AIFA: INTRODUCTION TO REINFORCEMENT LEARNING

04/04/2024

Koustav Rudra

Overview

- Supervised Learning:
  - Immediate feedback (labels provided for every input)

- Unsupervised Learning:
  - No feedback (no labels provided)

- Reinforcement Learning:
  - Delayed scalar feedback (a number called reward)

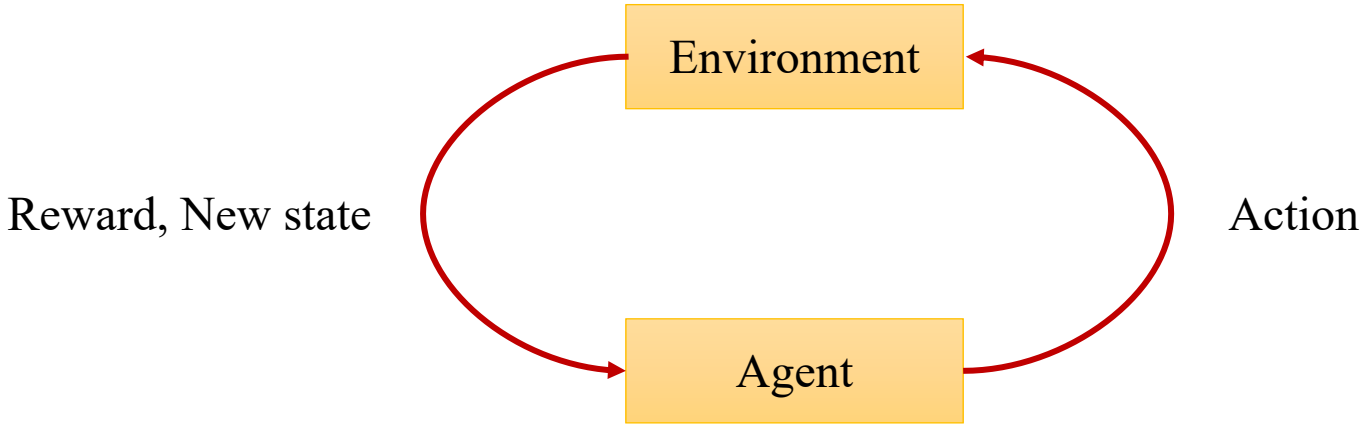
Reinforcement Learning

- RL deals with agents that must sense and act upon their environment
  - This combines classical AI and machine learning techniques
  - It the most comprehensive problem solving

- Example:
  - A robot cleaning the room and recharging its battery
  - How to invest in shares

The RL Framework

- Learn from interaction with environment to achieve a goal



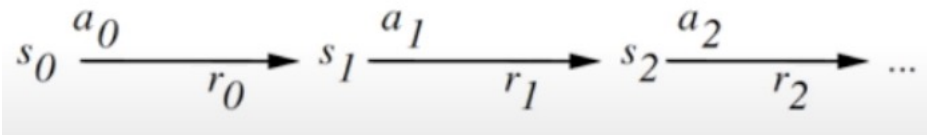
Your action influences the state of the world which determines its reward

Challenges

- The outcome of your actions may be uncertain
- You may not be able to perfectly sense the state of the world
- The reward may be stochastic
- The reward may be delayed (finding food in maze)
- You may have no clue (model) about how the world responds to your actions
- You may have no clue (model) of how rewards are being paid off
- The world may change while you try to learn it
- How much time do you need to explore uncharted territory before you exploit what you have learned?

The Task

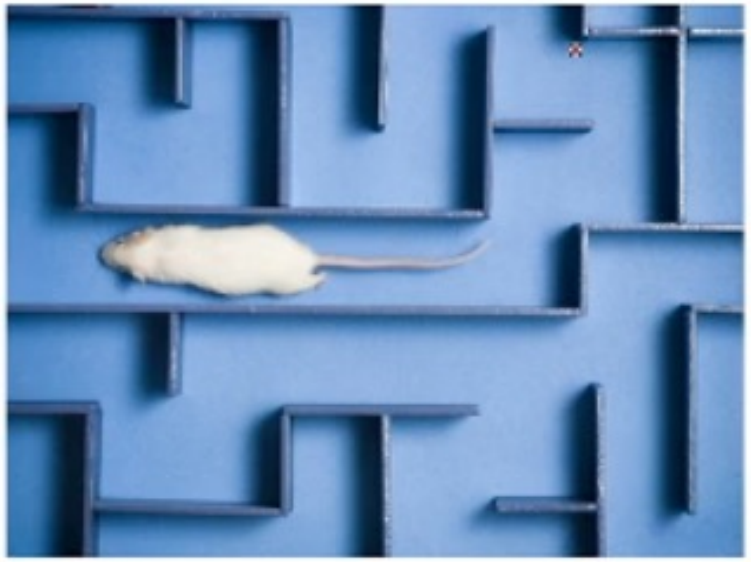
- To learn an optimal policy that maps states of the world to actions of the agent
  - For example: If this patch of room is dirty, I clean it. If my battery is empty, I recharge it.
  - $\pi: S \rightarrow A$



- What is it that the agent tries to optimize?
  - The total future discounted reward

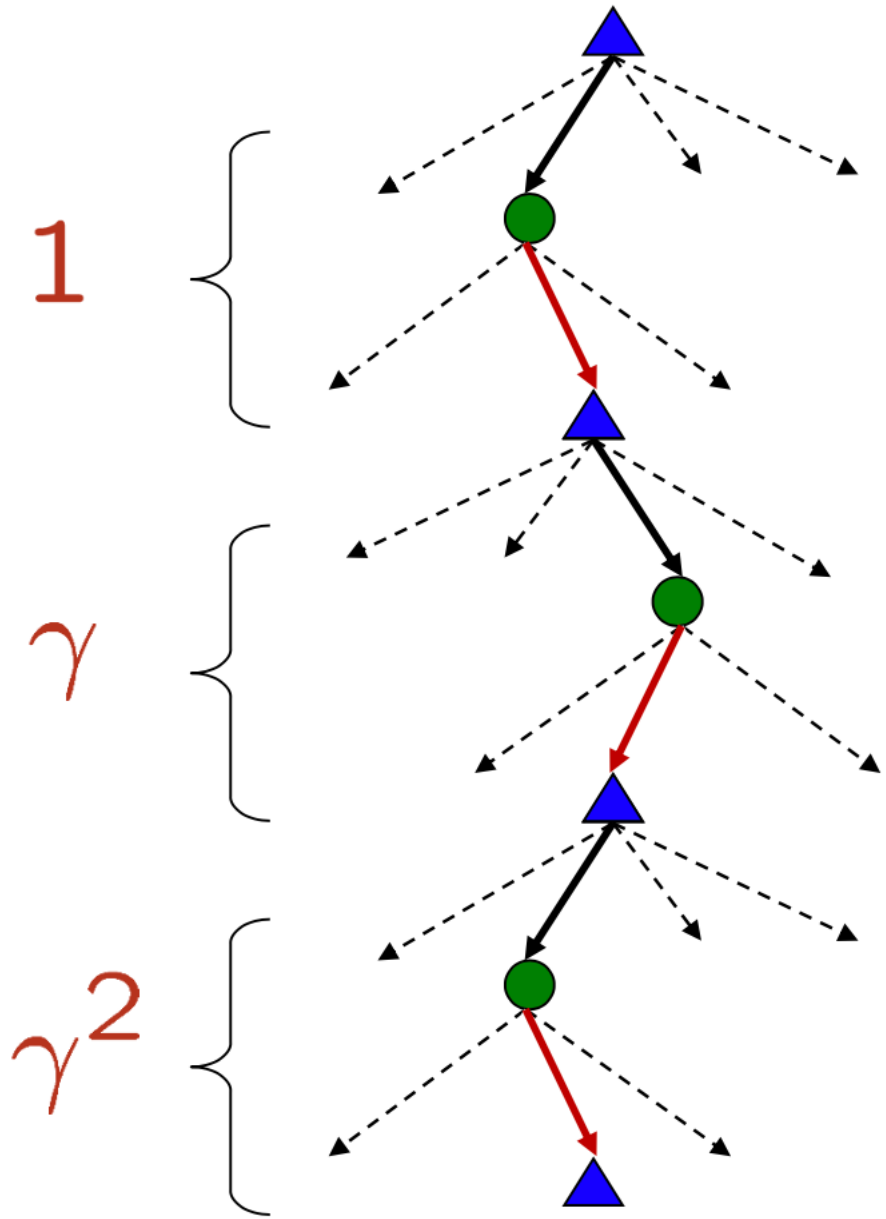
  - $V^{\pi}(S_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$
  - $V^{\pi}(S_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}, 0 \leq \gamma < 1$
  - Immediate reward is worth more than future reward

- What would happen to a mouse in a maze with γ=0



Discounting

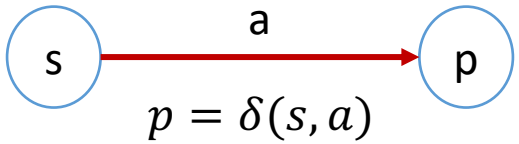
- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Reward now is better than later
  - Also helps our algorithms converge
- Example: discount of 0.5
  - U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - U([1,2,3]) < U([3,2,1])



Value Function

- Suppose we have access to the optimal value function that computes the total future discounted reward $V^*(s)$

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



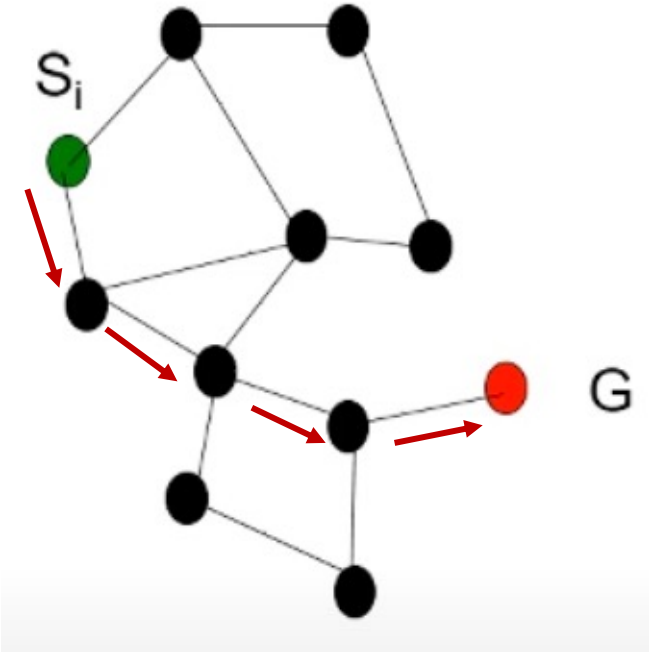
- We assume that we know what the reward will be if we perform action "a" in state "s": $r(s,a)$

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

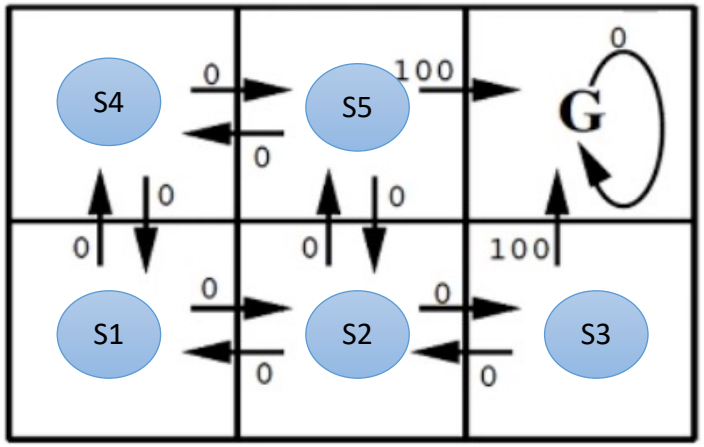
deterministic

Example 1

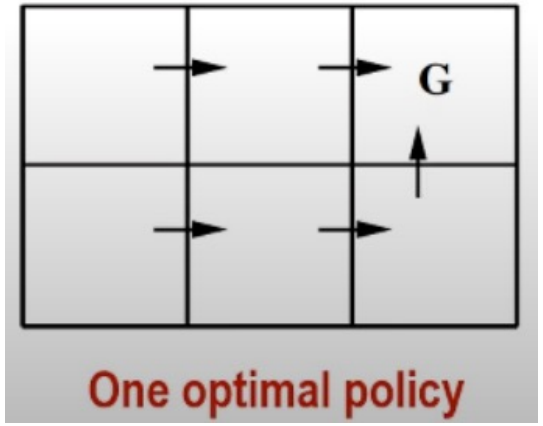
- Consider some complicated graph, and we would like to find the shortest path from a node S_i to a goal node G
- Traversing an edge will incur costs – this is the edge weight
- The value function encodes the total remaining distance to the goal node from any node s, that is:
  - $v(s) = {}^{1}/_{distance}$ to goal from s
- If we know v(s) the problem is trivial
  - Simply choose the node with highest $v(s)$

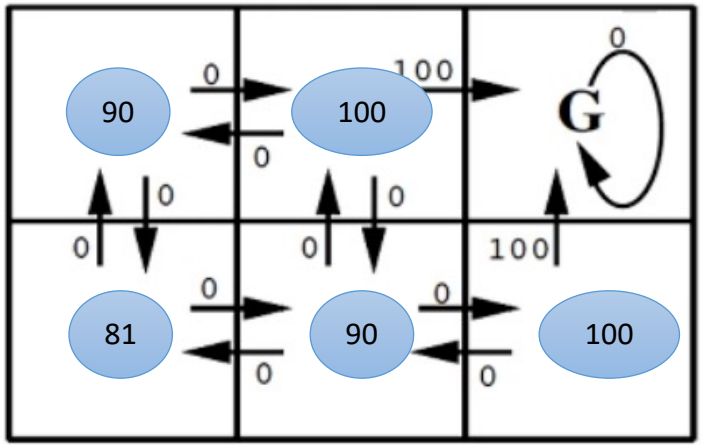


Example 2



r(s,a) : immediate reward values





$V^*(s)$

$V^*(s) = values$

$V^*(\mathrm{s1}) = 0 + 0.9{\times}0 + (0.9)^2 100 = 81$

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

The notion of Q-Function

- One approach to RL is then to try to estimate $V^*(\mathrm{s})$
  - We may use the Bellman Equation:
    - $V^*(s) = \max_a [r(s,a) + \gamma V^*(\delta(s,a))]$

- However, this approach requires us to know $r(s,a)$ and $\delta(s,a)$
  - This is unrealistic in many real problems because the state space may not be known

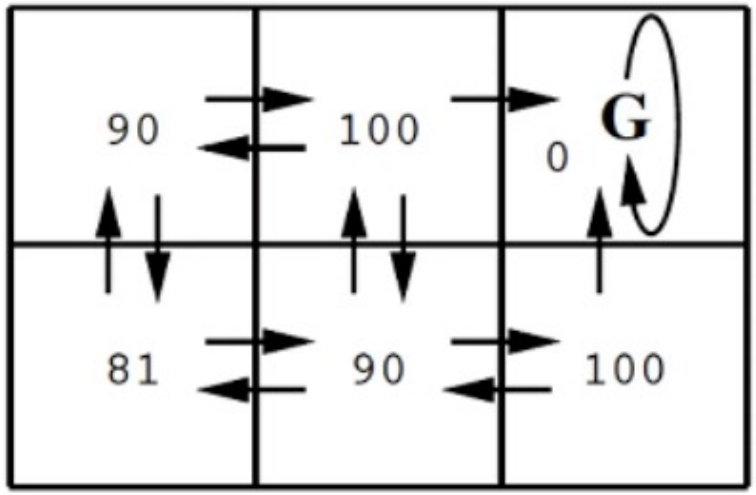
- This problem could be overcome by exploring and experiencing how the world reacts to our actions
  - We need to **learn values** of $r(s,a)$ and $\delta(s,a)$

The notion of Q-Function

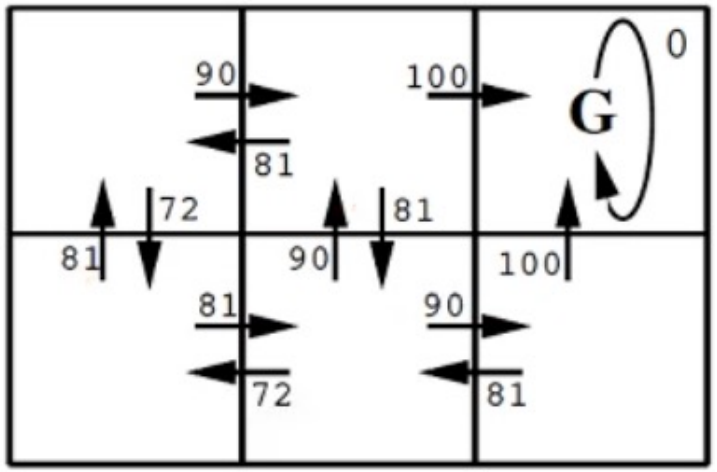
- We want a function that directly learns good state-action pairs, that is, what action should be taken in this state
  - Q(s,a)

- Given Q(s,a), it is trivial to evaluate to execute the optimal policy without knowing $r(s,a)$ and $\delta(s,a)$
  - $\pi^*(s) = argmax_a Q(s,a)$
  - $V^*(s) = max_a Q(s,a)$

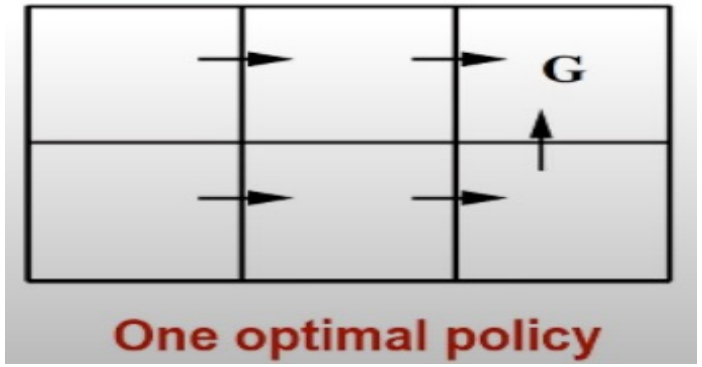
Example 2



V*(s) : values



Q(s,a) : values



- $\pi^*(s) = argmax_a Q(s,a)$
- $V^*(s) = max_a Q(s,a)$

Q-Learning

- $\pi^*(s) = argmax_a Q(s,a)$
- $V^*(s) = max_a Q(s,a)$

- $Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$
- $Q(s,a) = r(s,a) + \gamma \max_{a'} Q(\delta(s,a), a')$

- Still depends on $r(s,a)$ and $\delta(s,a)$

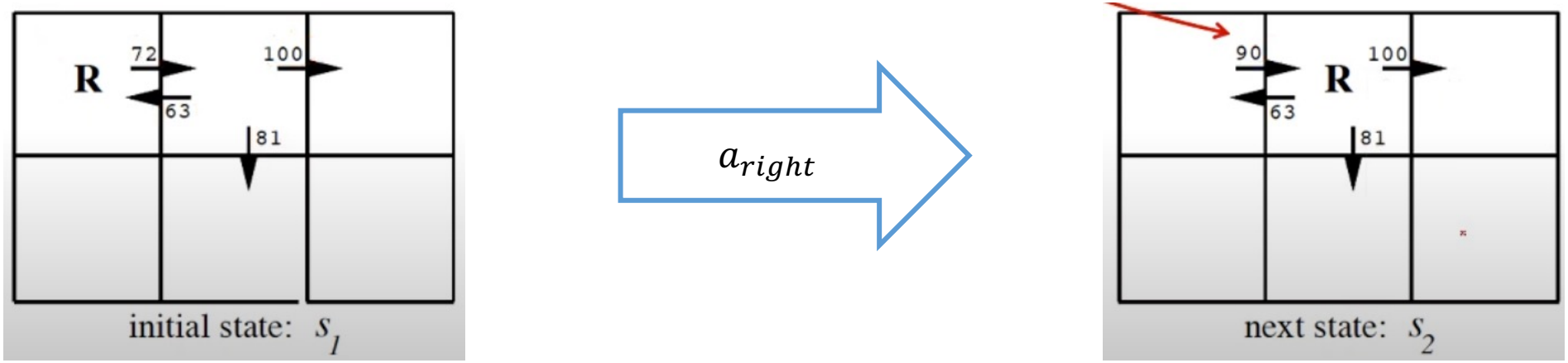
- $Imagine\ robot\ is\ exploring\ environment$
- At every step it receives reward "r" and it observes the environment changes to a new state s' for action a
- How can we use this observations (s, a, s', r) to learn a model?

- $\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$
- $s' = \delta(s,a)$

Q-Learning

- $\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$
- This equation continually estimates Q at state s consistent with an estimate of Q at state s', one step in the future
- Note that s' is closer to the goal, and hence more reliable, but still an estimate itself
- Updating estimate based on other heuristics is called bootstrapping
- We do an update after each state-action pair, i.e., we are learning online
- We are learning useful things about explored state-action pairs
  - Useful because they are likely to be encountered again
- Under suitable conditions, these updates will converge to real value

Example: One step Q Learning



$$\hat{Q}(S_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(S_2, a')$$

$$\hat{Q}(S_1, a_{right}) \leftarrow 0 + 0.9 \max_{a'}\{63,81,100\}$$

Limitations: Getting stuck in local optima

- It is very important that the agent does not simply follow the current policy when learning Q (off-policy learning)
  - The reason is that you may get stuck in a suboptimal solution
  - There may be other solutions out there that you have never seen

- Hence, it is good to try new things so now and then
  - For example, we could use something like:
  - $P(a|s) \propto e^{\frac{\widehat{Q}(s,a)}{T}}$
  - If T is large, lots of exploring
  - If T is small, follow current policy
  - We can decrease T over time

Improvements

- One can trade-off memory and computation by cashing (s, a, s', r) $for\ observed\ transitions$
  - After a while, as $Q(s', a') has\ changed$, you can "replay" the update
  - $\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$

Thank You