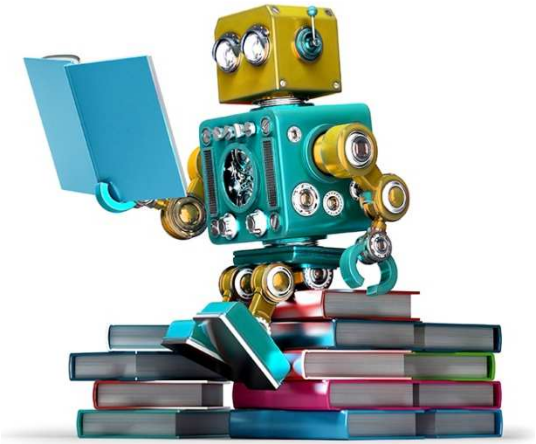
MACHINE LEARNING



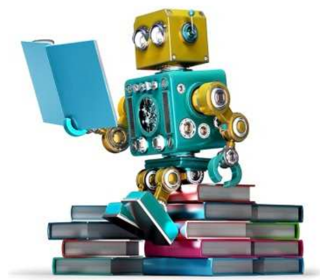
MODULE-V CHAPTER 8
REINFORCEMENT LEARNING

BY
HARIVINOD N
VIVEKANANDA COLLEGE OF ENGINEERING
TECHNOLOGY, PUTTUR

Module 5 - Outline

Chapter 13: Reinforcement Learning

1. **Introduction**
2. The Learning Task
3. Q Learning
4. Summary

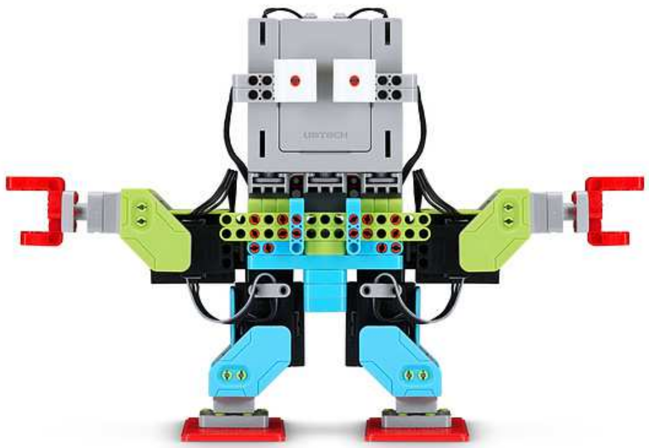


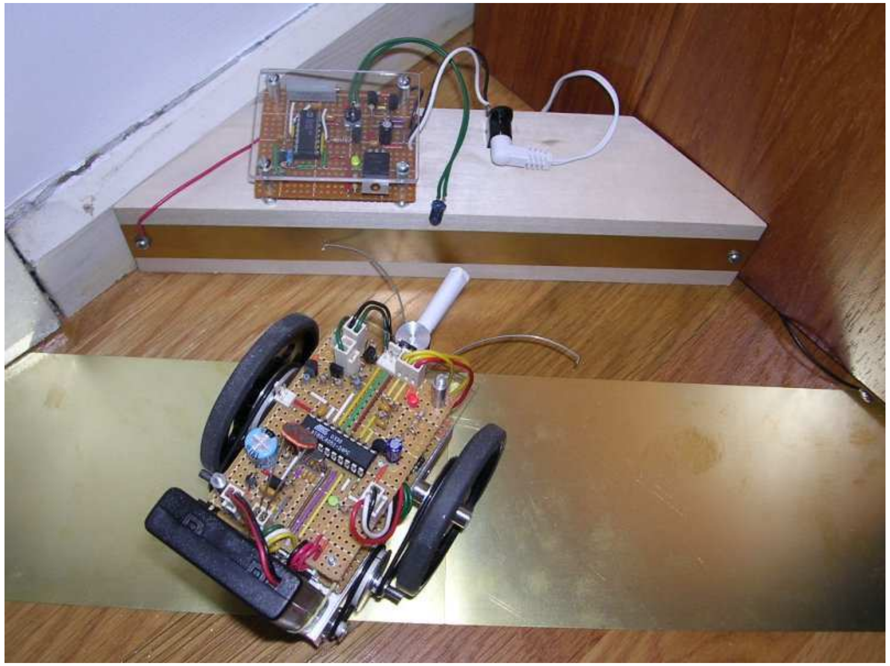
Introduction

- Reinforcement learning addresses the question of
  - how an autonomous agent that senses and
  - acts in its environment
  - can learn to choose optimal actions to achieve its goals.

- Applications
  - learning to control a mobile robot
  - learning to optimize operations in factories
  - learning to play board games.

Introduction





Introduction

- Consider building a learning robot called as agent.
- It has
  - a set of sensors to observe the state of its environment

    Ex: Camera, Sonar
  - a set of actions it can perform to alter this state

    Ex: “Move forward”, “Turn Right”
- Its task is to learn a control strategy, or policy, for choosing actions that achieve its goals.
- For example, the robot may have a goal of docking onto its battery charger whenever its battery level is low.

Introduction

- The goals of the agent can be defined by a reward function
- Reward function assigns a numerical value - an immediate payoff -to each distinct action the agent may take from each distinct state.
- For example, the goal of docking to the battery charger can be captured by
  - assigning a positive reward (e.g., +100) to state-action transitions that immediately result in a connection to the charger and
  - a reward of zero to every other state-action transition.

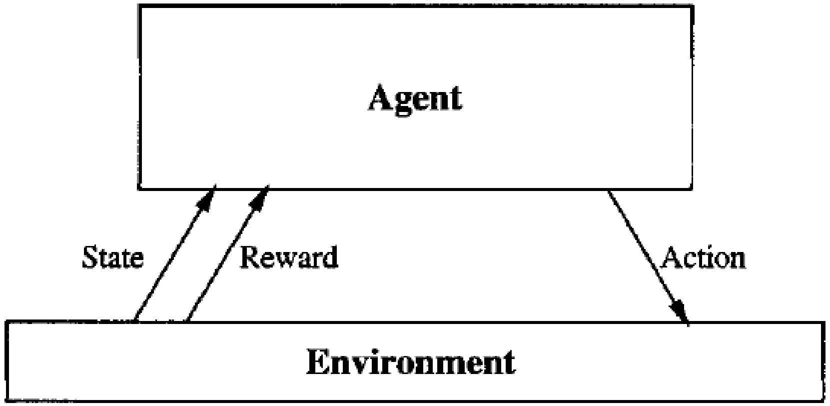
Introduction

- This reward function
  - may be built into the robot, or
  - known only to an external teacher who provides the reward value for each action performed by the robot.

- The task of the robot is to perform sequences of actions, observe their consequences, and learn a control policy.

- The control policy we desire is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agent.

Robot learning



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

Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \text{ , where } 0 \leq \gamma < 1$$

FIGURE 13.1
An agent interacting with its environment. The agent exists in an environment described by some set of possible states S. It can perform any of a set of possible actions A. Each time it performs an action a_t in some state s_t the agent receives a real-valued reward r_t that indicates the immediate value of this state-action transition. This produces a sequence of states s_i, actions a_i, and immediate rewards r_i as shown in the figure. The agent's task is to learn a control policy, $\pi : S \rightarrow A$, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.

Introduction

- **considered settings:**
  - deterministic or nondeterministic outcomes
  - prior backgound knowledge available or not
- **similarity to function approximation:**
  - approximating the function $\pi : S \rightarrow A$
    where S is the set of states and A the set of actions
- **differences to function approximation:**
  - Delayed reward: training information is not available in the form $< s, \pi(s) >$. Instead the trainer provides only a sequence of immediate reward values.
  - Temporal credit assignment: determining which actions in the sequence are to be credited with producing the eventual reward

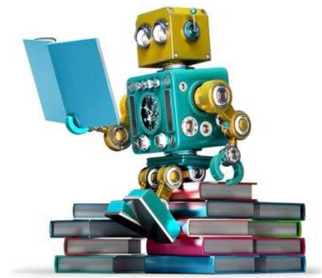
Introduction

- **differences to function approximation (cont.):**
  - exploration: distribution of training examples is influenced by the chosen action sequence
    - which is the most effective exploration strategy?
    - trade-off between exploration of unknown states and exploitation of already known states
  - partially observable states: sensors only provide partial information of the current state (e.g. forward-pointing camera, dirty lenses)
  - life-long learning: function approximation often is an isolated task, while robot learning requires to learn several related tasks within the same environment

Module 5 - Outline

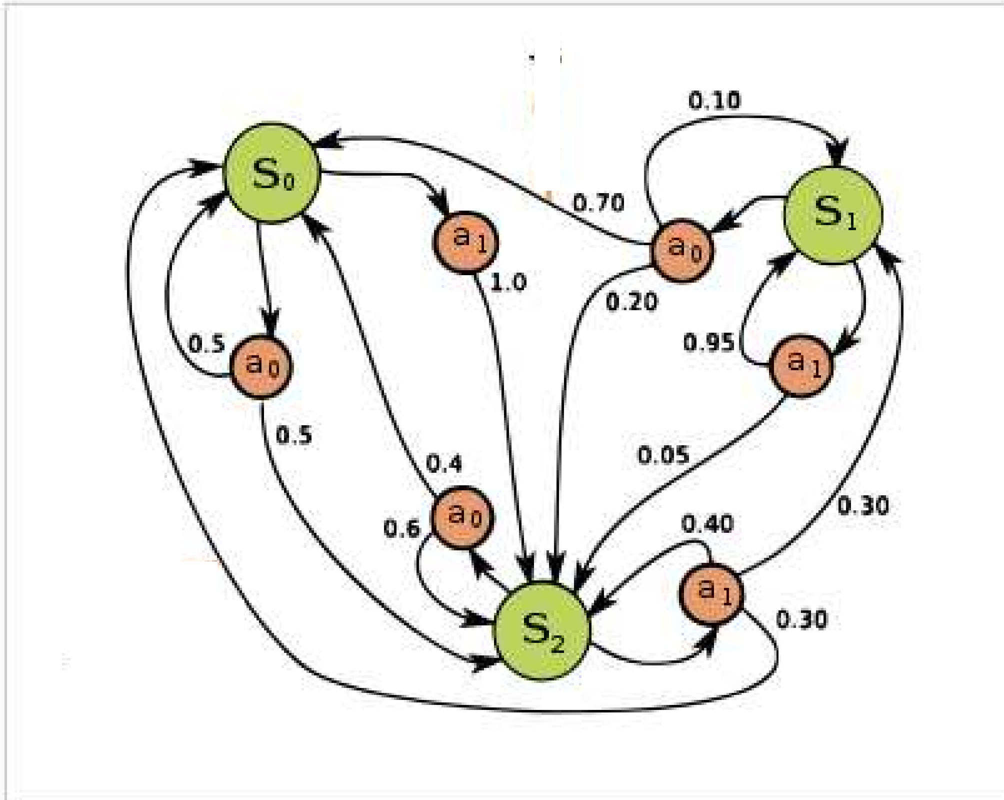
Chapter 13: Reinforcement Learning

1. Introduction
2. **The Learning Task**
3. Q Learning
4. Summary



The Learning Task

- based on Markov Decision Processes (MDP)
  - the agent can perceive a set S of distinct states of its environment and has a set A of actions that it can perform
  - at each discrete time step t, the agent senses the **current state** s_t, chooses a **current action** a_t and performs it
  - the environment responds by returning a **reward** $r_t = r(s_t, a_t)$ and by producing the **successor state** $s_{t+1} = \delta(s_t, a_t)$
  - the functions r and δ are part of the environment and not neccessarily known to the agent
  - in an MDP, the functions $r(s_t, a_t)$ and $\delta(s_t, a_t)$ depend only on the current state and action



Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows).

Source: Wikipedia

The Learning Task

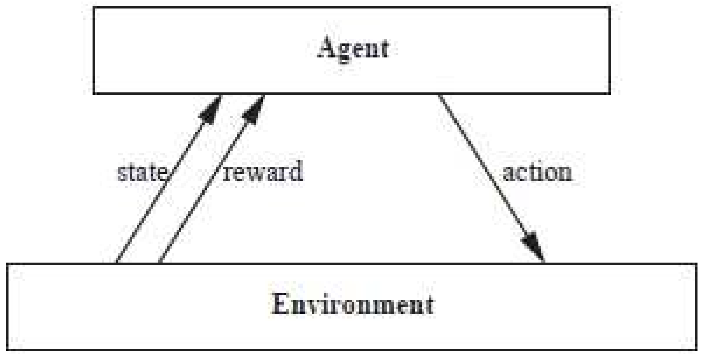
- the task is to learn a policy $\pi : S \rightarrow A$
- one approach to specify which policy π the agent should learn is to require the policy that produces the greatest possible cumulative reward over time (**discounted cumulative reward**)

$$V^{\pi}(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+1}$$
$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where $V^{\pi}(s_t)$ is the cumulative value achieved by following an arbitrary policy π from an arbitrary initial state s_t

r_{t+i} is generated by repeatedly using the policy π and γ $(0 \leq \gamma < 1)$ is a constant that determines the relative value of delayed versus immediate rewards

The Learning Task



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

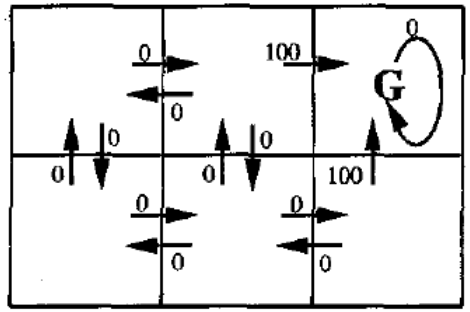
Goal: Learn to choose actions that maximize

$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$, where $0 < \gamma < 1$

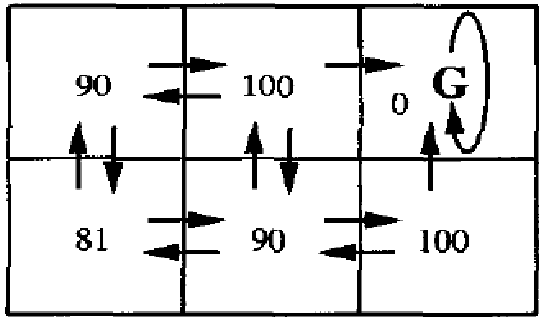
- hence, the agent's learning task can be formulated as

$$\pi^* \equiv \underset{\pi}{argmax}\, V^{\pi}(s), (\forall s)$$

Illustrative Example



$r(s, a)$ (immediate reward) values



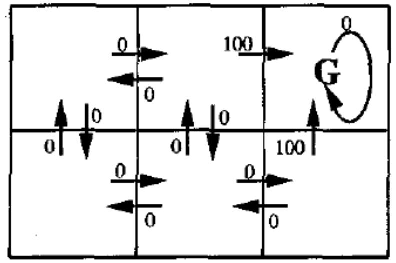
$V^*(s)$ values

- the left diagramm depicts a simple grid-world environment
  - squares $\approx$ states, locations
  - arrows $\approx$ possible transitions (with annotated $r(s, a)$)
  - $G \approx$ goal state (absorbing state)

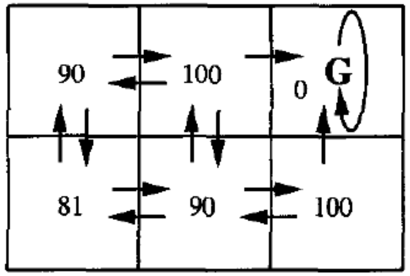
- $\gamma = 0.9$

- once states, actions and rewards are defined and γ is chosen, the optimal policy π^* with its value function $V^*(s)$ can be determined

Illustrative Example



$r(s, a)$ (immediate reward) values



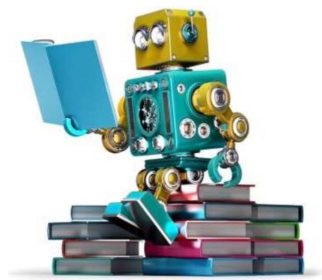
$V^*(s)$ values

- the right diagram shows the values of V^* for each state
- e.g. consider the bottom-right state
  - $V^* = 100$, because π^* selects the "move up" action that receives a reward of 100
  - thereafter, the agent will stay G and receive no further awards
  - $V^* = 100 + \gamma \cdot 0 + \gamma^2 \cdot 0 + ... = 100$
- e.g. consider the bottom-center state
  - $V^* = 90$, because π^* selects the "move right" and "move up" actions
  - $V^* = 0 + \gamma \cdot 100 + \gamma^2 \cdot 0 + ... = 90$
- recall that V^* is defined to be the sum of discounted future awards over **infinite** future

Module 5 - Outline

Chapter 13: Reinforcement Learning

1. Introduction
2. The Learning Task
3. **Q Learning**
4. Summary



Q Learning

- it is easier to learn a numerical evaluation function than implement the optimal policy in terms of the evaluation function
- **question:** What evaluation function should the agent attempt to learn?
- one obvious choice is V^*
- the agent should prefer s_1 to s_2 whenever $V^*(s_1) > V^*(s_2)$
- **problem:** the agent has to chose among actions, not among states

$$\pi^*(s) = \underset{a}{argmax}[r(s,a) + \gamma V^*(\delta(s,a))]$$

the optimal action in state s is the action a that maximizes the sum of the immediate reward $r(s,a)$ plus the value of V^* of the immediate successor, discounted by γ

Q Learning

- thus, the agent can acquire the optimal policy by learning V^*, *provided it has perfect knowledge of the immediate reward function* r and the state transition function δ
- in many problems, it is impossible to predict in advance the exact outcome of applying an arbitrary action to an arbitrary state
- the Q function provides a solution to this problem
  - $Q(s, a)$ indicates the maximum discounted reward that can be achieved starting from s and applying action a first

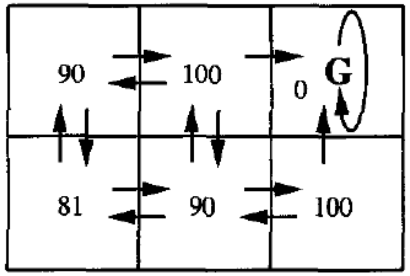
$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$

$$\Rightarrow \pi^*(s) = \underset{a}{argmax} Q(s, a)$$

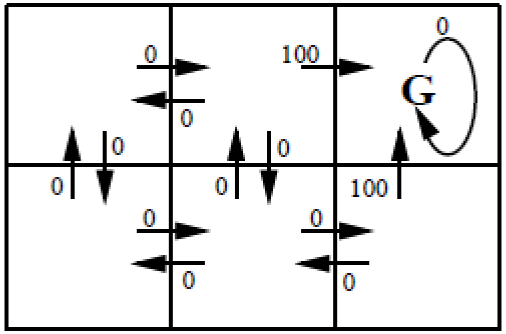
Q Learning

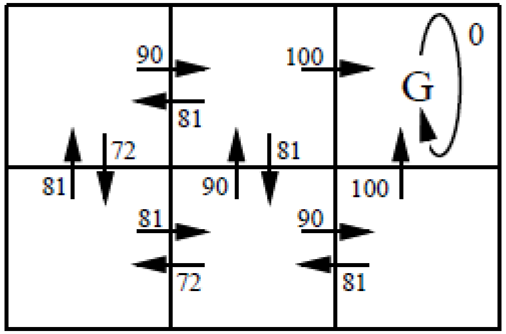
- hence, learning the Q function corresponds to learning the optimal policy π^*
- if the agent learns Q instead of V^*, it will be able to select optimal actions even when it has *no knowledge of* r and δ
- it only needs to consider each available action a in its current state s and chose the action that maximizes $Q(s, a)$
- the value of $Q(s, a)$ for the current state and action summarizes in one value all information needed to determine the discounted cumulative reward that will be gained in the future if a is selected in s

Q learning



$V^*(s)$ values





- the right diagramm shows the corresponding Q values
- the Q value for each state-action transition equals the r value for this transition plus the V^* value discounted by γ

Q Learning Algorithm

- **key idea:** iterative approximation
- relationship between Q and V^*

$$V^*(s) = \max_{a'} Q(s, a')$$

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

- this recursive definition is the basis for algorithms that use iterative approximation
- the learner's estimate $\hat{Q}(s, a)$ is represented by a large table with a separate entry for each state-action pair

Q Learning Algorithm

For each s, a initialize the table entry $\hat{Q}(s,a)$ to zero

Oberserve the current state s

Do forever:

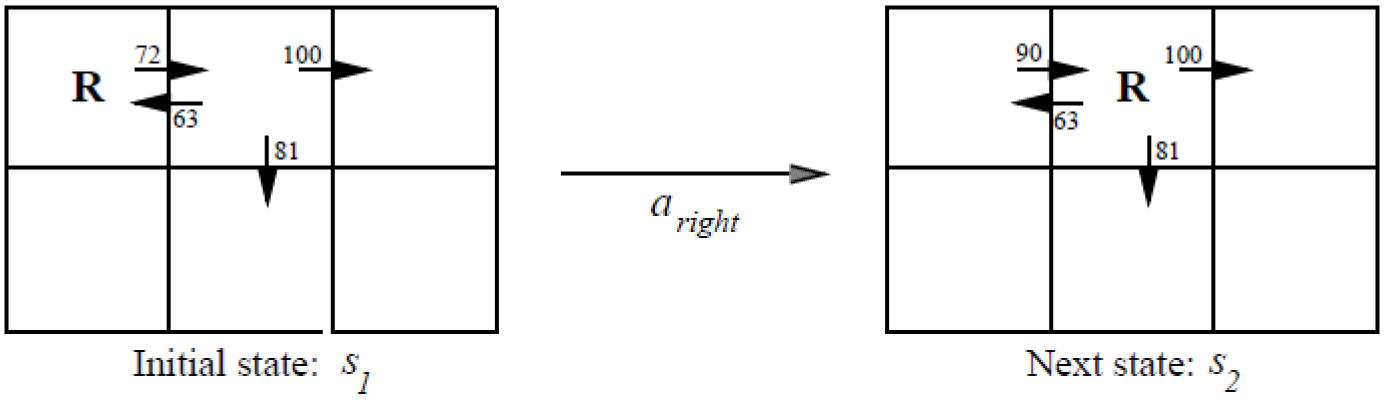
- Select an action a and execute it
- Receive immediate reward r
- Observe new state s'
- Update each table entry for $\hat{Q}(s,a)$ as follows

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

- $s \leftarrow s'$

$\Rightarrow$ using this algorithm the agent's estimate $\hat{Q}$ converges to the actual Q, provided the system can be modeled as a deterministic Markov decision process, r is bounded, and actions are chosen so that every state-action pair is visited infinitely often

Illustrative Example



$$
\begin{aligned}
\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \cdot \max_{a'} \hat{Q}(s_2, a') \\
&\leftarrow 0 + 0.9 \cdot \max\{66, 81, 100\} \\
&\leftarrow 90
\end{aligned}
$$

- each time the agent moves, Q Learning propagates $\hat{Q}$ estimates *backwards* from the new state to the old

Experiemntation Stages

- algorithm does not specify how actions are chosen by the agent
- **obvious strategy:** select action a that maximizes $\hat{Q}(s,a)$
  - risk of overcommiting to actions with high $\hat{Q}$ values during earlier trainings
  - exploration of yet unknown actions is neglected

- **alternative:** probabilistic selection

$$P(a_i|s) = \frac{k^{\hat{S}(s,a_i)}}{\sum_j k^{\hat{Q}(s,a_j)}}$$

k indicates how strongly the selection favors actions with high $\hat{Q}$ values

k large $\Rightarrow$ exploitation strategy

k small $\Rightarrow$ exploration strategy

Generalizing from Examples

- so far, the target function is represented as an explicit lookup table
- the algorithm performs a kind of rote learning and makes no attempt to estimate the Q value for yet unseen state-action pairs

$\Rightarrow$ unrealistic assumption in large or infinite spaces or when execution costs are very high

- incorporation of function approximation algorithms such as BACKPROPAGATION
  - table is replaced by a neural network using each $\hat{Q}(s, a)$ update as training example (s and a are inputs, $\hat{Q}$ the output)
  - a neural network for each action a

Relationship to Dynamic Programming

- Q Learning is closely related to dynamic programming approaches that solve Markov Decision Processes
- **dynamic programming**
  - assumption that $\delta(s, a)$ and $r(s, a)$ are known
  - focus on how to compute the optimal policy
  - mental model can be explored (no direct interaction with environment)

  $\Rightarrow$ *offline system*
- Q **Learning**
  - assumption that $\delta(s, a)$ and $r(s, a)$ are not known
  - direct interaction inevitable

  $\Rightarrow$ *online system*

Relationship to Dynamic Programming

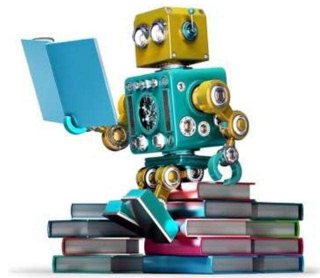
- relationship is appent by considering the Bellman's equation, which forms the foundation for many dynamic programming approaches solving Markov Decision Processes

$$(\forall s \in S) V^*(s) = E[r(s, \pi(s)) + \gamma V^*(\delta(s, \pi(s)))]$$

Module 5 - Outline

Chapter 13: Reinforcement Learning

1. Introduction
2. The Learning Task
3. Q Learning
4. **Summary**



Summary

- Reinforcement learning
    - Learning control strategies for autonomous agents.
    - It assumes that training information is available in the form of a real-valued reward signal given for each state-action transition.
    - The goal of the agent is to learn an action policy that maximizes the total reward it will receive from any starting state.

Summary

- The reinforcement learning algorithms addressed in this chapter fit a problem setting known as a Markov decision process.
- In Markov decision processes, the outcome of applying any action to any state depends only on this action and state (and not on preceding actions or states).
- Markov decision processes cover a wide range of problems including many robot control, factory automation, and scheduling problems.

Summary

- Q learning is one form of reinforcement learning in which the agent learns an evaluation function over states and actions.

- Evaluation function Q(s, a) is defined as the
  - maximum expected, discounted, cumulative reward
  - the agent can achieve by applying action a to state s.

- Advantage - it can-be employed even when the learner has no prior knowledge of how its actions affect its environment.

Thank You