02: Reinforcement Learning

Antorweep Chakravorty

Topics

- Dynamic Programming
- Monte Carlo Simulation
- Bandit Problems
    - One Arm
    - Multi Arm
    - Contextual
- Markov Property
- Overview
    - Value Functions
    - Policy Functions

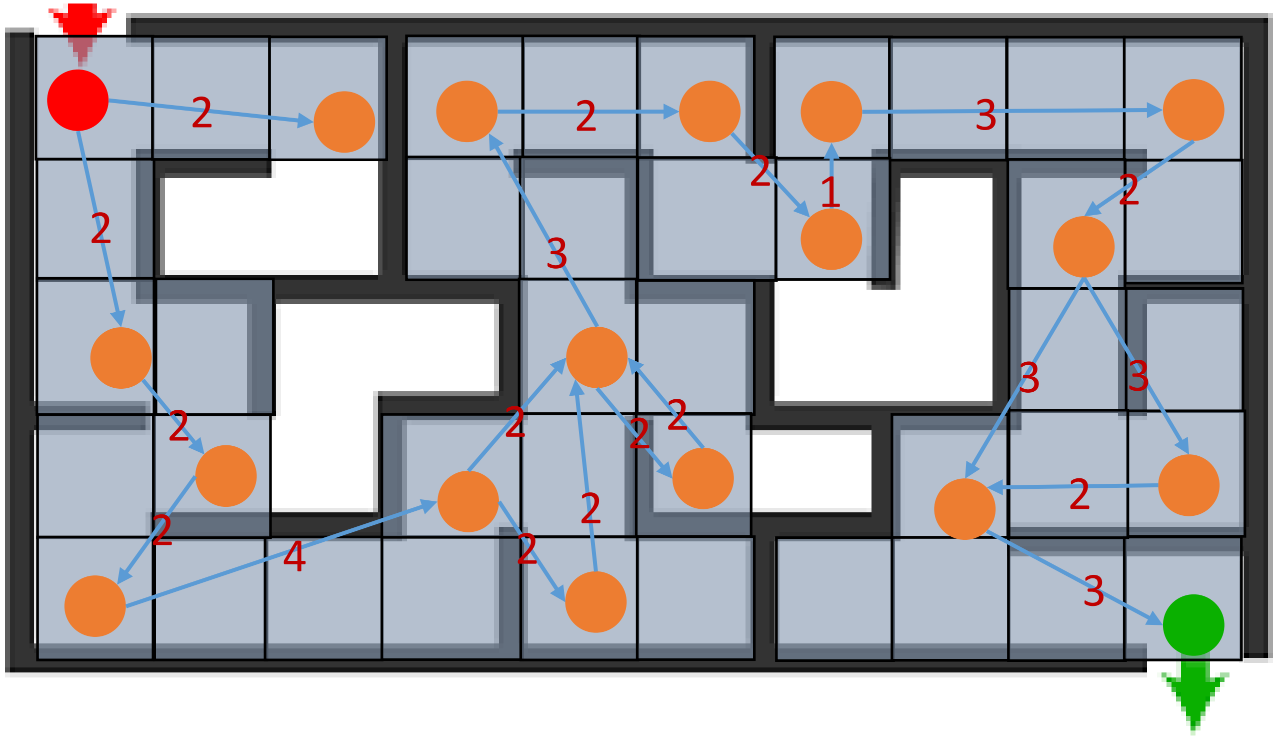
Dynamic Programming (DP)

- Suited in a situation where we have maximal knowledge of the environment
- A general method for solving certain kinds of control or decision problems
- A goal decomposition method
  - Solves complex high-level problems by decomposing them into smaller subproblems
  - Iteratively optimizes local subproblems and makes progress towards achieving the global objective
- Considered one extreme of a continuum of problem-solving techniques in Reinforcement Learning

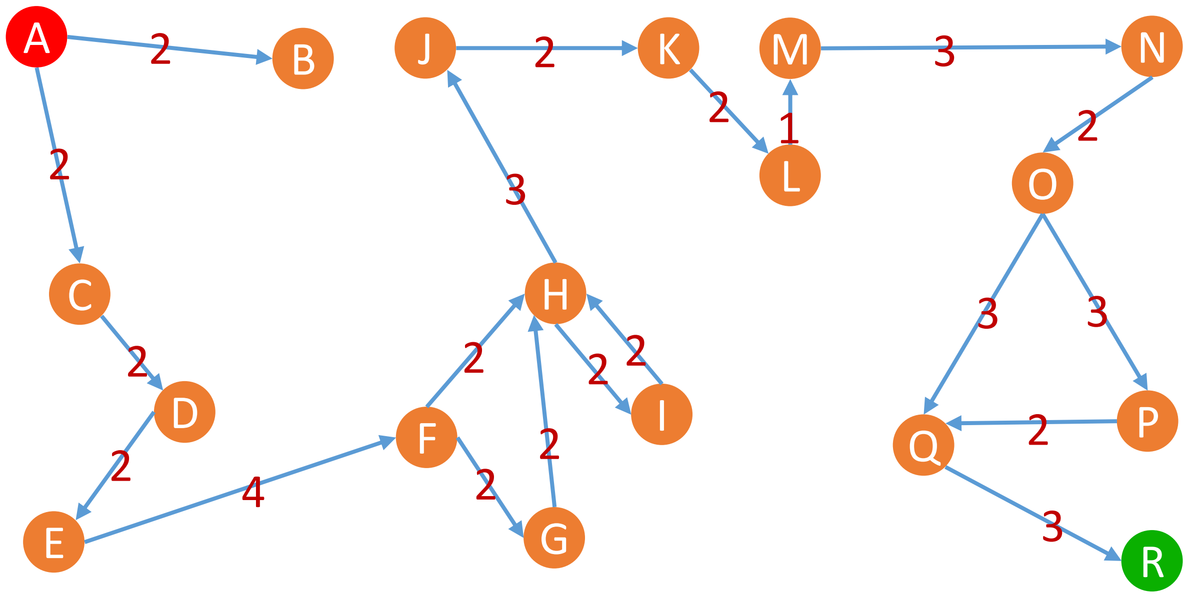
DP: Bellman Ford's Algorithm

- Start with a weighted graph
- Choose a starting vertex
  - Assign zero distance to this vertex
  - Assign infinite distance to all other vertices
- Visit each edge and relax the distances
  - If ***d[u] + c(u, v) < d[v] then d[v] = d[u] + c(u, v)***;
  - Here
    - ***u***: from vertex,
    - ***v***: to vertex;
    - ***d[*]***: distance from start to the specified vertex;
    - ***c***: a cost function that calculates the distance between two vertices
- Iterate '**V - 1**' times; where '***V***' is the total number of vertices

DP: Example



DP: Example



DP: Example

- CODE

Monte Carlo Simulation (MCS)

- Suited in a situation where we have minimum knowledge of the environment
- Employs a trial-and-error strategy Helps us deal with uncertainty in complex situations
- Involves random sampling from the environment
- The mean of the sampled distributions influences decisions
- Considered another extreme of a continuum of problem-solving techniques in Reinforcement Learning

MCS: Algorithm

- Update value table V for visited states
- We don’t need to know states ahead of time
- discover them as we sample
- Randomly sampled moves
- (OR) ideally intelligently sample what's most promising

MCS: Example

- WHITEBOARD

Exploitation VS Exploration

- <u>Exploitation</u>: Use current knowledge about the environment to choose the best course of action

- <u>Exploration</u>: The best course of action is decided based on randomly explored results of actions

➢ Proper balance of exploitation and exploration is essential for maximizing the rewards in RL

One-arm Bandit

Multi-arm Bandit

Multi-arm Bandit Problem

- Let us assume there are **ten** slot machines => *ten arm bandit*
- Each slot machine will give a reward between **0** and **10**
- Each machine has a different average payout
- Objective Function: Maximizing average rewards over multiple games

10-arm Bandit Problem

Let:

n possible actions: actions 0 – 9 correspond to pulling the respective lever of the slot machine

At each play ***k,*** we choose to pull one of the levers or perform an action ***a*** and receive a reward R_k.

Each lever has a unique probability distribution of payouts corresponding to their average payout.

Strategy:

- Play a few iterations by choosing random levers and observing the rewards
- In due course, we want only to choose those levers that produce the largest observed average reward
- The expected rewards at play ***k*** for action ***a*** can be calculated as $Q_k(a) = \frac{R_1 + R_2 + \ldots + R_k}{k_a}$; where k_a is the number of times action ***a*** was played.
- $Q_k(a)$ is also called as an ***action-value function*** as it provides the value of an action.
- At a current play ***k,*** $Q_k(a)$ is applied to all possible actions, and the action with the highest average expected reward is chosen for the next iteration ***k + 1***

Epsilon-Greedy Strategy

- Nudge the algorithm to discover the actual best action by introducing exploration
- We choose the best action based on an epsilon value, ***e***.
- We choose an action, ***a***, at random with a probability ***e***.
- The rest of the time, actions are chosen based on their expected value with probability ***1-e***.
- This strategy is moderately sensitive to the chosen **e** value. The value of **e** is always <u>*between 0 and 1*</u> and can be chosen intuitively.

SoftMax Selection Policy

- SoftMax chooses an action based on a probability distribution
- Action with the most significant probability will be equivalent to the best action
- However, the chosen best action may not be the one with the highest probability
- It allows the action to be chosen randomly while avoiding the worst action based on the given probability distributions of all actions $\mathbf{Pr}(\boldsymbol{A}) = \frac{e^{Q_k(A)/\tau}}{\sum_{i=1}^{n} e^{Q_k(i)/\tau}}$
- ***P(A)***, accepts an action-value vector and returns the probability distribution over the actions, such that the higher value action has higher probabilities.
- $\boldsymbol{\tau}$ (temperature) scales the probability distribution of action. A <u>*high τ value will cause the probabilities to be very similar*</u>, whereas a <u>*low τ value will exaggerate differences*</u> in probabilities between actions.
- The numerator of the fraction exponentiates the action value array divided by the parameter $\boldsymbol{\tau}$, yielding a vector of the same size as the input
- The denominator sums over the exponentiation of each action value divided by $\boldsymbol{\tau}$, yielding a scaler value
- Softmax is extremely sensitive to the $\boldsymbol{\tau}$ value. Choosing the right $\boldsymbol{\tau}$ is not intuitive and <u>*needs trial and error*</u>

10-arm Bandit Problem

- CODE

Contextual Bandit

- Adds a layer of complexity to the n-armed bandit problem by introducing *state spaces*
- The *traditional bandit problems* had *action spaces,* but *no concept of state* and actions were chosen based on action-value

A state is the set of information available in the environment that can be used to make decisions

- With the introduction of states, rewards are allocated based on chosen action in a specific state, often called *state-action-value*.

Contextual Bandit Problem

- Let us have ten e-commerce websites (10-arm bandit)
- Each website sells different categories of products: computers, shoes, jewelry, etc.
- Each website shows a recommendation for another website from its portfolio
- Objective Function: Increase traffic to sites by displaying the most relevant recommendation on each website

State, actions, rewards

- Action-value pairs ***(a, r)*** in the 10-armed bandit problems used a lookup table to store the average reward ***r*** for an action ***a*** over ***k*** plays.
- Lacking state information facilitates capturing of this information very efficiently in memory
- However, with the introduction of **states** with contextual bandits creates an explosion of possible *state-action-value* tuples
- Example: 10 actions and 100 states would require 1000 records
- Most real-world problems have much larger state spaces making use of a lookup table redundant

Deep Learning and Reinforcement Learning

- Deep Learning methods fit to the need to efficiently capturing complex relationship between state, action, and value pairs without having to maintain explicit lookup tables
- Deep Neural Networks can be trained to learn
  - Composite patterns and regularities in data
  - Compress a large amount of data while retaining its essential features

In Reinforcement Learning, we term this part of the RL algorithm that makes a decision based on state information as the **agent.**

Contextual Bandit Problem

- WHITEBOARD

Contextual Bandit Problem

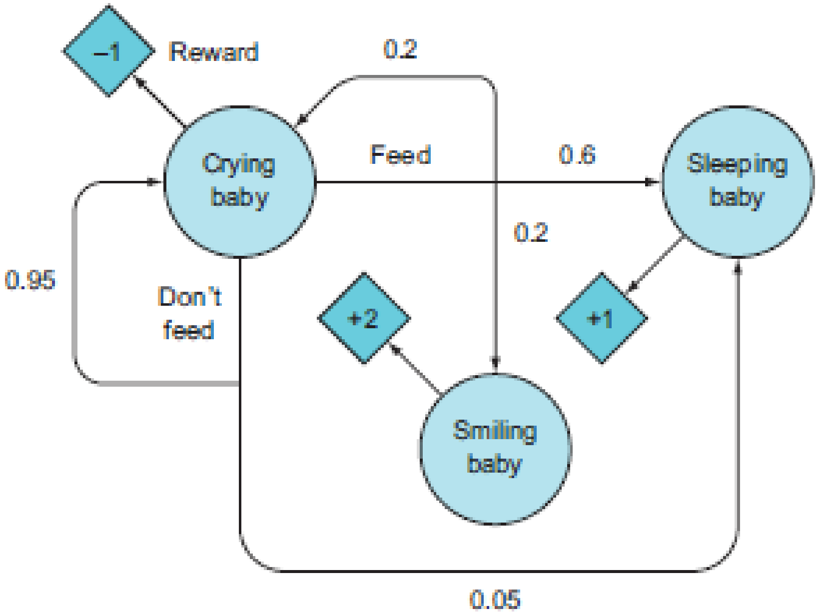
- CODE

The Markov Property

- The *Markov property* states that to execute decisions, all necessary information is captured in the current state without dependence on history
- A control task that exhibits the Markov property is said to be a *Markov Decision Process (MDP)*
- With an MDP, the current state alone contains enough information to choose optimal actions to maximize the future rewards
- Modelling a control task as an MDPs is a critical concept in Reinforcement Learning
- MDP simplifies an RL problem as only the current state is required for analysis as all previous states and actions are not required to be considered
- Hence in Reinforcement Learning problems are attempted (approximated) to be modeled as an MDP

Markov Property - Examples

- Which of the control tasks satisfy the Markov property
  - Driving a car
  - Deciding whether to invest in a stock or not
  - Choosing a medical treatment for a patient
  - Diagnosing a patient’s illness
  - Predicting which team will win a football game
  - Choosing the shortest route (by distance) to some destination
  - Aiming a gun to shoot a distant target



Transition Probabilities

- Maps an action to the probability of an outcome state
- The probability associated with mapping a state to a new state
- The agent receives a reward $\mathbf{r_t}$ for having taken action $\mathbf{a_t}$ in state $\mathbf{s_t}$ leading to a new state $\mathbf{s_{t+1}}$
- The generated reward depends on state transition $\mathbf{s_t}$ **->** $\mathbf{s_{t+1}}$ more so than the action taken since it may probabilistically lead to a bad state
- Therefore, the agent's goal could be to maximize the rewards by taking an action that leads to a better state with higher transition probabilities.
- This would require the agent to have a *model of the environment,* and this form of Reinforcement Learning is termed *model-based* learning

Value and Policy Functions

- Policy Functions (π):
  - A policy or **π** is the strategy of an agent in some environment
  - It maps a state to a probability distribution over the set of possible actions in that state
  - Example: Epsilon greedy strategy for n-arm bandit problems
  - $\boldsymbol{\pi}, \boldsymbol{s} \ \rightarrow \mathbf{Pr}(\boldsymbol{A} \mid \boldsymbol{s})$, where **s ∈ S**; **s** is a state, **P(A|a)** probability distribution over the set of actions **A** given state **s**
  - Optimal Policy:
    - Strategy that maximizes the rewards: $\boldsymbol{\pi}^* = \boldsymbol{argmax}\ \boldsymbol{E}(\boldsymbol{R} \mid \boldsymbol{\pi})$, where **E** is the expected rewards for any policy when followed produces the maximum rewards
- Value Functions
  - Value function maps a state or a state-action to the expected value of being in some state or taking some action in some state
  - Expected rewards are the long-term average of rewards received after being in some state or taking some action
  - $\boldsymbol{V_\pi}.\boldsymbol{s} \rightarrow \boldsymbol{E}(\boldsymbol{R}|\boldsymbol{s}, \boldsymbol{\pi})$, measures the expected reward for starting in **s** and taking actions according to policy $\boldsymbol{\pi}$.