

Reinforcement Learning

Why Reinforcement Learning?

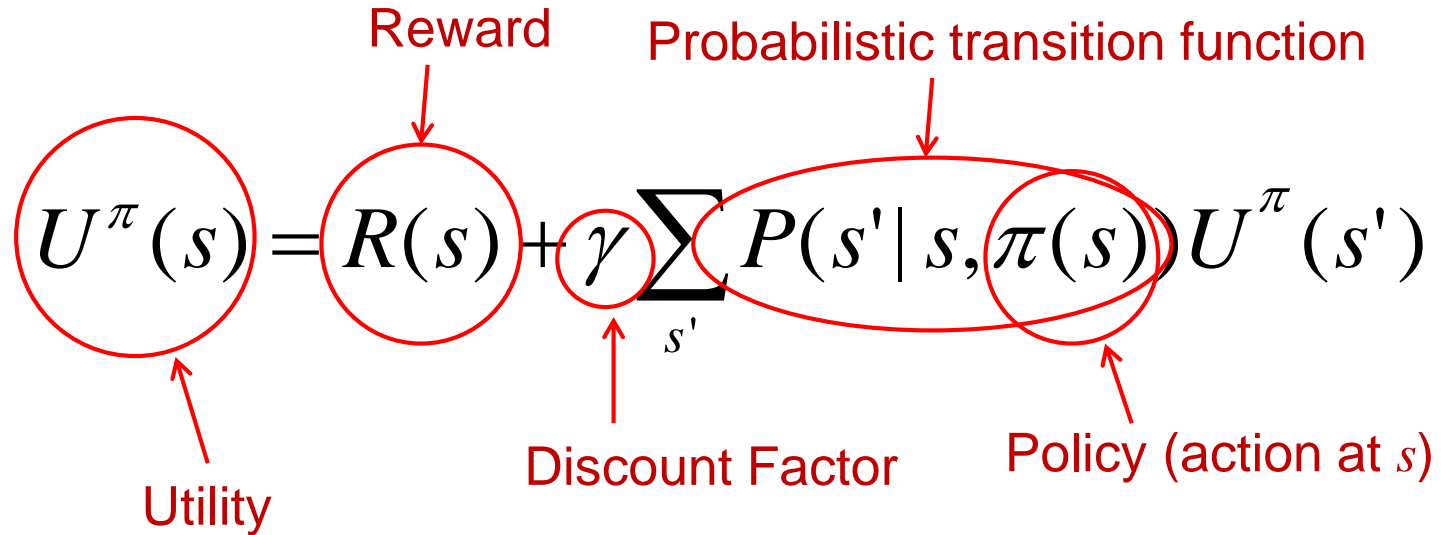
- The task involves a series of actions.
- For a given state, there is no teacher to tell the agent what the "correct" or "optimal" action is from that state.
- Feedbacks (reward/penalty) are available, but not after every action.
- Example scenario: Playing a game without a given evaluation function. Can the agent learn an evaluation function from its experience?

Policy, Reward and Utility

- **Policy**: A policy (chapter 17) determines the action to be taken at each state, that is, it is a function from states to actions.
- **Reward**: This can be represented as a reward function of a state. Penalties are just negative rewards.
- **Utility**: The "expected total reward" from a given state to terminal states.
 - This is like the averaged reward of all the possible paths from the given state to the terminal states, weighted by probabilities of the paths.
 - The utility of a state depends on the policy.

Policy, Reward and Utility

Bellman's equation (fixed policy):



The diagram shows the Bellman equation for a fixed policy, $U^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^\pi(s')$. Red annotations identify the components: 'Utility' points to $U^\pi(s)$; 'Reward' points to $R(s)$; 'Discount Factor' points to γ ; 'Probabilistic transition function' points to the summation term $\sum_{s'} P(s'|s, \pi(s))$; and 'Policy (action at s)' points to $\pi(s)$. Red circles and an oval highlight these specific parts of the equation.

$$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^\pi(s')$$

Utility

Reward

Discount Factor

Probabilistic transition function

Policy (action at s)

For deterministic environments (where the next state is determined by the current state and the policy), a simpler version:

$$U^\pi(s) = R(s) + \gamma U^\pi(\pi(s))$$

Passive Learning

- The policy is fixed during the learning.
- What can be learned?
 - Utilities of states
 - Transition functions of the states (i.e., the "model" of the environment); this is to account for any stochastic aspect of the environment.
- Multiple trials are necessary. (With enough trials, estimated utilities and/or transition functions will approach the true values.)

Active Learning

- The goal is to learn a "good" (hopefully "optimal") policy.
- Bellman's equation of an optimal policy:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s' | s, a) U(s')$$

- The agent needs to find a policy by experimentation.
- The role of the **discount factor**: instant or delayed rewards?
 - $0 \leq \gamma \leq 1$
 - Larger γ : Can learn to maximize rewards that are delayed (many steps later).
 - Smaller γ : Only learn to focus on immediate rewards.

Q-Learning

- A really popular approach of reinforcement learning.
- The goal: To learn the Q-function (action-utilities). Each entry, $Q(s,a)$, represents the expected total reward of taking action a at state s .
 - When the learning converges, the best policy is to follow the action with the best Q value.
 - The "expected total reward" assumes that the best-Q-value action is taken for all subsequent steps until a terminal state is reached.
- Q-functions can be learned without learning or knowing the model (transition functions) of the environment.
- Stochastic environment: The "next state" s' is not determined by the current s and the action a .

The Q-Learning Algorithm

- Initialize all the Q values (for example, to zero).
- A typical Q-learning procedure is to repeat the following many times (learning episodes):
 - Start at any valid initial state.
 - Repeat until a terminal state is reached:
 - ◆ Choose a valid action a from the current state s . Let s' be the resulting new state, and let r be the reward incurred for this action.
 - ◆ Update $Q(s,a)$: (This occurs only for non-terminal s .)

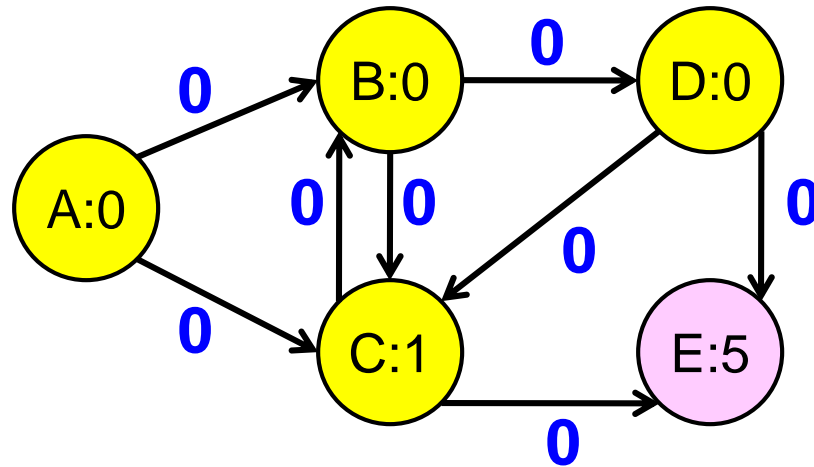
$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

Learning Rate

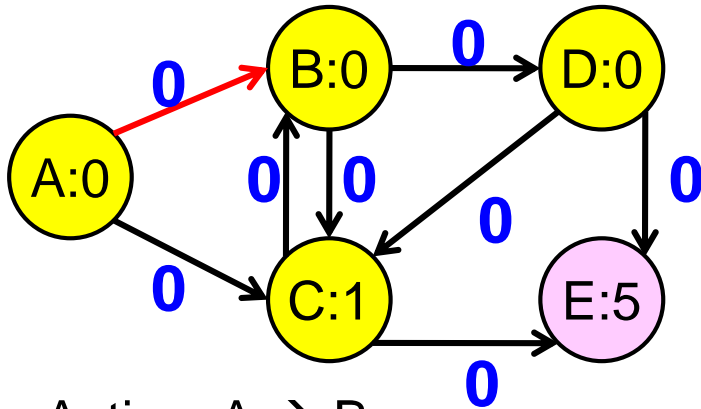
Discount Factor

Q-Learning Example

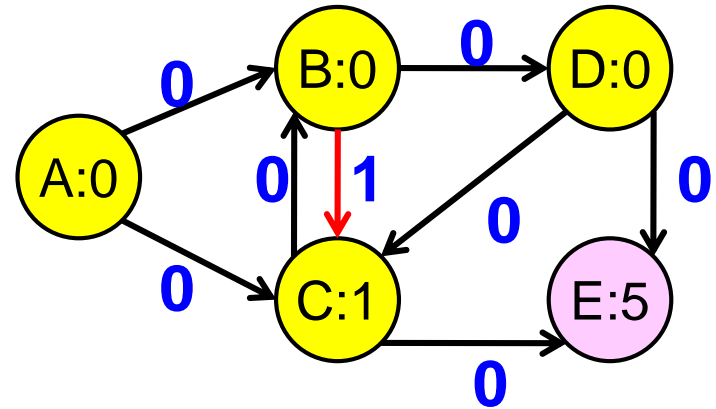
- For simplicity, we will consider only a deterministic environment here, and all the Q values are initialized to zero.
- Settings (state space given below): Start states = {A}.
Learning rate = 1. Discount factor = 1. Terminal states = {E}.



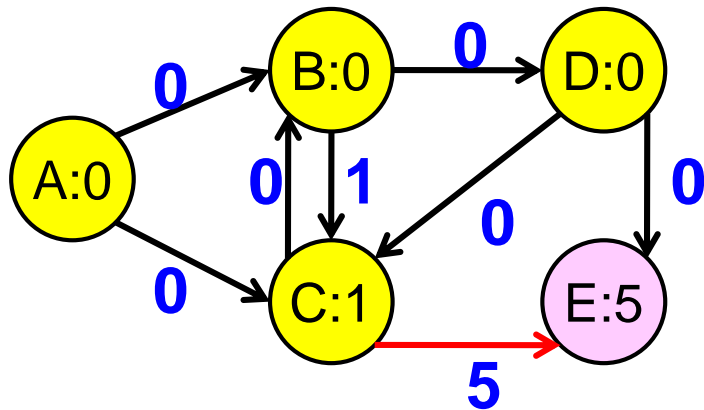
Q-Learning Example



Action: $A \rightarrow B$
 $Q(A, A \rightarrow B) \leftarrow 0$

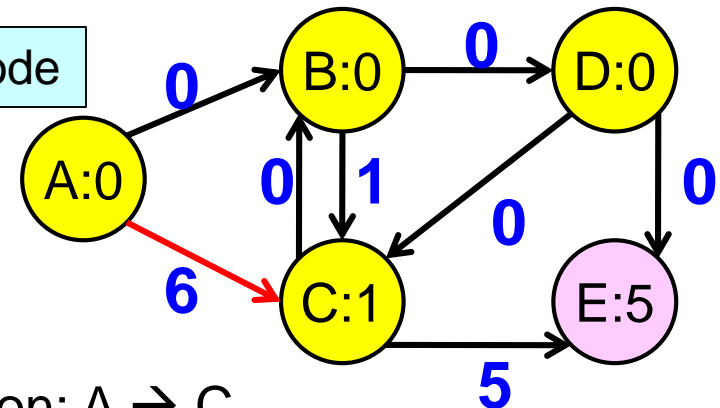


Action: $B \rightarrow C$
 $Q(B, B \rightarrow C) \leftarrow 1 + \max(0, 0) = 1$



Action: $C \rightarrow E$
 $Q(C, C \rightarrow E) \leftarrow 5$

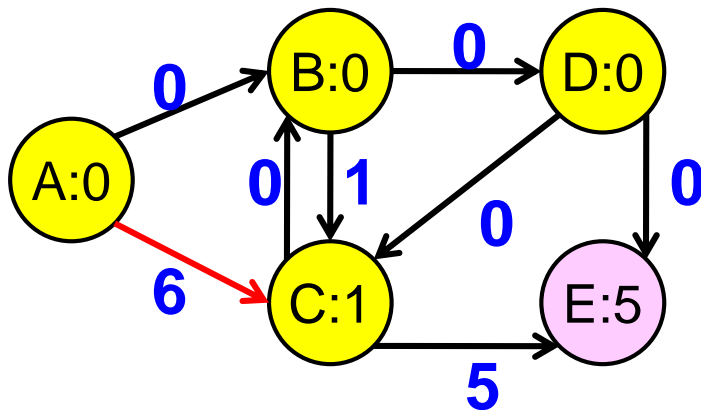
new episode



Action: $A \rightarrow C$
 $Q(A, A \rightarrow C) \leftarrow 1 + \max(0, 5) = 6$

Q Tables

- **Q-table** (the most common representation): A table of values of each combination of a state and a valid action from that state.
- Convenient for problems with finite states and finite valid actions per state.
- **Quantization** can be used for environments with continuous states and/or actions.



| | →A | →B | →C | →D | →E |
|---|----|----|----|----|----|
| A | - | 0 | 6 | - | - |
| B | - | - | 1 | 0 | - |
| C | - | 0 | - | - | 5 |
| D | - | - | 0 | - | 0 |
| E | - | - | - | - | - |

Q-Learning: Exploration vs. Exploitation

- Problem: During a training episode, how to choose the action from a state?
 - Best Q-value (greedy approach): Exploitation
 - Random action: Exploration
- ϵ -greedy: Use a probability (ϵ) to choose between the two. (More exploration initially, and more exploitation later to facilitate convergence.)
- (Optional) Adjustment of the discount factor: smaller initially (to avoid propagation of "noise") and larger later.

Generalization

In any realistic problem, it is impossible to learn the utility functions or Q-functions at all the possible states. What can we do?

- Express the utilities as a function of some "attributes" of the states. (Easier to generalize to new cases.)
 - Example: The piece-counting evaluation function for chess.
 - The weights (or function parameters) can be learned from the estimated utilities or Q-functions of actually visited states.

Deep Q-Learning

- Use a neural network to represent the Q function.
- The loss function:
$$(1/2) \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]^2$$
- The network weights are updated via backpropagation.
- DeepMind used DQL to make agents that play many Atari games to top human levels (2016).

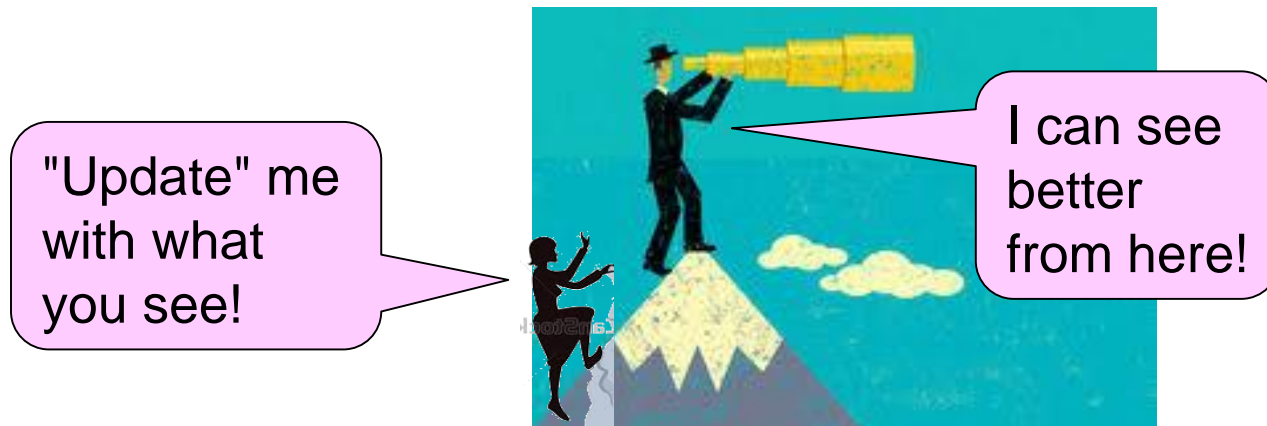


Temporal Difference (TD) Learning

- Idea: Use "estimated evaluations at future states" (which tend to be more accurate) to update the evaluation of the current state.
- A basic TD learning step (V is the evaluation function):

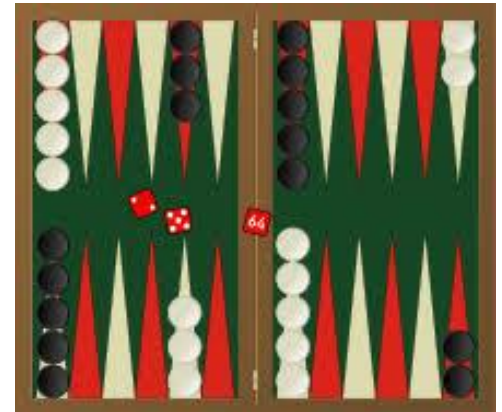
$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$$

- Q-learning also acts like TD learning.



Case Study: TD-Gammon

- Two-ply game tree search.
- Classical neural network (single hidden layer) for the evaluation function.
- First version: Pure reinforcement learning (**TD-lambda**) of the evaluation function without human knowledge. → Good but not top performance.
- Second version: The network learns a representation based on a set of expert-designed attributes. → As good as human world champions.



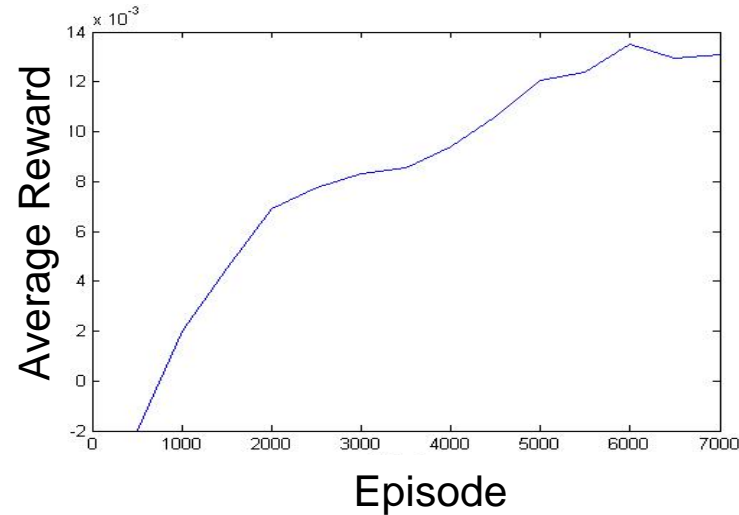
Case Study: Overtaking in Car Racing

- Played on a car racing game simulator.
- Quantization of features and actions:
 - Features: speed difference, cross-track positions of both cars, distance between the cars.
 - Actions: shift-left, shift-right, straight-ahead.
- Reward: success (+1), failure/crash (-1)
- Trained with Q-learning.



Case Study: Overtaking in Car Racing

- Typical learning curve:



- Example of a complex maneuver learned:

