

# Unit-5

Machine Learning (KOE073)

# Genetic Algorithm

- *A genetic algorithm is an adaptive heuristic search algorithm.* It is used to solve optimization problems in machine learning.
- It is one of the important algorithms as it helps solve complex problems that would take a long time to solve.
- Genetic Algorithms are being widely used in different real-world applications, for example, Designing electronic circuits, image processing, and artificial creativity etc.

# Basic Terminologies Used

- **Population:** Population is the subset of all possible or probable solutions, which can solve the given problem.
- **Chromosomes:** A chromosome is one of the solutions in the population for the given problem, and the collection of gene generate a chromosome.
- **Gene:** A chromosome is divided into a different gene, or it is an element of the chromosome.
- **Allele:** Allele is the value provided to the gene within a particular chromosome.
- **Fitness Function:** The fitness function is used to determine the individual's fitness level in the population. It means the ability of an individual to compete with other individuals. In every iteration, individuals are evaluated based on their fitness function.
- **Genetic Operators:** In a genetic algorithm, the best individual mate to regenerate offspring better than parents. Here genetic operators play a role in changing the genetic composition of the next generation.
- **Selection:** After calculating the fitness of every existent in the population, a selection process is used to determine which of the individualities in the population will get to reproduce and produce the seed that will form the coming generation.

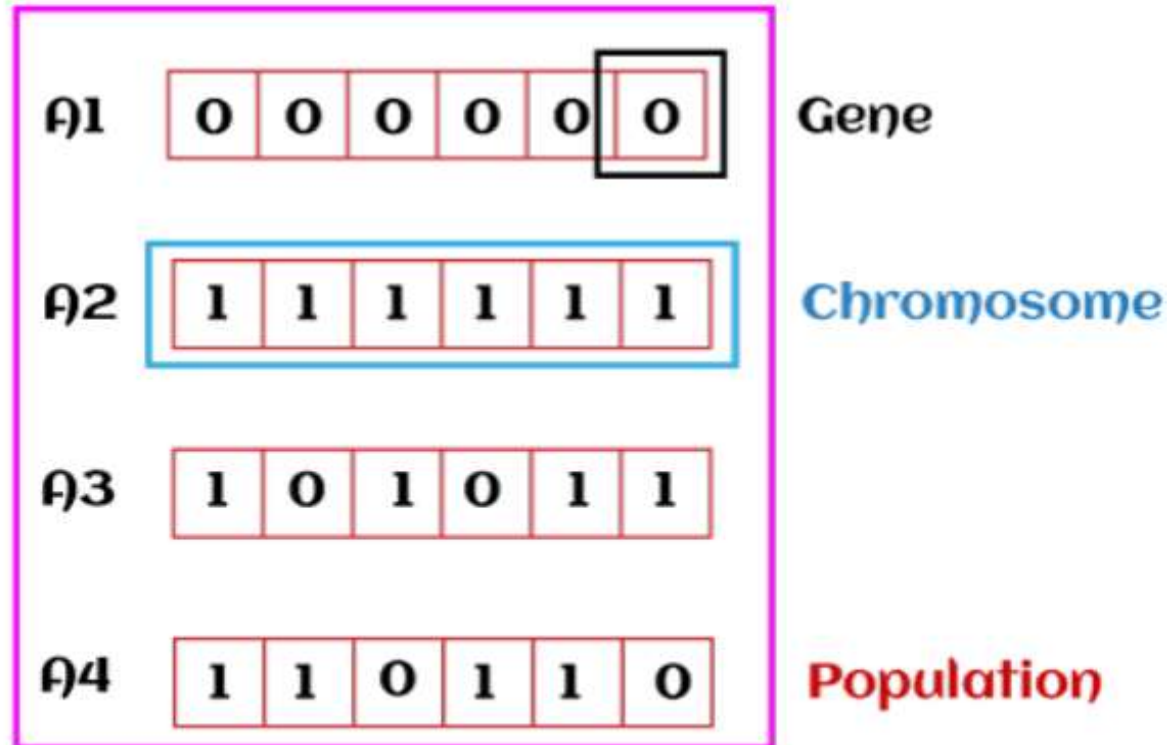
**A genetic algorithm uses genetic and natural selection concepts to solve optimization problems.**

# How Genetic Algorithm Work?

- 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.

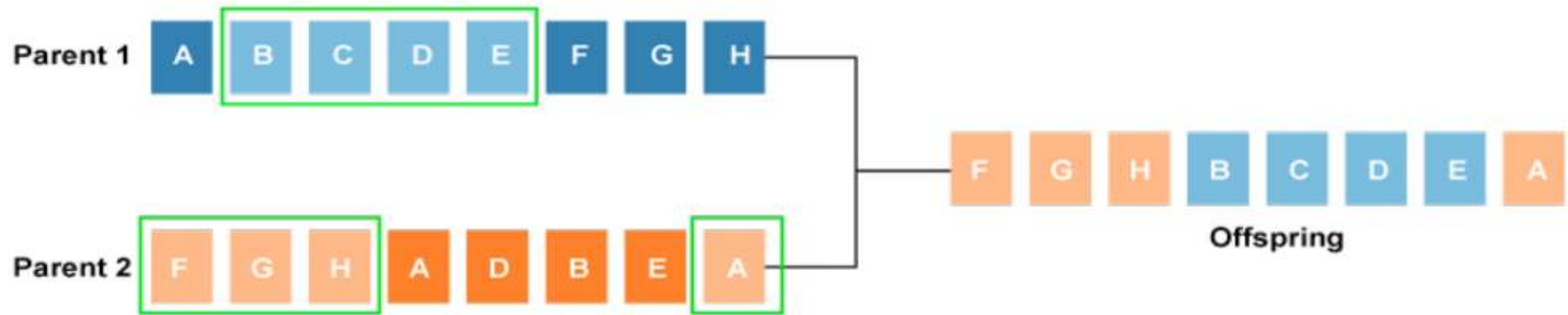
There are three types of Selection methods available, which are:

- 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.



- **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.



## Advantages of Genetic Algorithm

- The parallel capabilities of genetic algorithms are best.
- It helps in optimizing various problems such as discrete functions, multi-objective problems, and continuous functions.
- It provides a solution for a problem that improves over time.
- A genetic algorithm does not need derivative information.

## Limitations of Genetic Algorithms

- Genetic algorithms are not efficient algorithms for solving simple problems.
- It does not guarantee the quality of the final solution to a problem.
- Repetitive calculation of fitness values may generate some computational challenges.

## Question

Maximize the function  $f(x)=x^2$ , where x value range from 0-31

## Answer

### Step 1-

- Encoding technique- Binary encoding
- Selection operator- Roulette Wheel Selection
- Crossover operator- Single point crossover

### Step 2-

Population size (n) = 4

### Step 3-

Initial population (x value) = 13, 24, 8, 19

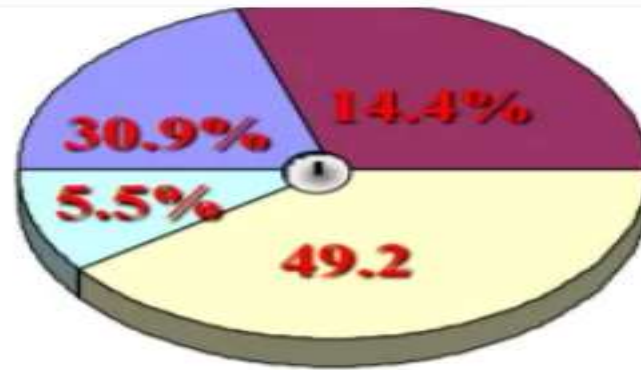
## Step 4-

defines the roulette wheel  
 $F(x)/\text{Total } F(x)$

$F(x)/\text{Average } F(x)$

Number of times the parent will get  
selected if wheel is spun 'n' times  
 $AC = \text{round}(EC)$

String No	Initial Population	X Value	F(x) value	Probability count	Expected count	Actual Count
1	01101	13	169	0.14	0.58	1
2	11000	24	576	0.49	1.97	2
3	01000	8	64	0.06	0.22	0
4	10011	19	361	0.31	1.23	1
Total			1170	1	4	
Average			293			



**Weighted Roulette wheel**

We see that if the Roulette wheel is spun four times, we'll get 24 twice and 13 and 19 once. So possible parental combinations are (24,13) and (24,19).

String 2	1 1 0 0 0	Parental combination 1
String 1	0 1 1 0 1	
String 2	1 1 0 0 0	Parental combination 2
String 4	1 0 0 1 1	

## Step 5-

Parents

Offsprings

String 2	1 1 0 0 0	1 1 0 0 1
String 1	0 1 1 0 1	0 1 1 0 0
String 2	1 1 0 0 0	1 1 0 1 1
String 4	1 0 0 1 1	1 0 0 0 0

### Step 6-

String No	Offspring 1	X Value	F(x) value
1	0 1 1 0 0	12	144
2	1 1 0 0 1	25	625
3	1 1 0 1 1	27	729
4	1 0 0 0 0	10	256

We can see that the maximum  $f(x)$  value has increased from 576 to 729.

### Step 7-

Now we'll take these four offsprings as parents and repeat the process until our termination condition is not satisfied.

# Genetic algorithm: Hypothesis space search

- Genetic algorithms employ a randomized search method to seek maximally fit hypotheses.
- It replaces the parent hypotheses with an offspring that can be very different from the parent. Due to this reason, genetic algorithm search has lower chances of it falling into the same kind of local minima.
- There is one practical difficulty that is often encountered in genetic algorithms, it is **crowding**.



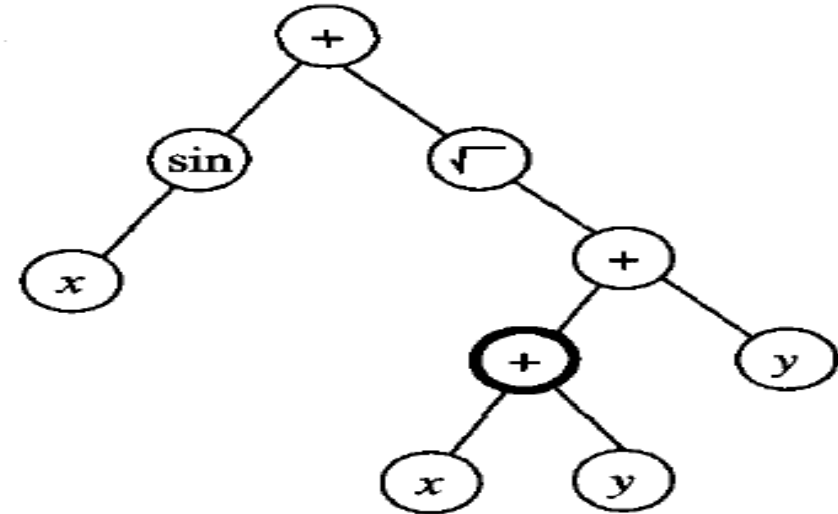
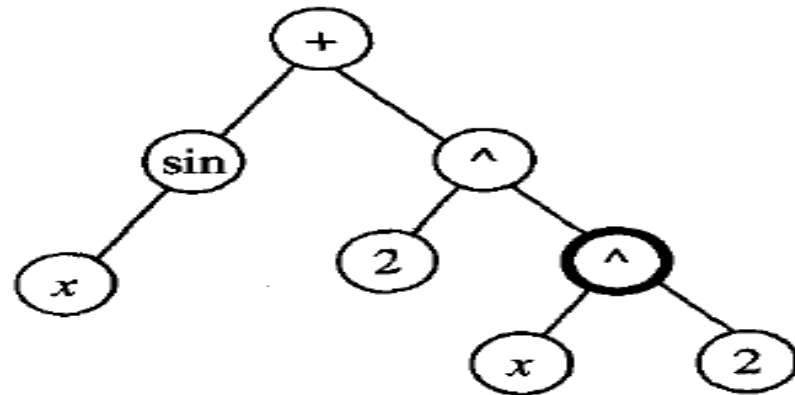
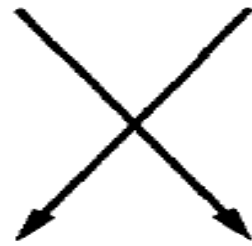
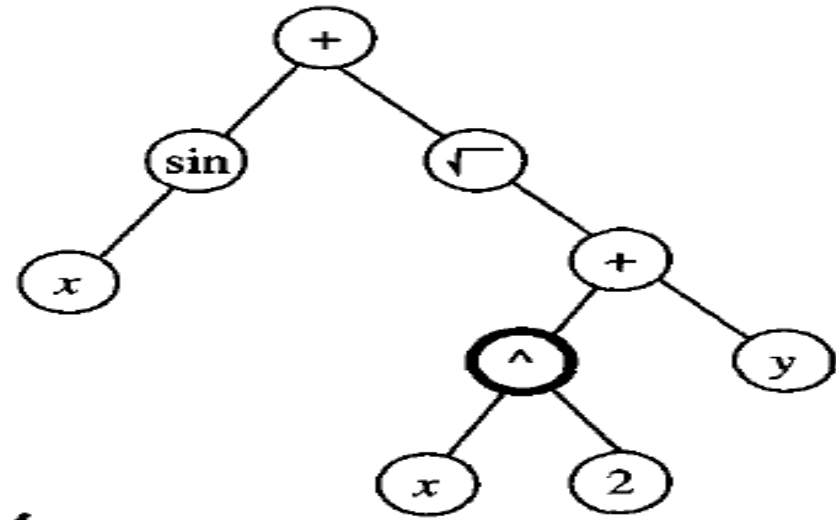
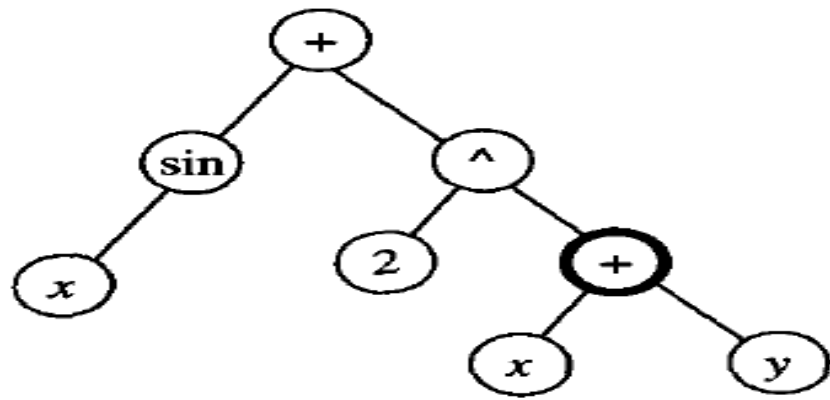
- Crowding can be defined as the phenomenon in which some individuals that are more fit in comparison to others, reproduce quickly, therefore the copies of this individual take over a larger fraction of the population.
- The negative impact of crowding is that it reduces the diversity of the population, thereby slowing further progress by the GA.



# Genetic Programming

- Genetic programming (GP) is a form of evolutionary computation in which the individuals in the evolving population are computer programs rather than bit strings.
- Koza (1992) describes the basic genetic programming approach and presents a broad range of simple programs that can be successfully learned by GP.
- Programs manipulated by a GP are typically represented by trees corresponding to the parse tree of the program. Each function call is represented by a node in the tree, and the arguments to the function are given by its descendant nodes.

- For Example: Tree representation for the function  $\sin(x) + \sqrt{x^2 + y}$ .
- To apply genetic programming to a particular domain, the user must define the primitive functions to be considered.
- As in a genetic algorithm, the prototypical genetic programming algorithm maintains a population of individuals (in this case, program trees).
- On each iteration, it produces a new generation of individuals using selection, crossover, and mutation.
- The fitness of a given individual program in the population is typically determined by executing the program on a set of training data.
- Crossover operations are performed by replacing a randomly chosen subtree of one parent program by a subtree from the other parent program.



# Models of Evolution And Learning

- In many natural systems, individual organisms learn to adapt significantly during their lifetime. At the same time, biological and social processes allow their species to adapt over a time frame of many generations. One interesting question regarding evolutionary systems is "What is the relationship between learning during the lifetime of a single individual, and the longer time frame species-level learning afforded by evolution?"
- **Lamarckian Evolution**
- **Baldwin Effect**

# Lamarckian Evolution

- Lamarck was a scientist who, in the late nineteenth century, proposed that evolution over many generations was directly influenced by the experiences of individual organisms during their lifetime.
- In particular, he proposed that experiences of a single organism directly affected the genetic makeup of their offspring.
- If an individual learned during its lifetime to avoid some toxic food, it could pass this trait on genetically to its offspring, which therefore would not need to learn the trait.

# Baldwin Effect

- It explains about the learning behavior.
- Baldwin proposed that individual learning can explain evolutionary phenomena that appear to require Lamarckian inheritance of acquired characteristics.
- The ability of individuals to learn can guide the evolutionary process.
- He focused on two things:
  - Genotype: genetic code (DNA)
  - Phenotype: characteristics (behavior)

# Parallelizing Genetic Algorithms

- GAS are naturally suited to parallel implementation, and a number of approaches to parallelization have been explored.
- *Coarse grain* approaches to parallelization subdivide the population into somewhat distinct groups of individuals, called *demes*. (*divided into fewer components*)
- Each deme is assigned to a different computational node, and a standard GA search is performed at each node. Communication and cross-fertilization between demes occurs on a less frequent basis than within demes.
- Transfer between demes occurs by a *migration* process, in which individuals from one deme are copied or transferred to other demes.
- One benefit of such approaches is that it reduces the crowding problem often encountered in nonparallel GAS, in which the system falls into a local optimum due to the early appearance of a genotype that comes to dominate the entire population.

- *Fine grain* approaches
- In contrast to coarse-grained parallel implementations of GAS, fine-grained implementations typically assign one processor per individual in the population.
- Detailed description which deals with much smaller components.



# Learning Sets of Rules

- It is useful to learn the target function represented as a set of if-then rules that jointly define the function.
- Many algorithms are there that directly learn rule sets defining two parameters:
  1. They are designed to learn first order rules that contain variables.
  2. The algorithm uses sequential covering algorithms that learn one rule at a time to incrementally grow the final set of rules.

# Sequential Covering 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 way, it covers all the rules involved with it in a sequential manner during the training phase.

# 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', where a rule is defined as: (Fig.2)*

## General Form of Rule

$$r_i : (\text{condition}_1, \dots, \text{condition}_i) \rightarrow y_i$$

*In the beginning,*

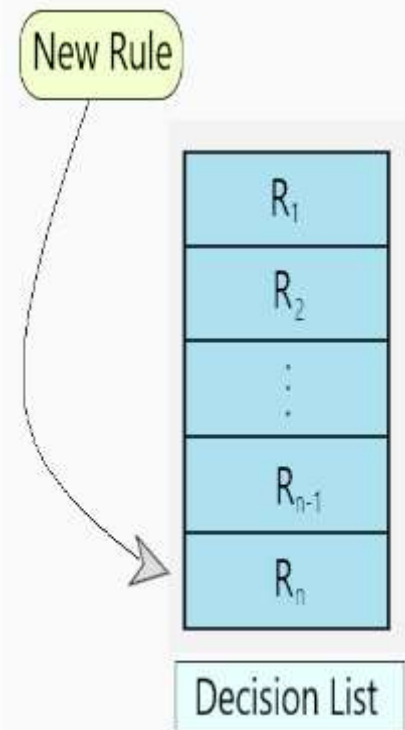
*Step 2.a – if all training examples  $\in$  class 'y', then it's classified as **positive example**.*

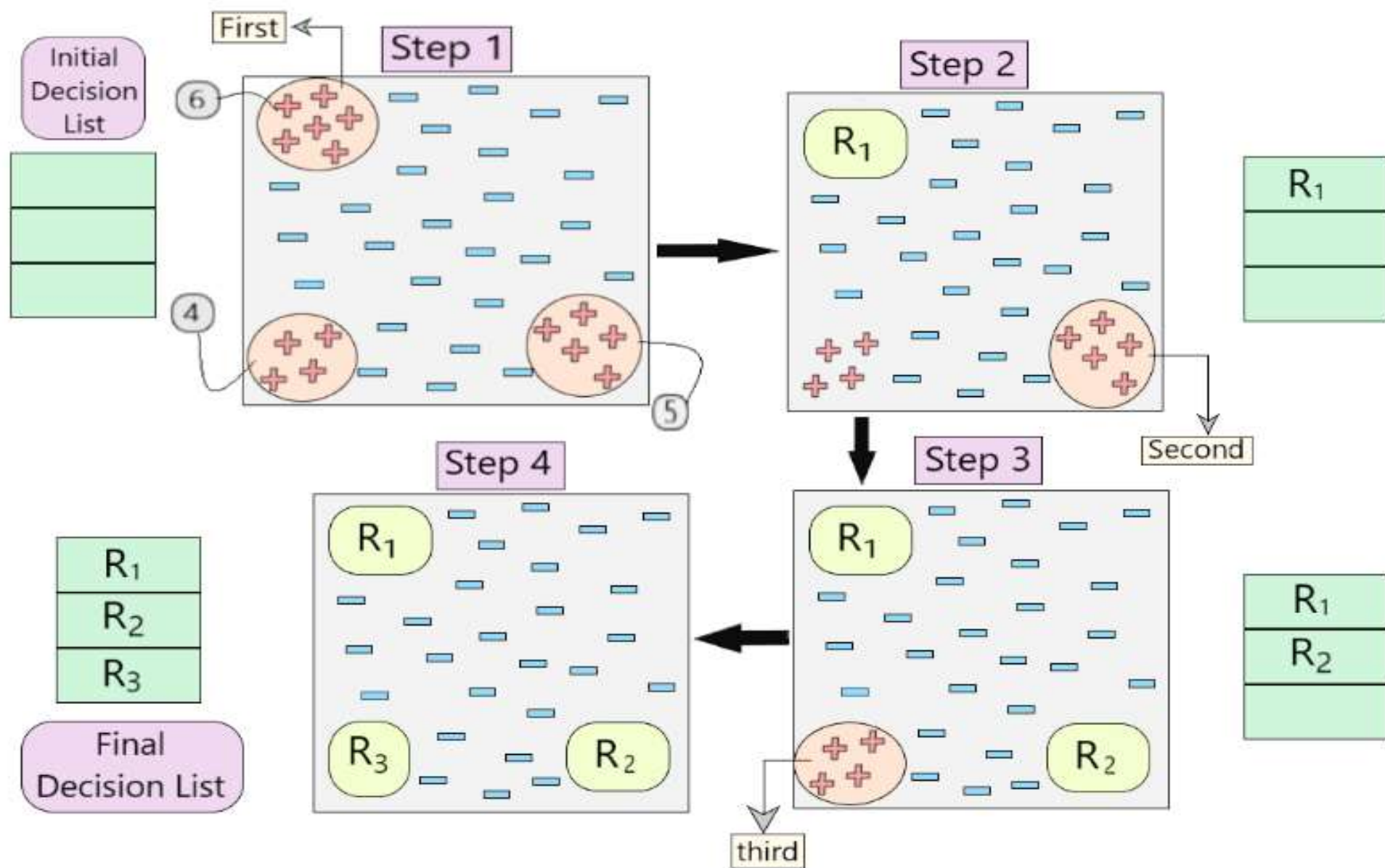
*Step 2.b – else if all training examples  $\notin$  class 'y', then it's classified as **negative example**.*

*Step 3 – The rule becomes '**desirable**' when it covers a majority of the positive examples.*

*Step 4 – When this rule is obtained, delete all the training data associated with that rule. (i.e. when the rule is applied to the dataset, it covers most of the training data, and has to be removed)*

*Step 5 – The new rule is added to the bottom of decision list, 'R'. (Fig.3)*





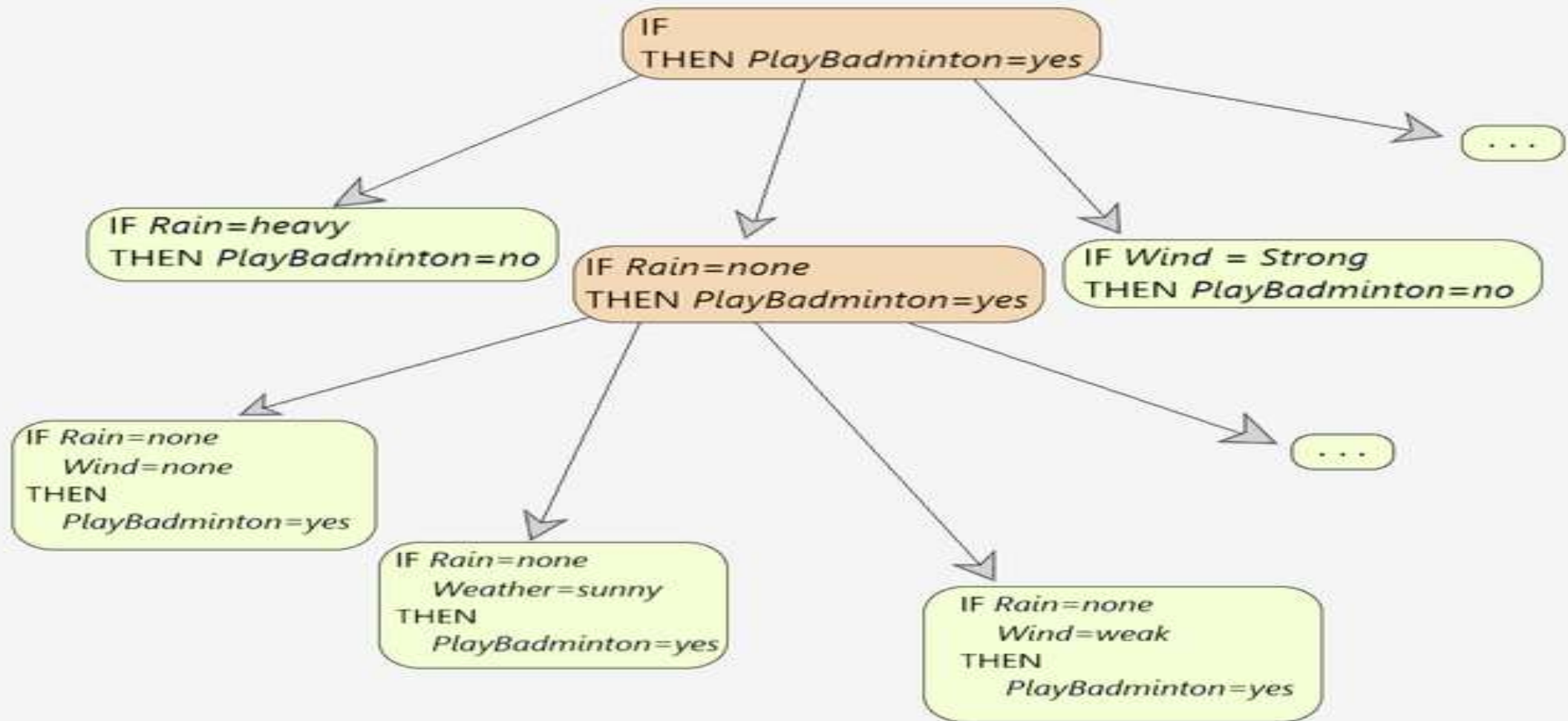
- Let us understand step by step how the algorithm is working in the example shown
  - First, we created an empty decision list. During Step 1, we see that there are three sets of positive examples present in the dataset. So, as per the algorithm, we consider the one with maximum no of positive example. (6, as shown in Step 1)
  - Once we cover these 6 positive examples, we get our first rule  $R_1$ , which is then pushed into the decision list and those positive examples are removed from the dataset. (as shown in Step 2)
  - Now, we take the next majority of positive examples (5, as shown in Step 2) and follow the same process until we get rule  $R_2$ . (Same for  $R_3$ )
  - In the end, we obtain our final decision list with all the desirable rules.
- 
- Sequential Learning is a powerful algorithm for generating rule-based classifiers in Machine Learning. It uses 'Learn-One-Rule' algorithm as its base to learn a sequence of disjunctive rules.

- **General to Specific Beam Search**

- One effective approach to implement LEARN-ONE-RULE is to organize the hypothesis space search in the same general fashion as the ID3 algorithm, but to follow only the most promising branch in the tree at each step.
- The search begins by considering the most general rule precondition possible (the empty test that matches every instance), then greedily adding the attribute test that most improves rule performance measured over the training examples.
- Once this test has been added, the process is repeated by greedily adding a second attribute test and so on.
- This way this process grows the hypothesis by greedily adding new attributes tests until the hypothesis reaches an acceptable level of performance.



# Learn-One-Rule



# First-Order Logic

- All expressions in first-order logic are composed of the following attributes:

1. constants — e.g. tyler, 23, a
2. variables — e.g. A, B, C
3. predicate symbols — e.g. male, father (True or False values only)
4. function symbols — e.g. age (can take on any constant as a value)
5. connectives — e.g.  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\rightarrow$ ,  $\leftarrow$
6. quantifiers — e.g.  $\forall$ ,  $\exists$

**Term:** It can be defined as any constant, variable or function applied to any term. e.g. **age(bob)**

**Literal:** It can be defined as any predicate or negated predicate applied to any terms. e.g. **female(sue), father(X, Y)**

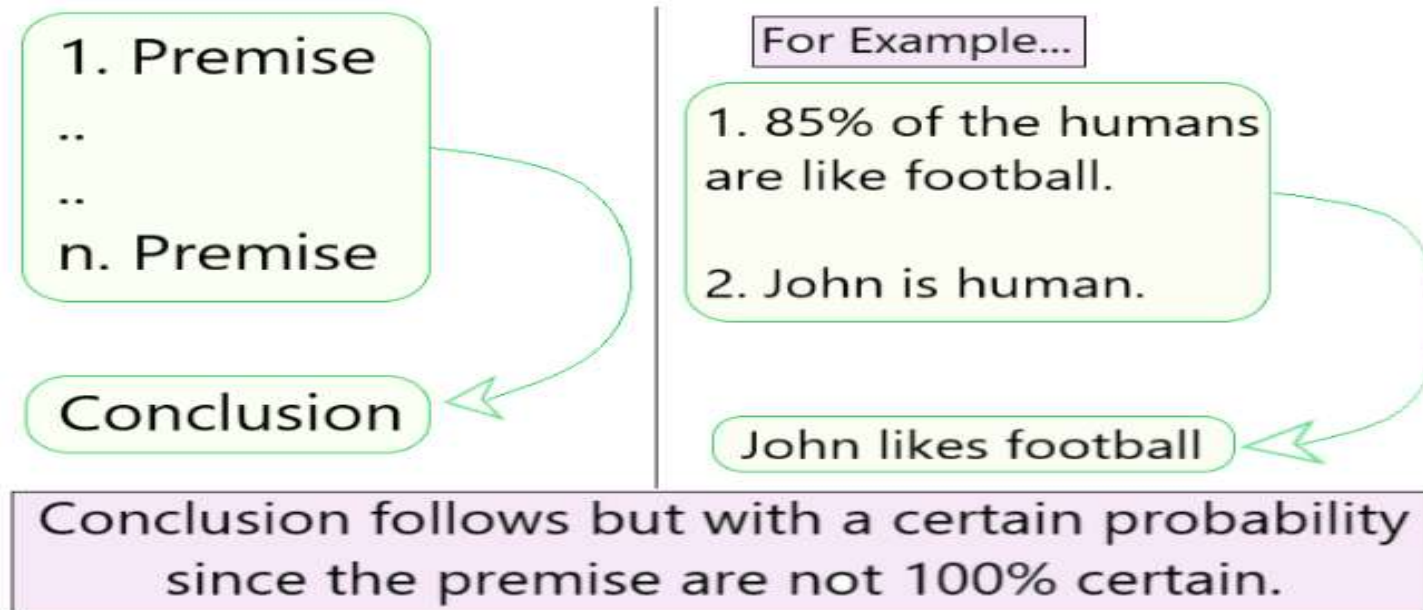


# First Order Inductive Learner (FOIL)

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.

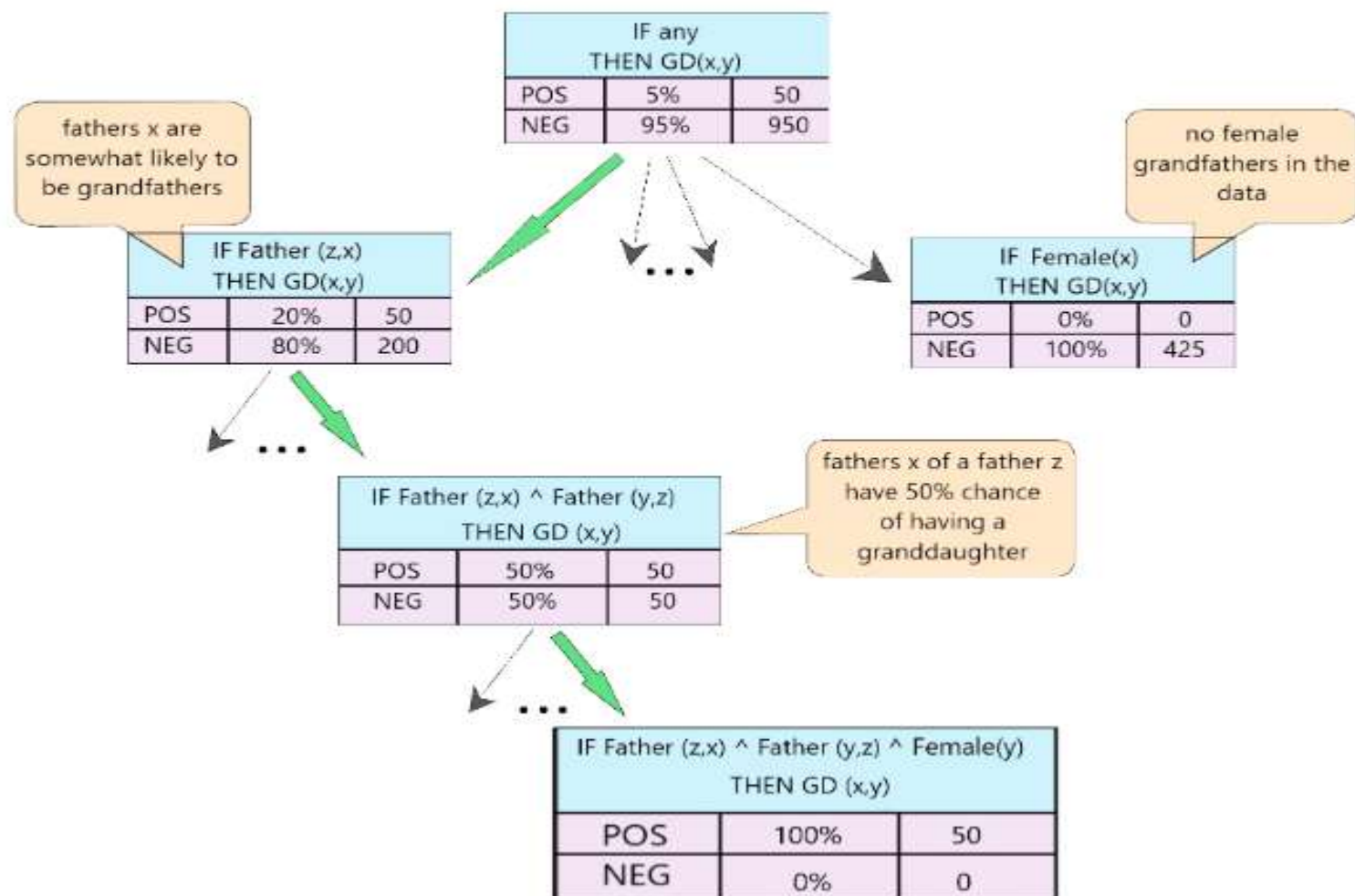


**FOIL(Target predicate, predicates, examples)**

- Pos  $\leftarrow$  positive examples
- Neg  $\leftarrow$  negative examples
- Learned rules  $\leftarrow \{\}$
- while Pos, do
  - //Learn a NewRule**
  - NewRule  $\leftarrow$  the rule that predicts target-predicate with no preconditions
  - NewRuleNeg  $\leftarrow$  Neg
  - while NewRuleNeg, do
    - Add a new literal to specialize NewRule
    - 1. Candidate\_literals  $\leftarrow$  generate candidates for newRule based on Predicates
    - 2. Best\_literal  $\leftarrow$   
$$\operatorname{argmax}_{L \in \text{Candidate literals}} \mathbf{Foil\_Gain}(L, \text{NewRule})$$
    - 3. add Best\_literal to NewRule preconditions
    - 4. NewRuleNeg  $\leftarrow$  subset of NewRuleNeg that satisfies NewRule preconditions
  - Learned rules  $\leftarrow$  Learned rules + NewRule
  - Pos  $\leftarrow$  Pos - {members of Pos covered by NewRule}
- Return Learned rules

## Working of the Algorithm:

In the algorithm, the inner loop is used to generate a new best rule. Let us consider an example and understand the step-by-step working of the algorithm.



Say we are trying to predict the **Target-predicate- *GrandDaughter(x,y)***.  
We perform the following steps: [Refer Fig 2]

**Step 1 - NewRule = GrandDaughter(x,y)**

**Step 2 -**

2.a - Generate the candidate\_literals.

(Female(x), Female(y), Father(x,y), Father(y.x),  
Father(x,z), Father(z,x), Father(y,z), Father(z,y))

2.b - Generate the respective candidate literal negations.

(¬Female(x), ¬Female(y), ¬Father(x,y), ¬Father(y.x),  
¬Father(x,z), ¬Father(z,x), ¬Father(y,z), ¬Father(z,y))

**Step 3 -** FOIL might greedily select Father(x,y) as most promising, then  
**NewRule = GrandDaughter(x,y) ← Father(y,z) [Greedy approach]**

**Step 4 -** Foil now considers all the literals from the previous step as well as:  
(Female(z), Father(z,w), Father(w,z), etc.) and their negations.

**Step 5 -** Foil might select Father(z,x), and on the next step Female(y) leading to  
**NewRule = GrandDaughter (x,y) ← Father(y,z) ∧ Father(z,x) ∧ Female(y)**

**Step 6 -** If this greedy approach covers only positive examples it terminates  
the search for further better results.

FOIL now **removes all positive examples covered by this new rule**.  
If more are left then the outer while loop continues.

## FOIL: Performance Evaluation Measure

The performance of a new rule is not defined by its entropy measure (like the *PERFORMANCE* method in Learn-One-Rule algorithm).

FOIL uses a gain algorithm to determine which new specialized rule to opt. Each rule's utility is estimated by the number of bits required to encode all the positive bindings. [Eq.1]

$$Foil\ Gain(L, R) \equiv t \left( \log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right)$$

where,

**L** is the candidate literal to add to rule **R**

$p_0$  = number of positive bindings of R

$n_0$  = number of negative bindings of R

$p_1$  = number of positive binding of R + L

$n_1$  = number of negative bindings of R + L

$t$  = number of positive bindings of R also covered by R + L

- FOIL Algorithm is another rule-based learning algorithm that extends on the Sequential Covering + Learn-One-Rule algorithms and uses a different Performance metrics (other than entropy/information gain) to determine the best rule possible.

# Reinforcement learning

- Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task.
- In the absence of a training dataset, it is bound to learn from its experience.
- Reinforcement Learning (RL) is the science of decision making. It is about learning the optimal behavior in an environment to obtain maximum reward.



- In RL, the data is accumulated from machine learning systems that use a trial-and-error method. Data is not part of the input that we would find in supervised or unsupervised machine learning.
- Reinforcement learning uses algorithms that learn from outcomes and decide which action to take next. After each action, the algorithm receives feedback that helps it determine whether the choice it made was correct, neutral or incorrect.
- It is a good technique to use for automated systems that have to make a lot of small decisions without human guidance.
- Reinforcement learning is an autonomous, self-teaching system that essentially learns by trial and error. It performs actions with the aim of maximizing rewards, or in other words, it is learning by doing in order to achieve the best outcomes.

## Main Points

- Input: The input should be an initial state from which the model will start
- Output: There are many possible outputs as there are a variety of solutions to a particular problem
- Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
- The model keeps continues to learn.
- The best solution is decided based on the maximum reward.



## Difference between Reinforcement learning and Supervised learning:

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

## Types of Reinforcement:

There are two types of Reinforcement:

1. **Positive:** Positive Reinforcement is defined as when an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior. In other words, it has a positive effect on behavior.

Advantages of reinforcement learning are:

- Maximizes Performance
- Sustain Change for a long period of time
- Too much Reinforcement can lead to an overload of states which can diminish the results

2. **Negative:** Negative Reinforcement is defined as strengthening of behavior because a negative condition is stopped or avoided.

Advantages of reinforcement learning:

- Increases Behavior
- Provide defiance to a minimum standard of performance
- It Only provides enough to meet up the minimum behavior

## Various Practical Applications of Reinforcement Learning –

- RL can be used in robotics for industrial automation.
- RL can be used in machine learning and data processing
- RL can be used to create training systems that provide custom instruction and materials according to the requirement of students.

## Advantages and Disadvantages of Reinforcement Learning

### Advantages of Reinforcement learning

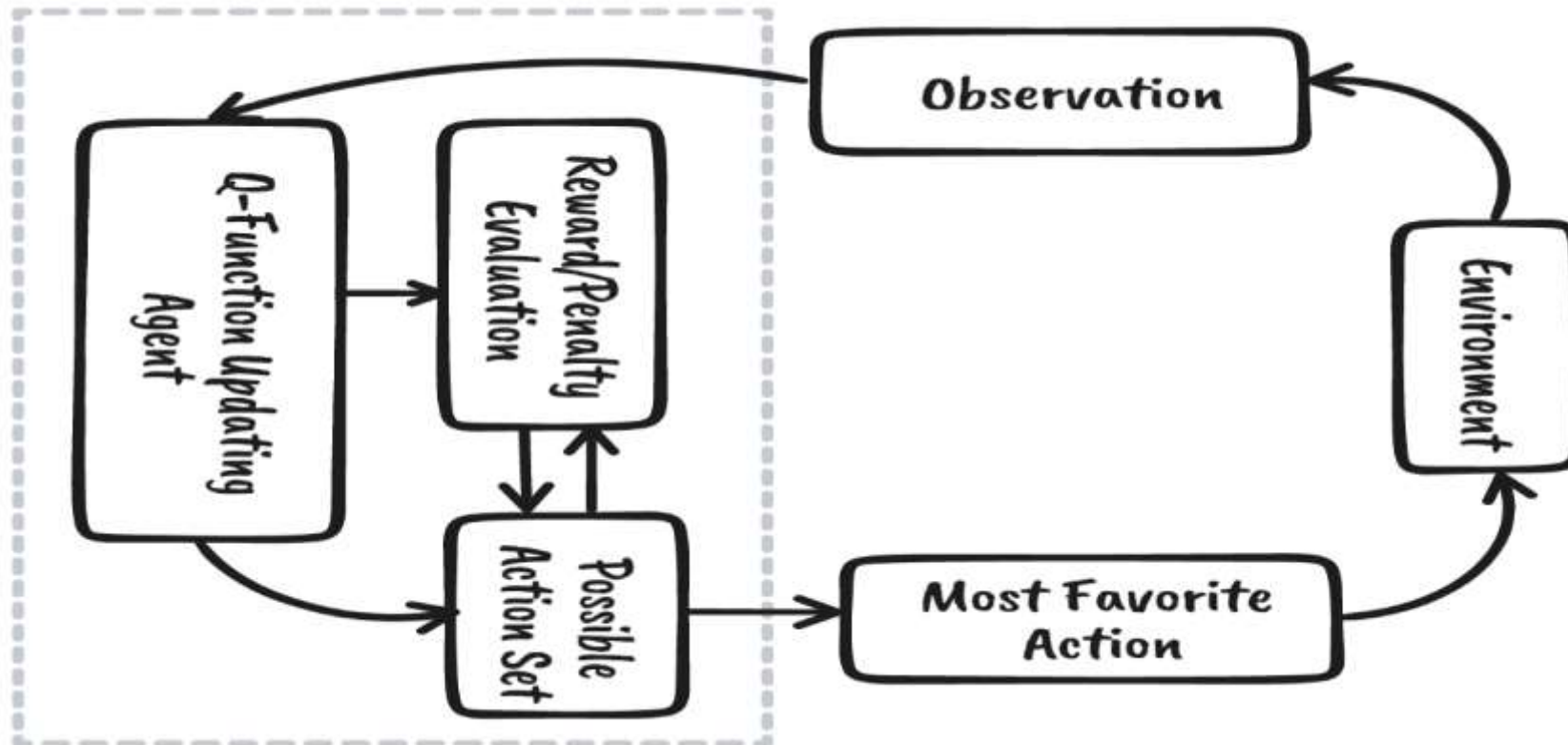
1. Reinforcement learning can be used to solve very complex problems that cannot be solved by conventional techniques.
2. The model can correct the errors that occurred during the training process.
3. In RL, training data is obtained via the direct interaction of the agent with the environment
4. Reinforcement learning can handle environments that are non-deterministic, meaning that the outcomes of actions are not always predictable. This is useful in real-world applications where the environment may change over time or is uncertain.
5. Reinforcement learning can be used to solve a wide range of problems, including those that involve decision making, control, and optimization.
6. Reinforcement learning is a flexible approach that can be combined with other machine learning techniques, such as deep learning, to improve performance.

### Disadvantages of Reinforcement learning

1. Reinforcement learning is not preferable to use for solving simple problems.
2. Reinforcement learning needs a lot of data and a lot of computation
3. Reinforcement learning is highly dependent on the quality of the reward function. If the reward function is poorly designed, the agent may not learn the desired behavior.
4. Reinforcement learning can be difficult to debug and interpret. It is not always clear why the agent is behaving in a certain way, which can make it difficult to diagnose and fix problems.

# Q-Learning Reinforcement learning

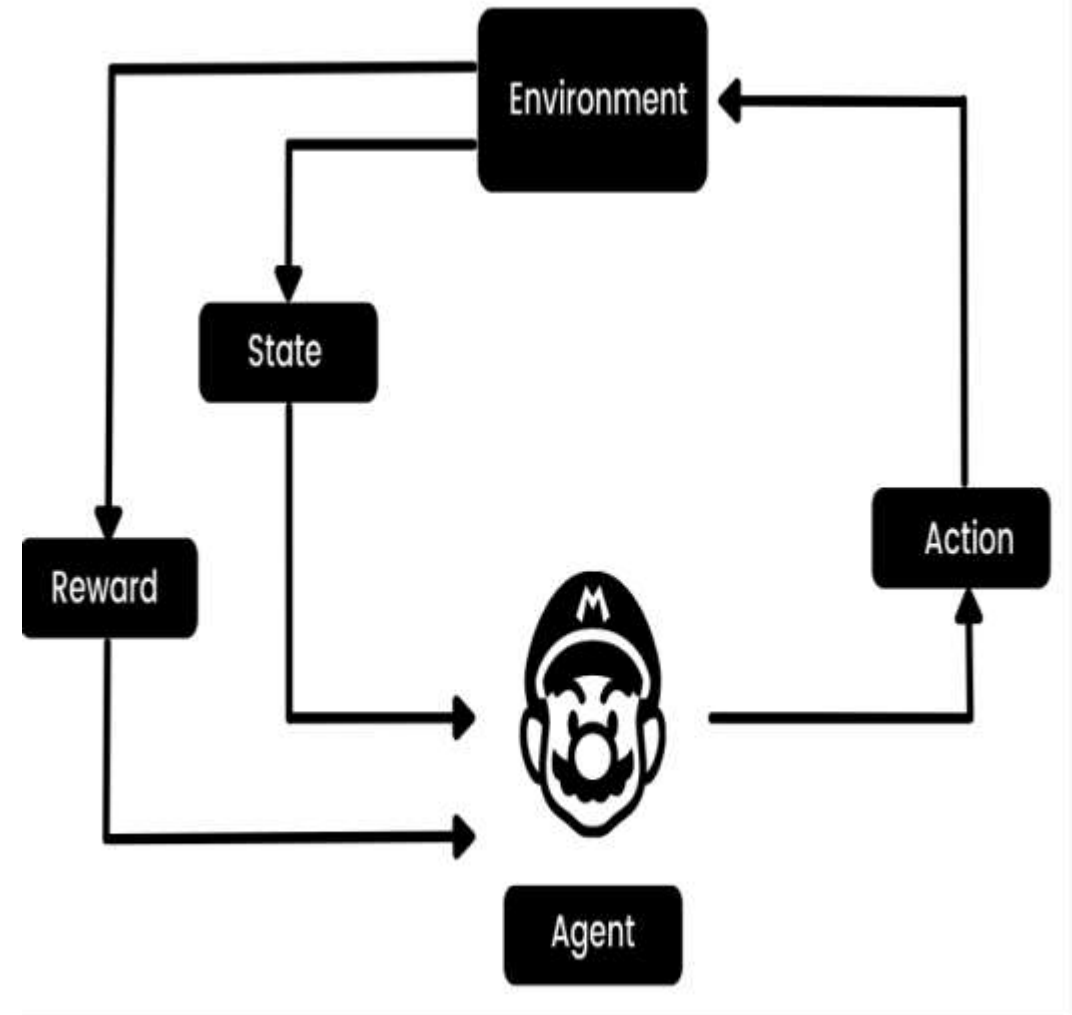
- Its a basic form of Reinforcement Learning which uses Q-values (also called action values) to iteratively improve the behavior of the learning agent.



- We can break down reinforcement learning into five simple steps:
  - 1.The agent is at state zero in an environment.
  - 2.It will take an action based on a specific strategy.
  - 3.It will receive a reward or punishment based on that action.
  - 4.By learning from previous moves and optimizing the strategy.
  - 5.The process will repeat until an optimal strategy is found.



- For example, in the Mario video game, if a character takes a random action (e.g. moving left), based on that action, it may receive a reward. After taking the action, the agent (Mario) is in a new state, and the process repeats until the game character reaches the end of the stage or dies.
- This episode will repeat multiple times until Mario learns to navigate the environment by maximizing the rewards.



# What is Q-Learning?

- Q-learning is a model-free, value-based, off-policy algorithm that will find the best series of actions based on the agent's current state. The “Q” stands for quality. Quality represents how valuable the action is in maximizing future rewards.
- The **model-based** algorithms use transition and reward functions to estimate the optimal policy and create the model. In contrast, **model-free** algorithms learn the consequences of their actions through the experience without transition and reward function.
- The **value-based** method trains the value function to learn which state is more valuable and take action. On the other hand, **policy-based** methods train the policy directly to learn which action to take in a given state.
- In the **off-policy**, the algorithm evaluates and updates a policy that differs from the policy used to take an action. Conversely, the **on-policy** algorithm evaluates and improves the same policy used to take an action.

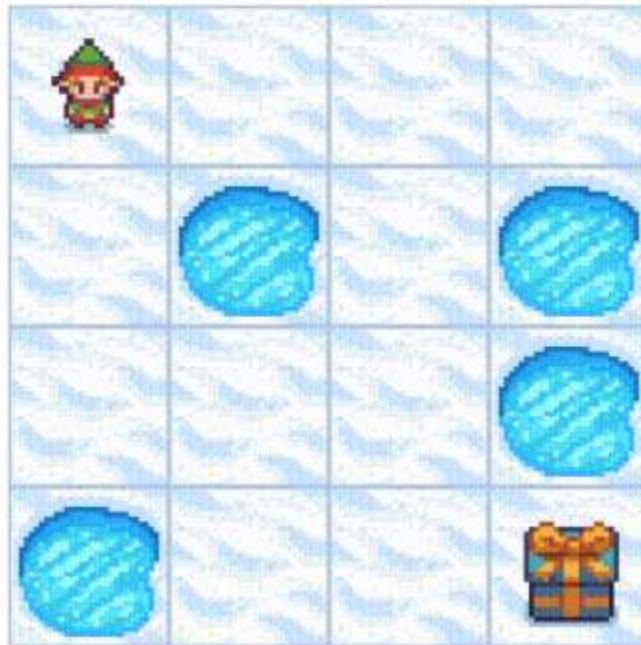
# Key Terminologies in Q-learning

- **States(s)**: the current position of the agent in the environment.
- **Action(a)**: a step taken by the agent in a particular state.
- **Rewards**: for every action, the agent receives a reward and penalty.
- **Episodes**: the end of the stage, where agents can't take new action. It happens when the agent has achieved the goal or failed.
- $Q(S_{t+1}, a)$ : expected optimal Q-value of doing the action in a particular state.
- $Q(S_t, A_t)$ : it is the current estimation of  $Q(S_{t+1}, a)$ .
- **Q-Table**: the agent maintains the Q-table of sets of states and actions.
- **Temporal Differences(TD)**: used to estimate the expected value of  $Q(S_{t+1}, a)$  by using the current state and action and previous state and action.



# How Does Q-Learning Work?

- Lets take an example of a frozen lake. In this environment, the agent must cross the frozen lake from the start to the goal, without falling into the holes. The best strategy is to reach goals by taking the shortest path.



## Q-Table

The agent will use a Q-table to take the best possible action based on the expected reward for each state in the environment. In simple words, a Q-table is a data structure of sets of actions and states, and we use the Q-learning algorithm to update the values in the table.

## Q-Function

The Q-function uses the Bellman equation and takes state(s) and action(a) as input. The equation simplifies the state values and state-action value calculation.

The diagram illustrates the Bellman equation for the Q-function,  $Q^\pi(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t, a_t]$ . The equation is presented with three colored boxes highlighting its components: a red box around  $Q^\pi(s_t, a_t)$ , a green box around the expectation term  $E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots]$ , and a purple box around the state-action pair  $| s_t, a_t]$ . Three red arrows point downwards from these boxes to their respective descriptions: 'Q-Values for the state given a particular state' for the red box, 'Expected discounted cumulative reward' for the green box, and 'Given the state and action' for the purple box.

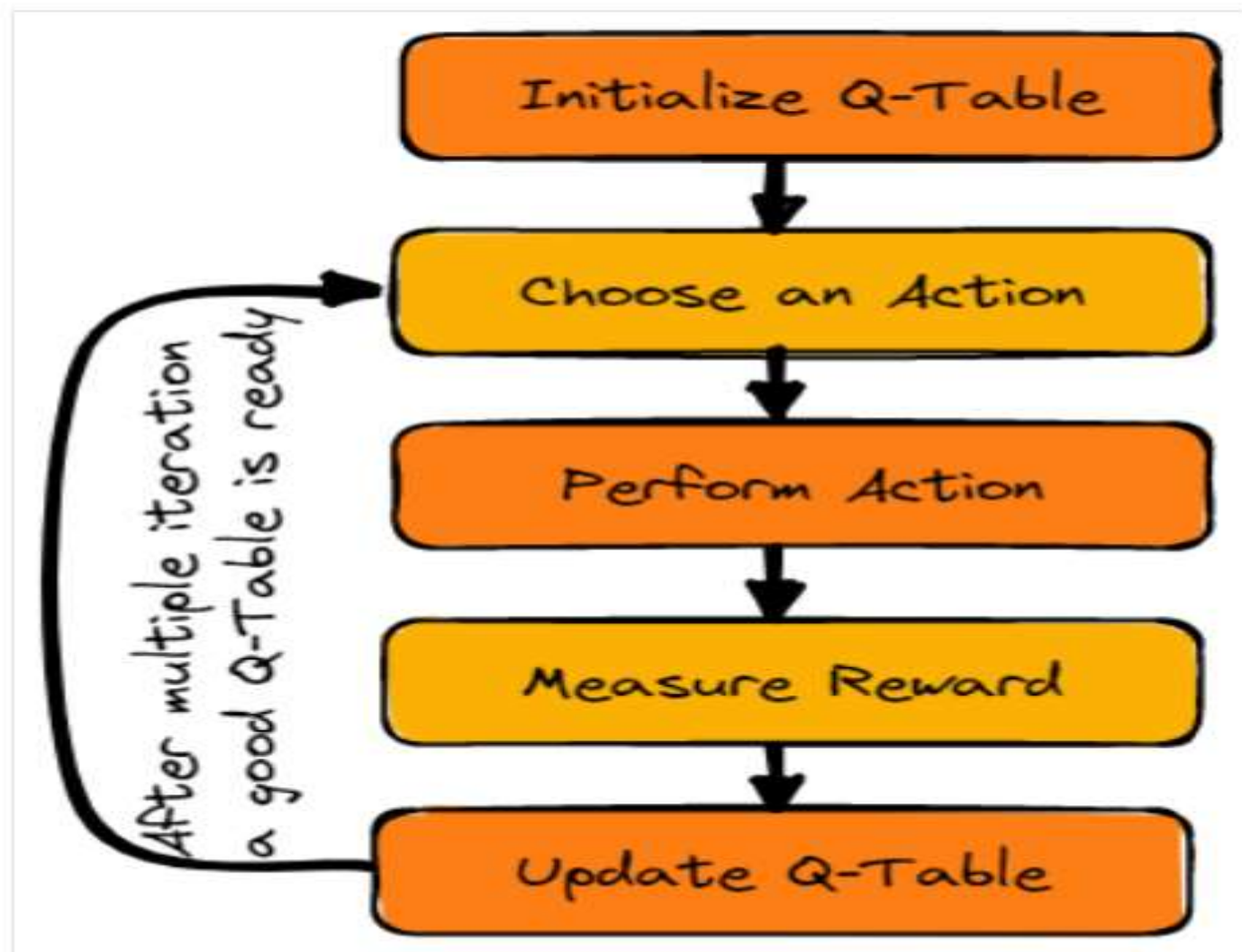
$$Q^\pi(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t, a_t]$$

Q-Values for the state given a particular state

Expected discounted cumulative reward

Given the state and action


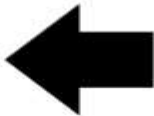


## Q-learning algorithm



## Initialize Q-Table

We will first initialize the Q-table. We will build the table with columns based on the number of actions and rows based on the number of states.

In our example, the character can move up, down, left, and right. We have four possible actions and four states(start, Idle, wrong path, and end). You can also consider the wrong path for falling into the hole. We will initialize the Q-Table with values at 0.

				
<b>Start</b>	0	0	0	0
<b>Idle</b>	0	0	0	0
<b>Hole</b>	0	0	0	0
<b>End</b>	0	0	0	0

## Choose an Action

The second step is quite simple. At the start, the agent will choose to take the random action(down or right), and on the second run, it will use an updated Q-Table to select the action.

## Perform an Action





Choosing an action and performing the action will repeat multiple times until the training loop stops. The first action and state are selected using the Q-Table. In our case, all values of the Q-Table are zero.

Then, the agent will move down and update the Q-Table using the Bellman equation. With every move, we will be updating values in the Q-Table and also using it for determining the best course of action.

Initially, the agent is in exploration mode and chooses a random action to explore the environment. The Epsilon Greedy Strategy is a simple method to balance exploration and exploitation. The epsilon stands for the probability of choosing to explore and exploits when there are smaller chances of exploring.

At the start, the epsilon rate is higher, meaning the agent is in exploration mode. While exploring the environment, the epsilon decreases, and agents start to exploit the environment. During exploration, with every iteration, the agent becomes more confident in estimating Q-values



				
<b>Start</b>	0	0	0	1
<b>Idle</b>	0	0	0	0
<b>Hole</b>	0	0	0	0
<b>End</b>	0	0	0	0

In the frozen lake example, the agent is unaware of the environment, so it takes random action (move down) to start. As we can see in the above image, the Q-Table is updated using the Bellman equation.

## Measuring the Rewards

After taking the action, we will measure the outcome and the reward.





- The reward for reaching the goal is +1
- The reward for taking the wrong path (falling into the hole) is 0
- The reward for Idle or moving on the frozen lake is also 0.

# Update Q-Table

We will update the function  $Q(S_t, A_t)$  using the equation. It uses the previous episode's estimated Q-values, learning rate, and Temporal Differences error. Temporal Differences error is calculated using Immediate reward, the discounted maximum expected future reward, and the former estimation Q-value.

The process is repeated multiple times until the Q-Table is updated and the Q-value function is maximized.

At the start, the agent is exploring the environment to update the Q-table. And when the Q-Table is ready, the agent will start exploiting and start taking better decisions.

				
Start	0	1	0	0
Idle	2	0	0	3
Hole	0	2	0	0
End	1	0	0	0

In the case of a frozen lake, the agent will learn to take the shortest path to reach the goal and avoid jumping into the holes.

