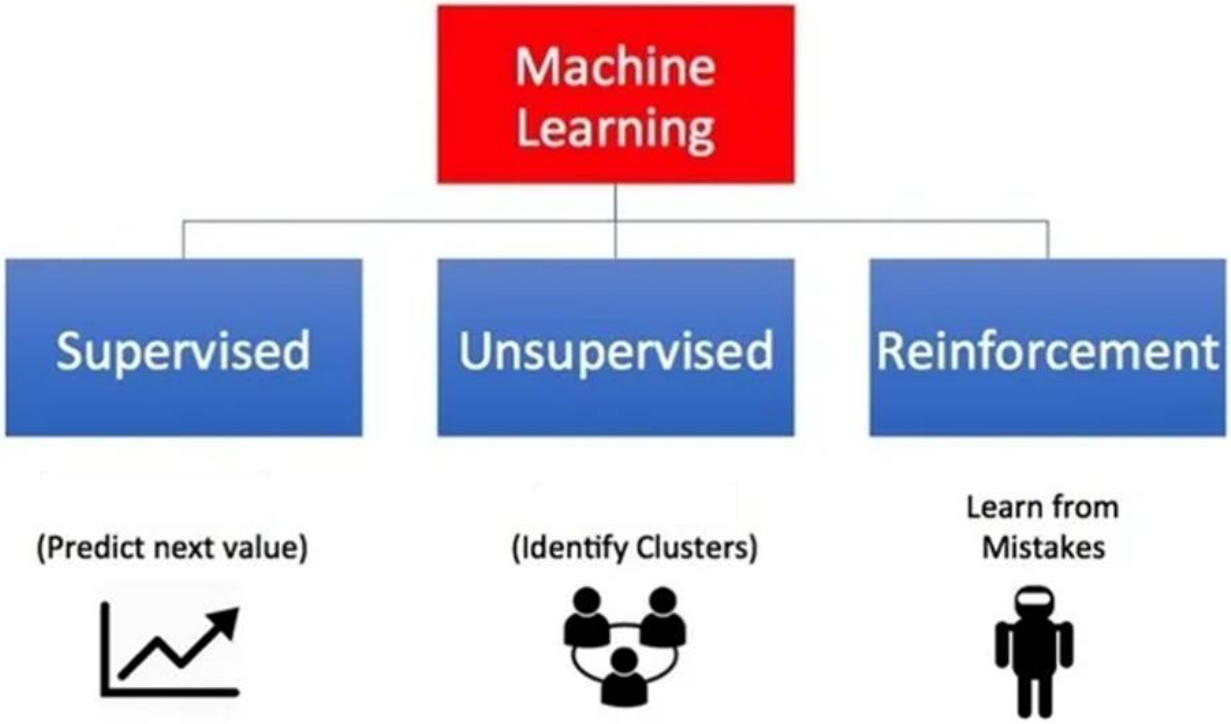
Reinforcement Learning

Types of Machine Learning

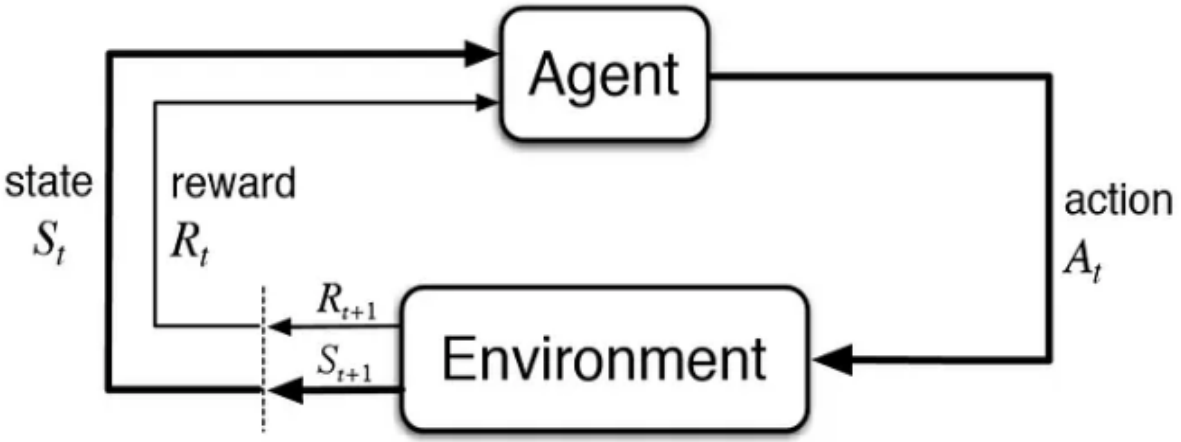


Introduction

- Reinforcement Learning (RL) overcomes the problem of data acquisition.
- In RL, the computer is simply given a goal to achieve.
- The computer then learns how to achieve that goal by trial-and-error by interacting with its environment.
- RL is a feedback-based Machine learning technique in which an agent learns based on this sensory input choosing an action to perform in the environment.
- There is no labeled data, so the agent is bound to learn by its experience only.
- By seeing the results of actions, the agent gets positive feedback for each good action, and for each bad action, the agent gets negative feedback or penalty.

Formulation of a Basic RL Problem

- The action changes the environment in some manner and this change is communicated to the agent through a scalar *reinforcement sig*nal(reward/penalty).



In reinforcement learning the goal is to find a suitable action model that would maximize the **total cumulative reward** of the agent.

Different Key Terms

- **Agent**: This is the algorithm/model that is going to perform the actions and learn over time.
- **Environment**: The surroundings that the agent interacts with.
- **Action**: This is what the agent performs. These are essentially the interactions of the agent in an environment.
- **Reward**: This is the outcome of an action and every action has a reward. A reward could be positive or negative (penalty).
- **State**: The current place of the agent in the environment. The actions that the agent performs can change its state.

The Environment

- Every RL system learns a mapping from situations to actions by trial-and-error interactions with a dynamic environment.
- This environment must at least be partially observable by the reinforcement learning system, and the observations may come in the form of sensor readings.
- If the RL system can observe perfectly all the information in the environment that might influence the choice of action to perform, then the RL system chooses actions based on true “states” of the environment.
- This ideal case is the best possible basis for reinforcement learning and, in fact, is a necessary condition for much of the associated theory.

- **Policy:** A plan that directs the agent's decision-making by mapping states to actions. Finding an ideal policy that maximises cumulative rewards is the objective.

- The policies could be deterministic (maps state to action) or non-deterministic (probability distribution of actions for a state).

- For deterministic policy: $a_t = \mu(s_t)$
  For stochastic policy: $a_t = \pi(. \mid s_t) = P[a_t = a \mid s_t = s]$

- **Value Function**: This function calculates the anticipated cumulative reward an agent can obtain from a specific state while adhering to a specific policy. It is beneficial in assessing and contrasting states and policies.

- **Model**: A depiction of the dynamics of the environment that enables the agent to simulate potential results of actions and states. Models are useful for planning and forecasting.

The Reinforcement Function

- The “goal” of the RL system is defined using the concept of a *reinforcement function.*
- There exists a mapping from state/action pairs to reinforcements; after performing an action in a given state the RL agent will receive some reinforcement (reward) in the form of a scalar value.
- The RL agent learns to perform actions that will maximize the sum of the reinforcements received when starting from some initial state and proceeding to a terminal state.
- It is the job of the RL system designer to define a reinforcement function that properly defines the goals of the RL agent.

Optimal Reinforcement Function

- Not always the learning agent attempts to maximize the reinforcement function.
- The learning agent could just as easily learn to minimize the reinforcement function due to limited resources.
- An alternative reinforcement function would be used in the context of a game environment, when there are two or more players with opposing goals.
- In a game scenario, the RL system can learn to generate optimal behavior for the players involved by finding the maximin, minimax, or saddlepoint of the reinforcement function.

Optimal Value Function

- Initially, the approximation of the optimal value function is poor.
- $V^*(s_t)$ *is the optimal value function where* s_t *is the state vector;* $V(s_t)$ *is the approximation of the value* function; γ is a discount factor in the range [0,1] that causes immediate reinforcement to have more importance.
- $V(s_t)$ *will be initialized to random values and will contain no information about the optimal* value function $V^*(s_t)$.
- *This means that the approximation of the optimal value function in a given state is* equal to the true value of that state $V^*(s_t)$ *plus some error in the approximation, as expressed*

$$V(s_t) = e(s_t) + V^*(s_t)$$

where $e(s_t)$ *is the error in the approximation of the value of the state occupied at time t. Likewise*

$$V(s_{t+1}) = e(s_{t+1}) + V^*(s_{t+1})$$

Bellman Equation

- The value of state s_t for the optimal policy is the sum of the reinforcement signal when starting from state s_t and performing optimal actions until a terminal state is reached.

- By this definition, a simple relationship exists between the values of successive states, s_t and s_{t+1} and defined by the Bellman equation as

$$V^*(s_t) = r(s_t) + \gamma V^*(s_{t+1})$$

where the discount factor γ is used to exponentially decrease the weight of reinforcements received in the future.

- The approximation $V(s_t)$ *also has the same relationship,*

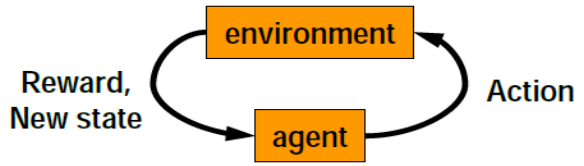
$$V(s_t) = r(s_t) + \gamma V(s_{t+1})$$
$$e(s_t) + V^*(s_t) = r(s_t) + \gamma(e(s_{t+1}) + V^*(s_{t+1}))$$
$$e(s_t) + V^*(s_t) = r(s_t) + \gamma e(s_{t+1}) + \gamma V^*(s_{t+1}))$$

- *$V^*(s_t)$ is subtracted from both sides to reveal the relationship in the* errors of successive states. This relationship is expressed

$$e(s_t) = \gamma e(s_{t+1})$$

Formulating Reinforcement Learning

- learn from interaction with environment to achieve a goal



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots$$

Your action influences the state of the world which determines its reward

- World described by a set of states and actions
- At every time step t, we are in a state s_t, and we:
  - Take an action a_t
  - Receive some reward r_{t+1}
  - Move into a new state s_{t+1}
- An RL agent may include one or more of these components:
  - Policy π: agent's behaviour function
  - Value function: how good is each state and/or action
  - Model: agent's representation of the environment

Value Function

- Value function is the expected future reward $V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\}$
- Used to evaluate the goodness/badness of states
- Our aim will be to maximize the value function (the total reward we receive over time): find the policy with the highest expected reward
- By following a policy π, the value function is defined as:

$$V^{\pi}(s_t) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots]$$

- γ is called a discount rate, and it is always $0 \leq \gamma \leq 1$
- If γ close to 1, rewards further in the future count more, and we say that the agent is "farsighted"
- γ is less than 1 because there is usually a time limit to the sequence of actions needed to solve a task (we prefer rewards sooner rather than later)

The Task

- To learn an optimal *policy* that maps states of the world to actions of the agent.
  - For example: *If this patch of room is dirty, I clean it. If my battery is empty, I recharge it.*

$$\pi : S \rightarrow A$$

- What is it that the agent tries to optimize?

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \ldots$$

Answer: The total future discounted reward:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$

$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i} \qquad 0 \leq \gamma < 1$$

Note that immediate reward is worth more than future reward.

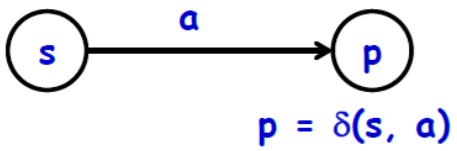
$V^{\pi}(s_t)$ is the optimal value function where s_t is the state vector; γ is a discount factor in the range [0,1] that causes immediate reinforcement to have more importance (weighted more heavily) than future reinforcement.

Optimal Policy

- Suppose we have access to the optimal value function that computes the total future discounted reward **V*(s)**
- What would be the optimal policy **π*(s)** ?

Answer: we choose the action that maximizes:

$$\pi^*(s) = \arg\max_a \left[r(s,a) + \gamma V^*(\delta(s,a)) \right]$$



- We assume that we know what the reward will be if we perform action "**a**" in state "**s**":
  r(s,a)
- We also assume we know what the next state of the world will be if we perform action "**a**" in state "**s**":
  $s_{t+1} = \delta(s_t, a)$

Learning

- Reinforcement learning is a difficult problem because the learning system may perform an action and not be told whether that action was good or bad.
- It will have to make many decisions and then acting on such decisions and the system learn from this experience.
- In the future, any time it chooses an action that leads to this particular situation, it will immediately learn that particular action is bad or good.
- The primary objective of learning is to find the correct mapping. Once this is completed, the optimal policy can easily be extracted.

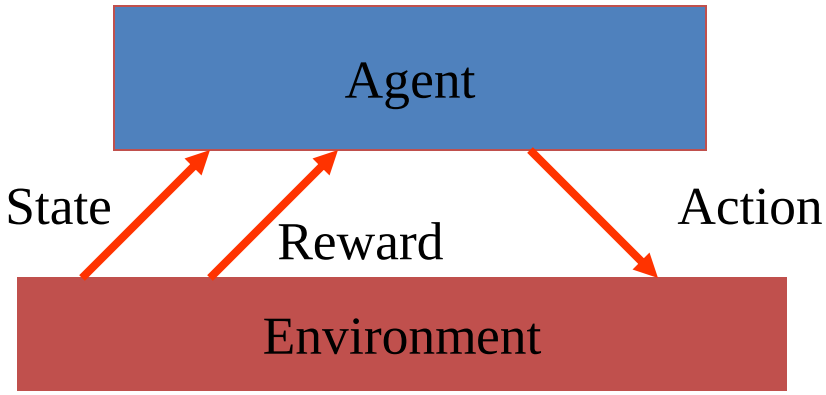
Markov Decision Process

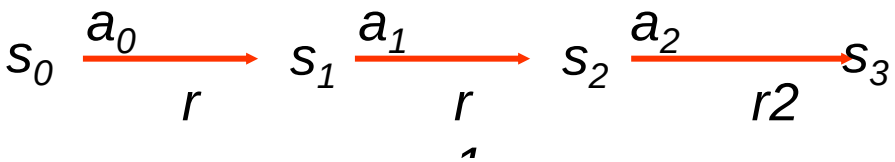
- The Markov decision process (MDP) is a mathematical framework used for modeling decision-making problems where the outcomes are partly random and partly controllable.
- It's a framework that can address most reinforcement learning (RL) problems.
- According to the Markov property, the current state of the robot depends only on its immediate previous state (or the previous time step).
- Formally, for a state S_t to be Markov, the probability of the next state $S_{(t+1)}$ being s' should only be dependent on the current state S_t

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

MDP Model

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





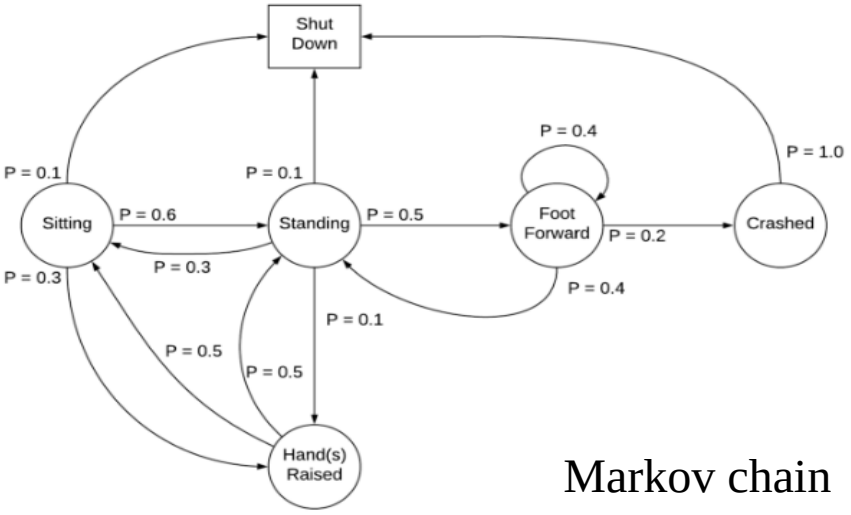
$$V(s) = \max [R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s')]$$

- 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*

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

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

- A Markov process is defined by (S, P) where S are the states, and *P* is the state-transition probability.
- It consists of a sequence of random states S_1, S_2, ... where all the states obey the Markov property.
- The state transition probability or $P_{ss'}$ is the probability of jumping to a state *s'* from the current state *s*.

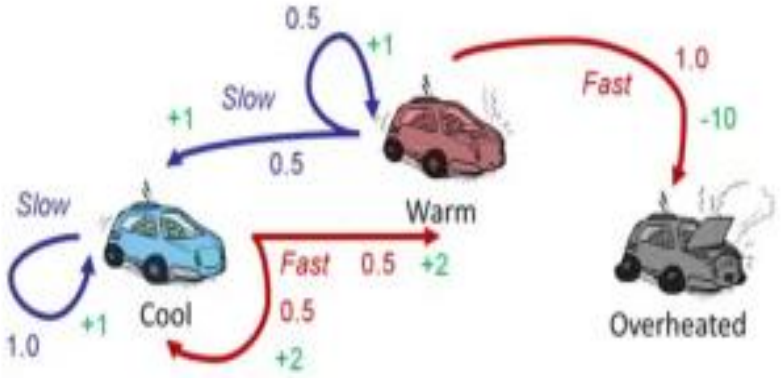


Markov chain

- The model describes the environment by a distribution over rewards and state transitions:

$$P(s_{t+1} = s', r_{t+1} = r' | s_t = s, a_t = a)$$

- We assume the Markov property: the future depends on the past only through the current state



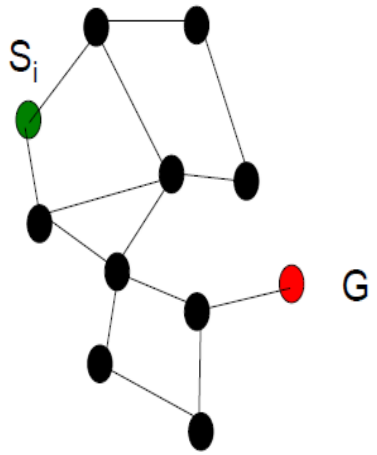
- Consider the game tic-tac-toe:
  - reward: win/lose/tie the game $(+1/-1/0)$ [only at final move in given game]
  - state: positions of X's and O's on the board
  - policy: mapping from states to actions
    - based on rules of game: choice of one open position
  - value function: prediction of reward in future, based on current state
- In tic-tac-toe, since state space is tractable, can use a table to represent value function
  - Each board position (taking into account symmetry) has some probability

State	Probability of a win (Computer plays "o")
	0.5
	0.5
	1.0
	0.0
	0.5
etc	

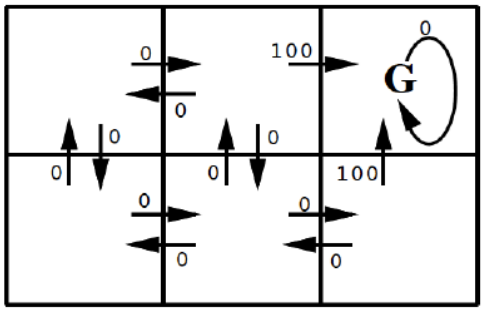
  - Simple learning process:
    - start with all values = 0.5
    - policy: choose move with highest probability of winning given current legal moves from current state
    - update entries in table based on outcome of each game
    - After many games value function will represent true probability of winning from each state
  - Can try alternative policy: sometimes select moves randomly (exploration)

Example 1

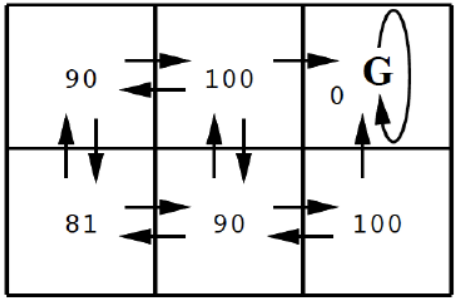
- Consider some complicated graph, and we would like to find the shortest path from a node S_i to a goal node G.
- Traversing an edge will incur costs – this is the edge weight.
- The value function encodes the total remaining distance to the goal node from any node s, that is: V(s) = "*1 / distance*" to goal from s.
- If you know V(s), the problem is trivial. You simply choose the node that has highest V(s).



Example 2



r(s,a) : immediate reward values

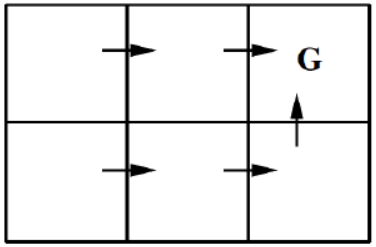


V*(s) : values

$\gamma = 0.9$

Therefore for the bottom left square:

$0 + (0.9 \times 0) + (0.9^2 \times 100) = 81$



One optimal policy

$$\pi^*(s) = \arg\max_a \left[r(s,a) + \gamma V^*(\delta(s,a)) \right]$$