Advanced Topics on Artificial Intelligence

Alban Grastien

The Australian National University

Second Semester, 2020

Model-free decision

- We are now considering the problem of decision under uncertainty when **the model is unknown**
  - We do not know the transition function $P(s, a, s')$
  - We do not know the reward function $R(s, a, s')$
  - But the state is still fully observable

My Previous Lecture: Bandit

Two important problems:

- How to update the knowledge
  - E.g., average outcome, temporal difference
- How to balance exploration / exploitation
  - E.g., ε-greedy strategy or UCB

Today

- Evaluation of a policy
- Computing the optimal (?) policy: SARSA / Q-learning

Small Number of States

- We now consider problems with small number of states.
- Small number of states $\rightarrow$ anywhere from 1 to millions, maybe billions.
- The idea is that all states can be enumerated and visited a few times.

Policy Evaluation

As with MDP, we first focus on policy evaluation:

- Given a policy $\pi : S \to A$
- calculate the value $V_\pi(s)$ of each (reachable) state, i.e., the expected discounted reward / cost from this state

Approaches

1. Run the system and record the output (new state, reward) to **estimate the parameters of the MDP**, and later use MDP techniques
   Issues:
   - Does not directly compute the value of the policy
   - Not good when the system dynamics change

2. Run the system and **record the long term outcome of each state** (using Temporal Difference)
   Issues:
   - Can be fairly imprecise if some states are visited rarely.

3. Run the system and **update state value estimate** using Temporal Difference (bootstrapping)

First Approach

Estimate-MDP

- *AddedUpReward*$[s, s'] = 0$ for all s, s'
- *AddedUpTransitions*$[s, s'] = 0$ for all s, s'
- **Repeat**:
  - Choose a state s and execute action $\pi(s)$
  - Let s' be the next state and r be the reward
  - *AddedUpReward*$[s, s'] += r$
  - *AddedUpTransitions*$[s, s'] += 1$
- **return** MDP $\langle S, A, P, R \rangle$ where
  - $P(s, \pi(s), s') = \frac{\mathit{AddedUpTransitions}[s,s']}{\Sigma_{q \in S} \mathit{AddedUpTransitions}[s,q]}$
  - $R(s, \pi(s), s') =$ *AddedUpReward*$[s, s']$/*AddedUpTransitions*$[s, s']$

Notice: here we use Average, but we could use Temporal Difference

Criticising the First Approach

- Computes an MDP and then we need to compute the value function
  - Very bad for incremental approach
- Is quadratic (keeps an entry for any pair of states)
  - That's potentially a problem from million of states on

Second Approach

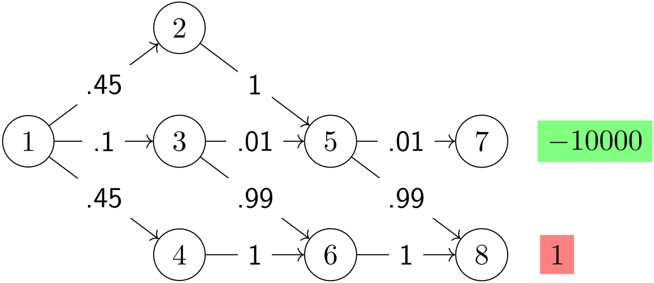
ESTIMATE-LONG-TERM-REWARDS

- For all state, $V(s) := 0$
- **Repeat:**
  - From initial state s, run policy π
  - Record rewards $r_1, \ldots, r_k$, states $s_1, \ldots, s_k$
  - For each index $i \in \{0, \ldots, k\}$,
    - *long-term-reward* $:= \Sigma_{j \geq i}\ \gamma^{j-i} r_j$
    - update $V(s_i) := V(s_i) + \alpha(\textit{long-term-reward} - V(s_i))$

Notice: here we use Temporal Difference, but we could use Average

Criticising the Second Approach

- Needs a significant amount of memory
- Fairly bad at estimating values in situations with low probability and high reward/cost



It will take a long time to get a good (accurate) estimate of 3

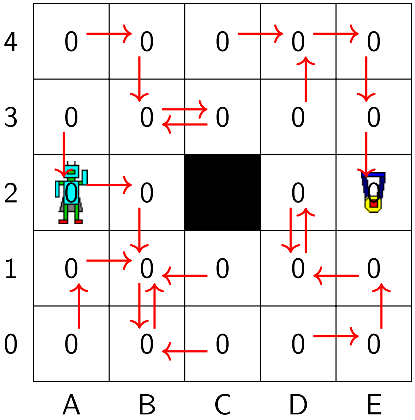
Third Approach: Bootstrapping and Temporal Difference

Temporal Difference

- For all state, $V(s) := 0$
- **Repeat:**
  - Choose a state s
  - Execute action $\pi(s)$ and observe state s' and reward r
  - Update state value: $V(s) := V(s) + \alpha\Big(r + \gamma V(s') - V(s)\Big)$

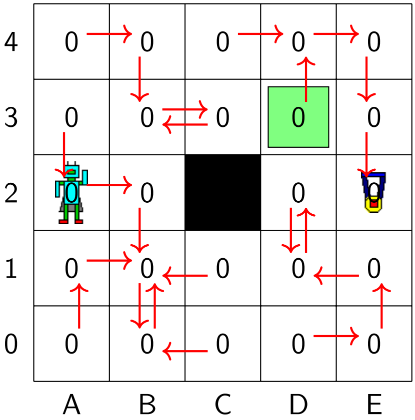
bootstrapping means that you compute the estimate of $V(s)$ from another estimate (here $V(s')$)

Example: Little Robot



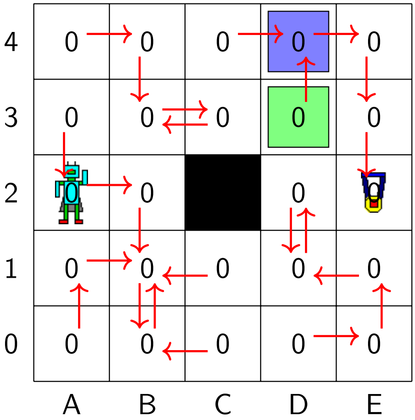
- Cost of action $= 1$;
  reward for getting in goal $= 0$;
  $\gamma = 1$
- Initialise value of states to 0
- $\alpha = 0.1$
- Arrows represent the policy, but the RL agent does not know what they mean!

Example: Little Robot



- Cost of action = 1;
  reward for getting in goal = 0;
  $\gamma = 1$
- Initialise value of states to 0
- $\alpha = 0.1$
- Arrows represent the policy, but the RL agent does not know what they mean!
- Select a state

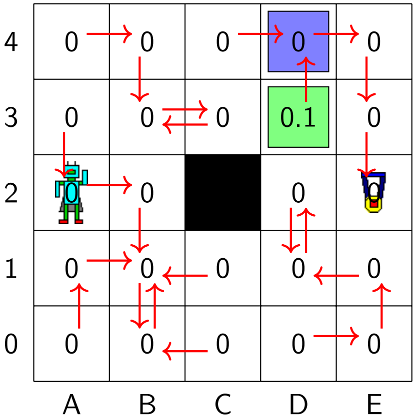
Example: Little Robot



- Cost of action = 1;
  reward for getting in goal = 0;
  $\gamma = 1$
- Initialise value of states to 0
- $\alpha = 0.1$
- Arrows represent the policy, but the RL agent does not know what they mean!

- Select a state
- Simulate action

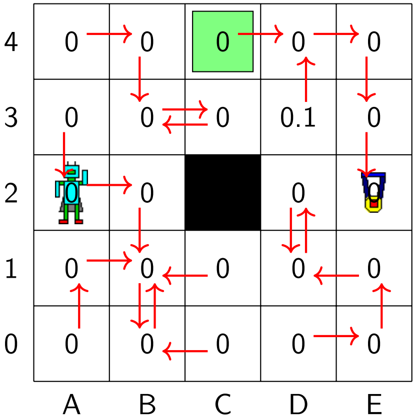
Example: Little Robot



- Cost of action = 1;
  reward for getting in goal = 0;
  $\gamma = 1$
- Initialise value of states to 0
- $\alpha = 0.1$
- Arrows represent the policy, but the RL agent does not know what they mean!

- Select a state
- Simulate action
- Update value

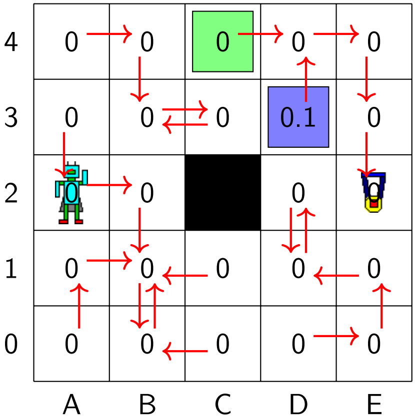
Example: Little Robot



- Cost of action = 1;
  reward for getting in goal = 0;
  $\gamma = 1$
- Initialise value of states to 0
- $\alpha = 0.1$
- Arrows represent the policy, but the RL agent does not know what they mean!

- Select a state
- Simulate action
- Update value

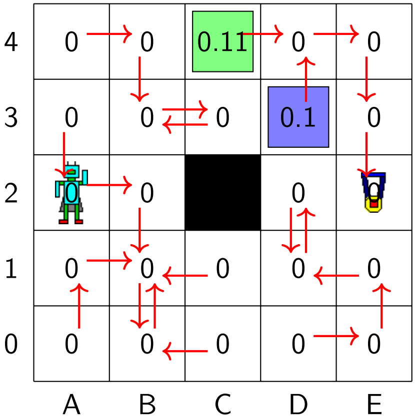
Example: Little Robot



- Cost of action = 1;
  reward for getting in goal = 0;
  $\gamma = 1$
- Initialise value of states to 0
- $\alpha = 0.1$
- Arrows represent the policy, but the RL agent does not know what they mean!

- Select a state
- Simulate action
- Update value

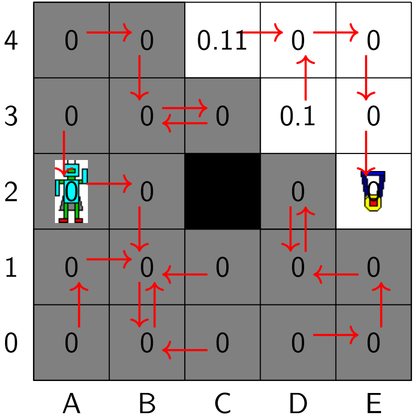
Example: Little Robot



- Cost of action $= 1$;
  reward for getting in goal $= 0$;
  $\gamma = 1$
- Initialise value of states to 0
- $\alpha = 0.1$
- Arrows represent the policy, but the RL agent does not know what they mean!

- Select a state
- Simulate action
- Update value

Example: Little Robot



- Cost of action $= 1$;
  reward for getting in goal $= 0$;
  $\gamma = 1$
- Initialise value of states to 0
- $\alpha = 0.1$
- Arrows represent the policy, but the RL agent does not know what they mean!

- Select a state
- Simulate action
- Update value

- Beware of states with ∞ value!

Properties of Temporal Difference

- Value estimate with TD converges to V_π in the mean for a constant α if α is sufficiently small
- Notice: only “in the mean”, but the estimate keeps fluctuating

Properties of Temporal Difference

- If α decreases according to the conditions presented in the section about Bandit,
  then it converges to V_π with probability 1

Finding Good / Optimal Policies

Problem definition:

- We assume given an MDP with unknown parameters
  - Fully observable
  - The list of applicable actions is known!
- Calculate a policy that maximises/minimises $V_\pi(s)$ for all states s

S A R S A

State Action Reward State Action

Q-Value

- With MDPs, we knew the probabilistic transition function P and the reward function R. Remember:
  - In MDPs, the optimal action is:

$$\arg\min_{a \in A(s)} \Sigma_{s' \in S} P(s,a,s') \times \left(R(s,a,s') + \gamma V^*(s') \right).$$

  It is easy to compute the optimal action from $V^*(s)$.

- In reinforcement learning, this is no longer true
  $\rightarrow$ If you don't know P and R, then knowing $V^*(s')$ for all s' is useless to decide the next action.

- Instead, we reason about the $\boxed{Q\text{-value}}$:
  - $Q(s,a)$ is the expected (discounted) value of applying a in s (assuming some policy / the best policy will be used after that).

How to Estimate the $Q(s, a)$?

Important factors:

- We want to boostrap
  - So we start with some estimate $Q(s, a)$
- We want to use temporal difference
  - Bootstrapping without temporal difference is a very bad idea because the initial estimated rewards are very bad
- We want to use the best moves (according to the current estimate)
- We also want to explore!

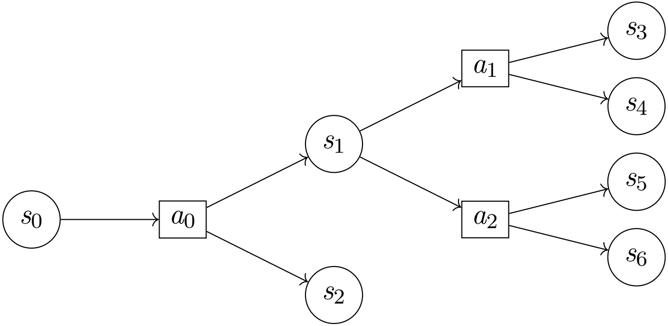
SARSA

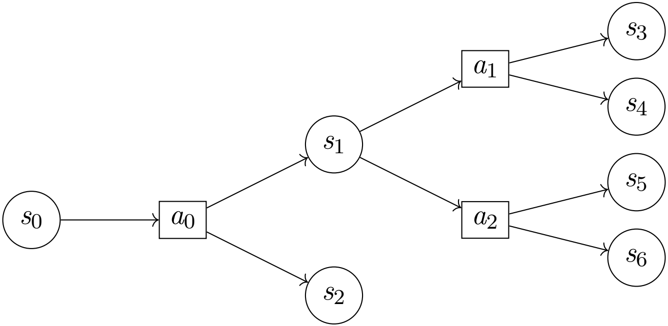
SARSA

- $Q(a, s) := 0$ for all $s \in S$ and $a \in A(s)$
- **Repeat**:
  - Start from initial state s
  - Let $a := \pi(s)$
  - Execute n times
    - $(r, s') :=$ *execute*(s, a)
    - Let $a' := \pi(s')$
    - $Q(s, a) := Q(s, a) + \alpha\Big(r + \gamma Q(s', a') - Q(s, a)\Big)$
    - $s := s'$ and $a := a'$

In general, π is the ϵ-greedy policy based on Q:

- With probability $1 - \epsilon$, $\pi(s) = \arg\max_{a \in A(s')} \; Q(s, a)$
- With probability ϵ, $\pi(s)$ is chosen (pseudo)-randomly


s_0
a_0
s_1
s_2
a_1
a_2
s_3
s_4
s_5
s_6



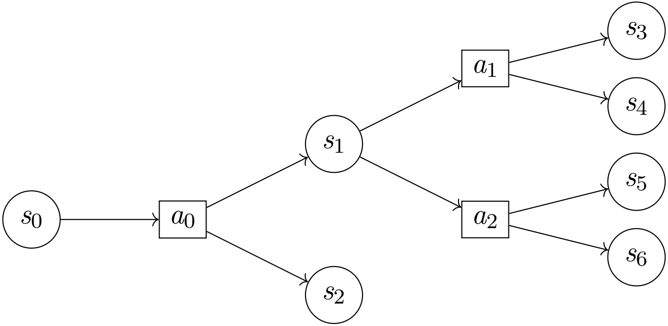
Example 1: executing a_0 in s_0

- $r = 1$
- $\gamma = .95$
- $\alpha = .05$
- Simulation $\rightarrow s_1$
- Next action: a_1

- $Q(s_0, a_0) = 10$
- $Q(s_1, a_1) = 10$
- $Q(s_1, a_2) = 3$

Updating the Q-value:

- $R := r + \gamma Q(s_1, a_1) = 10.5$
- $Q(s_0, a_0) := Q(s_0, a_0) + \alpha(R - Q(s_0, a_0)) = 10.025$



Example 2: executing a_0 in s_0

- $r = 1$
- $\gamma = .95$
- $\alpha = .05$
- Simulation $\rightarrow s_1$
- Next action: a_2

- $Q(s_0, a_0) = 10$
- $Q(s_1, a_1) = 10$
- $Q(s_1, a_2) = 3$

Updating the Q-value:

- $R := r + \gamma Q(s_1, a_2) = 3.85$
- $Q(s_0, a_0) := Q(s_0, a_0) + \alpha(R - Q(s_0, a_0)) = 9.6925$

Q-Learning

Q-Learning

Q-learning is a slight variation of SARSA.
We will discuss the practical differences later.

SARSA

- $Q(a, s) := 0$ for all $s \in S$ and $a \in A(s)$
- **Repeat**:
  - Start from initial state s
  - Execute n times
    - Let $a := \pi(s)$
    - $(r, s') :=$ *execute*(s, a)
    - $Q(s, a) := Q(s, a) + \alpha \Big(r + \gamma \max_{a' \in A(s')} \; Q(s', a') - Q(s, a) \Big)$
    - $s := s'$

In general, π is the ϵ-greedy policy based on Q

Differences between SARSA and Q-learning

The next discounted reward is estimated to:

- SARSA
  - $r + \gamma Q(s', a')$, which is the value that we expect to get this time
  - since a' is the action that we will perform from s'
  - We call this an *on-policy control*
- Q-learning
  - $r + \gamma \max_{a' \in A(s')} Q(s', a')$, which is the best outcome we can expect
  - Notice that we may then decide to perform an action different from the optimal action a', if we choose to explore
  - We call this an *off-policy control*

Which one to choose?

SARSA or Q-learning

- SARSA estimates the expected outcome assuming you will explore
- Q-learning estimates the expected outcome assuming you will take the best action

So...

- Choose SARSA if you want to use the system during the learning process
  - for instance, if you are driving your car

  Choose Q-learning if you don't mind making mistakes during the learning process
  - for instance, if you are practising for a competition