rules according to performance over Examples

    return Learned\_rules

The Sequential Learning algorithm takes care of to some extent, the low coverage problem in the Learn-One-Rule algorithm covering all the rules in a sequential manner.

#### Working on the Algorithm:

The algorithm involves a set of ‘ordered rules’ or ‘list of decisions’ to be made.

**Step 1** – *create an empty decision list, ‘R’.*

**Step 2** – *‘Learn-One-Rule’ Algorithm*

*It extracts the best rule for a particular class ‘y’.*

Learning rule sets: summary

- A sequential covering algorithm learns one rule at a time, removing the covered examples and repeating with the rest.
- Meanwhile, simultaneous covering algorithms like ID3 learn the entire set of disjuncts simultaneously.
- Which is better?
- ID3 chooses attributes by comparing the partitions of the data they generate.
- CN2 chooses among alternative attribute-value pairs by comparing the subsets of data they cover.
- Thus, CN2 makes a larger number of independent choices. So it is better if there is plenty of data.

#### Learning First-Order Rules

- Try to learn sets of rules such as

$\forall x, y : \text{Ancestor}(x, y) \leftarrow \text{Parent}(x, y)$

$\forall x, y : \text{Ancestor}(x, y) \leftarrow \text{Parent}(x, z) \wedge \text{Ancestor}(z, y)$

- This looks a lot like a Prolog program.

$Ancestor(x,y)$   $Parent(x,y)$   
 $Ancestor(x,y) :- Parent(x,z), Ancestor(z,y)$

- Which is why inductive learning of first-order rules is often referred to as **inductive logic programming**.
- Classifying web page A:

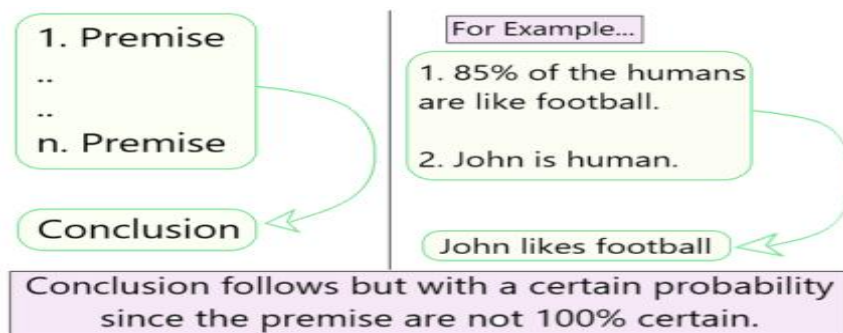
$course(A) \leftarrow has\_word(A, instructor) \wedge \neg has\_word(A, good) \wedge link\_from(A, B) \wedge has\_word(B, problem)$   
 $\wedge \neg link\_from(B, C)$

### First Order Inductive Learner (FOIL)

In machine learning, first-order inductive learner (FOIL) is a rule-based learning algorithm. It is a natural extension of SEQUENTIAL-COVERING and LEARN-ONE-RULE algorithms. It follows a Greedy approach.

#### Inductive Learning:

Inductive learning analyzing and understanding the evidence and then using it to determine the outcome. It is based on Inductive Logic.



#### Induction as Inverted Deduction

- A second approach to inductive logic programming is based on the observation that induction is the inverse of deduction. Induction is, in fact, the inverse operation of deduction, and cannot be conceived to exist without the corresponding operation, so that the question of relative importance cannot arise. Who thinks of asking whether addition or subtraction is the more important process in arithmetic? But at the same time much difference in difficulty may exist between a direct and inverse operation; ... it must be allowed that inductive investigations are of a far higher degree of difficulty and complexity than any questions of deduction.... (Jevons 1874)

- More formally, induction is finding a hypothesis  $h$  such that

$$(\forall h x_i, f(x_i) \mid i \in D) B \wedge h \wedge x_i \vdash f(x_i)$$

where

- $D$  is training data
- $x_i$  is  $i$ th training instance
- $f(x_i)$  is the target function value for  $x_i$
- $B$  is other background knowledge

“For each training data instance, the instance’s target classification is logically entailed by the background knowledge, together with hypothesis and the instance itself”

- Suggests we design an inductive algorithm by inverting operators for automated deduction .
- Concept to be learned: “pairs of people,  $h_u, v_i$  such that child of  $u$  is  $v$  ”
- Suppose we are given – one training example
  - \* describing two people Tim and Sharon in terms of their gender and their relation in terms of the predicate  $f$  ather
  - \* giving the target predicate  $\text{Child}(\text{Tim}, \text{Sharon})$  for these two people
  - background knowledge asserting that “if  $u$  is the father of  $v$  then  $u$  is a parent of  $v$  ”

### Inverting Resolution

- The inverse entailment operator must derive  $C_2$  given the resolvent  $C$  and  $C_1$ .
- Say  $C = A \vee B$ , and  $C_1 = B \vee D$ . How do we derive  $C_2$  s.t.  $C_1 \wedge C_2 \rightarrow C$ ?
- Find the  $L$  that appears in  $C_1$  and not in  $C$ , then form  $C_2$  by including the following literals  
 $C_2 = (C - (C_1 - \{L\})) \cup \{\neg L\}$
- so  
 $C_2 = A \vee \neg D$
- $C_2$  can also be  $A \vee \neg D \vee B$ .
- In general, inverse resolution can produce multiple clauses  $C_2$ .

### Learning With Inverted Resolution

- Use inverse entailment to construct hypotheses that, together with the background information, entail the training data.
  - Use sequential covering algorithm to iteratively learn a set of Horn clauses in this way.
1. Select a training example that is not yet covered by learned clauses.

2. Use inverse resolution rule to generate candidate hypothesis  $h$  that satisfies  $B \wedge h \wedge x \rightarrow f(x)$ , where  $B$  = background knowledge plus any learned clauses.

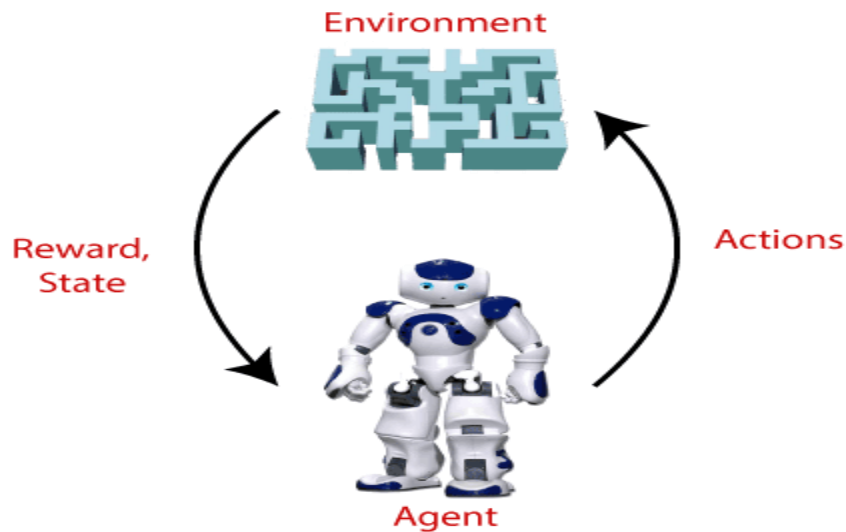
- This is example-driven search.
- If multiple candidate hypotheses then choose one with highest accuracy over the other examples

## Reinforcement Learning

### What is Reinforcement Learning?

- Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.
- In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.
- Since there is no labeled data, so the agent is bound to learn by its experience only.
- RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as **game-playing, robotics**, etc.
- The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.
- The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that **"Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that."** How a Robotic dog learns the movement of his arms is an example of Reinforcement learning.
- It is a core part of Artificial intelligence, and all AI agent works on the concept of reinforcement learning. Here we do not need to pre-program the agent, as it learns from its own experience without any human intervention.
- **Example:** Suppose there is an AI agent present within a maze environment, and his goal is to find the diamond. The agent interacts with the environment by performing some actions, and based on those actions, the state of the agent gets changed, and it also receives a reward or penalty as feedback.

- The agent continues doing these three things (**take action, change state/remain in the same state, and get feedback**), and by doing these actions, he learns and explores the environment.
- The agent learns that what actions lead to positive feedback or rewards and what actions lead to negative feedback penalty. As a positive reward, the agent gets a positive point, and as a penalty, it gets a negative point.



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.

## \* REINFORCEMENT LEARNING:

learns depending on changes occurring in environment

Goal: to achieve the best result

Example: chessboard  $R, L, U, D$

goal  $\rightarrow$  to win the game

Example 2: Agent

(2 ways - fire and water)  
① ②

## Terms used in Reinforcement Learning

- **Agent()**: An entity that can perceive/explore the environment and act upon it.
- **Environment()**: A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- **Action()**: Actions are the moves taken by an agent within the environment.
- **State()**: State is a situation returned by the environment after each action taken by the agent.
- **Reward()**: A feedback returned to the agent from the environment to evaluate the action of the agent.
- **Policy()**: Policy is a strategy applied by the agent for the next action based on the current state.
- **Value()**: It is expected long-term return with the discount factor and opposite to the short-term reward.

- **Q-value():** It is mostly similar to the value, but it takes one additional parameter as a current action (a).

## Types of Reinforcement learning

There are mainly two types of reinforcement learning, which are:

- **Positive Reinforcement**
- **Negative Reinforcement**

| Reinforcement Learning  | Supervised Learning  |
|---|--|
| RL works by interacting with the environment.                                 | Supervised learning works on the existing dataset.                                     |
| The RL algorithm works like the human brain works when making some decisions. | Supervised Learning works as when a human learns things in the supervision of a guide. |
| There is no labeled dataset is present  | The labeled dataset is present.  |
| No previous training is provided to the learning agent.                       | Training is provided to the algorithm so that it can predict the output.               |
| RL helps to take decisions sequentially.                                      | In Supervised learning, decisions are made when input is given.                        |

## Reinforcement Learning Applications



# Markov Decision Process

Markov Decision Process or MDP, is used to **formalize the reinforcement learning problems**. If the environment is completely observable, then its dynamic can be modeled as a **Markov Process**. In MDP, the agent constantly interacts with the environment and performs actions; at each action, the environment responds and generates a new state.



MDP is used to describe the environment for the RL, and almost all the RL problem can be formalized using MDP.

MDP contains a tuple of four elements ( $S$ ,  $A$ ,  $P_a$ ,  $R_a$ ):

- A set of finite States  $S$
- A set of finite Actions  $A$
- Rewards received after transitioning from state  $S$  to state  $S'$ , due to action  $a$ .
- Probability  $P_a$

## Learning Tasks

Learning tasks play an important role in instructional settings. They may be characterized as an interface between the learners and the information offered in the learning environment. They serve to activate and control learning processes in order to facilitate successful learning. They stimulate reactions referring to learning material, thus prompting the learners to engage intensively in the subject matter. Ideally, the learners receive feedback on how well they performed on a learning task and guidance on how to acquire the relevant information. While there is general agreement on the significant role of learning tasks, there is as yet little knowledge on how to design them appropriately.

## ***Q***-learning

***Q***-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment (hence "model-free"), and it can handle problems with stochastic transitions and rewards without requiring adaptations.

For any finite Markov decision process (FMDP), *Q*-learning finds an optimal policy in the sense of maximizing the expected value of the total reward over any and all successive steps, starting from the current state. *Q*-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy. "Q" refers to the function that the algorithm computes – the expected rewards for an action taken in a given state.

## **Nondeterministic**

a **nondeterministic algorithm** is an algorithm that, even for the same input, can exhibit different behaviors on different runs, as opposed to a deterministic algorithm. There are several ways an algorithm may behave differently from run to run. A concurrent algorithm can perform differently on different runs due to a race condition. A probabilistic algorithm's behaviors depends on a random number generator. An algorithm that solves a problem in nondeterministic polynomial time can run in polynomial time or exponential time depending on the choices it makes during execution. The nondeterministic algorithms are often used to find an approximation to a solution, when the exact solution would be too costly to obtain using a deterministic one.

## **Nondeterministic Rewards and Actions:**

- The functions  $V$  and  $Q$  can be viewed as first producing a probability distribution over outcomes based on  $V$  and  $Q$  and then drawing an outcome at random according to this distribution - nondeterministic Markov decision process
- but is not guaranteed to converge
- decaying weighted average of the current  $V$  and the revised estimate, where
- convergence long = 1.5 million games in Tesauro's backgammon program

## **Temporal difference learning**

• **Temporal difference (TD) learning** refers to a class of model-free reinforcement learning methods which learn by bootstrapping from the current estimate of the value function. These methods sample from the environment, like Monte Carlo methods, and perform updates based on current estimates, like dynamic programming methods.

- While Monte Carlo methods only adjust their estimates once the final outcome is known, TD methods adjust predictions to match later, more accurate, predictions about the future before the final outcome is known. This is a form of bootstrapping.

Generalizing from examples

- Every salesperson lies to make more money on a sale.
- Math homework is very easy.
- The United States is colder than Europe.
- Women all want to have large families.
- Men are all afraid of commitment.
- Politicians are greedy and manipulative.
- All cats are meaner than dogs.

## **Relationship to Dynamic Programming**

- agent possesses perfect knowledge of the functions  $V$  and  $Q$
- Focused on how to compute the optimal policy with the least computational effort, assuming the environment can be simulated
- *Q* learning has NO knowledge of the functions  $V$  and  $Q$

Focused on the number of real-world actions the agent must perform to converge to an acceptable policy

## Conclusion:

From the above discussion, we can say that Reinforcement Learning is one of the most interesting and useful parts of Machine learning. In RL, the agent explores the environment by exploring it without any human intervention. It is the main learning algorithm that is used in Artificial Intelligence. But there are some cases where it should not be used, such as if you have enough data to solve the problem, then other ML algorithms can be used more efficiently. The main issue with the RL algorithm is that some of the parameters may affect the speed of the learning, such as delayed feedback