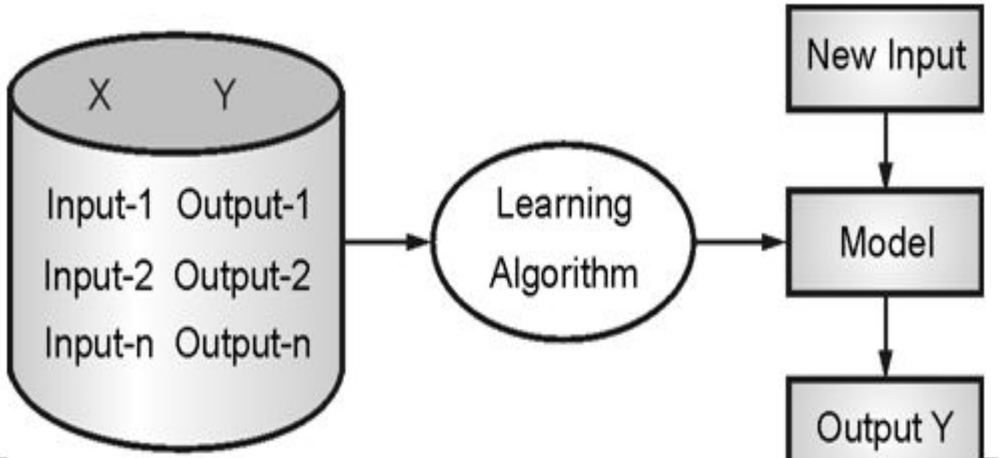
Learning

Learning: Forms of Learning, Theory of Learning, PAC learning.
Introduction to statistical learning (Introduction only)
Introduction to reinforcement learning: Learning from Rewards,
Passive Reinforcement Learning, Active reinforcement Learning

Forms of Learning

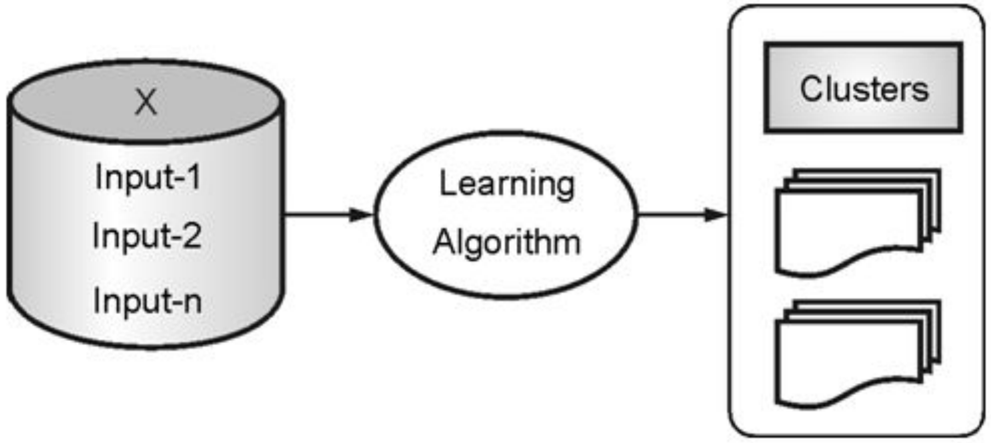
1. Supervised Learning

In this type of learning we use data which is comprises of input and corresponding output.
For every instance of data we can have input 'X' and corresponding output 'Y'.
From this ML system will build model so that given an observation 'X', it will try to find out what is corresponding 'Y'.
In supervised learning training data is labelled with the correct answers,
Two most important types of supervised learning are **classification** (where the outputs are discrete labels, as in spam filtering) and **regression** (where the outputs are real-valued).



2. Unsupervised learning

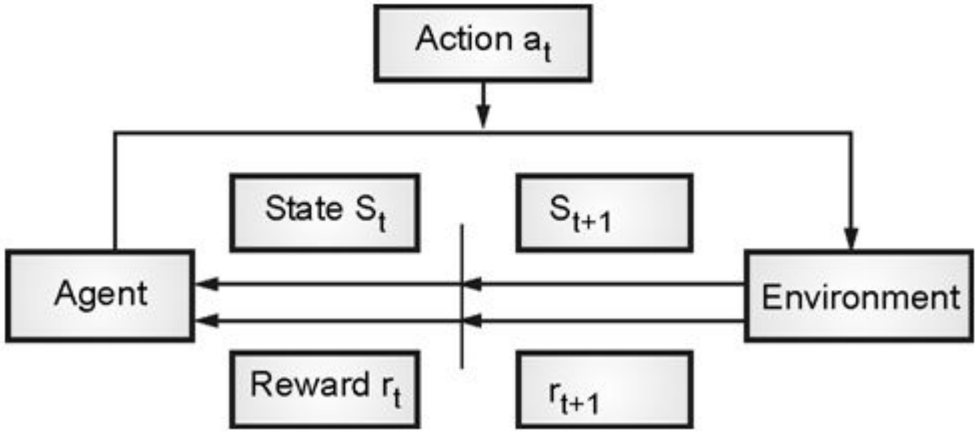
In unsupervised learning you are only given input 'X', there is no label to the data and given the data or different data points, you may want to form clusters or want to find some pattern**.**



3. Reinforcement learning

In reinforcement learning you have an agent who is acting in an environment and you want to find out what action the agent must take based on the reward or penalty that the agent gets it.

In this an agent (e.g., a robot or controller) seeks to learn the optimal actions to take based the outcomes of past actions.



Any component of an agent can be improved by learning from data.
The improvements, and the techniques used to make them, depend on four major factors:

1. Which component is to be improved
2. What prior knowledge the agent already has.
3. What representation is used for the data and the component.
4. What feedback is available to learn from

1.Components to be learned

The components of these agents include:

1. A direct mapping from conditions on the current state to actions.
2. A means to infer relevant properties of the world from the percept sequence.
3. Information about the way the world evolves and about the results of possible actions the agent can take.
4. Utility information indicating the desirability of world states.
5. Action-value information indicating the desirability of actions.
6. Goals that describe classes of states whose achievement maximizes the agent's utility.

Each of these components can be learned.

2. Representation and 3. prior knowledge

Several examples of representations for agent components:

- propositional and first-order logical sentences for the components in a logical agent;
- Bayesian networks for the inferential components of a decision-theoretic agent, and so on.

Effective learning algorithms have been devised for all of these representations.

Feedback to learn from

There are three types of feedback that determine the three main types of learning

- In **unsupervised learning** the agent learns patterns in the input even though no explicit feedback is supplied. The most common unsupervised learning task is clustering: detecting potentially useful clusters of input examples.
- In **reinforcement learning** the agent learns from a series of reinfo
- For example, the lack of a tip at the end of the journey gives the taxi agent an indication that it did something wrong —rewards or punishments
- In **supervised learning** the agent observes some example input–output pairs and learns a function that maps from input to output.

Theory of learning

- How can we be sure that our learning algorithm has produced a hypothesis that will predict the correct value for previously unseen inputs?
- In formal terms, how do we know that the hypothesis h is close to the target function f if we don’t know what f is?
- how many examples do we need to get a good h?
- What hypothesis space should we use?
- If the hypothesis space is very complex, can we even find the best h, or do we have to settle for a local maximum space of hypotheses?
- How complex should h be? How do we avoid overfitting?

PAC learning algorithm.

- Questions like this are addressed by computational learning theory, which lies at the intersection of AI, statistics and theoretical computer science.
- The underlying principle is that any hypothesis that is seriously wrong will almost certainly be "found out" with high probability after a small number of examples, because it will make an incorrect prediction.
- Thus, **any hypothesis that is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong: that is, it must be probably approximately correct.**
- Any learning algorithm that returns hypotheses that are **probably approximately correct** is called a PAC learning algorithm.

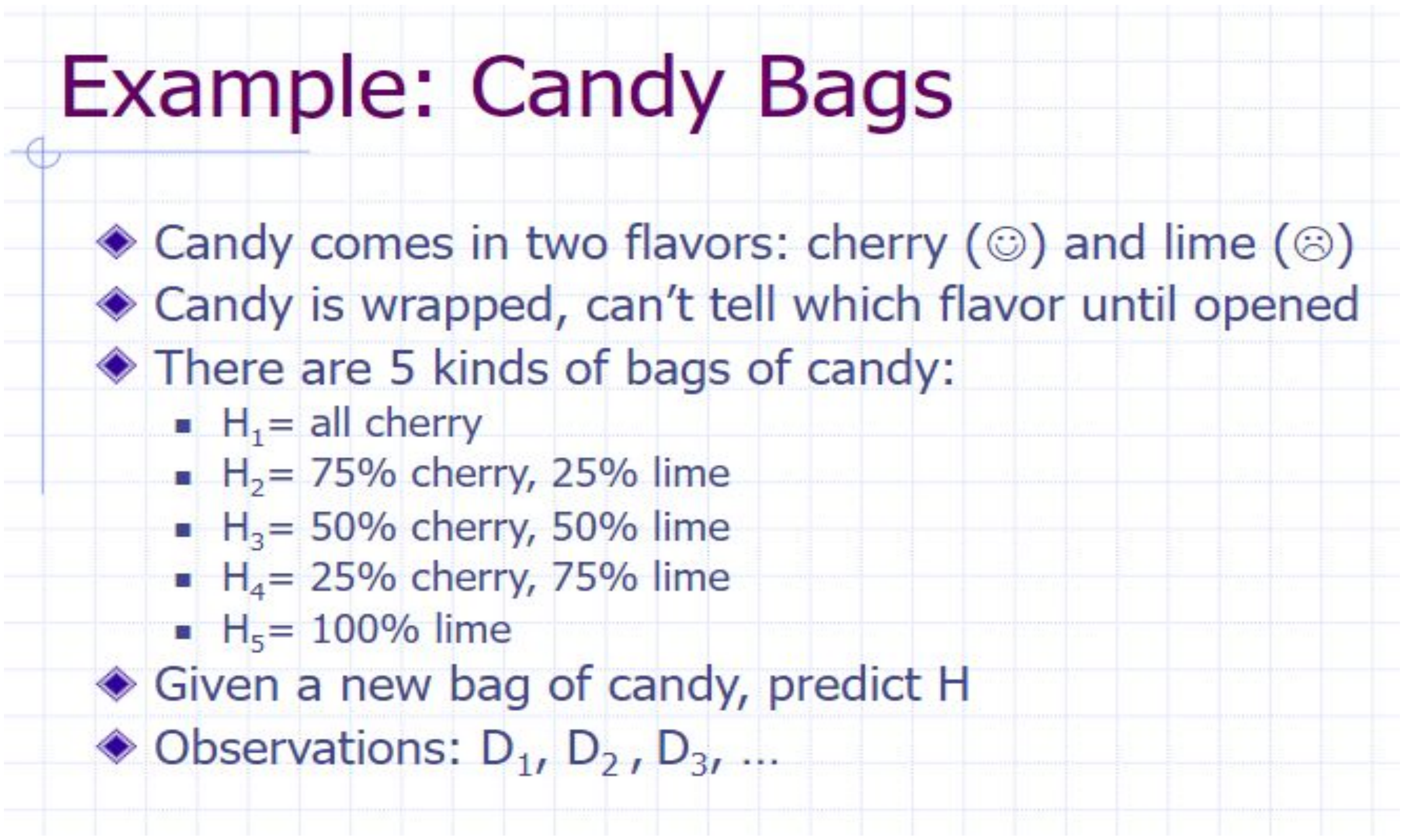
- PAC-learning theorems, like all theorems, are logical consequences of axioms.
- For PAC learning, the stationarity assumption which says that future examples are going to be drawn from the same fixed distribution **P**(E)=**P**(X, Y) as past examples. (Note that we do not have to know what distribution that is, just that it doesn't change.)

- The **error rate** of a hypothesis h, defined informally earlier, is defined formally
- here as the expected generalization error for examples drawn from the stationary distribution: $\text{error}(h) = GenLoss_{L_{0/1}}(h) = \sum_{x,y} L_{0/1}(y, h(x))\, P(x, y)$

- error(h) is the probability that h misclassifies a new example.

- A hypothesis h is called **approximately correct** if error(h) ≤ □, where □ is a small
- constant. We will show that we can find an N such that, after seeing N examples, with high probability, all consistent hypotheses will be approximately correct. One can think of an approximately correct hypothesis as being "close" to the true function in hypothesis space

Statistical Learning



Example: Candy Bags

- Candy comes in two flavors: cherry (☺) and lime (☹)
- Candy is wrapped, can't tell which flavor until opened
- There are 5 kinds of bags of candy:
  - H_1= all cherry
  - H_2= 75% cherry, 25% lime
  - H_3= 50% cherry, 50% lime
  - H_4= 25% cherry, 75% lime
  - H_5= 100% lime
- Given a new bag of candy, predict H
- Observations: D_1, D_2 , D_3, …

- **Bayesian learning** simply calculates the probability of each hypothesis, given the data,
- and makes predictions on that basis. That is, the predictions are made by using *all* the hypotheses,
- weighted by their probabilities, rather than by using just a single "best" hypothesis.
- In this way, learning is reduced to probabilistic inference. Let **D** represent all the data, with observed value **d**; then the probability of each hypothesis is obtained by Bayes' rule:

Bayesian Learning

- Calculate the probability of each hypothesis, given the data, and make prediction weighted by this probability (i.e. use all the hypothesis, not just the single best)

$$P(h_i \mid d) = \frac{P(d|h_i)P(h_i)}{P(d)} = \alpha P(d \mid h_i)P(h_i)$$

- Now, if we want to predict some unknown quantity X

$$P(X \mid d) = \sum_i P(X \mid h_i)P(h_i \mid d)$$

Bayesian Learning cont.

- Calculating P(h|d)

$$P(h_i \mid d) \propto \underset{\text{likelihood}}{\underset{\uparrow}{P(d \mid h_i)}}\,\underset{\text{prior}}{\underset{\uparrow}{P(h_i)}}$$

- Assume the observations are i.i.d.—independent and identically distributed

$$P(d \mid h_i) = \prod_j P(d_j \mid h_i)$$

Making Statistical Inferences

- Bayesian –
  - predictions made using all hypothesis, weighted by their probabilities
- MAP – maximum a posteriori
  - uses the single most probable hypothesis to make prediction
  - often much easier than Bayesian; as we get more and more data, closer to Bayesian optimal

Reinforcement learning

- E.g.the problem of learning to play chess.
- The agent needs to know that something good has happened when it (accidentally) checkmates the opponent, and that something bad has happened when it is checkmated—or vice versa, This kind of feedback is called a **reward**, or **reinforcement**.
- In games like chess, the reinforcement is received only at the end of the game. In other environments, the rewards come more frequently.
- In ping-pong, each point scored can be considered a reward;

- 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

- Reinforcement Learning(RL) is a type of learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences.

- Both active and passive reinforcement learning are types of RL.
- In case of passive RL, the agent's policy is fixed which means that it is ***told what to do***. In contrast to this,
- In case of active RL, an agent ***needs to decide what to do*** as there's no fixed policy that it can act on.

- Therefore, the goal of a passive RL agent is to execute a fixed policy (sequence of actions) and evaluate it while that of an active RL agent is to act and learn an optimal policy.

Key terms in **Reinforcement Learning problem**

- **Environment —** Physical world in which the agent operates
- **State —** Current situation of the agent
- **Reward —** Feedback from the environment
- **Policy —** Method to map agent’s state to actions
- **Value —** Future reward that an agent would receive by taking an action in a particular state

- **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

- **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

Types of Reinforcement learning

- Active and passive reinforcement learning are two approaches that differ in how they collect and use data to learn.
- ***Active reinforcement learning*** is an approach in which an agent interacts with an environment by selecting actions and observing the resulting rewards. The goal is for the agent to learn an optimal policy that maximizes its long-term cumulative reward. In **active reinforcement learning, the agent takes an active role in exploring the environment to learn how to make good decisions.**
- For example, imagine a robot tasked with navigating a maze to find a reward at the end. In active reinforcement learning, the robot would explore the maze by taking different paths and observing the resulting rewards (e.g., reaching the end of the maze). The robot would use this information to update its policy on which actions to take in different situations to maximize its reward

- ***Passive reinforcement learning***, on the other hand, is an approach in which an agent learns from a fixed dataset of examples. The goal is for the agent to learn a policy that maximizes its expected reward on this fixed dataset. In passive reinforcement learning, the agent does not interact with the environment but rather learns from the experiences of others (bcoz policy is fixed).

- For example, imagine a dataset of driving behaviors collected from human drivers. In passive reinforcement learning, an autonomous vehicle could learn to drive by observing this dataset and learning a policy that imitates the human drivers' behaviors.

- In summary, active reinforcement learning involves an agent actively exploring and learning from its environment, while passive reinforcement learning involves an agent learning from a fixed dataset of examples.

Types of active reinforcement learning

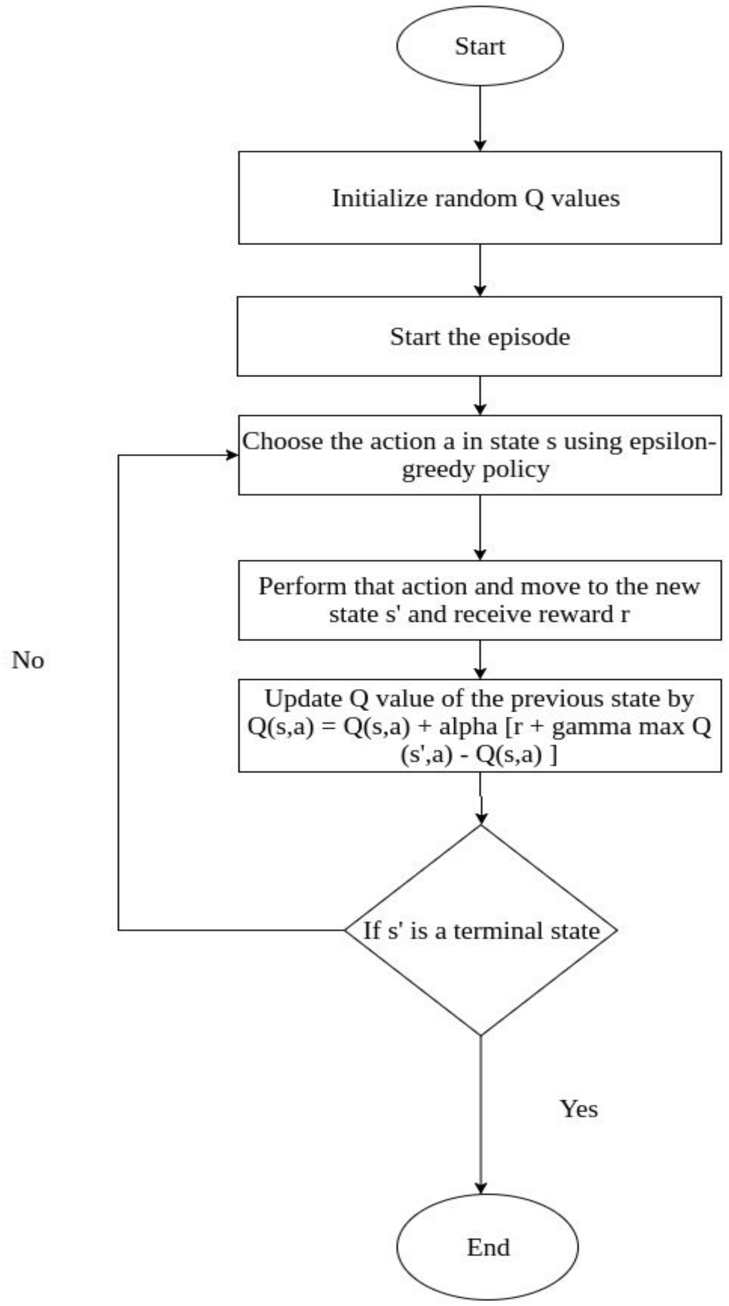
There are several types of active reinforcement learning, which differ in the way they explore the environment and learn from it:

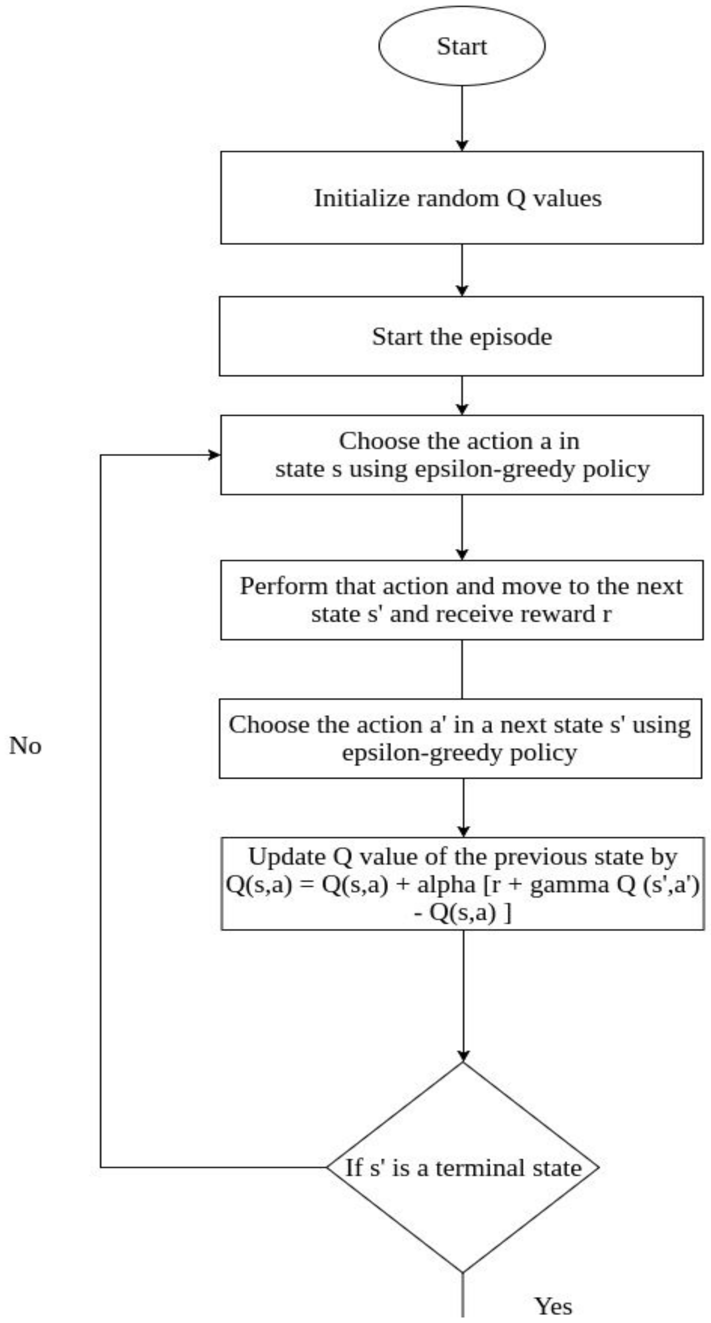
- ***Q-learning:*** Q-learning is a popular approach to active reinforcement learning in which the agent learns a state-action value function. The agent uses this function to select actions that maximize its expected reward. Q-learning is useful when the agent can easily observe the environment's state and the reward signal is immediate.

- ***SARSA (***State-action-reward-state-action)***:*** SARSA is another popular approach to active reinforcement learning that is similar to Q-learning but takes into account the agent's current policy. The agent learns a state-action value function and uses it to select actions that maximize its expected reward, given its current policy. SARSA is useful when the agent's policy needs to be taken into account.

- ***Temporal difference learning:*** Temporal difference learning is a family of algorithms that update the agent's value function based on the difference between the predicted and actual reward received. This approach is useful when the agent's policy can be learned incrementally over time.

- In summary, active reinforcement learning involves an agent actively exploring and learning from its environment, and there are several different types of active reinforcement learning that differ in the way they explore the environment and learn from it.


Start
Initialize random Q values
Start the episode
Choose the action a in state s using epsilon-greedy policy
Perform that action and move to the new state s' and receive reward r
Update Q value of the previous state by Q(s,a) = Q(s,a) + alpha [r + gamma max Q (s',a) - Q(s,a)]
If s' is a terminal state
No
Yes
End


Start
Initialize random Q values
Start the episode
Choose the action a in state s using epsilon-greedy policy
Perform that action and move to the next state s' and receive reward r
Choose the action a' in a next state s' using epsilon-greedy policy
Update Q value of the previous state by Q(s,a) = Q(s,a) + alpha [r + gamma Q (s',a') - Q(s,a)]
If s' is a terminal state
No
Yes

- The main difference between SARSA and Q-learning is the way they update their Q-values. In SARSA, the update rule depends on the next state and action that the agent takes, while in Q-learning, the update rule depends on the maximum Q-value of the next state and all possible actions.

- Specifically, the SARSA update rule is:
  Q(s, a) <- Q(s, a) + alpha * [r + gamma * Q(s', a') - Q(s, a)]
- where Q(s, a) is the estimated Q-value of taking action a in state s, r is the reward received from taking action a in state s, s' is the resulting state, a' is the next action taken by the agent, alpha is the learning rate, and gamma is the discount factor.

- In contrast, the Q-learning update rule is:
- Q(s, a) <- Q(s, a) + alpha * [r + gamma * max_a'(Q(s', a')) - Q(s, a)]
- where max_a'(Q(s', a')) is the maximum Q-value of the next state s' over all possible actions a', and all other terms have the same meaning as in the SARSA update rule.

- In the epsilon-greedy policy, where the agent selects the action with the highest Q-value with probability (1-ε), and selects a random action with probability ε. This allows the agent to balance exploration and exploitation, by occasionally selecting a random action to explore new parts of the environment, while still exploiting its current knowledge of the Q-values.

- The key difference between SARSA and Q-learning is that SARSA is an on-policy algorithm, meaning that it learns the Q-values for the policy that is currently being used to select actions. In contrast, Q-learning is an off-policy algorithm, meaning that it learns the Q-values for the optimal policy, even if a different policy is being used to select actions.

Types of Passive Reinforcement Learning

- Two types of passive reinforcement learning are: **direct utility estimation and adaptive dynamic programming.**

- In ***direct utility estimation***, the agent learns to estimate the expected cumulative reward (or utility) of taking a specific action in a given state, rather than estimating the optimal value function as in active reinforcement learning approaches like Q-learning and SARSA.
- The direct utility estimation method can be broken down into two steps:
- **Feature extraction:** In this step, the agent extracts relevant features from the input state to represent the environment's state. These features can be as simple as raw pixel values or more complex features like object positions or distances.
- **Utility function estimation:** In this step, the agent learns to estimate the expected cumulative reward (or utility) of taking a specific action in a given state using the extracted features. The agent can use various machine learning techniques like linear regression or neural networks to estimate the utility function.

- Once the utility function has been estimated, the agent can use it to select actions that maximize the expected cumulative reward. This approach is useful in situations where the optimal policy is difficult to learn directly, but a dataset of experiences is available.

- Passive learning with ***adaptive dynamic programming (ADP)*** is a type of passive reinforcement learning that involves learning an optimal policy from a fixed dataset of experiences by iteratively improving a value function estimate.

- The ADP algorithm can be broken down into three steps:
- **Policy evaluation**: In this step, the agent updates its value function estimate by iteratively improving its estimate of the expected cumulative reward for each state-action pair in the fixed dataset of experiences.
- **Policy improvement:** In this step, the agent improves its policy by selecting the action that maximizes the expected cumulative reward, based on its updated value function estimate.
- **Policy iteration:** In this step, the agent repeats steps 1 and 2 until the optimal policy is found.

- ADP is particularly useful in situations where the environment's dynamics are complex and unknown. By learning from a fixed dataset of experiences, the ADP algorithm can iteratively improve its value function estimate and policy without needing to interact with the environment.

MACHINE LEARNING	STATISTICAL LEARNING
Subfield of Artificial Intelligence	Subfield of mathematics
Uses algorithms	Uses equations
Requires minimum human effort; is automated	Requires a lot of human effort
Can learn from large data sets	Deals with smaller data sets
Has strong predictive abilities	Gives a best estimate: you gain some insights into one thing, but it is of little or no help with predictions
Makes predictions	Makes inferences
Learns from data and discovers patterns	Learns from samples, populations, and hypotheses