

Lecture 1: Introduction

Friday, October 22, 2021

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg



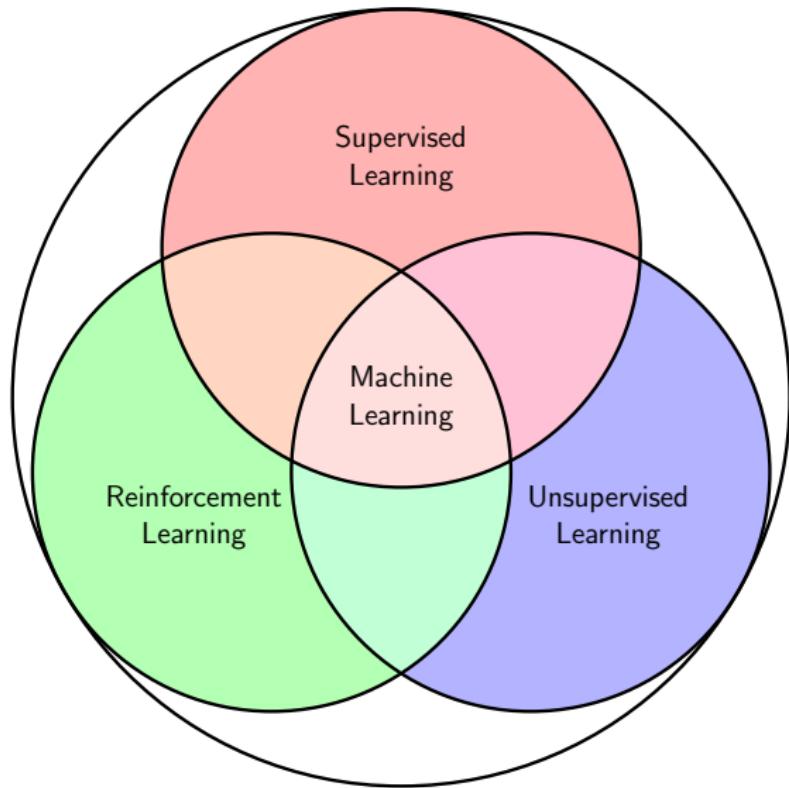
Lecture Overview

- 1 Reinforcement Learning
- 2 Organisation
- 3 Bandits

Lecture Overview

- 1 Reinforcement Learning
- 2 Organisation
- 3 Bandits

Reinforcement Learning in Machine Learning



Discussion

What are characteristics of Reinforcement Learning? What are the differences to Supervised Learning?



Characteristics of Reinforcement Learning

- Framework to solve sequential decision making problems
- No supervisor, trial-and-error
- Delayed feedback
- Behaviour affects subsequent data
- Exploration and exploitation

Examples

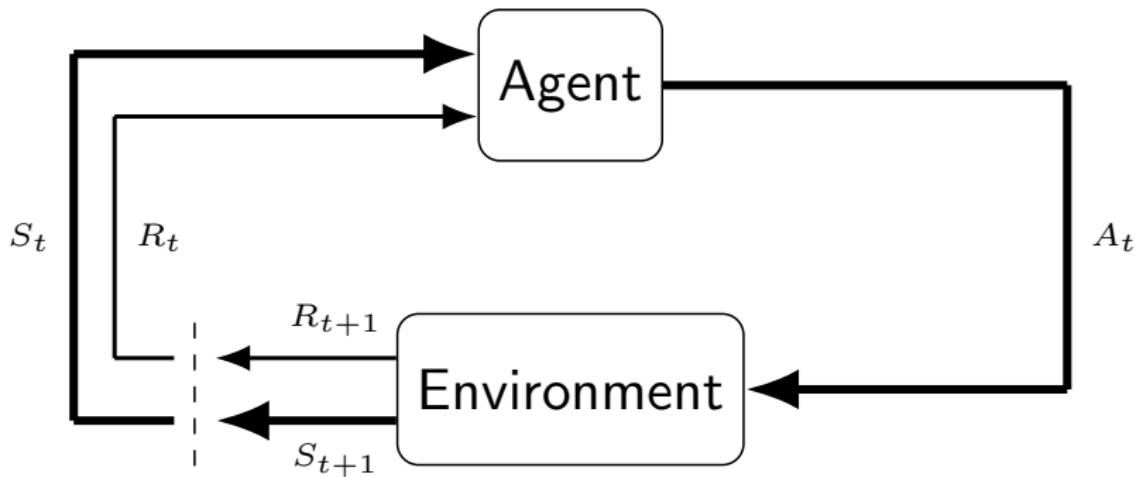
Videos in lecture recording.

Discussion

Where would you apply Reinforcement Learning?



Agent and Environment



Time steps $t: 0, 1, 2, \dots$

States: S_0, S_1, S_2, \dots

Actions: A_0, A_1, A_2, \dots

Rewards: R_1, R_2, R_3, \dots

Rewards

- A reward R_t in time step t is a **scalar** feedback signal.
- R_t indicates how well an agent is performing **at single time step t** .
- The agent aims at maximizing the expected discounted **cumulative reward** $G_t = R_{t+1} + \gamma^1 R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-(t+1)} R_T$.
 T can be infinite.

Reward Hypothesis

Everything we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

Examples:

- Chess: +1 for winning, -1 for losing
- Walking: +1 for every time step not falling over
- Investment Portfolio: difference in value between two time steps

Exploration and Exploitation

- Unique problem in Reinforcement Learning
- The agent has to exploit what it knows in order to obtain high reward (**Exploitation**)...
- ... but it has to explore to possibly do better in the future (**Exploration**).

Example: You want to go out for dinner. Do you...

- go to your favourite restaurant
- or try a new one?

Markov Decision Processes

A finite Markov Decision Process (MDP) is a 4-tuple $\langle \mathcal{S}, \mathcal{A}, p, \mathcal{R} \rangle$, where

- \mathcal{S} is a finite number of states,
- \mathcal{A} is a finite number of actions,
- p is the transition probability function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$,
- and \mathcal{R} is a finite set of scalar rewards. We can then define expected reward $r(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$ and $r(s, a, s') = \mathbb{E}[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s']$.

Markov Property

A state-reward pair (S_{t+1}, R_{t+1}) has the Markov property iff:

$$\Pr\{S_{t+1}, R_{t+1} | S_t, A_t\} = \Pr\{S_{t+1}, R_{t+1} | S_t, A_t, \dots, S_0, A_0\}.$$

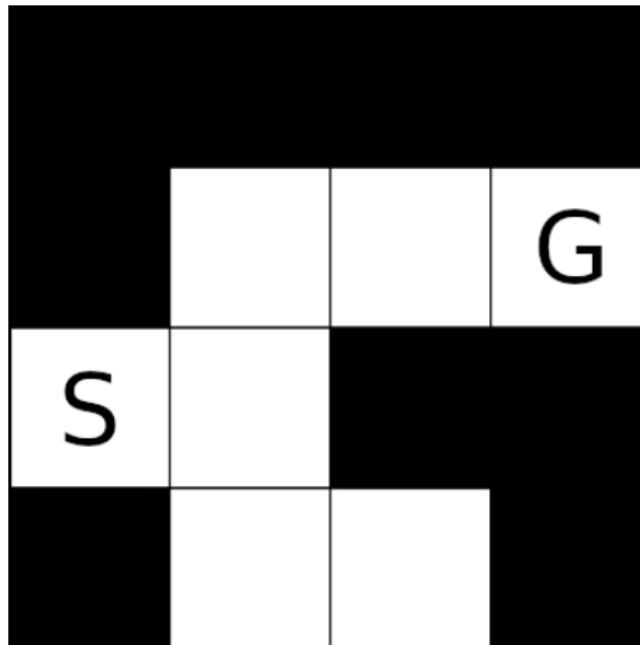
The future is independent of the past given the present.

Components of RL Systems

- Policy: defines the behaviour of the agent
 - is a mapping from a state to an action
 - can be stochastic: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$
 - or deterministic: $\pi(s) = a$
- Value-function: defines the expected value of a state or an action
 - $v_\pi(s) = \mathbb{E}[G_t|S_t = s]$ and $q_\pi(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a]$
 - Can be used to evaluate states or to extract a good policy
- Model: defines the transitions between states in an environment
 - p yields the next state and reward
 - $p(s', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a\}$

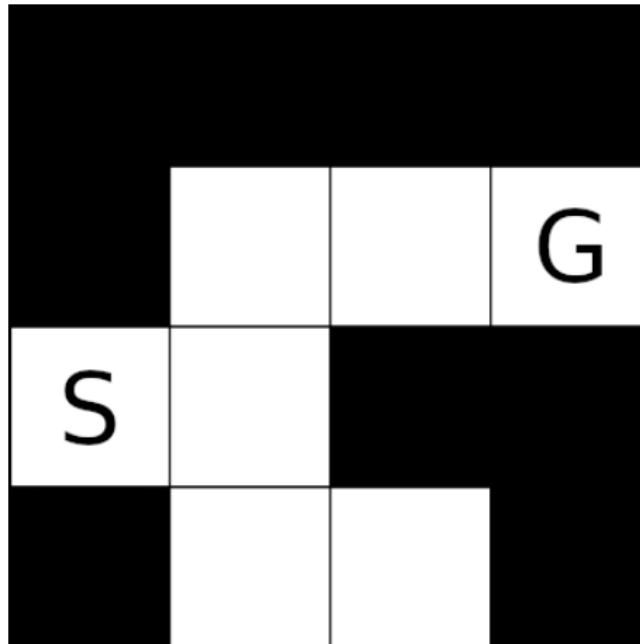
Maze Example: Policy

- Rewards: -1 per time step
- Actions: up, down, left, right
- States: location of the agent



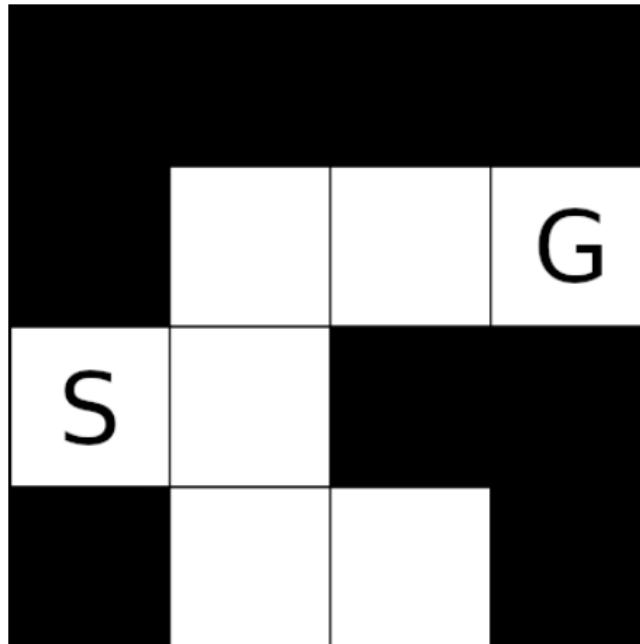
Maze Example: Value-function

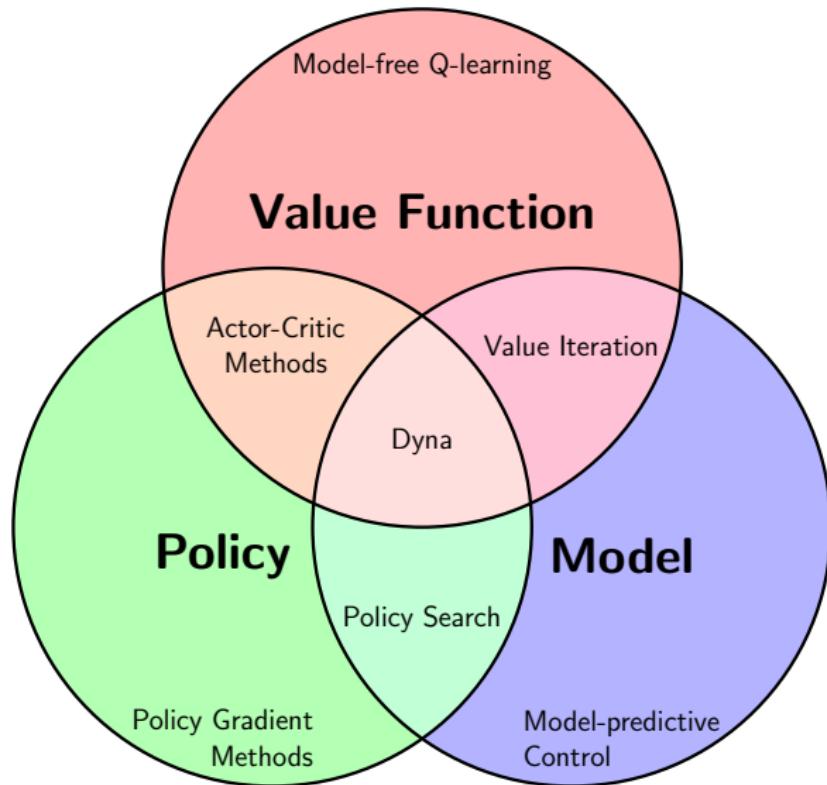
- Rewards: -1 per time step
- Actions: up, down, left, right
- States: location of the agent



Maze Example: Model

- Rewards: -1 per time step
- Actions: up, down, left, right
- States: location of the agent





Course Outline

- ① Introduction
- ② Markov Decision Processes
- ③ Dynamic Programming
- ④ Monte-Carlo Methods
- ⑤ Temporal-Difference Methods
- ⑥ Planning and Learning
- ⑦ On-policy Prediction and Control with Function Approximation
- ⑧ Off-policy Methods with Function Approximation
- ⑨ Pre-Winter Break Q&A
- ⑩ Eligibility Traces
- ⑪ Policy Gradient Methods
- ⑫ Advanced Value-based Methods with Function Approximation
- ⑬ Inverse Reinforcement Learning
- ⑭ Overview of RL research in our group
- ⑮ Invited Talk (hopefully)

Lecture Overview

1 Reinforcement Learning

2 Organisation

3 Bandits

Organisation

- 6 ECTS
- Flipped classroom with pre-recorded lectures
- Q & A online via Zoom, every week on Friday, 14:00 (s.t.) - 15:30
- Format of the exam: open-book, open-internet online ILIAS exam (90 min, date: TBA by Dec 31, 2021, likely time-period: March 2022)
- Course material and **forum** on ILIAS:
https://ilias.uni-freiburg.de/goto.php?target=crs_2376878&client_id=unifreiburg

Exercises

- Exercises are not mandatory, but highly recommended!
- Short walkthrough of sample solutions for most exercise sheets after the Q & A
- Please answer questions of others and ask your own questions in the **forum**

Exercises

- 6 ECTS = 180 working hours
- Lectures \approx 30 hours
- In addition: exercise sheets and exam preparation
- Time management options:

7-9 hours per ex.	little exam prep.	RECOMMENDED
5-6 hours per ex.	more exam prep.	MINIMUM
0 hours per ex.	∞ exam prep.	IMPOSSIBLE
- Slide taken from Hannah Bast

Team

Organizers of the Course



Joschka Boedecker



Gabriel Kalweit



Jasper Hoffmann

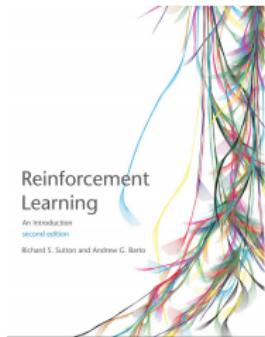


Andreas Saelinger

Links to other lectures

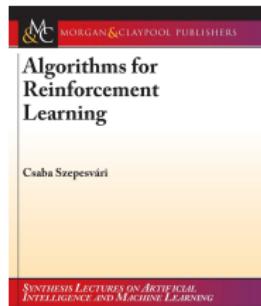
- Machine Learning Lecture (Summer term, recommended background)
- Deep Learning Lecture (Winter term, recommended background)
- Deep Learning Lab (Summer term)

Literature



Reinforcement Learning:
An Introduction (Sutton and Barto, 2018)
<http://incompleteideas.net/book/the-book.html>

Algorithms for Reinforcement Learning (Szepesvári, 2010)
<https://sites.ualberta.ca/~szepesva/RLBook.html>



Acknowledgement

Slide contents are partially based on the book *Reinforcement Learning: An Introduction* by Sutton and Barto and the Reinforcement Learning lecture by David Silver.

Lecture Overview

1 Reinforcement Learning

2 Organisation

3 Bandits

Multi-armed Bandits

- A multi-armed Bandit is a tuple $\langle \mathcal{A}, p \rangle$, where:
 - \mathcal{A} is a finite set of actions (or *arms*) and
 - $p(r|a) = \Pr\{R_t = r | A_t = a\}$ is an unknown probability distribution over rewards
- In each time step t , the agent selects an action A_t
- The environment then generates reward R_t
- The agent aims at maximizing the cumulative reward

Multi-armed Bandits

- The value of action a is the expected reward $q(a) = \mathbb{E}[R_t | A_t = a]$
- If the agent knew q , it could simply pick the action with highest value to solve the problem
- Assume we knew optimal action a_* . Then the total regret is defined as:

$$L_t = \mathbb{E} \left[\sum_{t=1}^T q(a_*) - q(a_t) \right]$$

- Maximize cumulative reward \equiv minimize total regret

Estimation of q

- We can estimate q given samples by $Q_t(a) = \frac{1}{N_t(a)} \sum_{t=1}^T R_t \mathbb{1}_{A_t=a}$, where $N_t(a)$ is the number of times a was taken in t time steps.
- Let R_i now denote the reward received after the i th selection of this action, and let Q_n denote the estimate of its action-value after it has been selected $n - 1$ times.
- Equally, we can use an incremental implementation:

$$Q_{n+1} = Q_n + \frac{1}{n} [R_n - Q_n]$$

- Generally, we can use some step size α :

$$Q_{n+1} = Q_n + \alpha [R_n - Q_n]$$

- A constant α can also be used for non-stationary problems (i.e. problems for which the reward probabilities change over time)
- Having an estimate of q , we can do greedy action selection by:

$$A_t = \arg \max_a Q_t(a)$$

ϵ -greedy

- One of the simplest ways to explore:
 - With probability $(1 - \epsilon)$ select the greedy action
 - With probability ϵ select a random action
- Can be coupled with a decay schedule for ϵ , so as to explore less when the estimate is quite accurate after some time
- Exemplary schedule: $\epsilon_{t+1} = (1 - \frac{t}{T})\epsilon_{\text{init}}$, where t is the current round and T is the maximum considered number of rounds

Softmax Action Selection

- Instead of choosing equally among actions, like ϵ -greedy, we can pick actions proportional to their value.
- This is called Softmax Action Selection and is modeled by a Boltzmann distribution over the Q-values:

$$\frac{\exp(\frac{Q_t(a)}{\tau})}{\sum_{i=1}^k \exp(\frac{Q_t(i)}{\tau})},$$

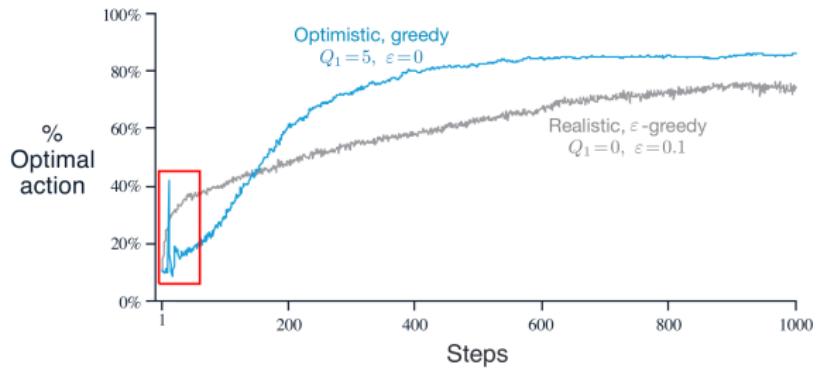
where τ is called the *temperature*.

- High temperatures cause the actions to be all (nearly) equiprobable.
- Low temperatures cause a greater difference in selection.

Optimistic Initial Values

- Initialize $Q_0(a)$ to high values for all $a \in \mathcal{A}$
- The result is that all actions are tried several times before the value estimates converge
- Encourages exploration only in the beginning, which is especially bad for non-stationary problems

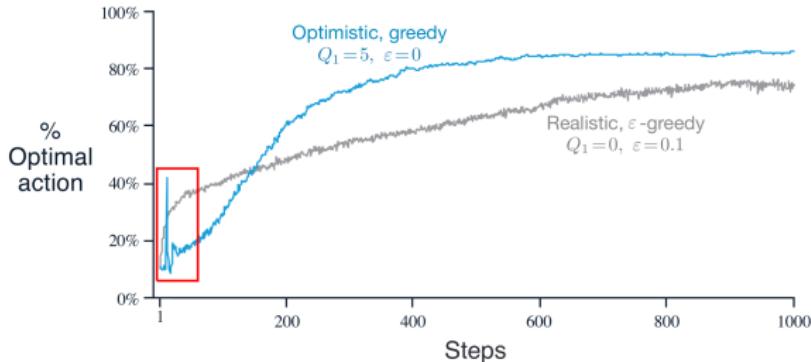
Question



Question (5 min)

The results should be quite reliable because they are averages over 2000 individual, randomly chosen 10-armed bandit tasks. Why, then, are there oscillations and spikes in the early part of the curve for the optimistic method? In other words, what might make this method perform particularly better or worse, on average, on particular early steps?

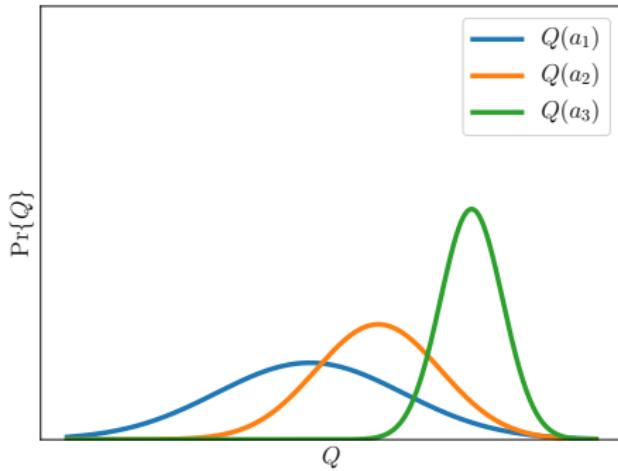
Solution



Solution

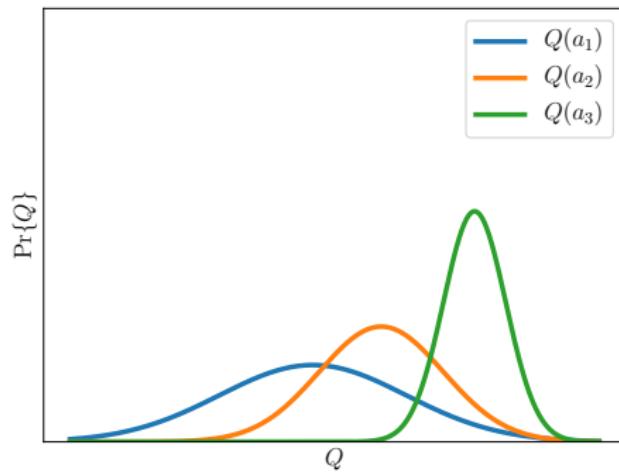
In the beginning, all action-values are overly optimistic. Therefore, each execution will reduce the value of the corresponding action. The value of the best action, however, will reduce the least in expectation. Thus, it will be focused by the greedy policy for some time – hence the peak. This will in turn reduce the value of the best action more, which is the reason for the drop. This procedure will repeat, but with smaller peaks the more the values converge.

Optimism in the Face of Uncertainty Principle



Which action should we pick?

Optimism in the Face of Uncertainty Principle



Which action should we pick?

Optimism in the Face of Uncertainty Principle

The more uncertain we are about an action-value, the more important it is to explore that action – since it could turn out to be the best action.

Upper Confidence Bound

- Exploration is needed because there is always uncertainty about the accuracy of the action-value estimates
- Idea: take into account how close their estimates are to being maximal and the uncertainties in those estimates
- For example by the *Upper Confidence Bound* (UCB) action selection:

$$A_t = \arg \max_a Q_t(a) + c \sqrt{\frac{\log t}{N_t(a)}}$$

- Each time a is selected, the uncertainty is presumably reduced
- On the other hand, each time an action other than a is selected, t increases but $N_t(a)$ does not
- One implementation of the *Optimism in the Face of Uncertainty Principle*

Contextual Bandits

- A contextual bandit is a tuple $\langle \mathcal{A}, p_S, p_R \rangle$, where:
 - $p_S(s) = \Pr\{S_t = s\}$ is an unknown distribution over states (or *contexts*) and
 - $p_R(r|s, a) = \Pr\{R_t = r | S_t = s, A_t = a\}$ is an unknown probability distribution over rewards
- In each time step t , the environment generates s_t and the agent selects action a_t
- The environment then generates reward r_t
- Example: Imagine a slot machine that changes its display color each time it changes its action-values
 - If you have no information about the task, the described bandit algorithms might not work very well
 - If you know the color, however, you can have different policies for different colors
- If actions are allowed to affect the next state as well as the immediate reward, then we have the full reinforcement learning problem



Lecture 01: Bandits and MDPs

#cs #rl

108 plays · 185 players

A public kahoot

Questions (10)

1 - Quiz

Exploration vs. Exploitation: What is correct?

60 sec

- | | | |
|--|---|--|
| | Exploitation: Take a random action. | |
| | Exploitation: Make best decision given current information. | |
| | Exploration: Try new actions. | |
| | Exploration: Apply best known action. | |

2 - Quiz

UCB can be categorized as ...

60 sec

- | | | |
|--|-------------------------------------|--|
| | Naive Exploration | |
| | Optimism in the Face of Uncertainty | |

3 - True or false

The more often an action has been chosen, the more likely it is chosen again by UCB.

20 sec

- | | | |
|--|-------|--|
| | True | |
| | False | |

4 - Quiz

Bandit strategies for solving the exploration-exploitation issue for RL are suboptimal,...

60 sec

- because they ignore the sequence of actions to be made. ✓
- because they are not an one-step decision-making approach. ✗
- because they are a multi-step decision-making approach. ✗
- because they consider the sequence of actions to be made. ✗

5 - Quiz

In contextual bandits, ...

60 sec

- we also consider the current state (i.e. context). ✓
- the best action depends on the given context. ✓
- the best action does not depend on the given state. ✗

6 - Quiz

A Markov Decision Process is defined by a set of states and ...

20 sec

- a value function. ✗
- transition probabilities between states. ✓
- a set of actions and a reward function. ✓
- a policy. ✗

7 - Quiz

A discount factor smaller 1 is required in a Markov Decision Process for

20 sec

- focusing on future rewards. ✗
- taking uncertainty about the future into account. ✓
- avoiding infinite rewards. ✓
- ensuring infinite rewards in the limit. ✗

8 - Quiz

The state value function $v(s)$ of an MDP is

20 sec

-  the expected return starting from state s and a specific action a. X
-  the expected return starting from s given a policy π . ✓

9 - Quiz

Which one is correct?

60 sec

-  (1) X
-  (2) X
-  (3) X
-  (4) ✓

10 - Quiz

To "solve" an MDP, we have to determine ...

20 sec

-  the state-value function for an arbitrary policy. X
-  the action-value function for an arbitrary policy. X
-  the state-value function for an optimal policy. ✓
-  the unique optimal policy. X

Lecture 2: Markov Decision Processes

Friday, October 29, 2021

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg



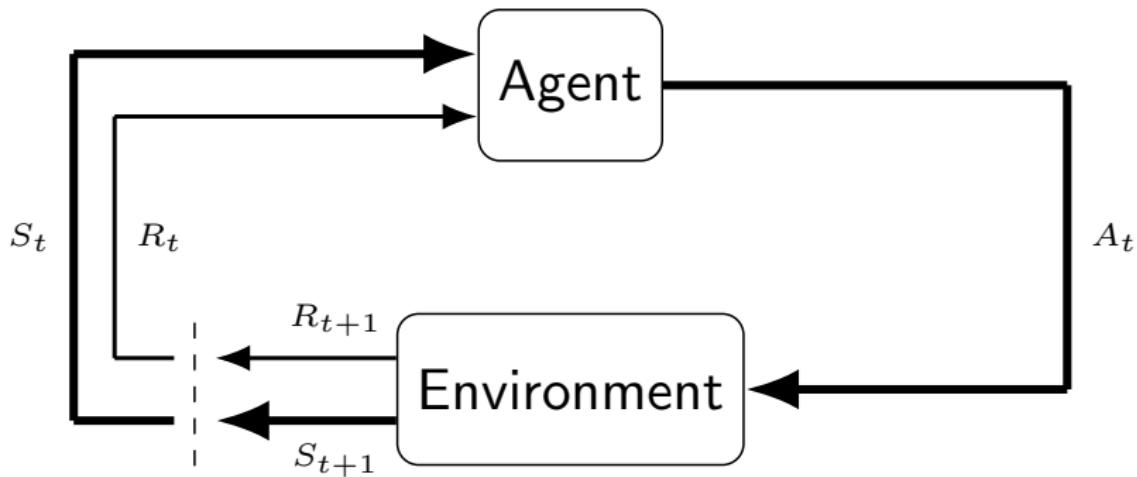
Lecture Overview

- 1 Markov Decision Processes
- 2 Policies and Value Functions
- 3 Optimal Policies and Value Functions
- 4 Extensions of the MDP framework
- 5 Wrapup

Lecture Overview

- 1 Markov Decision Processes
- 2 Policies and Value Functions
- 3 Optimal Policies and Value Functions
- 4 Extensions of the MDP framework
- 5 Wrapup

Agent and Environment



Time steps $t: 0, 1, 2, \dots$

States: S_0, S_1, S_2, \dots

Actions: A_0, A_1, A_2, \dots

Rewards: R_1, R_2, R_3, \dots

Markov Decision Processes

A finite Markov Decision Process (MDP) is a 4-tuple $\langle \mathcal{S}, \mathcal{A}, p, \mathcal{R} \rangle$, where

- \mathcal{S} is a finite number of states,
- \mathcal{A} is a finite number of actions,
- p is the transition probability function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$,
- and \mathcal{R} is a finite set of scalar rewards. We can then define expected reward $r(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$ and $r(s, a, s') = \mathbb{E}[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s']$.

Markov Property

A state-reward pair (S_{t+1}, R_{t+1}) has the Markov property iff:

$$\Pr\{S_{t+1}, R_{t+1} | S_t, A_t\} = \Pr\{S_{t+1}, R_{t+1} | S_t, A_t, \dots, S_0, A_0\}.$$

The future is independent of the past given the present.

Markov Property: Example

Question (3 min)

Imagine you want to apply the algorithms from this lecture on a real physical system. You get sensor input after each 0.01 seconds, but the execution of actions has a delay of 0.2 seconds.

Which adjustments to the state and/or action space would you have to do to fulfill the Markov Property?

Markov Property: Example

Solution (3 min)

Consider the history of the last 0.2 seconds as part of the state space. However, this blows up the state space which makes computations more challenging (*curse of dimensionality*).

Rewards

- A reward R_t in time step t is a **scalar** feedback signal.
- R_t indicates how well an agent is performing **at single time step t** .

Reward Hypothesis

All of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

Examples:

- Chess: +1 for winning, -1 for losing
- Walking: +1 for every time step not falling over
- Investment Portfolio: difference in value between two time steps

Return

- The agent aims at maximizing the expected **cumulative reward**
- Non-discounted: $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T$
- Discounted: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
- Discounting with $\gamma \in [0, 1]$ to prevent from infinite returns (e.g. in infinite horizon control problems)
- Returns at successive time steps are related to each other:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

MDP: Example

Question (5 min)

Imagine a house cleaning robot. It can have three charge levels: *high*, *low* and *none*. At every point in time, the robot can decide to *recharge* or to *explore* unless it has no battery. When exploring, the charge level can reduce with probability ρ . Exploring is preferable to recharging, however it has to avoid running out of battery.

Formalize the above problem as an MDP.

Solution

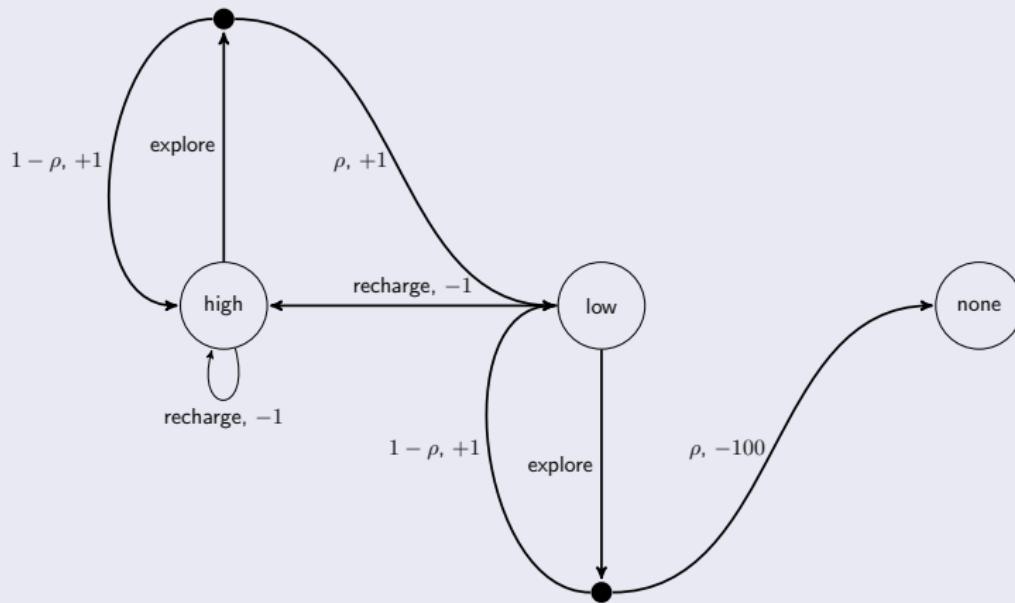
For the given problem, we set:

- $\mathcal{S} = \{\text{high, low, none}\}$
- $\mathcal{A} = \{\text{explore, recharge}\}$
- $\mathcal{R} = \{+1, -1, -100\}$ for exploring, recharging, and transitions leading to none, respectively.
- p has entries with value 1 for transitions (high, -1 , high, recharge), (low, -1 , high, recharge) and (none, 0 , none, \cdot). It further has entries with value ρ for transitions (high, $+1$, low, explore) and (low, -100 , none, explore) and entries with value $1 - \rho$ for transitions (high, $+1$, high, explore) and (low, $+1$, low, explore).

MDP: Example

Solution

The transition graph therefore is:



Lecture Overview

- 1 Markov Decision Processes
- 2 Policies and Value Functions
- 3 Optimal Policies and Value Functions
- 4 Extensions of the MDP framework
- 5 Wrapup

- The policy defines the behaviour of the agent:
 - can be stochastic: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$
 - or deterministic: $\pi(s) = a$
- Due to the Markov property, knowledge of the current state s is sufficient to make an informed decision.

Value Functions

- Value Function $v_\pi(s)$ is the expected return when starting in s and following π :

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right]$$

- Action-Value Function q_π is the expected return when starting in s , taking action a and following π thereafter:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right]$$

- Simple connection:

$$v_\pi(s) = \mathbb{E}_\pi[q_\pi(s, \pi(s))] \tag{1}$$

Bellman Equation

- The Bellman Equation expresses a relationship between the value of a state and the values of its successor states
- The value function v_π is the unique solution to its Bellman Equation

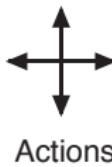
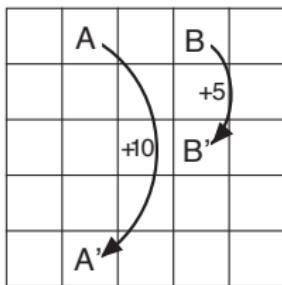
$$\begin{aligned}v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\&= \mathbb{E}_\pi[R_t + \gamma G_{t+1} | S_t = s] \\&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) [r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s']] \\&= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) [r + \gamma v_\pi(s')]\end{aligned}$$

Bellman Equation for v_π

The Bellman Equation for v_π is defined as:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) [r + \gamma v_\pi(s')].$$

Bellman Equation: Example

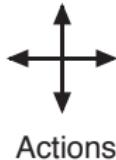
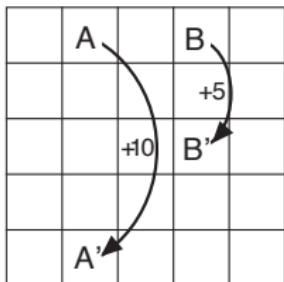


3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Question (5 min)

Actions move the agent deterministically. Actions that would move the agent off the grid cost -1 with no state change. All other actions are free. However, every action performed by the agent in A moves it to A' with a reward of $+10$, each action in B moves it to B' with a reward of $+5$. Assume a uniform policy. v_π with a discounting factor of $\gamma = 0.9$ is to the right. Show exemplary for state $s_{0,0}$ with $v_\pi(s_{0,0}) = 3.3$ that the Bellman equation is satisfied.

Bellman Equation: Example



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Solution

$$\begin{aligned}v_{\pi}(s_{0,0}) &= 0.25 \cdot (-1 + \gamma \cdot 3.3) + \\&\quad 0.25 \cdot (+0 + \gamma \cdot 8.8) + \\&\quad 0.25 \cdot (+0 + \gamma \cdot 1.5) + \\&\quad 0.25 \cdot (-1 + \gamma \cdot 3.3) \\&= 3.3025 \\&\approx 3.3\end{aligned}$$

Bellman Equation

We equivalently obtain a corresponding system of equations for the Q-function:

Bellman Equation for q_π

The Bellman Equation for action-value function q_π is defined as:

$$q_\pi(s, a) = \sum_{s', r} p(s', r | s, a) \left[r + \gamma \sum_{a'} \pi(a' | s') q_\pi(s', a') \right].$$

Lecture Overview

- 1 Markov Decision Processes
- 2 Policies and Value Functions
- 3 Optimal Policies and Value Functions
- 4 Extensions of the MDP framework
- 5 Wrapup

Optimality of Policies

We consider a policy as optimal if the value (i.e. its expected return under the policy) in every state is at least as high as for any other policy:

Optimality of a policy π_*

A policy π_* is called *optimal* : \Leftrightarrow

For all $s \in S$:

$$v_{\pi_*}(s) \geq v_{\pi}(s) \text{ for all } \pi \quad (2)$$

The corresponding *optimal value function* is denoted by v_* .

- This requires a search among all, possibly infinitely many, policies.
This seems to be rather impractical.
- Is there an easier way to check if a policy π and corresponding value function v_{π} is actually optimal?

Bellman Optimality Equation

Intuitively, the Bellman Optimality Equation expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state:

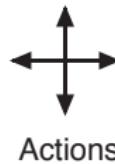
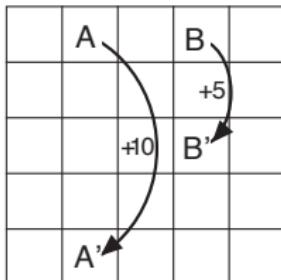
$$\begin{aligned}v_*(s) &= \max_a q_{\pi_*}(s, a) \\&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\&= \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma v_*(s')]\end{aligned}$$

Bellman Optimality Equation for v_*

The Bellman Equation for the optimal value function v_* is defined as:

$$v_*(s) = \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma v_*(s')].$$

Bellman Optimality Equation: Example



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Non-optimality of the uniform random policy

$$\begin{aligned}v_{\pi}(s_{0,0}) &= 3.3025 \neq \max\{-1 + \gamma \cdot 3.3, 0 + \gamma \cdot 8.8, \\&\quad 0 + \gamma \cdot 1.5, -1 + \gamma \cdot 3.3\} \\&= 7.92\end{aligned}$$

⇒ random policy π is *not* optimal.

Bellman Optimality Equation

Equivalently, there exists a Bellman optimality equation for Q-functions:

Bellman Optimality Equation for q_*

The Bellman Equation for the optimal action-value function q_* is:

$$q_*(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} q_*(s', a')].$$

Lecture Overview

- 1 Markov Decision Processes
- 2 Policies and Value Functions
- 3 Optimal Policies and Value Functions
- 4 Extensions of the MDP framework
- 5 Wrapup

Extensions to the MDP framework: POMDPs

A Partially Observable Markov Decision Process (POMDP) is a 6-tuple $\langle \mathcal{S}, \mathcal{A}, p, \mathcal{R}, \Omega, p_O \rangle$, where

- \mathcal{S} is a finite number of states,
- \mathcal{A} is a finite number of actions,
- Ω is a finite number of observations,
- p is the transition probability function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$,
- \mathcal{R} is a finite set of scalar rewards. We can then define expected reward $r(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$ and $r(s, a, s') = \mathbb{E}[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s']$.
- and p_O is the observation probability function, $p_O : \Omega \times \mathcal{S} \mapsto [0, 1]$

The transitions of the true state S_t still obey the Markov property, *but* observations O_t do not.

Lecture Overview

- 1 Markov Decision Processes
- 2 Policies and Value Functions
- 3 Optimal Policies and Value Functions
- 4 Extensions of the MDP framework
- 5 Wrapup

Summary by Learning Goals

Having heard this lecture, you can now . . .

- formulate a given problem as an MDP
- explain the Markov-Property
- describe the relationship between rewards and values
- state the Bellmann Expectation and Optimality equations
- explain the relationship between optimal value function and optimal policy



Lecture 02: Dynamic Programming

#cs #rl

92 plays · 155 players

A public kahoot

Questions (6)

1 - Quiz

Dynamic programming can be applied if ...

60 sec

- the principle of suboptimality holds true. ✗
- subproblems are independent. ✗
- the optimal solution can be decomposed into subproblems. ✓
- subproblems occur only once. ✗

2 - Quiz

Prediction vs Control: what is correct?

60 sec

- For control, the policy is an input. ✗
- For control, an arbitrary policy is the output. ✗
- For prediction, the optimal value function is the output. ✗
- For prediction, a value function for given π is the output. ✓

3 - Quiz

For iterative policy evaluation, we iteratively apply the ...

30 sec

- Bellman expectation equation ✓
- Bellman optimality equation ✗

4 - Quiz

In policy iteration, we interleave ...

60 sec

- policy evaluation and improvement. ✓
- policy evaluation and value iteration. ✗
- policy improvement and value iteration. ✗
- policy evaluation and dynamic programming. ✗

5 - Quiz

In value iteration, we...

60 sec

- iteratively apply the Bellman expectation equation. ✗
- iteratively apply the Bellman optimality equation. ✓
- have corresponding policies in each iteration. ✗
- iteratively improve the value function until convergence. ✓

6 - Quiz

How to break the curse of dimensionality for DP?

60 sec

- Use value iteration instead of policy iteration. ✗
- Sample backups instead of full-width backups. ✓
- Learn a model of the environment. ✗

Lecture 3: Dynamic Programming

Friday, November 12, 2021

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg



Lecture Overview

1 Policy Iteration

2 Value Iteration

3 Wrapup

Lecture Overview

1 Policy Iteration

2 Value Iteration

3 Wrapup

Recap: Bellman Optimality Equations

Bellman Optimality Equation for v_*

The Bellman Equation for the optimal value function v_* is defined as:

$$v_*(s) = \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma v_*(s')].$$

Bellman Optimality Equation for q_*

The Bellman Equation for the optimal action-value function q_* is:

$$q_*(s, a) = \sum_{s',r} p(s', r | s, a) [r + \gamma \max_{a'} q_*(s', a')].$$

How can we turn these equations into practical algorithms to find optimal policies π_* ?

Policy Iteration: Overview

Idea: Alternate **evaluating** the value function v_π and **improving** the policy π to convergence.

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

Policy Evaluation

Compute the state-value function v_π for an arbitrary policy π .

$\forall s \in S :$

$$v_\pi(s) \doteq \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_\pi(s')]$$

If the environments dynamics are completely known, this is a system of $|\mathcal{S}|$ simultaneous linear equations in $|\mathcal{S}|$ unknowns. With the Bellman equation, we can iteratively update an initial approximation v_0 :

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi [R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')] \end{aligned}$$

Policy Evaluation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π , the policy to be evaluated

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

$$\Delta \leftarrow 0$$

Loop for each $s \in \mathcal{S}$:

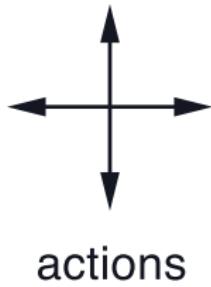
$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$

Example: Gridworld



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R_t = -1$
on all transitions

Policy Improvement

Once we have the value function for a policy, we consider which action a to select in a state s when we follow our old policy π afterwards. To decide this, we look at the Bellman equation of the state-action value function:

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')] \end{aligned}$$

Policy improvement theorem

Let π and π' be any pair of deterministic policies. If, $\forall s \in S$,

$$q_\pi(s, \pi'(s)) \geq v_\pi(s),$$

then the policy π' must be as good as, or better than, π . It follows that, $\forall s \in S$:

$$v_{\pi'}(s) \geq v_\pi(s)$$

Policy Improvement

To implement this, we compute $q_{\pi}(s, a)$ for *all* states and *all* actions, and consider the greedy policy:

$$\begin{aligned}\pi'(s) &\doteq \arg \max_a q_{\pi}(s, a) \\&= \arg \max_a \mathbb{E} [R_{t+1} + \gamma v_{\pi}(S_{t_1}) | S_t = s, A_t = a] \\&= \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma v_{\pi}(s')]\end{aligned}$$

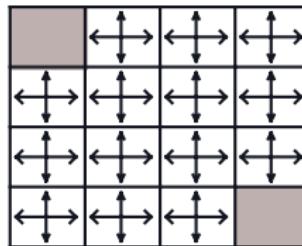
Example: Gridworld

v_k for the
random policy

$k = 0$

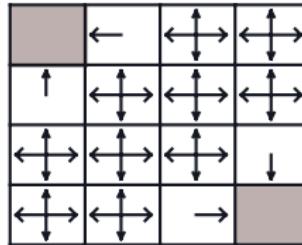
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

greedy policy
w.r.t. v_k



$k = 1$

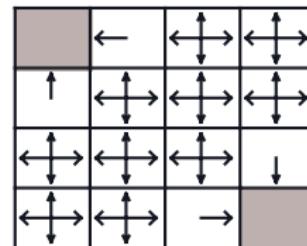
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



Example: Gridworld

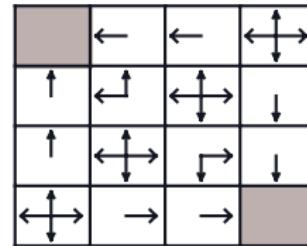
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

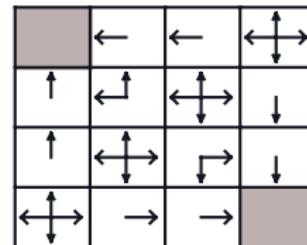
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Example: Gridworld

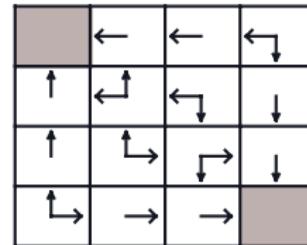
$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



Example: Gridworld

$k = 3$

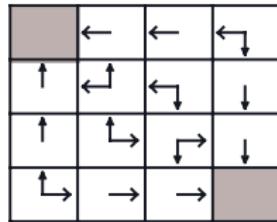
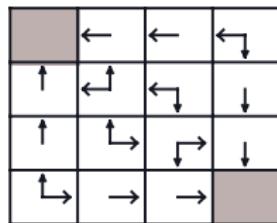
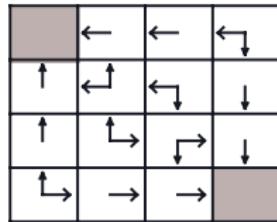
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



optimal policy

Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

Loop:

$$\Delta \leftarrow 0$$

Loop for each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_{s',r} p(s', r | s, \pi(s)) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

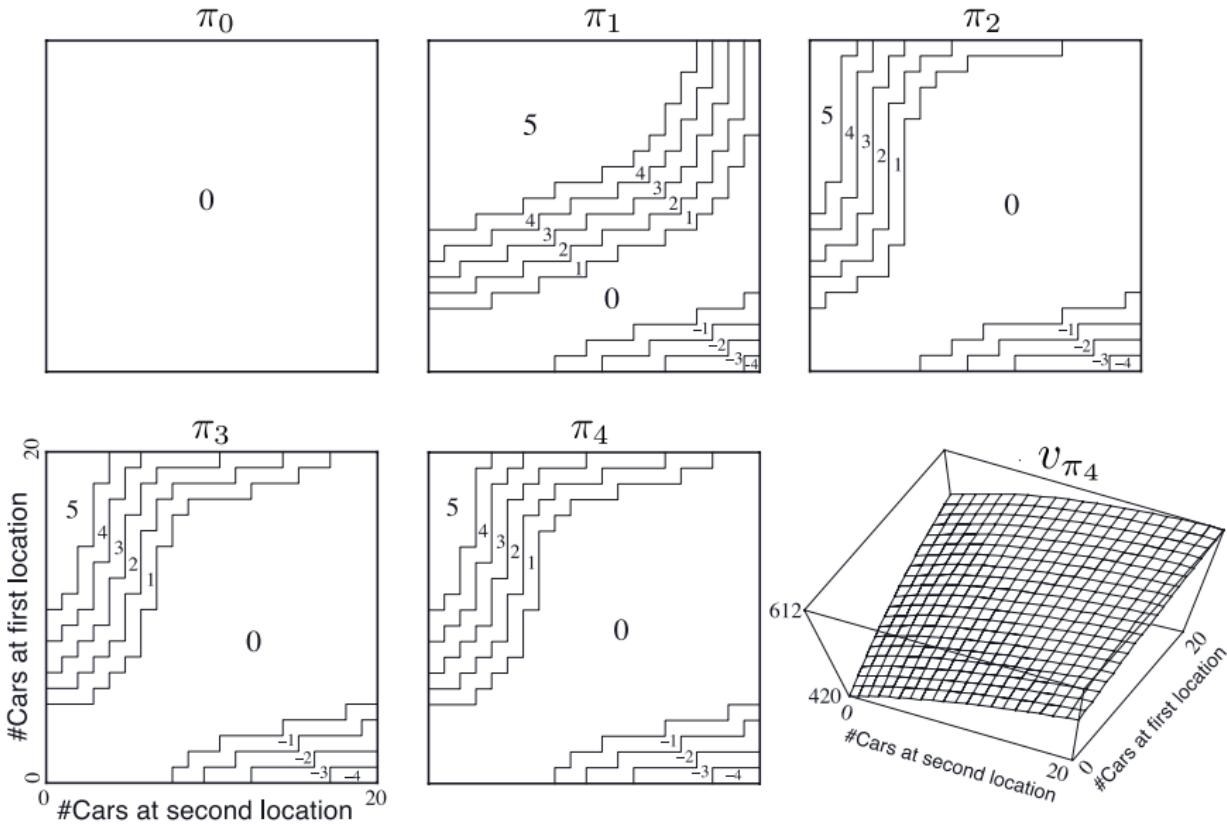
$$\text{old-action} \leftarrow \pi(s)$$

$$\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

If $\text{old-action} \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Example: Car rental



Lecture Overview

1 Policy Iteration

2 Value Iteration

3 Wrapup

Value Iteration

Performing policy evaluation to convergence *in every iteration* is costly and often not necessary. A special case is to evaluate just once and combine it with the policy improvement step:

$$\begin{aligned}v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\&= \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma v_k(s')]\end{aligned}$$

Value Iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

| $\Delta \leftarrow 0$

| Loop for each $s \in \mathcal{S}$:

| | $v \leftarrow V(s)$

| | $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$

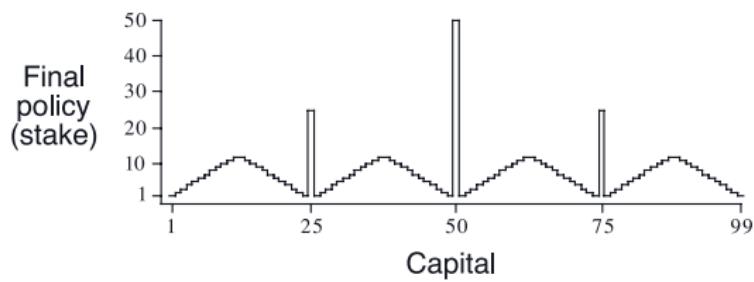
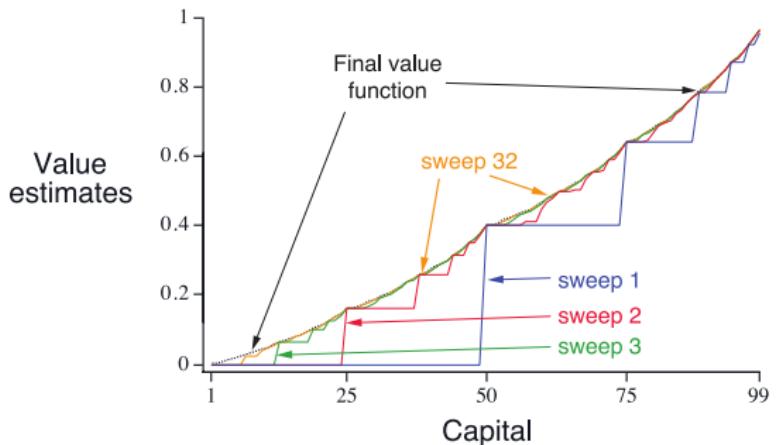
| | $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that

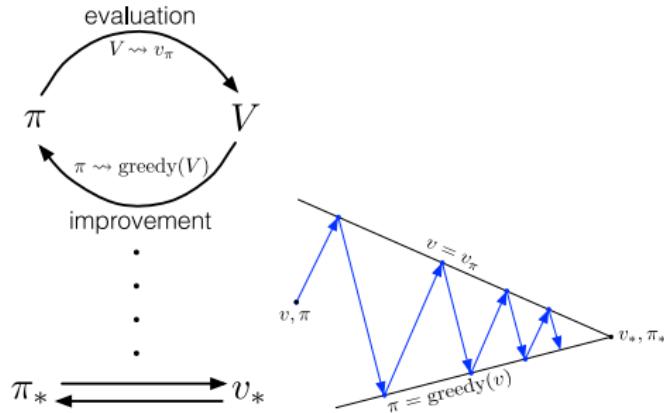
$$\pi(s) = \arg \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$$

Gambler's Problem



Additional topics

- Asynchronous Dynamic Programming
- Generalized Policy Iteration
- Efficiency of Dynamic Programming



Lecture Overview

1 Policy Iteration

2 Value Iteration

3 Wrapup

Summary by Learning Goals

Having heard this lecture, you can now...

- formulate and apply Policy Iteration
- formulate and apply Value Iteration



Lecture 03: Monte Carlo Methods

83 plays · 132 players

A public kahoot

Questions (7)

1 - Quiz

For Monte-Carlo Policy Evaluation, we have to...

60 sec

- update only the first time-step that a state is visited. ✗
- update every time-step that a state is visited. ✗
- store all returns, we ever observed. ✗
- use the empirical mean return instead of expected return. ✓

2 - Quiz

Compared to Policy Evaluation, Monte Carlo evaluation...

60 sec

- returns a more accurate value-function. ✗
- converges faster. ✗
- does not require a model. ✓
- needs less exploration in the real world. ✗

3 - True or false

When MC eval. sees each state infinitely often, it converges to the true value-function ind. from the learning rate.

60 sec

- True ✗
- False ✓

4 - Quiz

If we base our improvement on $Q(s, a)$, we...

60 sec

-  ... don't need transition and reward probabilities. 
-  ... can ignore exploration. 

5 - True or false

MC methods can only be used in on-policy settings.

30 sec

-  True 
-  False 

6 - True or false

The IS ratio for MC control only needs to take the current time step of the updated state into account.

30 sec

-  True 
-  False 

7 - True or false

By adding the IS ratio, bias and variance are reduced to make learning off-policy possible.

30 sec

-  True 
-  False 

Lecture 4: Monte Carlo Methods

Friday, November 19, 2021

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg



Lecture Overview

1 Recap

2 Monte Carlo Prediction

3 Monte Carlo Control

4 Wrapup

Lecture Overview

1 Recap

2 Monte Carlo Prediction

3 Monte Carlo Control

4 Wrapup

Recap: Dynamic Programming

Last lecture: Planning by dynamic programming, solve a *known* MDP.

Policy Iteration

Alternate **evaluating** the value function v_π and **improving** the policy π to convergence.

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

Value Iteration

Evaluate just once and combine it with the policy improvement step.

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma v_k(s')] \end{aligned}$$

Monte Carlo Reinforcement Learning

This lecture: Model-free prediction and control. Estimate/ Optimize the value function of an *unknown* MDP.

- MC methods learn from episodes of *experiences*
experiences = sequences of states, actions, and rewards
- MC is model-free: no knowledge required about MDP dynamics
- MC learns from complete episodes (no bootstrapping), based on averaging sample returns

Lecture Overview

1 Recap

2 Monte Carlo Prediction

3 Monte Carlo Control

4 Wrapup

Monte Carlo Prediction

- Goal: learn the state-value function v_π for a given policy π

$$S_0, A_0, R_1, \dots, S_T \sim \pi$$

- Idea: estimate it from experience by average the returns observed after visits to that state
- Recall: the *return* is the total discounted reward

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Recall: the value function is the expected return

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

- Monte-Carlo policy prediction uses the empirical mean return instead of expected return

Incremental and Running Mean

- We can compute the mean of a sequence x_1, x_2, \dots incrementally:

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum_{j=1}^k x_j \\ &= \frac{1}{k} \left(x_k + \sum_{j=1}^{k-1} x_j \right) \\ &= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})\end{aligned}$$

Incremental and Running Mean

- Thus, we can update $V(s)$ incrementally by:

$$V(s) \leftarrow V(s) + \frac{1}{N(s)}(G_t - V(s)),$$

where $\frac{1}{N(s)}$ is the state-visitation counter

- Instead $\frac{1}{k}$, we can use step size α to calculate a running mean:

$$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$$

Monte Carlo Prediction

- First-visit MC method: Estimates $v_\pi(s)$ as the average of the returns following first visits to s .
- Every-visit MC method: Estimates $v_\pi(s)$ as the average of the returns following all visits to s .

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in S_0, S_1, \dots, S_{t-1} :

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

Example: Blackjack

- States (200 of them) :
 - Current sum (12 - 21)
 - Dealer's showing card (ace-10)
 - Do I have a *usable* ace? (yes-no)
- Action stick: Stop receiving cards (and terminate)
- Action twist: Take another card (no replacement)
- Reward for stick:
 - +1 if sum of cards > sum of dealer cards
 - 0 if sum of cards = sum of dealer cards
 - -1 if sum of cards < sum of dealer cards
- Reward for twist:
 - -1 if sum of cards > 21 (and terminate)
 - 0 otherwise
- Transitions: automatically twist if sum of cards < 12

Example: Blackjack

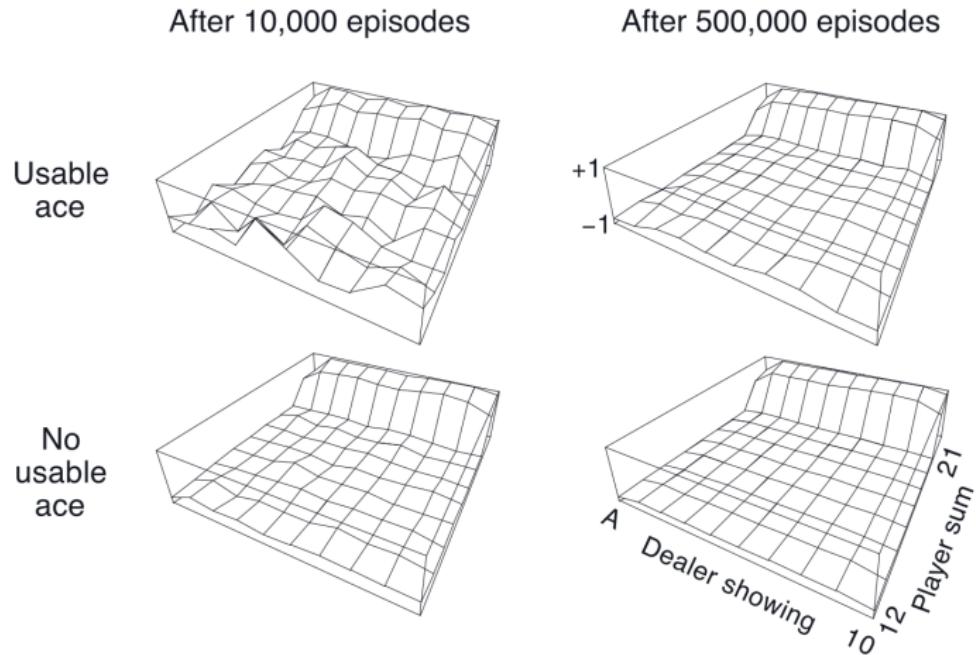


Figure 5.1: Approximate state-value functions for the blackjack policy that sticks only on 20 or 21, computed by Monte Carlo policy evaluation. ■

Lecture Overview

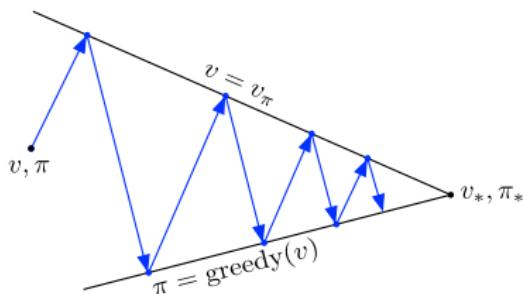
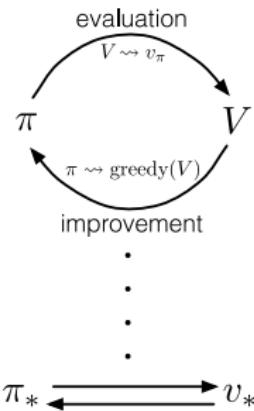
1 Recap

2 Monte Carlo Prediction

3 Monte Carlo Control

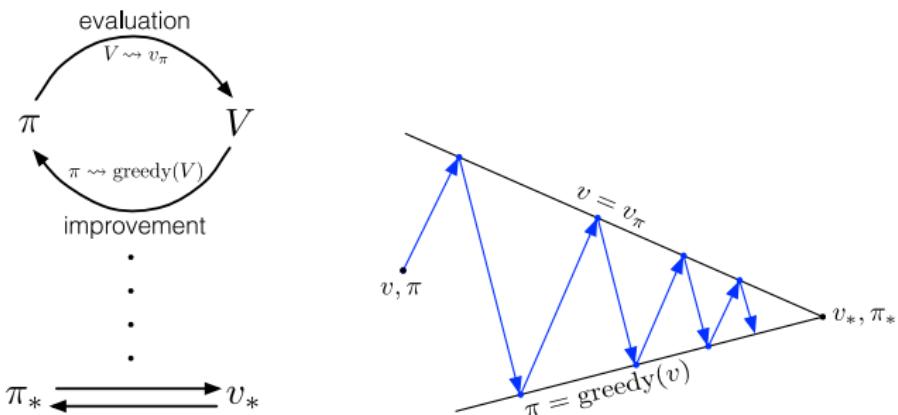
4 Wrapup

Generalized Policy Iteration



- Policy Evaluation: estimate v_π
- Policy Improvement: greedy

Generalized Policy Iteration with MC Evaluation



- Monte Carlo Policy Evaluation: $V \approx v_\pi$
- Policy Improvement: greedy?

Monte Carlo Estimation of Action Values

- Greedy policy improvement over $V(s)$ requires a model of the MDP

$$\pi(s) = \arg \max_{a \in \mathcal{A}} \sum_{s', r} p(s', r | s, a) [r + \gamma V(s')]$$

- Greedy policy improvement over $Q(s, a)$ is model-free

$$\pi(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)$$

Generalized Policy Iteration with action-value function:

- Monte Carlo Policy Evaluation: $Q \approx q_\pi$
- Policy Improvement: greedy?

ϵ -greedy Policy Improvement

- We have to ensure that each state-action pair is visited a sufficient (infinite) number of times
- Simple idea: ϵ -greedy
- All actions have non-zero probability
- With probability ϵ choose a random action, with probability $1 - \epsilon$ take the greedy action.

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{|\mathcal{A}|} + 1 - \epsilon & \text{if } a = \arg \max_{a' \in \mathcal{A}} Q(s, a') \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases}$$

ϵ -greedy Policy Improvement

$$\begin{aligned} q_{\pi}(s, \pi'(s)) &= \sum_a \pi'(a|s) q_{\pi}(s, a) \\ &= \frac{\epsilon}{|\mathcal{A}|} \sum_a q_{\pi}(s, a) + (1 - \epsilon) \max_a q_{\pi}(s, a) \\ &\geq \frac{\epsilon}{|\mathcal{A}|} \sum_a q_{\pi}(s, a) + (1 - \epsilon) \sum_a \frac{\pi(a|s) - \frac{\epsilon}{|\mathcal{A}|}}{1 - \epsilon} q_{\pi}(s, a) \\ &= \frac{\epsilon}{|\mathcal{A}|} \sum_a q_{\pi}(s, a) - \frac{\epsilon}{|\mathcal{A}|} \sum_a q_{\pi}(s, a) + \sum_a \pi(a|s) q_{\pi}(s, a) \\ &= v_{\pi}(s) \end{aligned}$$

On-policy First-visit MC Control

On-policy first-visit MC control (for ε -soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε -soft policy

$Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \dots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow$ average($Returns(S_t, A_t)$)

$A^* \leftarrow \arg \max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Off-policy Learning

- We want to learn the optimal policy, but we have to account for the problem of *maintaining exploration*
- We call the (optimal) policy to be learned the *target policy* π and the policy used to generate behaviour the *behaviour policy* b
- We say that learning is from data *off* the target policy – thus, those methods are referred to as *off-policy learning*

Importance Sampling

- Weight returns according to the relative probability of target and behaviour policy
- Define state-transition probabilities $p(s'|s, a)$ as
$$p(s'|s, a) = \Pr\{S_t = s' | S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r | s, a)$$
- The probability of the subsequent trajectory under any policy π , starting in S_t , then is:

$$\begin{aligned} & \Pr\{A_t, S_{t+1}, A_{t+1}, \dots, S_T | S_t, A_{t:T-1} \sim \pi\} \\ &= \pi(A_t | S_t) p(S_{t+1} | S_t, A_t) \pi(A_{t+1} | S_{t+1}) \cdots p(S_T | S_{T-1}, A_{T-1}) \\ &= \prod_{k=t}^{T-1} \pi(A_k | S_k) p(S_{k+1} | S_k, A_k) \end{aligned}$$

Importance Sampling

The relative probability therefore is:

Definition: Importance Sampling Ratio

$$\rho_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k) p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k) p(S_{k+1}|S_k, A_k)} = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)}$$

The expectation of the returns by b is $\mathbb{E}[G_t|S_t = s] = v_b(s)$. However, we want to estimate the expectation under π . Given the importance sampling ratio, we can transform the returns by b to yield the expectation under π :

$$\mathbb{E}[\rho_{t:T-1} G_t | S_t = s] = v_\pi(s).$$

Importance Sampling can come with a vast increase in variance.

Off-policy MC Control

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$$Q(s, a) \in \mathbb{R} \text{ (arbitrarily)}$$

$$C(s, a) \leftarrow 0$$

$$\pi(s) \leftarrow \operatorname{argmax}_a Q(s, a) \quad (\text{with ties broken consistently})$$

Loop forever (for each episode):

$b \leftarrow$ any soft policy

Generate an episode using b : $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$$G \leftarrow 0$$

$$W \leftarrow 1$$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$$G \leftarrow \gamma G + R_{t+1}$$

$$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$$

$$\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a) \quad (\text{with ties broken consistently})$$

If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)

$$W \leftarrow W \frac{1}{b(A_t | S_t)}$$

Lecture Overview

1 Recap

2 Monte Carlo Prediction

3 Monte Carlo Control

4 Wrapup

Summary by Learning Goals

Having heard this lecture, you can now...

- describe how to evaluate a given policy with Monte Carlo rollouts
- explain policy improvement by on-policy Monte Carlo control



Lecture 04: Temporal Difference Methods

#cs #rl

83 kez oynandi · 136 oyuncu

Herkese açık bir kahoot

Sorular (8)

1 - Quiz

MC vs TD: What is correct?

60 sn

- | | | |
|--|---|--|
| | MC is more sensitive to initial values than TD. | |
| | TD has a lower variance than MC. | |
| | MC is usually more efficient than TD. | |
| | MC has a larger bias. | |

2 - Quiz

Bootstrapping vs Sampling: What is correct?

60 sn

- | | | |
|--|---------------------------------------|--|
| | MC bootstraps and TD samples | |
| | MC and TD do not bootstrap | |
| | MC samples, but TD does not. | |
| | MC does not bootstrap and TD samples. | |

3 - Quiz

Temporal difference learning is biased because ...

60 sn

- | | | |
|--|---|--|
| | the update uses the expected return of the next state. | |
| | the update uses the estimated return of the next state. | |
| | it bootstraps its estimation of the immediate reward. | |
| | it has a lower variance. | |

4 - Doğru/Yanlış

In contrast to MC Control, SARSA can be easily used in an off-policy setting.

60 sn

- True ✗
- False ✓

5 - Quiz

What is the correct SARSA equation?

60 sn

- 1 ✗
- 2 ✗
- 3 ✗
- 4 ✓

6 - Quiz

Off-policy Learning...

60 sn

- is more sample-efficient, but can be less stable. ✓
- always needs some off-policy correction like Imp. Sam. ✗
- can reuse experience generated from old policies. ✓
- can only learn from older policies of the same agent. ✗

7 - Quiz

Q-Learning: What is correct?

60 sn

- A_t is sampled accordingly to target policy. ✗
- A_t is sampled accordingly to behavior policy. ✓
- A' is sampled accordingly to target policy. ✓
- A' is sampled accordingly to behavior policy. ✗

8 - Quiz

How does Double Q-learning help?

60 sn

- It decorrelates the noise of Q and argmax_a Q. ✓
- It is always underestimating the true Q-value. ✗
- By the estimation of two Q-functions, the noise adds to 0. ✗
- It is always overestimating the true Q-value. ✗

Lecture 5: Temporal-Difference Learning

Friday, November 26, 2021

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg



Lecture Overview

1 Recap

2 TD Prediction

3 TD Control

4 Wrapup

Lecture Overview

1 Recap

2 TD Prediction

3 TD Control

4 Wrapup

Recap: Monte-Carlo Reinforcement Learning

Last lecture: Estimate/ Optimize the value function of an *unknown* MDP using Monte-Carlo Prediction.

- MC methods learn from episodes of *experiences*
- MC is model-free: no knowledge required about MDP dynamics
- MC learns from complete episodes based on averaging sample returns
- First-visit MC method: Estimates $v_\pi(s)$ as the average of the returns following first visits to s .
- Every-visit MC method: Estimates $v_\pi(s)$ as the average of the returns following all visits to s .

Example: Blackjack

Temporal Difference Learning

This lecture: Estimate/ optimize the value function of an *unknown* MDP using Temporal Difference Learning.

- TD is a combination of Monte Carlo and dynamic programming ideas
- Similar to MC methods, TD methods learn directly raw experiences without a dynamic model
- TD learns from *incomplete* episodes by bootstrapping
- Bootstrapping: update estimated based on other estimates without waiting for a final outcome (update a guess towards a guess)

Lecture Overview

1 Recap

2 TD Prediction

3 TD Control

4 Wrapup

Monte Carlo Update

Update value $V(S_t)$ towards the *actual* return G_t .

$$V(s_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

α is a step-size parameter.

Simplest temporal-difference learning algorithm: $TD(0)$

Update value $V(S_t)$ towards the *estimated* return $R_{t+1} + \gamma V(S_{t+1})$.

$$V(s_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

MC Error and TD Error

If V does not change during an episode, then the MC error can be decomposed of consecutive TD errors.

$$\begin{aligned} G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\ &= \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\ &= \delta_t + \gamma\delta_{t+1} + \gamma^2(G_{t+2} - V(S_{t+2})) \\ &= \delta_t + \gamma\delta_{t+1} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(G_T - V(S_T)) \\ &= \delta_t + \gamma\delta_{t+1} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(0 - 0)) \\ &= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k \end{aligned}$$

TD Prediction

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated

Algorithm parameter: step size $\alpha \in (0, 1]$

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

$A \leftarrow$ action given by π for S

 Take action A , observe R, S'

$V(S) \leftarrow V(S) + \alpha [R + \gamma V(S') - V(S)]$

$S \leftarrow S'$

 until S is terminal

Driving Home Example

State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

TD Prediction

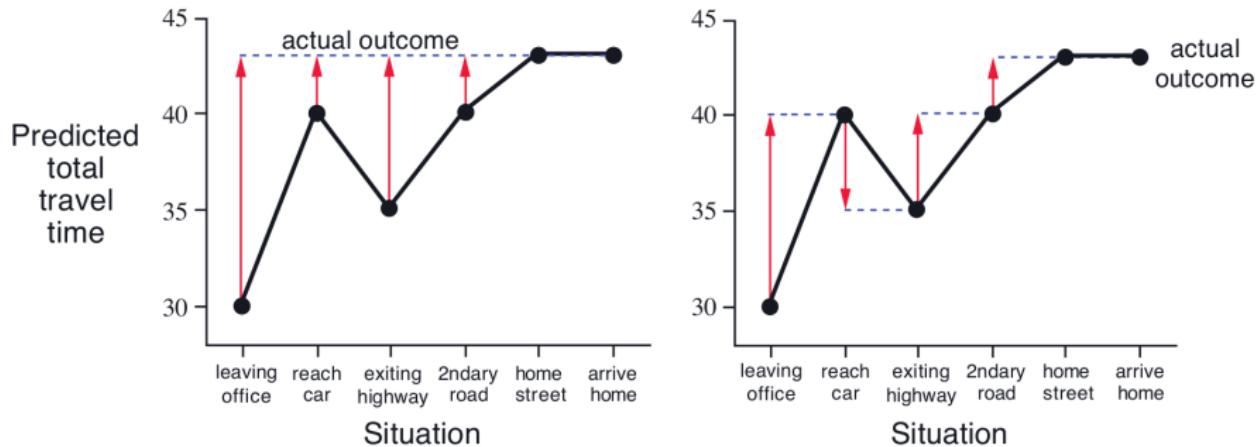


Figure 6.1: Changes recommended in the driving home example by Monte Carlo methods (left) and TD methods (right).

MC vs TD

- TD can learn online after every step, MC has to wait for the final outcome/return
- TD can even learn without ever getting a *final* outcome, which is especially important for infinite horizon tasks
- The return G_t depends on many random actions, transitions and rewards, the TD-target depends on one random action, transition and reward
- Therefore, the TD-target has lower *variance* than the return
- But the TD-target is a *biased* estimate of v_π
- This is known as the bias/variance trade-off

MC vs TD

- MC and TD converge if every state and every action are visited an infinite number of times
- What about finite experience?

Imagine two states, A and B , and the following transitions:

$A,0$	$B,0$	$B,1$
$B,1$		$B,1$
$B,1$		$B,1$
$B,1$		$B,0$

What are the values of A and B given this data?

MC vs TD

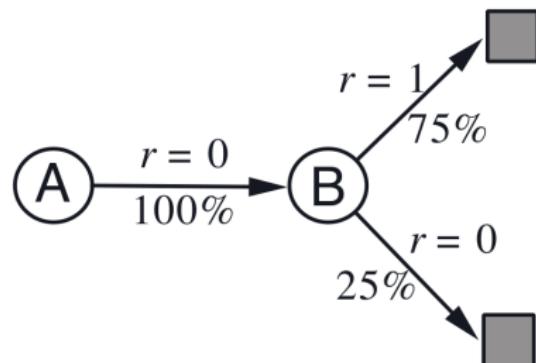
- W.r.t. B , the process terminated immediately 6/8 times with a return of 1, 0 otherwise
- Thus, it is reasonable to assume a value of 0.75
- What about A ?

MC vs TD

- W.r.t. B , the process terminated immediately 6/8 times with a return of 1, 0 otherwise
- Thus, it is reasonable to assume a value of 0.75
- What about A ?

A led in all

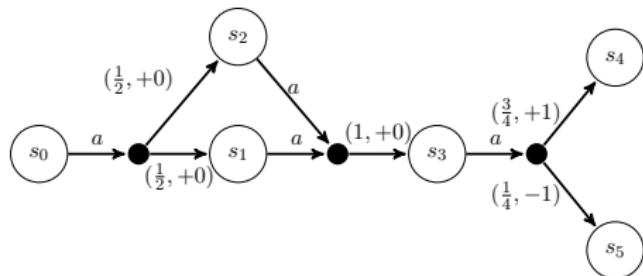
cases to B . Thus, A could have a value of 0.75 as well. This answer is based on first modelling the Markov Process and then computing the values given the model. TD is leading to this value. MC gives a value of 0 – which is also the solution with 0 MSE on the given data. One can assume, however, that the former gives lower error on *future* data.



MC vs TD

- Batch MC converges to the solution with minimum MSE on the observed returns
- Batch TD converges to the solution of the maximum-likelihood Markov model
- Given this model, we can compute the estimate of the value-function that would be exactly correct, if the model were exactly correct
- This is called the *Certainty Equivalence*

MC vs TD

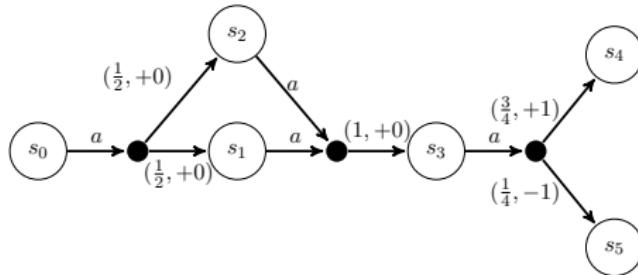


Assume that
the agent encounters the following
set of trajectories at every
iteration i (where $i \bmod 4 = 0$):

$$\begin{aligned} t_i : \quad & s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_4 \\ t_{i+1} : \quad & s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_4 \\ t_{i+2} : \quad & s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_4 \\ t_{i+3} : \quad & s_0 \rightarrow s_2 \rightarrow s_3 \rightarrow s_5 \end{aligned}$$

Question (5 min)

Given these trajectories, explain why TD-learning is better fitted to estimate the value function compared to MC. Assume no discount and that the value function is initialized with zeros. To which value function is MC going to converge, given a suitable learning rate α ? What about TD?



Solution

MC always takes the full return to update its values. Therefore, \$s_1\$ only updates on return \$+1\$, whereas \$s_2\$ only updates on return \$-1\$. TD takes this into account due to bootstrapping. MC converges to:

$v(s_0) = v(s_3) = 0.5$, $v(s_1) = v(s_4) = +1$ and $v(s_2) = v(s_5) = -1$. TD converges to the true value function $v(s_0) = v(s_1) = v(s_2) = v(s_3) = 0.5$ and $v(s_4) = v(s_5) = 0$, since $\frac{3}{4}$ trajectories end with a return of \$+1\$ and $\frac{1}{4}$ with a return of \$-1\$ – which corresponds to the true distribution of the MDP.

Lecture Overview

1 Recap

2 TD Prediction

3 TD Control

4 Wrapup

SARSA

- SARSA: State, Action, Reward, State, Action
- Why is it considered an on-policy method?

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Loop for each step of episode:

 Take action A , observe R, S'

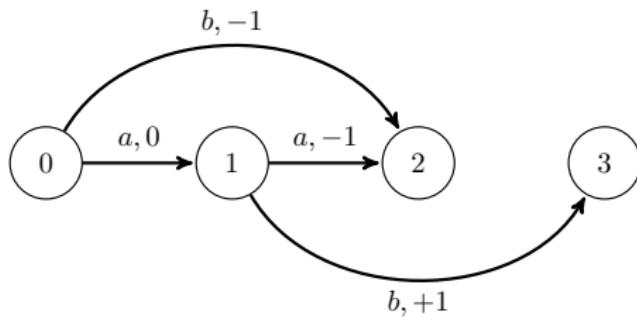
 Choose A' from S' using policy derived from Q (e.g., ε -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal

SARSA Example



$\text{traj}_1 : 0 \rightarrow 1 \rightarrow 2$

$\text{traj}_2 : 0 \rightarrow 1 \rightarrow 3$

$\text{traj}_3 : 0 \rightarrow 1 \rightarrow 2$

Q-learning

- Why is it considered an off-policy method?

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

 Choose A from S using policy derived from Q (e.g., ε -greedy)

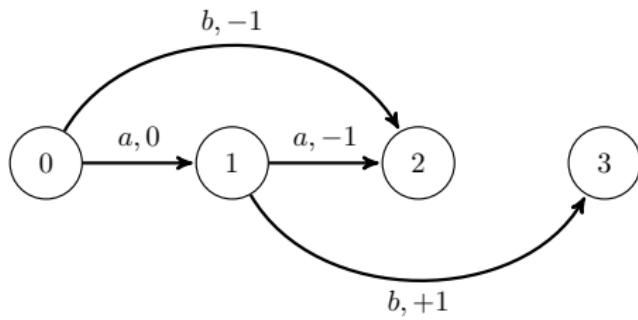
 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$S \leftarrow S'$

 until S is terminal

Q-learning Example



$\text{traj}_1 : 0 \rightarrow 1 \rightarrow 2$

$\text{traj}_2 : 0 \rightarrow 1 \rightarrow 3$

$\text{traj}_3 : 0 \rightarrow 1 \rightarrow 2$

Expected SARSA

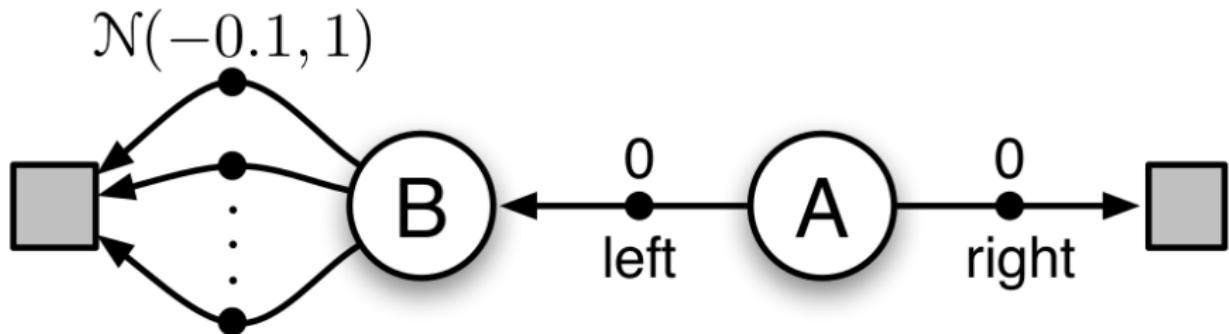
- Similar to Q-learning, but uses the expectation
- *Expected SARSA*-Update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \underbrace{\mathbb{E}_\pi [Q(S_{t+1}, A_{t+1}) | S_{t+1}]}_{\sum_a \pi(a|S_{t+1}) Q(S_{t+1}, A_{t+1})} - Q(S_t, A_t)]$$

- If π is greedy, but behaviour comes from an exploratory policy, then *Expected SARSA* is equivalent to Q-learning and is thus a generalization of this concept

Overestimation Bias

- In all control algorithms so far, the target policy is created by the *maximization* of a value-function
- We thus consider the maximum over estimated values as an estimate of the maximum value
- This can lead to the so-called *overestimation bias*



Overestimation Bias

- Recall the Q-learning target: $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- Imagine two random variables X_1 and X_2 :

$$\mathbb{E}[\max(X_1, X_2)] \geq \max(\mathbb{E}[X_1], \mathbb{E}[X_2])$$

- $Q(S_{t+1}, a)$ is not perfect – it can be *noisy*:

$$\max_a Q(S_{t+1}, a) = \overbrace{Q(S_{t+1}, \arg \max_a Q(S_{t+1}, a))}^{\substack{\text{value comes from } Q \\ a}} \underbrace{\quad}_{\substack{\text{action comes from } Q}}$$

- If the noise in these is decorrelated, the problem goes away!

Double Q-learning

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, such that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

 Choose A from S using the policy ε -greedy in $Q_1 + Q_2$

 Take action A , observe R, S'

 With 0.5 probability:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha \left(R + \gamma Q_2(S', \arg \max_a Q_1(S', a)) - Q_1(S, A) \right)$$

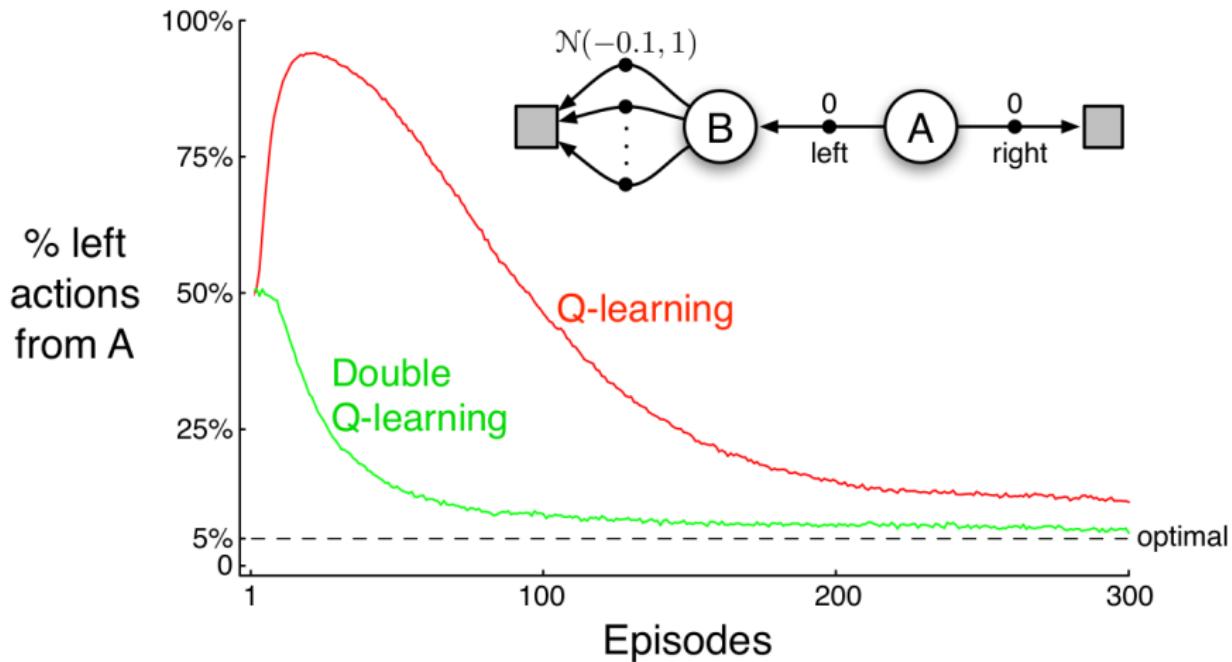
 else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha \left(R + \gamma Q_1(S', \arg \max_a Q_2(S', a)) - Q_2(S, A) \right)$$

$S \leftarrow S'$

until S is terminal

Double Q-learning



Lecture Overview

1 Recap

2 TD Prediction

3 TD Control

4 Wrapup

Summary by Learning Goals

Having heard this lecture, you can now...

- Explain the differences between MC and TD
- Use TD to predict a value function
- Apply on-policy TD-control
- Apply off-policy TD-control
- Explain the overestimation bias



Lecture 05: Planning and Learning

#cs #rl

74 plays · 119 players

A public kahoot

Questions (7)

1 - Quiz

In model-based reinforcement learning,

60 sec

- the MDP of the environment is given. ✗
- we learn a model of the underlying environment. ✓
- we learn a value function from samples from the environment. ✗
- we can use planning on a model to obtain a value function. ✓

2 - True or false

It is always easier to learn a dynamics model than a policy.

60 sec

- True ✗
- False ✓

3 - Quiz

It can be a good choice to learn the state difference rather than the transition to a global state -- why?

60 sec

- The numbers are usually smaller. ✗
- The model suffers less from accumulating errors. ✗
- There can be local similarities w.r.t. the state differences. ✓

4 - Quiz

In sample-based planning, we ...

60 sec

-  we solve the MDP directly. ✗
-  apply planning to the MDP. ✗
-  we apply model-free RL to sampled experience. ✓
-  suffer less from the curse of dimensionality. ✓

5 - Quiz

In Dyna, we learn the value function/ policy from

60 sec

-  samples from the learned model. ✗
-  samples of the real environment. ✗
-  imaginations of the real world. ✗
-  samples from the learned model and the real environment. ✓

6 - Quiz

In Prioritized Sweeping, we update (s,a)-pairs according to ...

60 sec

-  their absolute Q-value. ✗
-  their absolute TD-error. ✓
-  the number of states that lead to them. ✗
-  their negative distance to the goal. ✗

7 - Quiz

In monte carlo tree search (MCTS), we ...

60 sec

- combine in-tree policies and out-of-tree policies. ✓
- traverse the tree randomly to obtain MC simulations. ✗
- use a greedy policy as an in-tree policy. ✗
- use a greedy policy as an out-of-tree policy. ✗

Lecture 6: Planning and Learning

Friday, December 3, 2021

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg



Lecture Overview

- 1 Recap
- 2 Model Learning
- 3 Dyna
- 4 Simulation-based Search
- 5 Wrapup

Lecture Overview

1 Recap

2 Model Learning

3 Dyna

4 Simulation-based Search

5 Wrapup

Recap

- TD is a combination of Monte Carlo and dynamic programming ideas
- Similar to MC methods, TD methods learn directly raw experiences without a dynamic model
- TD learns from *incomplete* episodes by bootstrapping
- Bootstrapping: update estimated based on other estimates without waiting for a final outcome (update a guess towards a guess)

Simplest temporal-difference learning algorithm: $TD(0)$

Update value $V(S_t)$ towards the *estimated* return $R_{t+1} + \gamma V(S_{t+1})$.

$$V(s_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

Lecture Overview

1 Recap

2 Model Learning

3 Dyna

4 Simulation-based Search

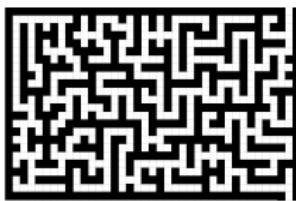
5 Wrapup

Components of RL Systems

- Policy: defines the behaviour of the agent
 - is a mapping from a state to an action
 - can be stochastic: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$
 - or deterministic: $\pi(s) = a$
- Value-function: defines the expected value of a state or an action
 - $v_\pi(s) = \mathbb{E}[G_t|S_t = s]$ and $q_\pi(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a]$
 - can be used to evaluate states or to extract a good policy
- Model: defines the transitions between states in an environment
 - p yields the next state and reward
 - $p(s', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a\}$

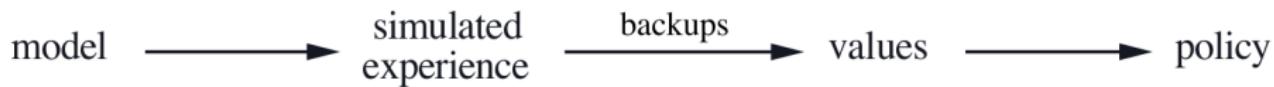
Learning Models

- Depending on the task, the dynamics model can be much easier than the value-function or the policy
- We can estimate it via supervised learning methods
- However, the model can also be more complex than policy and value-function
- In practice, modelling state-changes can even be easier than the global state
- In a nutshell:
 - Learning a model: data-efficient, hard to extract an optimal policy
 - Learning a value function: less data-efficient, easier to extract an optimal policy
 - Learning a policy: data-inefficient, directly estimate an optimal policy



Models and Planning

- Given a state and an action, a model generates the next state and the corresponding reward (can also be used to generate sequences of states and rewards)
- It can either give the probabilities of all possible next states and rewards (*distribution model*), or only one (*sample model*)
- Which one was used in Dynamic Programming?
- Extracting a policy from a model is called *planning*
- Here: *state-space planning*



Models and Planning

- Planning: Uses simulated experience generated by a model
- Learning: Uses real experiences from the environment
- But we can also apply learning methods to simulated experience

Random-sample one-step tabular Q-planning

Loop forever:

1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send S, A to a sample model, and obtain
a sample next reward, R , and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S' :
$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

- Converges to the optimal policy *for the model*

Lecture Overview

1 Recap

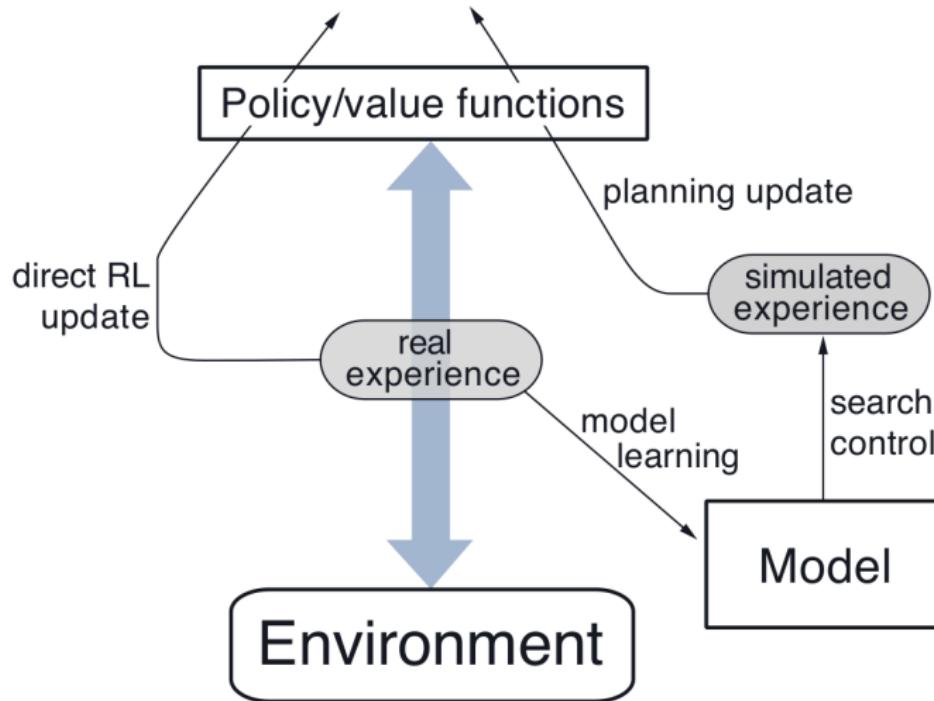
2 Model Learning

3 Dyna

4 Simulation-based Search

5 Wrapup

- Real experience can be used to optimize the value function (or the policy)
 - *directly* (model-free RL) or
 - *indirectly* (model-based RL) via a model
- *Indirect methods* are often more data-efficient
- But they introduce additional bias through the model
- Idea of Dyna: try to combine the best of both worlds



Dyna

Tabular Dyna-Q

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Loop forever:

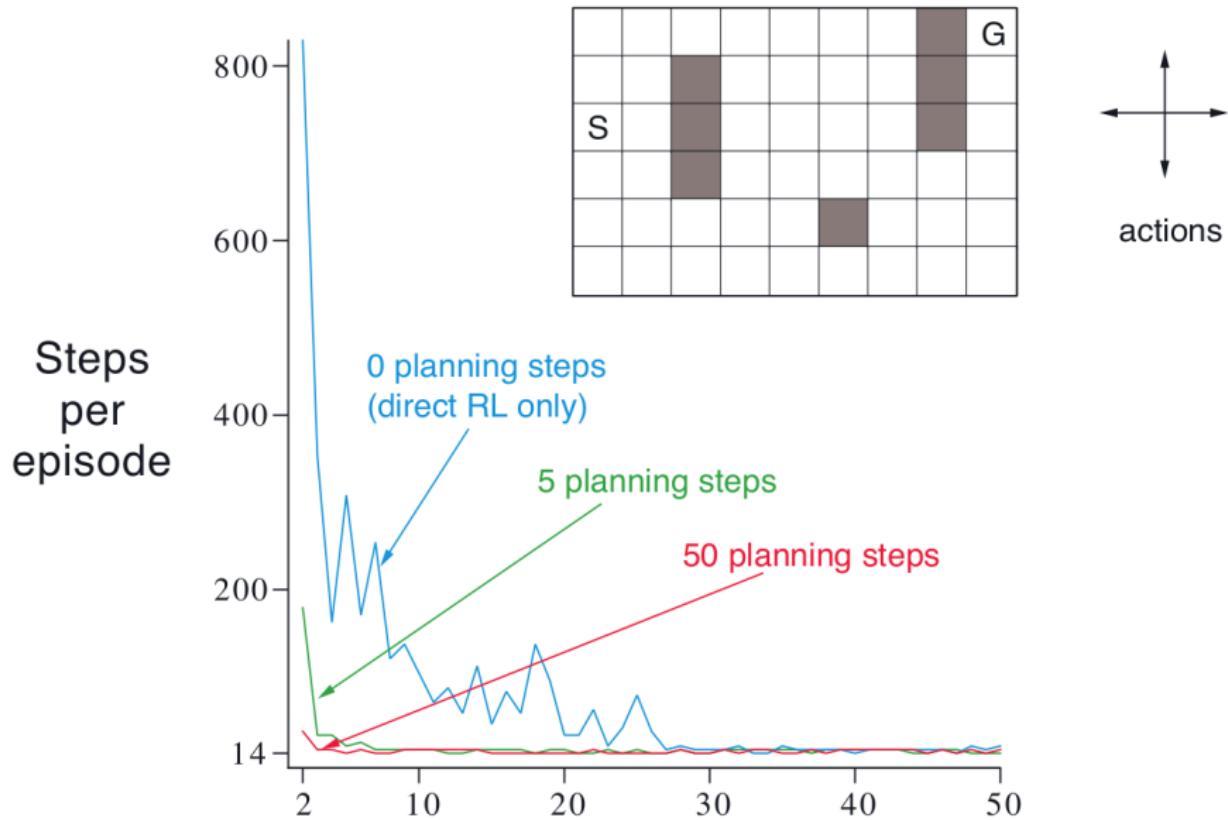
- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \varepsilon\text{-greedy}(S, Q)$
- (c) Take action A ; observe resultant reward, R , and state, S'
- (d) $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
- (e) $Model(S, A) \leftarrow R, S'$ (assuming deterministic environment)
- (f) Loop repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S, A)$

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$



When the model is wrong

- Models can be incorrect (limited number of samples, environment has changed, function approximation)
- Especially in areas where the agent has not explored
- There can be a *Distribution Mismatch* when the agent enters new areas of the state-action space
- When the model is incorrect, the planning process is likely to find a suboptimal policy

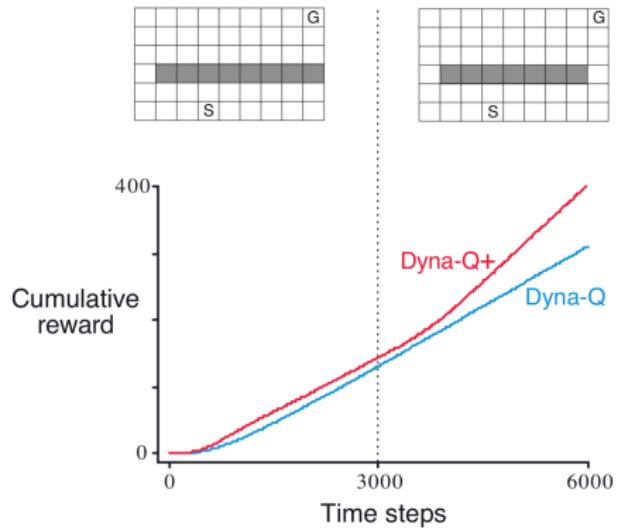
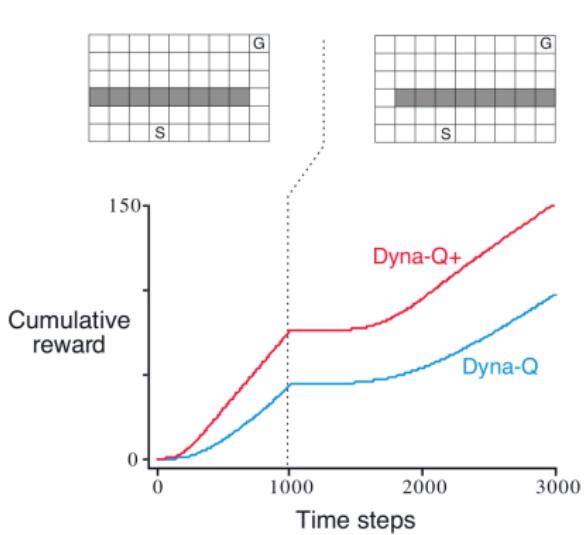
When the model is wrong

- Dyna-Q+: Add an exploration bonus for transitions that have not been visited recently
- Let r be the reward, κ the weight of the exploration bonus and τ the number of time steps in which a certain transition has not been visited
- Then Dyna-Q+ modifies the internal reward function to:

$$r + \kappa\sqrt{\tau}$$

- To which exploration technique from Lecture 01 (Bandits) does this share great similarity?

When the model is wrong



Prioritized Sweeping

- Update action-values for state-action pairs with high priority
- Here: we want to work back from states whose values have changed

Prioritized sweeping for a deterministic environment

Initialize $Q(s, a)$, $Model(s, a)$, for all s, a , and $PQueue$ to empty

Loop forever:

- $S \leftarrow$ current (nonterminal) state
- $A \leftarrow policy(S, Q)$
- Take action A ; observe resultant reward, R , and state, S'
- $Model(S, A) \leftarrow R, S'$
- $P \leftarrow |R + \gamma \max_a Q(S', a) - Q(S, A)|$.
- if $P > \theta$, then insert S, A into $PQueue$ with priority P
- Loop repeat n times, while $PQueue$ is not empty:

$S, A \leftarrow first(PQueue)$

$R, S' \leftarrow Model(S, A)$

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

Loop for all \bar{S}, \bar{A} predicted to lead to S :

$\bar{R} \leftarrow$ predicted reward for \bar{S}, \bar{A}, S

$P \leftarrow |\bar{R} + \gamma \max_a Q(S, a) - Q(\bar{S}, \bar{A})|$.

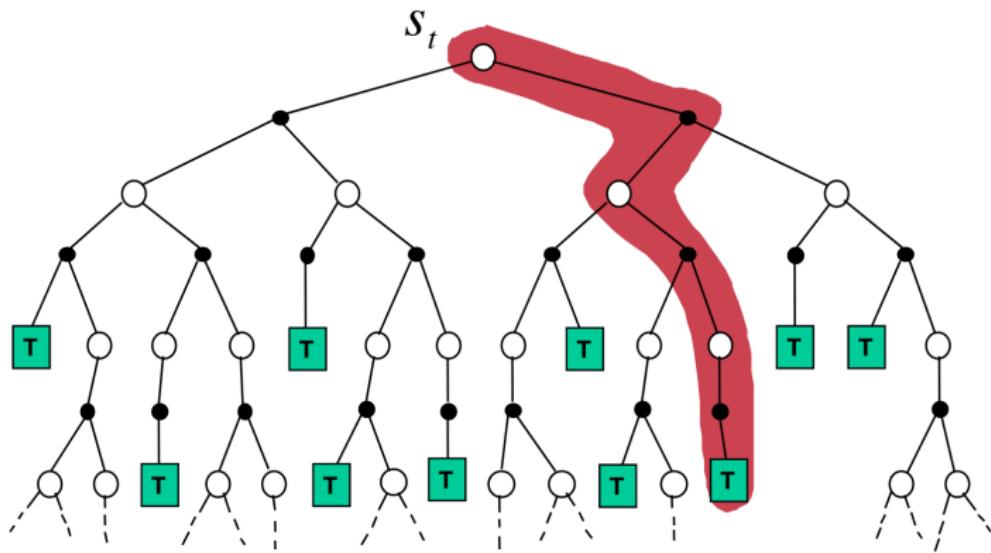
if $P > \theta$ then insert \bar{S}, \bar{A} into $PQueue$ with priority P

Lecture Overview

- 1 Recap
- 2 Model Learning
- 3 Dyna
- 4 Simulation-based Search
- 5 Wrapup

Simulation-based Search

- Forward search paradigm using sample-based planning
- Simulate episodes of experience from now with the model
- Apply model-free RL to simulated episodes



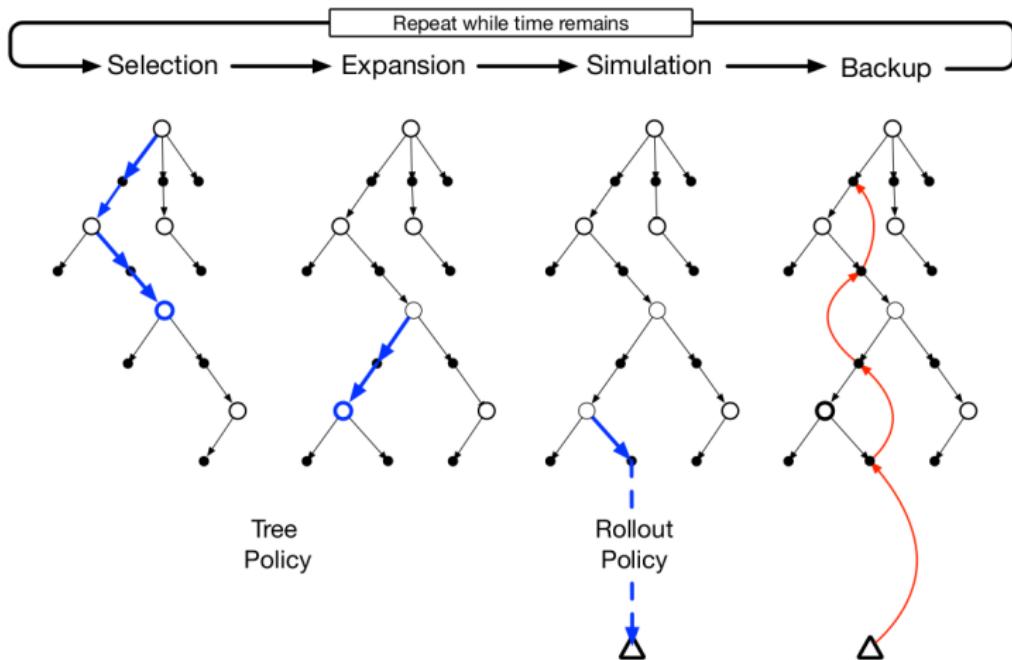
Monte Carlo Tree Search

- Build a search tree containing visited states and actions using the model (simulate episodes from current state)
- Two policies: tree policy (improving, e.g. ϵ -greedy) and out-of-tree rollout policy (random)
- Monte-Carlo control applied to simulated experience
- One of the key ingredients of AlphaGo (2016)

Monte Carlo Tree Search

- ① Selection: starting at the root, traverse to a leaf node following the *tree policy*
- ② Expansion: expand the tree by one or multiple child nodes reached from the selected leaf in some iterations
- ③ Simulation: simulate an episode following the *rollout policy*
- ④ Backup: update the action-values for all nodes visited **in the tree**

Monte Carlo Tree Search



Temporal-Difference Search

- Simulation-based search
- Using TD instead of MC
- MCTS applies MC control to sub-MDP from now
- TD search applies SARSA to sub-MDP from now

Lecture Overview

1 Recap

2 Model Learning

3 Dyna

4 Simulation-based Search

5 Wrapup

Summary by Learning Goals

Having heard this lecture, you can now...

- Explain the difference between model-based and model-free RL
- Explain how to make use of models in RL
- Explain the Dyna-architecture
- Explain simulation-based forward search and its variants MCTS and TD Search



Lecture 06: On-Policy Prediction and Control with Function Approximation

Lecture 05: Value Function Approximation

76 plays · 119 players

A public kahoot

Questions (6)

1 - Quiz

We introduce function approximators, because...

60 sec

- they are more stable to train. ✗
- we want to generalize over similar states and actions. ✓
- they converge to better value functions. ✗
- they can cope with very large state and action spaces. ✓

2 - Quiz

Differentiable function approximators are often preferred, because...

60 sec

- others do not work. ✗
- they are guaranteed to find the global optimum. ✗
- the gradients yield the influence of parameters. ✓
- we have to handle non-stationary data. ✗

3 - True or false

In contrast to gradient MC prediction, on-policy linear semi-gradient TD is not guaranteed to converge.

30 sec

- True ✗
- False ✓

4 - Quiz

When can we directly calculate the least squares solution?

30 sec

-  Linear combination of non-linear features. ✓
 -  Non-linear combination of linear features. ✗
-

5 - Quiz

What do we estimate in semi-gradient SARSA via function approximation?

60 sec

-  The state-value function. ✗
 -  The policy. ✗
 -  The action-value function. ✓
 -  The model. ✗
-

6 - True or false

Similar to tabular methods, memory-based function approximation does not generalize.

30 sec

-  True ✗
-  False ✓

Lecture 7: On-policy Prediction and Control with Function Approximation

Friday, December 10, 2021

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg



Lecture Overview

- 1 Recap
- 2 Function Approximation in Reinforcement Learning
- 3 Linear Methods
- 4 Memory-based Function Approximation
- 5 On-policy Control with Function Approximation
- 6 Organization
- 7 Wrapup

Lecture Overview

1 Recap

2 Function Approximation in Reinforcement Learning

3 Linear Methods

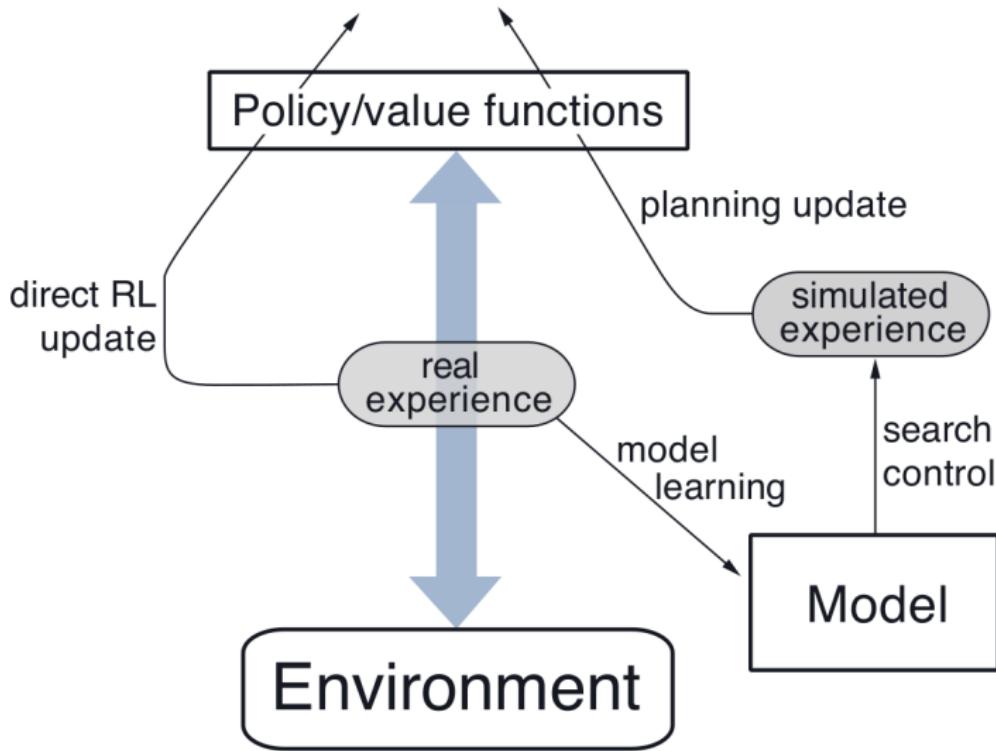
4 Memory-based Function Approximation

5 On-policy Control with Function Approximation

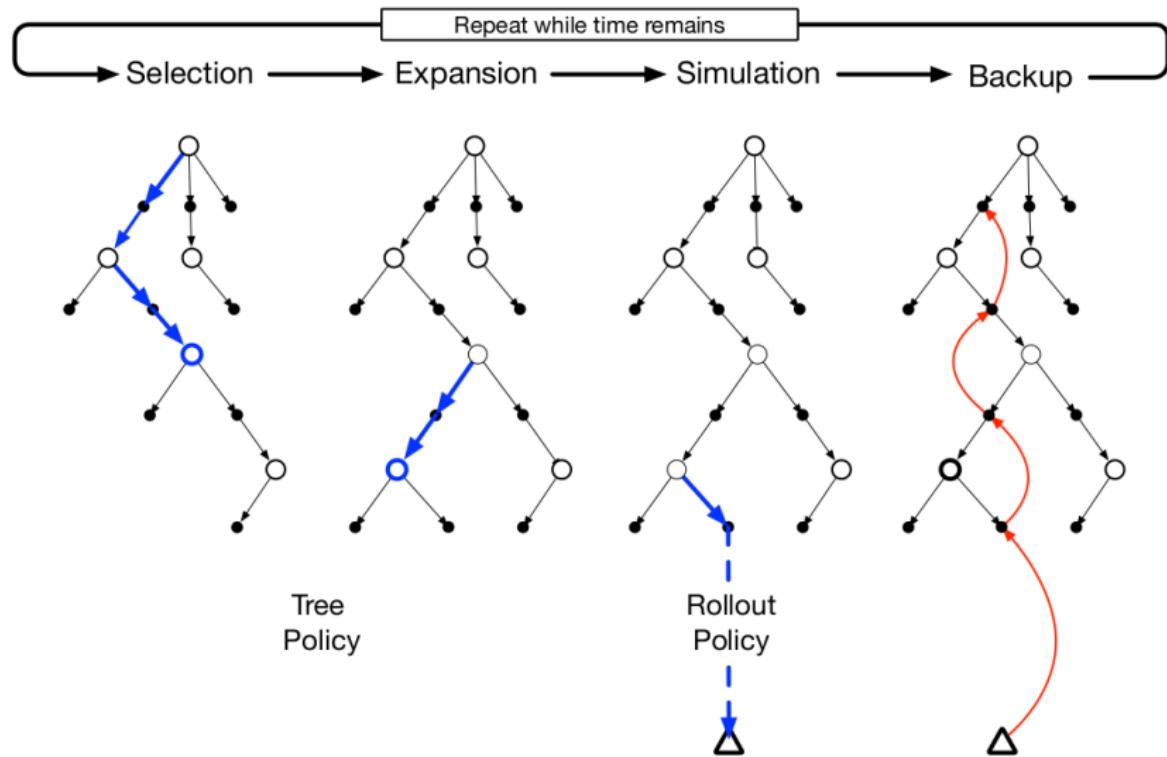
6 Organization

7 Wrapup

Recap: Dyna



Recap: MCTS



Lecture Overview

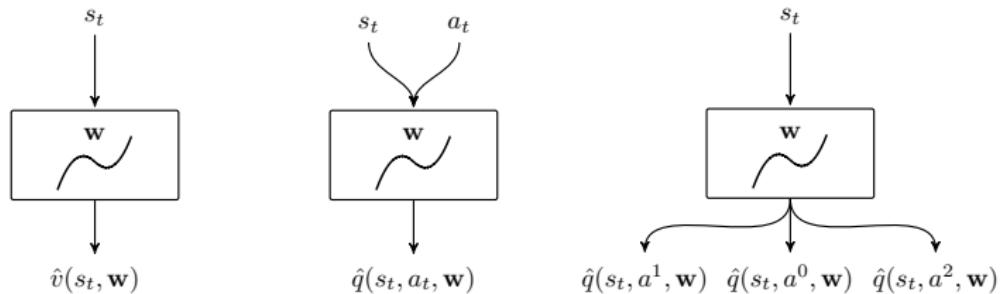
- 1 Recap
- 2 Function Approximation in Reinforcement Learning
- 3 Linear Methods
- 4 Memory-based Function Approximation
- 5 On-policy Control with Function Approximation
- 6 Organization
- 7 Wrapup

Function Approximation in Reinforcement Learning

- Up to this point, we represented all elements of our RL systems by tables (value functions, models and policies)
- If the state and action spaces are very large or infinite, this is not a feasible solution
- We can apply function approximation to find a more compact representation of RL components and to generalize over states and actions
- Reinforcement Learning with function approximation comes with new issues that do not arise in Supervised Learning – such as non-stationarity, bootstrapping and delayed targets

Function Approximation in Reinforcement Learning

- Here: we estimate value-functions $v_\pi(\cdot)$ and $q_\pi(\cdot, \cdot)$ by function approximators $\hat{v}(\cdot, \mathbf{w})$ and $\hat{q}(\cdot, \cdot, \mathbf{w})$, parameterized by weights \mathbf{w}



- But we can also represent models or policies

Function Approximation in Reinforcement Learning

We can use different types of function approximators:

- Linear combinations of features
- Neural networks
- Decision trees
- Gaussian processes
- Nearest neighbor methods
- ...

Here: We focus on differentiable FAs and update the weights via gradient descent.

Function Approximation in Reinforcement Learning

We want to update our weights w.r.t. the *Mean Squared Value Error* of our prediction:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)]^2 \\ &= \mathbf{w}_t + \alpha[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)]\nabla\hat{v}(S_t, \mathbf{w}_t)\end{aligned}$$

However, we don't have $v_\pi(S_t)$.

Function Approximation in Reinforcement Learning

Gradient MC

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [G_t - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})$$

Semi-gradient TD(0)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})$$

Why are bootstrapping methods, defined this way, called *semi-gradient methods*?

Function Approximation in Reinforcement Learning

Gradient MC

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [G_t - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})$$

Semi-gradient TD(0)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})$$

Why are bootstrapping methods, defined this way, called *semi-gradient methods*?

They take into account the effects of changing \mathbf{w} w.r.t. the prediction, but not w.r.t. the target!

Lecture Overview

- 1 Recap
- 2 Function Approximation in Reinforcement Learning
- 3 Linear Methods
- 4 Memory-based Function Approximation
- 5 On-policy Control with Function Approximation
- 6 Organization
- 7 Wrapup

Linear Methods

- Represent state s by feature vector $\mathbf{x}(s) = (x^1(s), x^2(s), \dots, x^d(s))^\top$
- These features can also be non-linear functions/combinations of state dimensions
- Linear methods approximate the value function by a linear combination of these features

$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s) = \sum_{i=1}^d w^i x^i(s)$$

- Therefore, $\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w}) = \mathbf{x}(s)$
- Gradient MC prediction converges under linear FA
- What about TD?

Proof of Convergence of on-policy linear semi-gradient TD

- The update at each time step t is:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha \left(R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \right) \mathbf{x}_t \\ &= \mathbf{w}_t + \alpha \left(R_{t+1} \mathbf{x}_t - \mathbf{x}_t (\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top \mathbf{w}_t \right)\end{aligned}$$

- The expected next weight vector can thus be written:

$$\mathbb{E}[\mathbf{w}_{t+1} | \mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t),$$

where $\mathbf{b} = \mathbb{E}[R_{t+1} \mathbf{x}_t]$ and $\mathbf{A} = \mathbb{E}[\mathbf{x}_t (\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top]$

- If the system converges, it has to converge to the *fixed point*:

$$\mathbf{w}_{\text{TD}} = \mathbf{A}^{-1} \mathbf{b}$$

Proof of Convergence of on-policy linear semi-gradient TD

- Rewrite the expected update: $\mathbb{E}[\mathbf{w}_{t+1} | \mathbf{w}_t] = (\mathbf{I} - \alpha \mathbf{A})\mathbf{w}_t + \alpha \mathbf{b}$
- It can be seen that \mathbf{b} is not important to convergence, only \mathbf{A}
- \mathbf{w}_t will be reduced towards 0 whenever \mathbf{A} is positive definite
- \mathbf{A} is defined as:

$$\begin{aligned}\mathbf{A} &= \mathbb{E}[\mathbf{x}_t(\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top] \\ &= \sum_s \mu(s) \sum_a \pi(a|s) \sum_{r,s'} p(r, s'|s, a) \mathbf{x}(s) (\mathbf{x}(s) - \gamma \mathbf{x}(s'))^\top \\ &= \sum_s \mu(s) \sum_{s'} p(s'|s) \mathbf{x}(s) (\mathbf{x}(s) - \gamma \mathbf{x}(s'))^\top \\ &= \sum_s \mu(s) \mathbf{x}(s) (\mathbf{x}(s) - \gamma \sum_{s'} p(s'|s) \mathbf{x}(s'))^\top \\ &= \mathbf{X}^\top \mathbf{D}(\mathbf{I} - \gamma \mathbf{P}) \mathbf{X}\end{aligned}$$

- We thus have to analyze $\mathbf{D}(\mathbf{I} - \gamma \mathbf{P})$ to determine the positive definiteness of \mathbf{A}

Proof of Convergence of on-policy linear semi-gradient TD

- The diagonal entries of $\mathbf{D}(\mathbf{I} - \gamma\mathbf{P})$ are positive and the off-diagonal entries are negative
- We have to show that the row and corresponding column sums are positive
- The row sums are positive, since \mathbf{P} is a stochastic matrix and $\gamma < 1$
- The column sums are:

$$\begin{aligned}\mathbf{1}^\top \mathbf{D}(\mathbf{I} - \gamma\mathbf{P}) &= \boldsymbol{\mu}^\top (\mathbf{I} - \gamma\mathbf{P}) \\ &= \boldsymbol{\mu}^\top - \gamma \boldsymbol{\mu}^\top \mathbf{P} \\ &= \boldsymbol{\mu}^\top - \gamma \boldsymbol{\mu}^\top \\ &= (1 - \gamma)\boldsymbol{\mu}^\top\end{aligned}$$

- It follows that \mathbf{A} is positive definite and thus on-policy linear semi-gradient TD(0) is stable
- Unfortunately, this does not hold for non-linear FA

Least Squares TD

- Recall the *fixed point*: $\mathbf{w}_{\text{TD}} = \mathbf{A}^{-1}\mathbf{b}$
- Why don't we calculate \mathbf{A} and \mathbf{b} directly?
- LSTD does exactly that:

$$\hat{\mathbf{A}}_t = \sum_{k=0}^{t-1} \mathbf{x}_k (\mathbf{x}_k - \gamma \mathbf{x}_{k+1})^\top + \varepsilon \mathbf{I} \text{ and } \hat{\mathbf{b}}_t = \sum_{k=0}^{t-1} R_{k+1} \mathbf{x}_k$$

- LSTD is more data-efficient, but also has quadratic runtime (compared to semi-gradient TD(0) – which is linear)

Least Squares TD

LSTD for estimating $\hat{v} = \mathbf{w}^\top \mathbf{x}(\cdot) \approx v_\pi$ ($O(d^2)$ version)

Input: feature representation $\mathbf{x} : \mathcal{S}^+ \rightarrow \mathbb{R}^d$ such that $\mathbf{x}(\text{terminal}) = \mathbf{0}$

Algorithm parameter: small $\varepsilon > 0$

$$\widehat{\mathbf{A}^{-1}} \leftarrow \varepsilon^{-1} \mathbf{I}$$

A $d \times d$ matrix

$$\widehat{\mathbf{b}} \leftarrow \mathbf{0}$$

A d -dimensional vector

Loop for each episode:

Initialize S ; $\mathbf{x} \leftarrow \mathbf{x}(S)$

Loop for each step of episode:

Choose and take action $A \sim \pi(\cdot | S)$, observe R, S' ; $\mathbf{x}' \leftarrow \mathbf{x}(S')$

$$\mathbf{v} \leftarrow \widehat{\mathbf{A}^{-1}}^\top (\mathbf{x} - \gamma \mathbf{x}')$$

$$\widehat{\mathbf{A}^{-1}} \leftarrow \widehat{\mathbf{A}^{-1}} - (\widehat{\mathbf{A}^{-1}} \mathbf{x}) \mathbf{v}^\top / (1 + \mathbf{v}^\top \mathbf{x})$$

$$\widehat{\mathbf{b}} \leftarrow \widehat{\mathbf{b}} + R \mathbf{x}$$

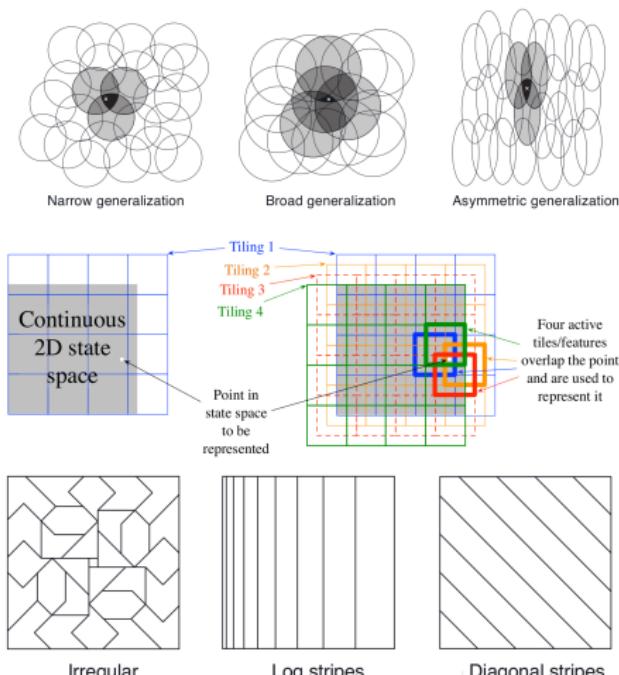
$$\mathbf{w} \leftarrow \widehat{\mathbf{A}^{-1}} \widehat{\mathbf{b}}$$

$$S \leftarrow S'; \mathbf{x} \leftarrow \mathbf{x}'$$

until S' is terminal

Coarse Coding

Divide the state space in circles/tiles/shapes and check in which some state is inside. This is a binary representation of the location of a state and leads to generalization.



Lecture Overview

- 1 Recap
- 2 Function Approximation in Reinforcement Learning
- 3 Linear Methods
- 4 Memory-based Function Approximation
- 5 On-policy Control with Function Approximation
- 6 Organization
- 7 Wrapup

Memory-based Function Approximation

- So far, we discussed the parametric approach to represent value functions
- Memory-based methods simply store collected examples and their values in memory and retrieve samples in order to estimate the value for a query state
- The simplest examples are the nearest neighbor method or the weighted average method over a subset of nearest neighbors
- Similarity between states can be defined by a *kernel* $k(s, s')$
- The value of a query state then is

$$\hat{v}(s, \mathcal{D}) = \sum_{s' \in \mathcal{D}} k(s, s') g(s'),$$

where $g(s')$ is the stored value of s'

Lecture Overview

- 1 Recap
- 2 Function Approximation in Reinforcement Learning
- 3 Linear Methods
- 4 Memory-based Function Approximation
- 5 On-policy Control with Function Approximation
- 6 Organization
- 7 Wrapup

On-policy Control with Function Approximation

- Again, up to this point we discussed Policy Evaluation based on state value functions
- In order to apply FA in control, we parameterize the action-value function

Semi-gradient SARSA

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})] \nabla \hat{q}(S_t, A_t, \mathbf{w})$$

Semi-gradient SARSA

This is your exercise (with neural networks as FA)

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$

Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:

$S, A \leftarrow$ initial state and action of episode (e.g., ε -greedy)

Loop for each step of episode:

Take action A , observe R, S'

If S' is terminal:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R - \hat{q}(S, A, \mathbf{w})] \nabla \hat{q}(S, A, \mathbf{w})$$

Go to next episode

Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε -greedy)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R + \gamma \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})] \nabla \hat{q}(S, A, \mathbf{w})$$

$$S \leftarrow S'$$

$$A \leftarrow A'$$

Semi-gradient SARSA

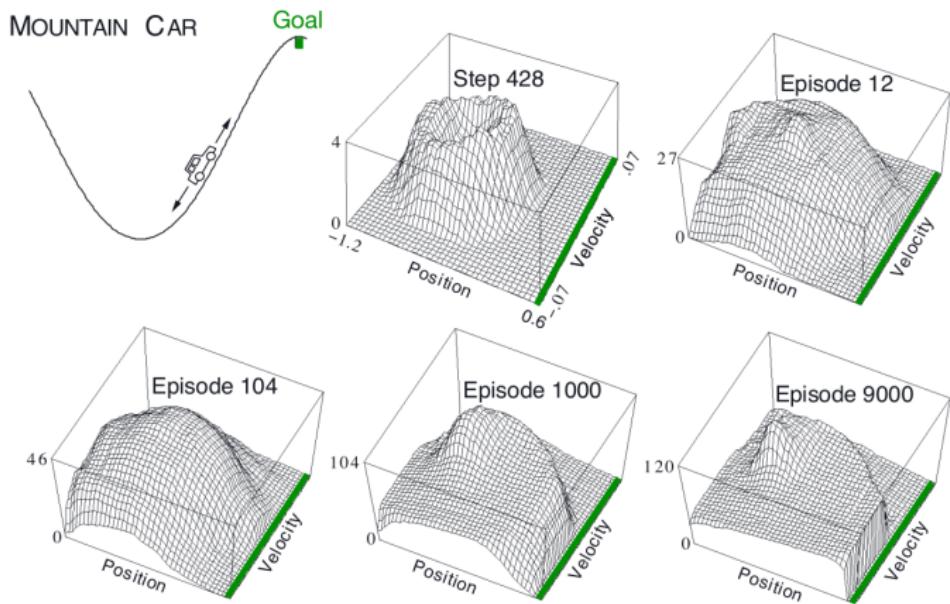
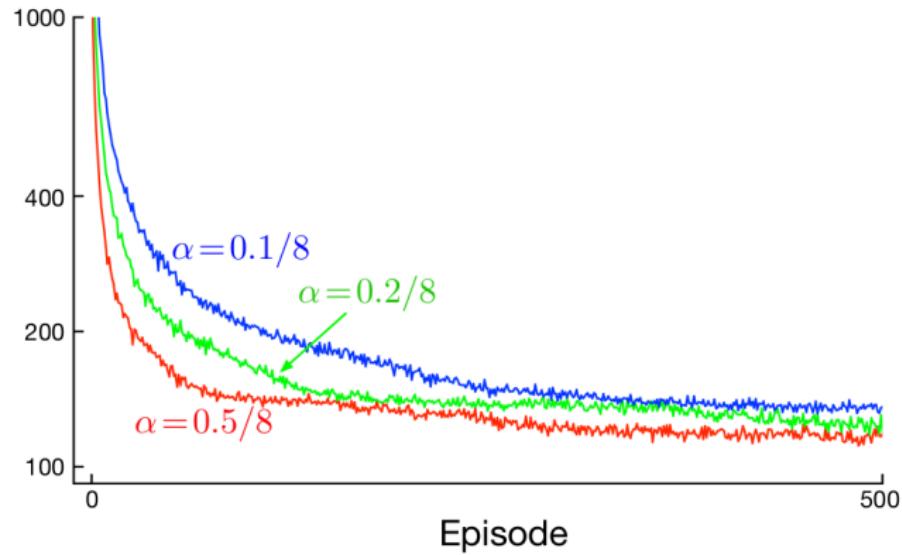


Figure 10.1: The Mountain Car task (upper left panel) and the cost-to-go function ($-\max_a \hat{q}(s, a, w)$) learned during one run.

Semi-gradient SARSA

Mountain Car
Steps per episode
log scale
averaged over 100 runs



Lecture Overview

- 1 Recap
- 2 Function Approximation in Reinforcement Learning
- 3 Linear Methods
- 4 Memory-based Function Approximation
- 5 On-policy Control with Function Approximation
- 6 Organization
- 7 Wrapup

Organization

Our exercises and solutions will be based on PyTorch:
<https://pytorch.org/get-started/locally/>

START LOCALLY

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, 1.3 builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also **install previous versions of PyTorch**. Note that LibTorch is only available for C++.

PyTorch Build	Stable (1.3)		Preview (Nightly)		
Your OS	Linux		Mac	Windows	
Package	Conda	Pip	LibTorch		Source
Language	Python 2.7	Python 3.5	Python 3.6	Python 3.7	C++
CUDA	9.2	10.1		None	
Run this Command:	<code>pip3 install torch torchvision</code>				

Lecture Overview

- 1 Recap
- 2 Function Approximation in Reinforcement Learning
- 3 Linear Methods
- 4 Memory-based Function Approximation
- 5 On-policy Control with Function Approximation
- 6 Organization
- 7 Wrapup

Summary by Learning Goals

Having heard this lecture, you can now...

- explain which parts we usually represent via FA in RL systems and why
- explain how to update the parameters of a differentiable FA via gradient descent in MC and TD algorithms
- apply FAs in on-policy control



Lecture 07: Off-policy Methods with Function Approximation

74 plays · 108 players

A public kahoot

Questions (5)

1 - Quiz

What is not part of the Deadly Triad?

60 sec

- Off-policy Learning ✗
- Bootstrapping ✗
- Infinite Control Problems ✓
- Function Approximation ✗

2 - Quiz

What is not true?

60 sec

- NFQ is a full-batch and DQN a minibatch approach. ✗
- NFQ makes full use of old experience, DQN does not. ✓
- DQN calculates its target based on distinct target networks. ✗
- NFQ reinitializes the weights between iterations. ✗

3 - Quiz

Why do we use target networks in DQN?

20 sec

- To fix the otherwise moving target. ✓
- To transform TD-learning more to a regression. ✓
- To get the true Q-target. ✗
- To speed up the learning process. ✗

4 - True or false

Off-policy Non-linear Gradient TD is guaranteed to converge.

20 sec

- | | |
|--|---|
|  True |  |
|  False |  |

5 - True or false

It is enough to evaluate a common Deep Q-learning algorithm once.

20 sec

- | | |
|--|---|
|  True |  |
|  False |  |

Lecture 8: Off-policy Methods with Function Approximation

Friday, December 17, 2021

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg



Lecture Overview

- 1 Recap
- 2 Off-policy Learning with Function Approximation
- 3 Problems of Off-policy Learning with Function Approximation
- 4 Gradient-TD Methods
- 5 Deep Q-learning
- 6 Wrapup

Lecture Overview

- 1 Recap
- 2 Off-policy Learning with Function Approximation
- 3 Problems of Off-policy Learning with Function Approximation
- 4 Gradient-TD Methods
- 5 Deep Q-learning
- 6 Wrapup

Recap: Function Approximation in Reinforcement Learning

We want to update the weights w.r.t. the *Mean Squared Value Error* of the prediction:

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \frac{1}{2} \alpha \nabla [v_\pi(S_t) - \hat{v}(S_t, \mathbf{w})]^2 \\ &\leftarrow \mathbf{w} + \alpha [v_\pi(S_t) - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})\end{aligned}$$

However, we don't have $v_\pi(S_t)$.

Recap: Function Approximation in Reinforcement Learning

Gradient MC

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[\mathcal{G}_t - \hat{v}(S_t, \mathbf{w})]\nabla\hat{v}(S_t, \mathbf{w})$$

Semi-gradient TD(0)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[\mathcal{R}_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})]\nabla\hat{v}(S_t, \mathbf{w})$$

Why are bootstrapping methods, defined this way, called *semi-gradient methods*? They take into account the effects of changing \mathbf{w} w.r.t. the prediction, but not w.r.t. the target!

Recap: Semi-gradient SARSA

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$

Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:

$S, A \leftarrow$ initial state and action of episode (e.g., ε -greedy)

Loop for each step of episode:

Take action A , observe R, S'

If S' is terminal:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R - \hat{q}(S, A, \mathbf{w})] \nabla \hat{q}(S, A, \mathbf{w})$$

Go to next episode

Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε -greedy)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R + \gamma \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})] \nabla \hat{q}(S, A, \mathbf{w})$$

$$S \leftarrow S'$$

$$A \leftarrow A'$$

Lecture Overview

- 1 Recap
- 2 Off-policy Learning with Function Approximation
- 3 Problems of Off-policy Learning with Function Approximation
- 4 Gradient-TD Methods
- 5 Deep Q-learning
- 6 Wrapup

Off-policy Learning

- We want to learn the optimal policy, but we have to account for the problem of *maintaining exploration*
- We call the (optimal) policy to be learned the *target policy* π and the policy used to generate behaviour the *behaviour policy* b
- We say that learning is from data *off* the target policy – thus, those methods are referred to as *off-policy learning*
- Today: Off-policy learning methods with function approximation

Semi-gradient Off-policy TD(0)

Replace the update to an array to an update to weight vector \mathbf{w} .

Recap: Importance Sampling Ratio

$$\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

Semi-gradient Off-policy TD(0)

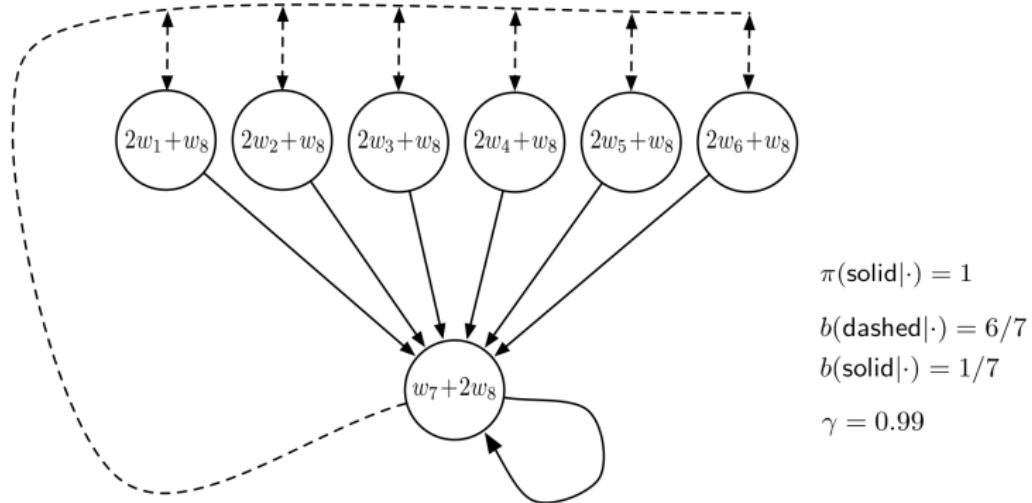
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \rho_t \delta_t \nabla \hat{v}(S_t, \mathbf{w})$$

$$\text{where } \delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})$$

Lecture Overview

- 1 Recap
- 2 Off-policy Learning with Function Approximation
- 3 Problems of Off-policy Learning with Function Approximation
- 4 Gradient-TD Methods
- 5 Deep Q-learning
- 6 Wrapup

Baird's Counterexample

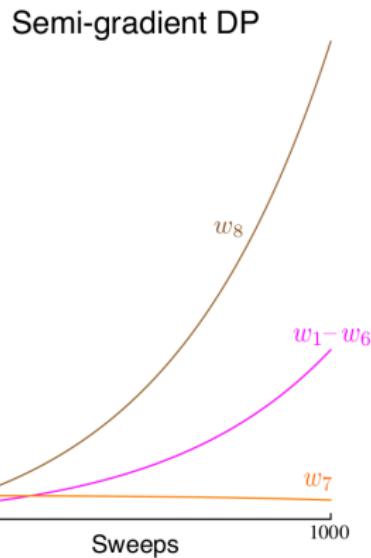
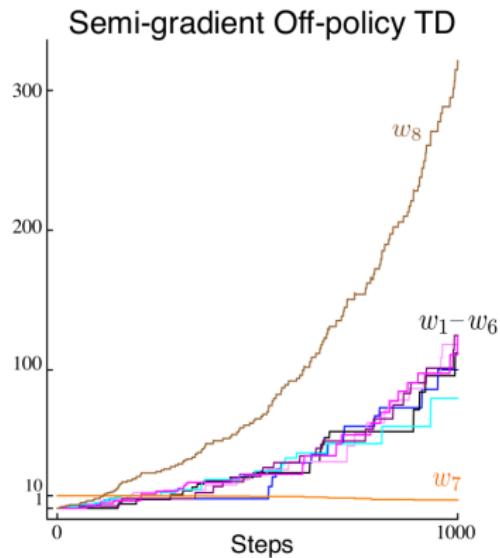


The reward is 0 for all transitions, hence $v_\pi(s) = 0$. This could be exactly approximated by $\mathbf{w} = \mathbf{0}$.

Baird's Counterexample

Semi-gradient DP

$$\mathbf{w} \leftarrow \mathbf{w} + \frac{\alpha}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} (\mathbb{E}[R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) | S_t = s] - \hat{v}(s, \mathbf{w})) \nabla \hat{v}(s, \mathbf{w})$$



The Deadly Triad

The combination of

- Function Approximation,
- Bootstrapping and
- Off-policy Learning

is known as the *Deadly Triad*, since it can lead to stability issues and divergence.

Lecture Overview

- 1 Recap
- 2 Off-policy Learning with Function Approximation
- 3 Problems of Off-policy Learning with Function Approximation
- 4 Gradient-TD Methods
- 5 Deep Q-learning
- 6 Wrapup

Gradient-TD Methods

To this point, the TD-methods discussed did not leverage the true gradient (they are called semi-gradient methods). Recall the Bellman Equation:

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_{\pi}(s')]$$

Bellman Error

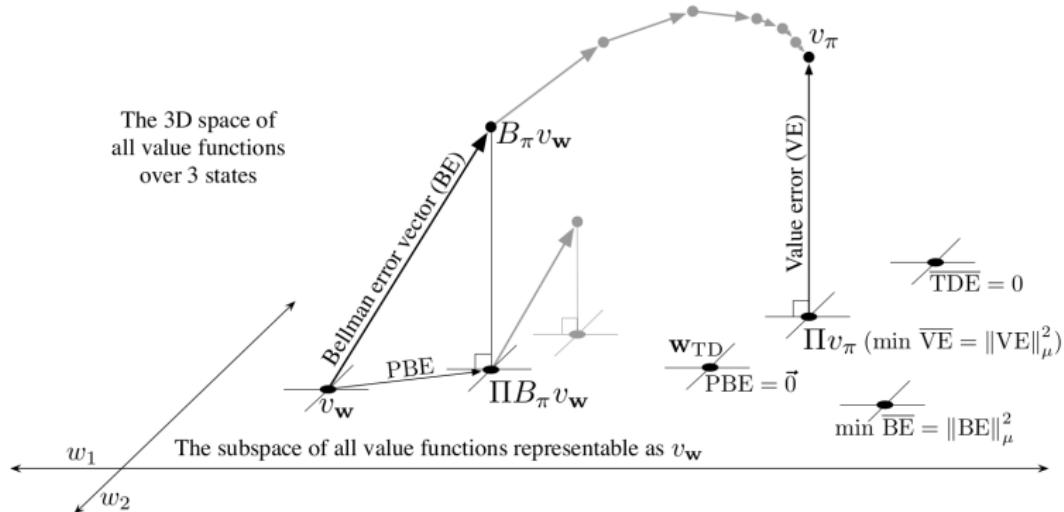
$$\begin{aligned}\bar{\delta}_{\mathbf{w}}(s) &= \left(\sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_{\mathbf{w}}(s')] \right) - v_{\mathbf{w}}(s) \\ &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t) | S_t = s, A_t \sim \pi]\end{aligned}$$

Gradient-TD Methods

Projected Bellman Error

The Projected Bellman Error is the projection of the Bellman Error back into the representable space:

$$\overline{\text{PBE}} = \|\Pi \bar{\delta}_w\|_\mu^2$$



Gradient-TD Methods

Projection Matrix for linear FA

The projection matrix for linear FA can be represented as an $|\mathcal{S}| \times |\mathcal{S}|$ matrix:

$$\Pi = \mathbf{X}(\mathbf{X}^\top \mathbf{D} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{D}$$

$$\begin{aligned}\overline{\text{PBE}}(\mathbf{w}) &= \|\Pi \bar{\delta}_{\mathbf{w}}\|_{\mu}^2 \\ &= (\Pi \bar{\delta}_{\mathbf{w}})^\top \mathbf{D} \Pi \bar{\delta}_{\mathbf{w}} \\ &= \bar{\delta}_{\mathbf{w}}^\top \Pi^\top \mathbf{D} \Pi \bar{\delta}_{\mathbf{w}} \\ &= \bar{\delta}_{\mathbf{w}}^\top \mathbf{D} \mathbf{X} (\mathbf{X}^\top \mathbf{D} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}} \\ &= (\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}})^\top (\mathbf{X}^\top \mathbf{D} \mathbf{X})^{-1} (\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}})\end{aligned}$$

The gradient with respect to \mathbf{w} is:

$$\nabla \overline{\text{PBE}}(\mathbf{w}) = 2 \nabla [\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}}]^\top (\mathbf{X}^\top \mathbf{D} \mathbf{X})^{-1} (\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}})$$

Gradient-TD Methods

Assume μ to be the distribution of state visited under the behaviour policy.

①

$$\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}} = \sum_s \mu(s) \mathbf{x}(s) \bar{\delta}_{\mathbf{w}}(s) = \mathbb{E}[\rho_t \delta_t \mathbf{x}_t]$$

②

$$\begin{aligned}\nabla [\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}}]^\top &= \mathbb{E}[\rho_t \nabla \delta_t^\top \mathbf{x}_t^\top] \\ &= \mathbb{E}[\rho_t \nabla (R_{t+1} + \gamma \mathbf{w}^\top \mathbf{x}_{t+1} - \mathbf{w}^\top \mathbf{x}_t)^\top \mathbf{x}_t^\top] \\ &= \mathbb{E}[\rho_t (\gamma \mathbf{x}_{t+1} - \mathbf{x}_t) \mathbf{x}_t^\top]\end{aligned}$$

③

$$\mathbf{X}^\top \mathbf{D} \mathbf{X} = \sum_s \mu(s) \mathbf{x}_s \mathbf{x}_s^\top = \mathbb{E}[\mathbf{x}_t \mathbf{x}_t^\top]$$

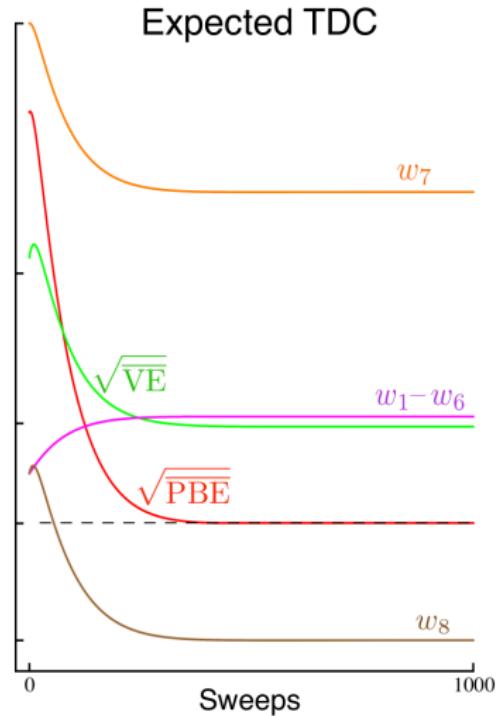
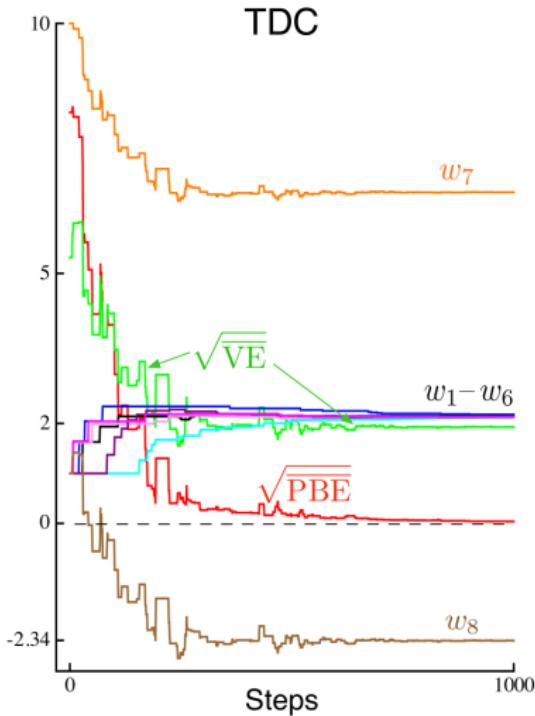
Gradient-TD Methods

- We can thus rewrite the gradient as:

$$\nabla \overline{\text{PBE}}(\mathbf{w}) = 2\mathbb{E}[\rho_t(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t)\mathbf{x}_t^\top] \mathbb{E}[\mathbf{x}_t \mathbf{x}_t^\top] \mathbb{E}[\rho_t \delta_t \mathbf{x}_t]$$

- Store the last two factors in d -vector $\mathbf{v} \approx \mathbb{E}[\mathbf{x}_t \mathbf{x}_t^\top]^{-1} \mathbb{E}[\rho_t \delta_t \mathbf{x}_t]$ and update via $\mathbf{v} \leftarrow \mathbf{v} + \beta \rho_t (\delta_t - \mathbf{v}^\top \mathbf{x}_t) \mathbf{x}_t$
- Estimate the first part via sampling
- This method and variants of it are called Gradient-TD methods

Gradient-TD Methods



Lecture Overview

- 1 Recap
- 2 Off-policy Learning with Function Approximation
- 3 Problems of Off-policy Learning with Function Approximation
- 4 Gradient-TD Methods
- 5 Deep Q-learning
- 6 Wrapup

Neural Fitted-Q Iteration (NFQ) [Riedmiller 2005]

- Model-free off-policy RL algorithm that works on continuous state and discrete action spaces
- Q-function is represented by a multi-layer perceptron
- One of the first approaches that combined RL with ANNs, predecessor of DQN

Neural Fitted-Q Iteration (NFQ) [Riedmiller 2005]

Algorithm 1 NFQ

for iteration $i = 1, \dots, N$ **do**

sample trajectory with ϵ -greedy exploration and add to memory D

initialize network weights randomly

generate pattern set $P = \{(x_j, y_j) | j = 1..|D|\}$ with

$$x_j = (s_j, a_j) \text{ and } y_j = \begin{cases} r_j & \text{if } s_j \text{ is terminal} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a', \mathbf{w}_i) & \text{else} \end{cases}$$

for iteration $k = 1, \dots, K$ **do**

Fit weights according to:

$$L(\mathbf{w}_i) = \frac{1}{|D|} \sum_{j=1}^{|D|} (y_j - Q(x_j, \mathbf{w}_i))^2$$

Deep Q-Networks (DQN)

DQN provides a stable solution to deep RL:

- Use experience replay (as in NFQ)
- Sample minibatches (as opposed to Full Batch in NFQ)
- Freeze target Q-networks (no target networks in NFQ)
- Optional: Clip rewards or normalize network adaptively to sensible range

Deep Q-Networks: Experience Replay

To remove correlations, build data set from agent's own experience

- Take action a_t according to ϵ -greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D
- Sample random mini-batch of transitions (s, a, r, s') from D
- Optimize MSE between Q-network and Q-learning targets, e.g.

$$L(\mathbf{w}) = \mathbb{E}_{s,a,r,s' \sim D} [(r + \gamma \max_{a'} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}))^2]$$

Deep Q-Networks: Target Networks

To avoid oscillations, fix parameters used in Q-learning target

- Compute Q-learning targets w.r.t. old, fixed parameters \mathbf{w}^-

$$r + \gamma \arg \max_{a'} Q(s', a', \mathbf{w}^-)$$

- Optimize MSE between Q-network and Q-learning targets

$$L(\mathbf{w}) = \mathbb{E}_{s,a,r,s' \sim D} [(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}))^2]$$

- Periodically update fixed parameters $\mathbf{w}^- \leftarrow \mathbf{w}$

- hard update: update target network every N steps
- slow update: slowly update weights of target network every step by

$$\mathbf{w}^- \leftarrow (1 - \tau)\mathbf{w}^- + \tau\mathbf{w}$$

Deep Q-Networks (DQN)

Algorithm 2 DQN

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights

for episode $i = 1, \dots, M$ **do**

for $t = 1, \dots, T$ **do**

 select action a_t ϵ -greedily

 Store transition (s_t, a_t, s_{t+1}, r_t) in D

 Sample minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D

 Set $y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a', \mathbf{w}^-) & \text{else} \end{cases}$

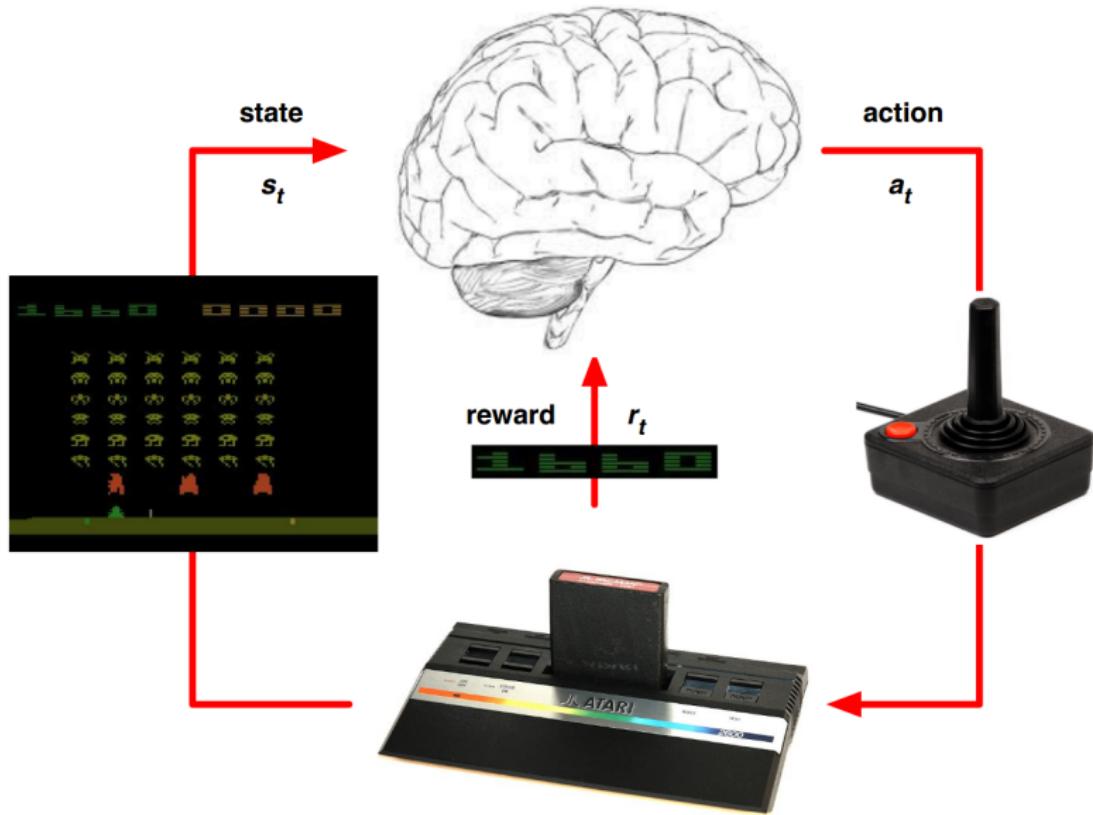
 Update the parameters of Q according to:

$$\begin{aligned} \nabla_{\mathbf{w}_i} L_i(\mathbf{w}_i) = \mathbb{E}_{s, a, s, r \sim D} [& (r + \gamma \max_{a'} Q(s', a', \mathbf{w}_i) \\ & - Q(s, a, \mathbf{w}_i)) \nabla_{\mathbf{w}_i} Q(s, a, \mathbf{w}_i)] \end{aligned}$$

 Update target network

This is your
exercise

Deep Q-Networks: Reinforcement Learning in Atari



Deep Q-Networks: Reinforcement Learning in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is a stack of raw pixels from the last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step

How much does DQN help?

	Q-Learning	Q-Learning + Target Q	Q-Learning + Replay	DQN + Replay + Target Q
Breakout	3	10	241	317
Enduro	29	142	831	1006
River Raid	1453	2868	4103	7447
Seaquest	276	1003	831	2894
Space Invaders	302	373	826	1089

Lecture Overview

- 1 Recap
- 2 Off-policy Learning with Function Approximation
- 3 Problems of Off-policy Learning with Function Approximation
- 4 Gradient-TD Methods
- 5 Deep Q-learning
- 6 Wrapup

Summary by Learning Goals

Having heard this lecture, you can now...

- Explain the difficulties that may arise in off-policy learning with FA
- Apply deep Q-learning and explain why we include replay memory and target networks



Lecture 08: n-Step Bootstrapping and Eligibility Traces

75 plays · 105 players

A public kahoot

Questions (6)

1 - Quiz

MC vs TD: What is correct?

60 sec

- MC is more sensitive to initial values than TD. ✗
- TD has a lower variance than MC. ✓
- MC is usually more efficient than TD. ✗
- MC has a larger bias. ✗

2 - Quiz

n-step returns: what is correct?

60 sec

- $n=1$ is equivalent to the TD-target and $n=\infty$ to the MC-target. ✓
- $n=1$ is equivalent to the MC-target and $n=\infty$ to the TD-target. ✗

3 - Quiz

Why are n-step targets not trivially applicable in Q-learning?

60 sec

- The subtrajectory does not follow the target policy. ✓
- n-step subreturn and current Q-estimation can have a different scale. ✗
- The Q-prediction for some state $s_{(t+n)}$ is less accurate than for $s_{(t+1)}$. ✗

4 - True or false

$\lambda=0.5$ is always the best choice for TD(λ).

30 sec

- | | |
|--|---|
|  True |  |
|  False |  |

5 - True or false

In order to perform an update with λ -returns, we have to wait until the end of an episode.

30 sec

- | | |
|--|---|
|  True |  |
|  False |  |

6 - Quiz

Eligibility Traces: what is correct?

60 sec

- | | |
|---|---|
|  (1) encodes Frequency and (2) Recency. |  |
|  (1) encodes Recency and (2) Frequency. |  |

Lecture 9: Eligibility Traces

Friday, January 14, 2022

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg



Lecture Overview

1 Recap

2 n -step Bootstrapping

3 Eligibility Traces

4 Wrapup

Lecture Overview

1 Recap

2 n -step Bootstrapping

3 Eligibility Traces

4 Wrapup

Recap: Off-policy Methods with Function Approximation

- *Deadly Triad*: The combination of Function Approximation, Bootstrapping and Off-policy Learning
- Gradient TD methods: TD(0) with gradient correction (TDC)
- NFQ (experience replay, full batch)
- DQN (experience replay, minibatches, target networks)

Lecture Overview

1 Recap

2 n -step Bootstrapping

3 Eligibility Traces

4 Wrapup

n -step Bootstrapping

- Recall:

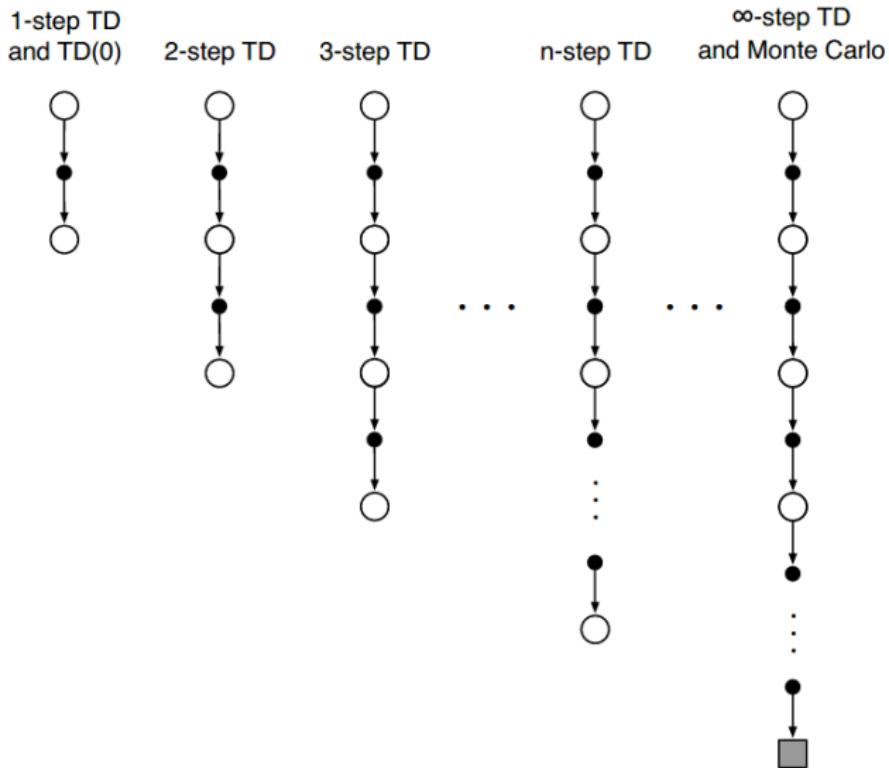
- MC-target: $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T$
- TD-target: $R_{t+1} + \gamma V(S_{t+1})$

- n -step Return:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- $n = 0$: TD
- $n = \infty$: MC

n -step TD Prediction



n -step TD Prediction

n -step TD for estimating $V \approx v_\pi$

Input: a policy π

Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n

Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$

All store and access operations (for S_t and R_t) can take their index mod $n + 1$

Loop for each episode:

 Initialize and store $S_0 \neq$ terminal

$T \leftarrow \infty$

 Loop for $t = 0, 1, 2, \dots$:

 If $t < T$, then:

 Take an action according to $\pi(\cdot | S_t)$

 Observe and store the next reward as R_{t+1} and the next state as S_{t+1}

 If S_{t+1} is terminal, then $T \leftarrow t + 1$

$\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)

 If $\tau \geq 0$:

$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$

 If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$

$V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$

 Until $\tau = T - 1$

n-step Sarsa

How can *n*-step methods be used not just for prediction but for control?

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

where $n \geq 1$ and $0 \leq t < T - n$ and $G_{t:t+n} = G_t$ if $t + n \geq T$.

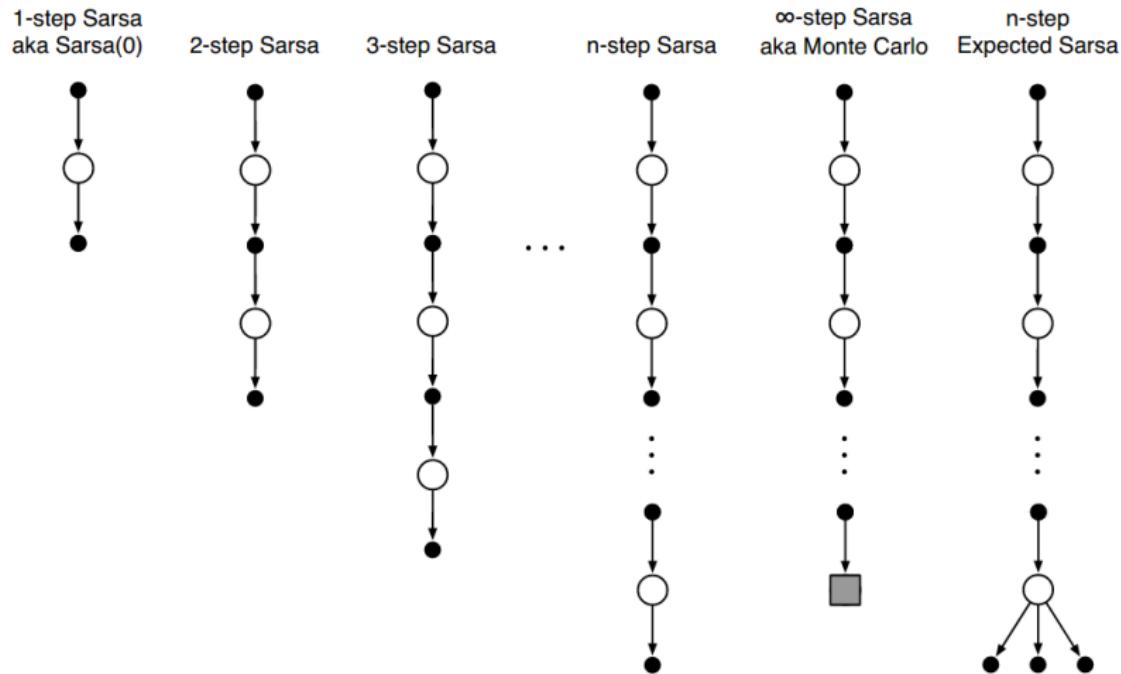
n-step Sarsa

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)], \quad 0 \leq t < T$$

while the values of all other states remain unchanged:

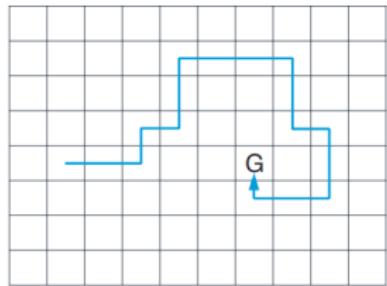
$$Q_{t+n}(s, a) = Q_{t+n-1}(s, a) \quad \forall s, a \text{ s.t. } s \neq S_t \text{ or } a \neq A_t$$

n -step Sarsa

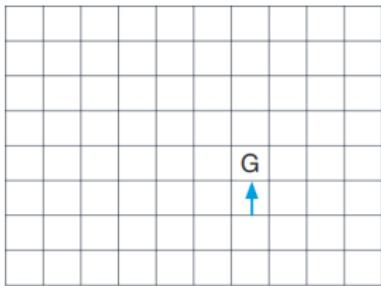


Example: Gridworld

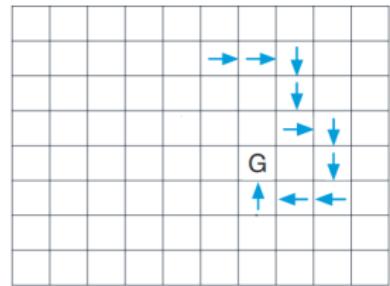
Path taken



Action values increased
by one-step Sarsa



Action values increased
by 10-step Sarsa



n -step Off-policy Learning

Importance sampling ratio (the relative probability under the two policies of taking the n actions from A_t to A_{t+n-1}):

$$\rho_{t:h} = \prod_{k=t}^{\min(h, T-1)} \frac{\pi(A_k | S_k)}{b(A_k | S_k)}$$

Previous n -step Sarsa update can be replaced by an off-policy form:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n} [G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$$

with $0 \leq t < T$.

Off-policy n -step Sarsa

Off-policy n -step Sarsa for estimating $Q \approx q_*$ or q_π

Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$

Initialize π to be greedy with respect to Q , or as a fixed given policy

Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n

All store and access operations (for S_t , A_t , and R_t) can take their index mod $n + 1$

Loop for each episode:

 Initialize and store $S_0 \neq$ terminal

 Select and store an action $A_0 \sim b(\cdot|S_0)$

$T \leftarrow \infty$

 Loop for $t = 0, 1, 2, \dots$:

 If $t < T$, then:

 Take action A_t

 Observe and store the next reward as R_{t+1} and the next state as S_{t+1}

 If S_{t+1} is terminal, then:

$T \leftarrow t + 1$

 else:

 Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$

$\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)

 If $\tau \geq 0$:

$$\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1, T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)} \quad (\rho_{\tau+1:t+n-1})$$

$$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i \quad (G_{\tau:\tau+n})$$

 If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$

$$Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \rho [G - Q(S_\tau, A_\tau)]$$

 If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q

 Until $\tau = T - 1$

Averaging n -step Returns

- We can average n -step returns over different n
- e.g. average the 2-step and 4-step returns

$$\frac{1}{2}G_{t:t+2} + \frac{1}{2}G_{t:t+4}$$

- Combines information from two different time steps
- Can we efficiently combine information from all time-steps?

Lecture Overview

1 Recap

2 n -step Bootstrapping

3 Eligibility Traces

4 Wrapup

Eligibility Traces and λ -return

Eligibility traces unify and generalize TD and Monte Carlo methods

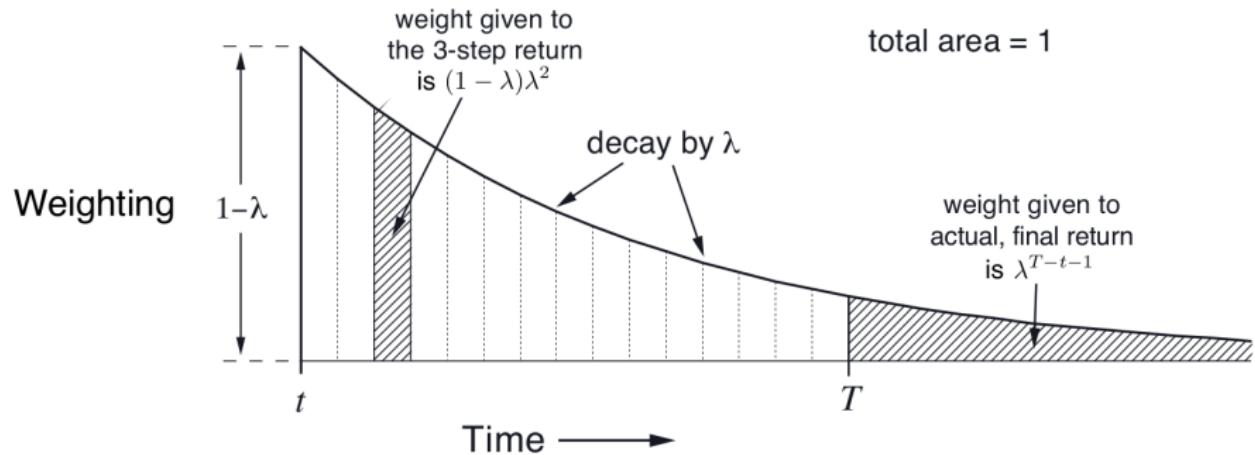
- MC methods at one end ($\lambda = 1$) and one-step TD methods at the other ($\lambda = 0$)
- almost any temporal-difference (TD) method can be combined with eligibility traces to (maybe) learn more efficiently

λ -return

- For infinite control tasks: $G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$
- For episodic control tasks:

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t$$

λ -return



Offline λ -Return Algorithm

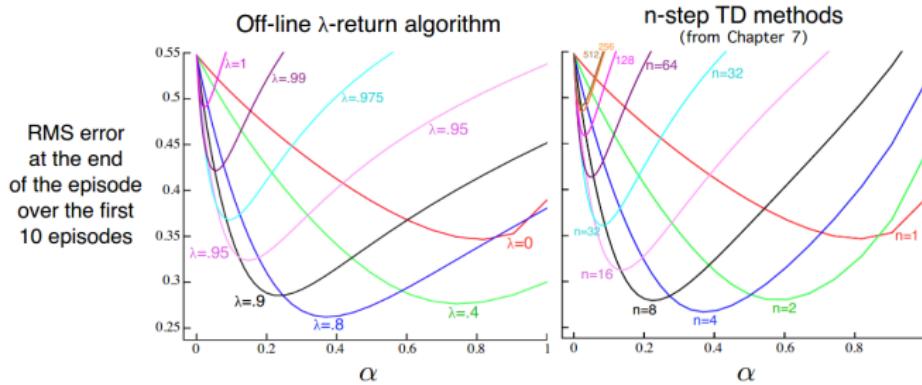
At the end of the episode, updates are made according to:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^\lambda - V(S_t)),$$

or, in the FA setting, to the semi-gradient rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[G_t^\lambda - \hat{v}(S_t, \mathbf{w}_t)]\nabla\hat{v}(S_t, \mathbf{w}_t), t = 0, \dots, T-1$$

Alternative way of moving smoothly between MC and one-step TD, can be compared to n -step TD methods.



Offline λ -Return Algorithm

The Offline λ -update can be converted to TD-form:

$$V(S_t) \leftarrow V(S_t) + \alpha \sum_{i=0}^{\infty} \lambda^i \delta_{t+i+1}$$

Offline λ -Return Algorithm

Lemma 9.1

$$(1 - \lambda) \sum_{n=m}^{\infty} \lambda^{n-1} = \lambda^{m-1}$$

Proof.

$$\begin{aligned}(1 - \lambda) \sum_{n=m}^{\infty} \lambda^{n-1} &= (1 - \lambda) \left(\sum_{n=1}^{\infty} \lambda^{n-1} - \sum_{k=1}^{m-1} \lambda^{k-1} \right) \\&= (1 - \lambda) \left(\sum_{n=0}^{\infty} \lambda^n - \sum_{k=0}^{m-2} \lambda^k \right) \\&= (1 - \lambda) \left(\frac{1}{1 - \lambda} - \frac{1 - \lambda^{m-1}}{1 - \lambda} \right) \\&= 1 - (1 - \lambda^{m-1}) \\&= \lambda^{m-1}\end{aligned}$$

Offline λ -Return Algorithm

Lemma 9.2

$$\sum_{n=1}^{\infty} \lambda^{n-1} \sum_{m=1}^n R_{t+m} = \sum_{m=1}^{\infty} R_{t+m} \sum_{n=m}^{\infty} \lambda^{n-1}$$

Proof.

$$\begin{aligned} \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{m=1}^n R_{t+m} &= \lambda^0 R_{t+1} + \lambda^1 (R_{t+1} + R_{t+2}) \\ &\quad + \lambda^2 (R_{t+1} + R_{t+2} + R_{t+3}) + \dots \\ &= R_{t+1}(\lambda^0 + \lambda^1 + \lambda^2 + \dots) + R_{t+2}(\lambda^1 + \lambda^2 + \dots) \\ &\quad + R_{t+3}(\lambda^2 + \dots) + \dots \\ &= \sum_{m=1}^{\infty} R_{t+m} \sum_{n=m}^{\infty} \lambda^{n-1} \end{aligned}$$

□

Offline λ -Return Algorithm

Lemma 9.3

$$(1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{m=1}^n R_{t+m} = \sum_{m=1}^{\infty} \lambda^{m-1} R_{t+m}$$

Proof.

$$\begin{aligned}(1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{m=1}^n R_{t+m} &\stackrel{2}{=} (1 - \lambda) \sum_{m=1}^{\infty} R_{t+m} \sum_{n=m}^{\infty} \lambda^{n-1} \\&= \sum_{m=1}^{\infty} R_{t+m} (1 - \lambda) \sum_{n=m}^{\infty} \lambda^{n-1} \\&\stackrel{1}{=} \sum_{m=1}^{\infty} \lambda^{m-1} R_{t+m}\end{aligned}$$

□

Offline λ -Return Algorithm

Lemma 9.4

$$(1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} V(S_{t+n}) = V(S_t) + \sum_{m=1}^{\infty} \lambda^{m-1} [V(S_{t+m}) - V(S_{t+m-1})]$$

Proof.

$$\begin{aligned}(1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} V(S_{t+n}) &= \sum_{n=1}^{\infty} V(S_{t+n})(\lambda^{n-1} - \lambda^n) \\&= V(S_{t+1})(\lambda^0 - \lambda^1) + V(S_{t+2})(\lambda^1 - \lambda^2) + \dots \\&= \lambda^0 V(S_{t+1}) + \lambda^1 V(S_{t+2}) - \lambda^1 V(S_{t+1}) + \\&\quad \lambda^2 V(S_{t+3}) - \lambda^2 V(S_{t+2}) + \dots + \\&\quad \lambda^0 V(S_t) - \lambda^0 V(S_t) \\&= V(S_t) + \sum_{m=1}^{\infty} \lambda^{m-1} [V(S_{t+m}) - V(S_{t+m-1})]\end{aligned}$$

Offline λ -Return Algorithm

The Offline λ -update can be converted to TD-form:

$$V(S_t) \leftarrow V(S_t) + \alpha \sum_{i=0}^{\infty} \lambda^i \delta_{t+i+1}$$

Proof. For simplification, we will ignore the discount.

$$\begin{aligned} V(S_t) &= (1 - \lambda) \cdot \mathbb{E} \left[\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \right] \\ &= (1 - \lambda) \cdot \mathbb{E} \left[\sum_{n=1}^{\infty} \lambda^{n-1} \left(\sum_{m=1}^n R_{t+m} + V(S_{t+n}) \right) \right] \\ &\stackrel{3,4}{=} \mathbb{E} \left[\sum_{m=1}^{\infty} \lambda^{m-1} [R_{t+m} + V(S_{t+m}) - V(S_{t+m-1})] + V(S_t) \right] \\ &= \mathbb{E} \left[\sum_{m=1}^{\infty} \lambda^{m-1} \delta_{t+m} + V(S_t) \right] \end{aligned}$$

Offline λ -Return Algorithm

The Offline λ -update can be converted to TD-form:

$$V(S_t) \leftarrow V(S_t) + \alpha \sum_{i=0}^{\infty} \lambda^i \delta_{t+i+1}$$

Proof. For simplification, we will ignore the discount.
If we plug this into our value function update, we get:

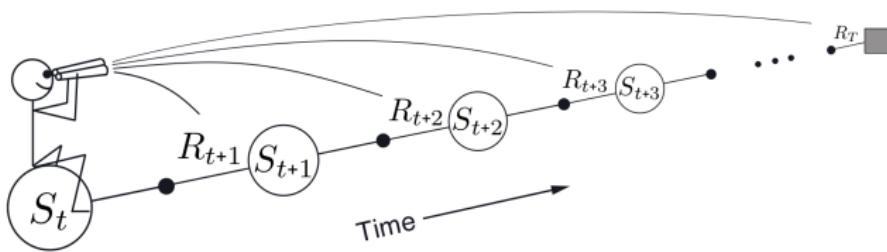
$$V(S_t) \leftarrow V(S_t) + \alpha \left(\sum_{m=1}^{\infty} \lambda^{m-1} \delta_{t+m} + V(S_t) - V(S_t) \right),$$

which leads to:

$$V(S_t) \leftarrow V(S_t) + \alpha \sum_{m=0}^{\infty} \lambda^m \delta_{t+m+1}.$$

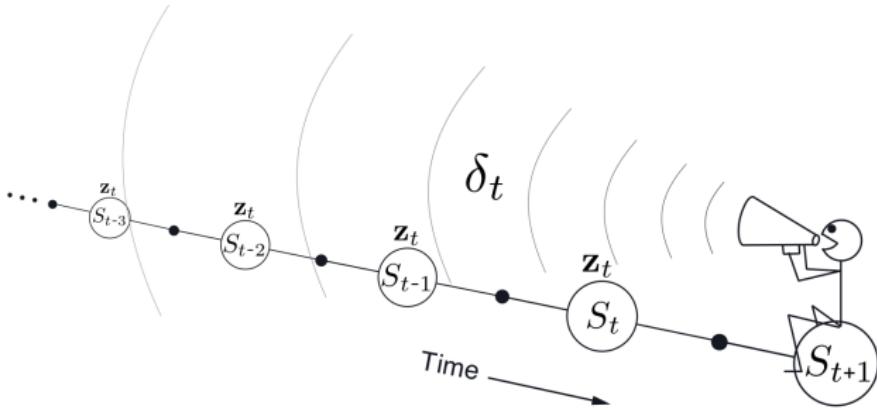
□

Forward View



- Update value function towards the λ -return
- Forward-view looks into the future to compute G_t^λ
- Like MC, can only be computed from complete episodes

Backward View



- look at the current TD error δ_t
- assign it backward to each prior state according to how much that state contributed to the current eligibility trace at that time

Eligibility Traces

- Eligibility Traces assign credit to components of the weight vector according to their contribution to state valuations
- They combine heuristics of *Frequency* and *Recency* (implemented by a λ -decay)
- With function approximation, the eligibility trace is a vector $\mathbf{z}_t \in \mathbb{R}$, initialized by $\mathbf{z}_{-1} = \mathbf{0}$ and incremented on each time step by:

$$\mathbf{z}_t = \gamma \lambda \mathbf{z}_{t-1} + \nabla \hat{v}(S_t, \mathbf{w}_t), \quad 0 \leq t \leq T,$$

where λ is called trace-decay parameter.

TD(λ)

- TD(λ) updates the weight vector on every step of an episode rather than only at the end and is not limited to episodic problems
- With the TD error

$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t),$$

in TD(λ), the weight vector is updated by:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t.$$

TD(λ)

Semi-gradient TD(λ) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$

Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$

Initialize value-function weights \mathbf{w} arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:

 Initialize S

$\mathbf{z} \leftarrow \mathbf{0}$ (a d -dimensional vector)

 Loop for each step of episode:

 | Choose $A \sim \pi(\cdot | S)$

 | Take action A , observe R, S'

 | $\mathbf{z} \leftarrow \gamma \lambda \mathbf{z} + \nabla \hat{v}(S, \mathbf{w})$

 | $\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$

 | $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \mathbf{z}$

 | $S \leftarrow S'$

 until S' is terminal

n -step Truncated λ -return

- offline λ -return algorithm is limited because:
 - the λ -return is not known until the end of the episode, or
 - technically never known in the continuing case
- dependence becomes weaker for longer-delayed reward, falling by $\gamma\lambda$ for each step of delay
- natural approximation: truncate the sequence after some number of steps:

truncated λ -return

$$G_{t:h}^\lambda = (1 - \lambda) \sum_{n=1}^{h-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{h-t-1} G_{t:h}, \quad 0 \leq t < h \leq T$$

n -step Truncated λ -return

- Now: updates are delayed by n steps and only take into account the first n rewards, but now all the k -step returns are included for $1 \leq k \leq n$.
- State-value case: truncated TD(λ) or TTD(λ)

TTD(λ)

$$\mathbf{w}_{t+n} = \mathbf{w}_{t+n-1} + \alpha [G_{t:t+n}^\lambda - \hat{v}(S_t, \mathbf{w}_{t+n-1})] \nabla \hat{v}(S_t, \mathbf{w}_{t+n-1}), 0 \leq t < T.$$

The k -step λ -return can efficiently implemented as:

$$G_{t:t+k} = \hat{v}(S_t, \mathbf{w}_{t-1}) + \sum_{i=t}^{t+k-1} (\gamma \lambda)^{i-t} \delta'_i$$

with $\delta'_i = R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_{t-1}) - \hat{v}(S_t, \mathbf{w}_{t-1})$.

Sarsa(λ)

Extension of eligibility traces to action-value methods.

- Action-value form of the n -step return:

$$G_{t:t+n} = R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}), \quad t+n < T$$

with $G_{t:t+n} = G_t$ if $t+n \geq T$.

- Action-value form of the truncated λ -return:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha [G_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}_t)] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t), \quad t = 0, \dots, T-1,$$

where $G_t^\lambda = G_{t:\infty}^\lambda$.

Sarsa(λ)

- With the TD error

$$\delta_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t),$$

- And the action-value form of the TD error:

$$\mathbf{z}_{-1} = \mathbf{0}$$

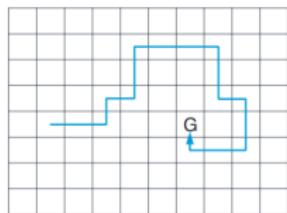
$$\mathbf{z}_t = \gamma \lambda \mathbf{z}_{t-1} + \nabla \hat{q}(S_t, A_t, \mathbf{w}_t), \quad 0 \leq t \leq T,$$

in Sarsa(λ), the weight vector is updated by:

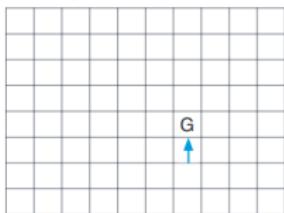
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t.$$

Example: Traces in Gridworld

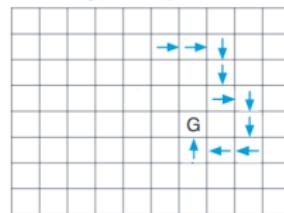
Path taken



Action values increased by one-step Sarsa



Action values increased by 10-step Sarsa



Action values increased by Sarsa(λ) with $\lambda=0.9$



Lecture Overview

1 Recap

2 n -step Bootstrapping

3 Eligibility Traces

4 Wrapup

Summary by Learning Goals

Having heard this lecture, you can now...

- explain two different ways of shifting and choosing between Monte Carlo and TD methods
- why eligibility trace methods are more general and often faster to learn



Lecture 09: Policy Gradient Methods

78 plays · 108 players

A public kahoot

Questions (6)

1 - Quiz

In Policy Gradient Methods, we...

60 sec

- always parameterize value-function and policy. ✗
- implicitly learn the policy by a value-function. ✗
- explicitly parameterize the policy. ✓
- never use a value-function. ✗

2 - True or false

We can only get policy gradients for continuous policies.

60 sec

- True ✗
- False ✓

3 - Quiz

We can employ a baseline...

60 sec

- to reduce variance. ✓
- to reduce bias. ✗

4 - Quiz

What is true about baselines?

60 sec

- Baselines change the expectation of the policy gradient. ✗
- Baselines must not depend on actions. ✓
- The baseline can be a constant scalar. ✓
- Only the true value function can act as a baseline. ✗

5 - Quiz

Why do we introduce the surrogate in PPO?

60 sec

- We can't sample states from a policy we haven't applied. ✓
- To make more cautious updates. ✗
- π_{old} yields better trajectories than π . ✗

6 - Quiz

Why do we constrain the surrogate (KL-divergence, clipping)?

60 sec

- The optimization becomes infeasible otherwise. ✗
- To make more cautious updates. ✓
- The surrogate is only a local approximation of x_i . ✓
- The surrogate is an overestimation of x_i . ✗

Lecture 10: Policy Gradient Methods

Friday, January 21, 2022

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg



Lecture Overview

- 1 Recap
- 2 Policy Gradient Methods
- 3 REINFORCE
- 4 Actor-Critic Methods
- 5 Proximal Policy Optimization
- 6 Exam
- 7 Wrapup

Lecture Overview

1 Recap

2 Policy Gradient Methods

3 REINFORCE

4 Actor-Critic Methods

5 Proximal Policy Optimization

6 Exam

7 Wrapup

Recap: Eligibility Traces and λ -return

Eligibility traces unify and generalize TD and Monte Carlo methods

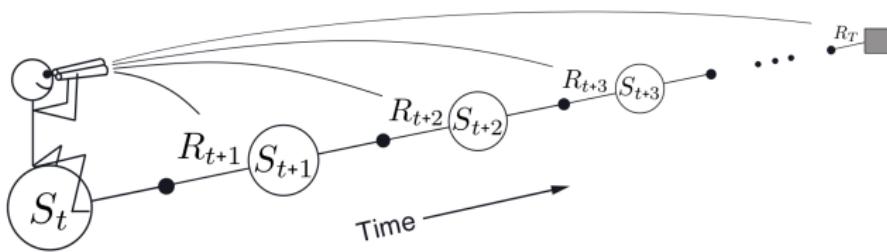
- MC methods at one end ($\lambda = 1$) and one-step TD methods at the other ($\lambda = 0$)
- almost any temporal-difference (TD) method can be combined with eligibility traces to (maybe) learn more efficiently

λ -return

- For infinite control tasks: $G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$
- For episodic control tasks:

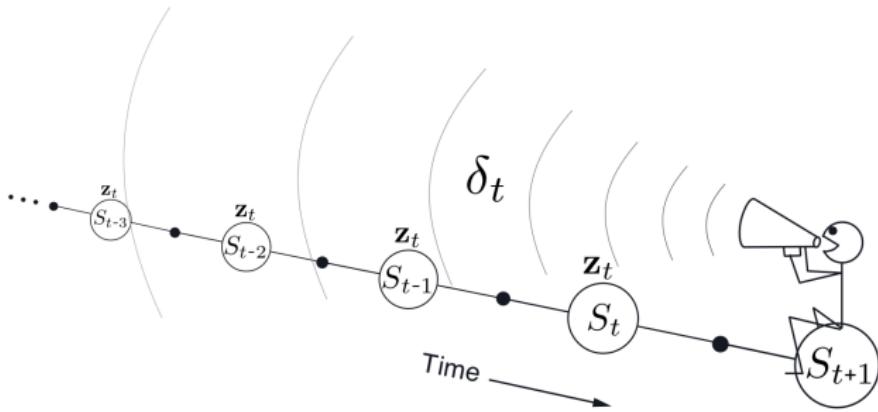
$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t$$

Recap: Forward View



- Update value function towards the λ -return
- Forward-view looks into the future to compute G_t^λ
- Like MC, can only be computed from complete episodes

Recap: Backward View



- look at the current TD error δ_t
- assign it backward to each prior state according to how much that state contributed to the current eligibility trace at that time

Recap: Eligibility Traces

- Eligibility Traces assign credit to components of the weight vector according to their contribution to state valuations
- They combine heuristics of *Frequency* and *Recency* (implemented by a λ -decay)
- With function approximation, the eligibility trace is a vector $\mathbf{z}_t \in \mathbb{R}$, initialized by $\mathbf{z}_{-1} = \mathbf{0}$ and incremented on each time step by:

$$\mathbf{z}_t = \gamma \lambda \mathbf{z}_{t-1} + \nabla \hat{v}(S_t, \mathbf{w}_t), \quad 0 \leq t \leq T,$$

where λ is called trace-decay parameter.

Lecture Overview

1 Recap

2 Policy Gradient Methods

3 REINFORCE

4 Actor-Critic Methods

5 Proximal Policy Optimization

6 Exam

7 Wrapup

Policy Gradient Methods

- Up to this point, we represented a model or a value function by some parameterized function approximator and extracted the policy implicitly
- Today, we are going to talk about *Policy Gradient Methods*: methods which consider a parameterized *policy*

$$\pi(a|s, \theta) = \Pr\{A_t = a | S_t = s, \theta_t = \theta\},$$

with parameters θ

- Policy Gradient Methods are able to represent stochastic policies and scale naturally to very large or continuous action spaces

Policy Gradient Methods

- We update these parameters based on the gradient of some performance measure $J(\theta)$ that we want to maximize, i.e. via *gradient ascent*:

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)},$$

where $\widehat{\nabla J(\theta_t)} \in \mathbb{R}^d$ is a stochastic estimate whose expectation approximates the gradient of the performance measure w.r.t. θ_t

Score Function

- Likelihood ratios exploit the following identity:

$$\overbrace{\nabla_{\theta} \pi(a|s, \theta)}^{\text{We want the expectation of this}} = \pi(a|s, \theta) \frac{\nabla_{\theta} \pi(a|s, \theta)}{\pi(a|s, \theta)}$$
$$= \underbrace{\pi(a|s, \theta) \nabla_{\theta} \log \pi(a|s, \theta)}_{\text{Easy to take the expectation because we can sample from } \pi!}$$

- $\nabla_{\theta} \log \pi(a|s, \theta)$ is called the **score function**

Score Function: Example

Consider a Gaussian policy, where the mean is a linear combination of state features: $\pi(a|s, \theta) \sim \mathcal{N}(s^\top \theta, \sigma^2)$, i.e.

$$\pi(a|s, \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} \frac{(s^\top \theta - a)^2}{\sigma^2}\right)$$

Exercise (5min)

Derive the score function.

Score Function: Example

Consider a Gaussian policy, where the mean is a linear combination of state features: $\pi(a|s, \boldsymbol{\theta}) \sim \mathcal{N}(s^\top \boldsymbol{\theta}, \sigma^2)$, i.e.

$$\pi(a|s, \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} \frac{(s^\top \boldsymbol{\theta} - a)^2}{\sigma^2}\right)$$

Solution

The log yields

$$\log \pi(a|s, \boldsymbol{\theta}) = -\frac{1}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} (s^\top \boldsymbol{\theta} - a)^2$$

and the gradient

$$\nabla_{\boldsymbol{\theta}} \log \pi(a|s, \boldsymbol{\theta}) = -\frac{1}{2\sigma^2} (s^\top \boldsymbol{\theta} - a) 2s = \frac{(a - s^\top \boldsymbol{\theta})s}{\sigma^2}.$$

Policy Gradient Theorem

Policy Objective Functions:

- For episodic problems we define performance as:
$$J(\boldsymbol{\theta}) = \eta(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{s_0 \sim \rho_0}[v_{\pi_{\boldsymbol{\theta}}}(s_0)]$$
- For continuing problems:
$$J(\boldsymbol{\theta}) = \sum_s \mu(s)v_{\pi_{\boldsymbol{\theta}}}(s)$$

Policy Gradient Theorem

For any differentiable policy $\pi(a|s, \boldsymbol{\theta})$ and any of the above policy objective functions, the policy gradient is:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi}[\nabla_{\boldsymbol{\theta}} \log \pi(a|s, \boldsymbol{\theta}) q_{\pi}(s, a)]$$

Reminder: $v_{\pi_{\boldsymbol{\theta}}} = \sum_a \pi(a|s)q_{\pi}(s, a)$

Policy Gradient Theorem

Proof (episodic case):

$$\begin{aligned}\nabla v_\pi(s) &= \nabla \left[\sum_a \pi(a|s) q_\pi(s, a) \right], \quad \text{for all } s \in \mathcal{S} \\ &= \sum_a [\nabla \pi(a|s) q_\pi(s, a) + \pi(a|s) \nabla q_\pi(s, a)] \quad (\text{product rule of calculus}) \\ &= \sum_a \left[\nabla \pi(a|s) q_\pi(s, a) + \pi(a|s) \nabla \sum_{s', r} p(s', r|s, a) (r + v_\pi(s')) \right] \\ &= \sum_a \left[\nabla \pi(a|s) q_\pi(s, a) + \pi(a|s) \sum_{s'} p(s'|s, a) \nabla v_\pi(s') \right] \\ &= \sum_a \left[\nabla \pi(a|s) q_\pi(s, a) + \pi(a|s) \sum_{s'} p(s'|s, a) \right. \\ &\quad \left. \sum_{a'} [\nabla \pi(a'|s') q_\pi(s', a') + \pi(a'|s') \sum_{s''} p(s''|s', a') \nabla v_\pi(s'')] \right] \\ &= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \Pr(s \rightarrow x, k, \pi) \sum_a \nabla \pi(a|x) q_\pi(x, a)\end{aligned}$$

Policy Gradient Theorem

Proof (episodic case):

$$\begin{aligned}\nabla J(\boldsymbol{\theta}) &= \nabla v_{\pi}(s_0) \\&= \sum_s \left(\sum_{k=0}^{\infty} \Pr(s_0 \rightarrow s, k, \pi) \right) \sum_a \nabla \pi(a|s) q_{\pi}(s, a) \\&= \sum_s \eta(s) \sum_a \nabla \pi(a|s) q_{\pi}(s, a) \\&= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla \pi(a|s) q_{\pi}(s, a) \\&= \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_{\pi}(s, a) \\&\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_{\pi}(s, a) \\&\quad (\text{Q.E.D.})\end{aligned}$$

Lecture Overview

- 1 Recap
- 2 Policy Gradient Methods
- 3 REINFORCE
- 4 Actor-Critic Methods
- 5 Proximal Policy Optimization
- 6 Exam
- 7 Wrapup

REINFORCE

- REINFORCE: Monte Carlo Policy Gradient
- Builds upon Monte Carlo returns as an unbiased sample of q_π
- However, therefore REINFORCE can suffer from high variance

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Algorithm parameter: step size $\alpha > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|s, \theta)$

Loop for each step of the episode $t = 0, 1, \dots, T - 1$:

$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k \tag{G_t}$$

$$\theta \leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t | S_t, \theta)$$

Variance Reduction with Baselines

- Vanilla REINFORCE provides *unbiased* estimates of the gradient $\nabla J(\theta)$, but it can suffer from high variance
- Goal: reduce variance while remaining unbiased
- Observation: we can generalize the policy gradient theorem by including an arbitrary *action-independent baseline* $b(s)$, i.e.

$$\begin{aligned}\nabla_{\theta} J(\theta) &\propto \sum_s \mu(s) \sum_a (q_{\pi}(s, a) - b(s)) \nabla \pi(a|s) \\&= \sum_s \mu(s) \left[\sum_a q_{\pi}(s, a) \nabla \pi(a|s) - b(s) \underbrace{\nabla \sum_a \pi(a|s)}_{=0} \right] \\&= \sum_s \mu(s) \sum_a q_{\pi}(s, a) \nabla \pi(a|s)\end{aligned}$$

- Baselines can reduce the variance of gradient estimates significantly!

Variance Reduction with Baselines

- A constant value can be used as a baseline
- The state-value function can be used as a baseline

Question

Is the Q-function a valid baseline?

Question

Assume an approximation of the state-value function as a baseline. Is REINFORCE then biased?

REINFORCE with Baselines

Indeed, an estimate of the state value function, $\hat{v}(S_t, w)$, is a very reasonable choice for $b(s)$:

REINFORCE with Baseline (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, w)$

Algorithm parameters: step sizes $\alpha^\theta > 0$, $\alpha^w > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot| \cdot, \theta)$

Loop for each step of the episode $t = 0, 1, \dots, T - 1$:

$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k \tag{G_t}$$

$$\delta \leftarrow G - \hat{v}(S_t, w)$$

$$w \leftarrow w + \alpha^w \delta \nabla \hat{v}(S_t, w)$$

$$\theta \leftarrow \theta + \alpha^\theta \gamma^t \delta \nabla \ln \pi(A_t | S_t, \theta)$$

Lecture Overview

- 1 Recap
- 2 Policy Gradient Methods
- 3 REINFORCE
- 4 Actor-Critic Methods
- 5 Proximal Policy Optimization
- 6 Exam
- 7 Wrapup

Actor-Critic Methods

- Methods that learn approximations to both policy and value functions are called actor-critic methods

actor: learned policy

critic: learned value function (usually a state-value function)

Question: Is REINFORCE-with-baseline considered as an actor-critic method?

Actor-Critic Methods

- REINFORCE-with-baseline is unbiased, but tends to learn slowly and has high variance
- To gain from advantages of TD methods we use actor-critic methods with a bootstrapping critic

One-step actor-critic methods

Replace the full return of REINFORCE with one-step return as follows:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha (G_{t:t+1} - \hat{v}(S_t, \mathbf{w})) \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha (R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})) \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha \delta_t \frac{\nabla \pi(A_t | S_t, \boldsymbol{\theta}_t)}{\pi(A_t | S_t, \boldsymbol{\theta}_t)}\end{aligned}$$

Actor-Critic Methods

One-step Actor–Critic (episodic), for estimating $\pi_{\theta} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, w)$

Parameters: step sizes $\alpha^{\theta} > 0$, $\alpha^w > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

 Initialize S (first state of episode)

$I \leftarrow 1$

 Loop while S is not terminal (for each time step):

$A \sim \pi(\cdot|S, \theta)$

 Take action A , observe S', R

$\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$ (if S' is terminal, then $\hat{v}(S', w) \doteq 0$)

$w \leftarrow w + \alpha^w \delta \nabla \hat{v}(S, w)$

$\theta \leftarrow \theta + \alpha^{\theta} I \delta \nabla \ln \pi(A|S, \theta)$

$I \leftarrow \gamma I$

$S \leftarrow S'$

Actor-Critic Methods

Actor-Critic with Eligibility Traces (episodic), for estimating $\pi_{\theta} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, w)$

Parameters: trace-decay rates $\lambda^{\theta} \in [0, 1]$, $\lambda^w \in [0, 1]$; step sizes $\alpha^{\theta} > 0$, $\alpha^w > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

 Initialize S (first state of episode)

$z^{\theta} \leftarrow \mathbf{0}$ (d' -component eligibility trace vector)

$z^w \leftarrow \mathbf{0}$ (d -component eligibility trace vector)

$I \leftarrow 1$

 Loop while S is not terminal (for each time step):

$A \sim \pi(\cdot|S, \theta)$

 Take action A , observe S', R

$\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$ (if S' is terminal, then $\hat{v}(S', w) \doteq 0$)

$z^w \leftarrow \gamma \lambda^w z^w + \nabla \hat{v}(S, w)$

$z^{\theta} \leftarrow \gamma \lambda^{\theta} z^{\theta} + I \nabla \ln \pi(A|S, \theta)$

$w \leftarrow w + \alpha^w \delta z^w$

$\theta \leftarrow \theta + \alpha^{\theta} \delta z^{\theta}$

$I \leftarrow \gamma I$

$S \leftarrow S'$

Lecture Overview

- 1 Recap
- 2 Policy Gradient Methods
- 3 REINFORCE
- 4 Actor-Critic Methods
- 5 Proximal Policy Optimization
- 6 Exam
- 7 Wrapup

Proximal Policy Optimization

- We collect data with $\pi_{\theta_{\text{old}}}$
- And we want to optimize some objective to get a new policy π_{θ}
- We can write $\eta(\pi_{\theta})$ in terms of $\pi_{\theta_{\text{old}}}$:

$$\eta(\pi_{\theta}) = \eta(\pi_{\theta_{\text{old}}}) + \mathbb{E}_{\pi_{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{A}_{\pi_{\theta_{\text{old}}}}(s_t, a_t) \right]$$

where the **advantage function** is defined as

$$\begin{aligned}\mathcal{A}_{\pi_{\theta_{\text{old}}}}(s, a) &= \mathbb{E}_{\pi_{\theta}, s_{t+1} \sim p} [q_{\pi_{\theta_{\text{old}}}}(s, a) - v_{\pi_{\theta_{\text{old}}}}(s)] \\ &= \mathbb{E}_{\pi_{\theta}, s_{t+1} \sim p} [r(s, a) + \gamma v_{\pi_{\theta_{\text{old}}}}(s') - v_{\pi_{\theta_{\text{old}}}}(s)]\end{aligned}$$

- Advantage: how much better or worse is every action than average?

Proximal Policy Optimization

Proof:

$$\begin{aligned} & \mathbb{E}_{\pi_{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{A}_{\pi_{\theta_{\text{old}}}}(s_t, a_t) \right] \\ &= \mathbb{E}_{\pi_{\theta}, s_{t+1} \sim p} \left[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t) + \gamma v_{\pi_{\theta_{\text{old}}}}(s_{t+1}) - v_{\pi_{\theta_{\text{old}}}}(s_t)) \right] \\ &= \mathbb{E}_{\pi_{\theta}, s_{t+1} \sim p} \left[-v_{\pi_{\theta_{\text{old}}}}(s_0) + \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \\ &= \mathbb{E}_{s_0 \sim p_0} \left[-v_{\pi_{\theta_{\text{old}}}}(s_0) \right] + \mathbb{E}_{\pi_{\theta}, s_{t+1} \sim p} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \\ &= -\eta(\pi_{\theta_{\text{old}}}) + \eta(\pi_{\theta}) \end{aligned}$$

Proximal Policy Optimization

- In PPO, we *ignore* the change in state distribution and optimize a **surrogate objective**:

$$\begin{aligned} J_{\text{old}}(\theta) &= \mathbb{E}_{s \sim \pi_{\theta_{\text{old}}}, a \sim \pi_{\theta}} [\mathcal{A}_{\pi_{\theta_{\text{old}}}}(s, a)] \\ &= \mathbb{E}_{(s, a) \sim \pi_{\theta_{\text{old}}}} \left[\frac{\pi_{\theta}}{\pi_{\theta_{\text{old}}}} \mathcal{A}_{\pi_{\theta_{\text{old}}}}(s, a) \right] \end{aligned}$$

- Improvement Theory: $\eta(\pi_{\theta}) \geq J_{\text{old}}(\theta) - c \cdot \max_s \text{KL}[\pi_{\theta_{\text{old}}} || \pi_{\theta}]$
- If we keep the KL-divergence between our old and new policies small, optimizing the surrogate is close to optimizing $\eta(\pi_{\theta})$!

Proximal Policy Optimization

- Clipped Surrogate Objective:

$$\mathbb{E}_{(s,a) \sim \pi_{\theta_{\text{old}}}} \left[\min\left(\frac{\pi_{\theta}}{\pi_{\theta_{\text{old}}}} \mathcal{A}_{\pi_{\theta_{\text{old}}}}(s, a), \text{clip}\left(\frac{\pi_{\theta}}{\pi_{\theta_{\text{old}}}}, 1 - \epsilon, 1 + \epsilon\right) \mathcal{A}_{\pi_{\theta_{\text{old}}}}(s, a)\right) \right]$$

- Adaptive Penalty Surrogate Objective:

$$\mathbb{E}_{(s,a) \sim \pi_{\theta_{\text{old}}}} \left[\frac{\pi_{\theta}}{\pi_{\theta_{\text{old}}}} \mathcal{A}_{\pi_{\theta_{\text{old}}}}(s, a) - \beta \text{KL}[\pi_{\theta_{\text{old}}} || \pi_{\theta}] \right]$$

Algorithm 1 PPO

for $i = 1, 2, \dots$ **do**

 Run policy for T timesteps of N trajectories

 Estimate advantage function at all timesteps

 Do SGD on one of the above objectives for some number of epochs

 In case of the Adaptive Penalty Surrogate: Increase β if KL-divergence too high, otherwise decrease β

Lecture Overview

- 1 Recap
- 2 Policy Gradient Methods
- 3 REINFORCE
- 4 Actor-Critic Methods
- 5 Proximal Policy Optimization
- 6 Exam
- 7 Wrapup

Exam

- There will be oral exams, the dates are March 23-25
- The first six minutes will be about the project, talk and discussion
- The project is designed as an exercise sheet for the last three weeks of the lecture (January 31, February 07, February 14)
- Final grade: $\frac{1}{3}$ project, $\frac{2}{3}$ questions about the rest of the lecture

Project

- You can choose to implement and apply any reinforcement learning algorithm (from the lecture or beyond) to solve this problem
- The evaluation should at least include learning curves (i.e. the return over time) of your chosen approach and settings – you can additionally think of your own metric and evaluate that as well
- It is important that your evaluation builds the basis for discussion and scientifically analyzes which are the important aspects and characteristics of your approach – your talk has to highlight your findings in a convincing manner
- You can prepare two slides, one with your approach and one with results (prepare as many backup slides for the discussion as you want)

Lecture Overview

- 1 Recap
- 2 Policy Gradient Methods
- 3 REINFORCE
- 4 Actor-Critic Methods
- 5 Proximal Policy Optimization
- 6 Exam
- 7 Wrapup

Summary by Learning Goals

Having heard this lecture, you can now...

- explain the Policy Gradient Theorem and derive score functions for a given policy.
- explain Actor-Critic Methods.
- apply Policy Gradient algorithms, such as REINFORCE and PPO.



Lecture 10: Advanced Value-based Methods

70 plays · 99 players

A public kahoot

Questions (5)

1 - Quiz

PER samples transitions with probability relative to their importance on the basis of...

60 sec

- their Q-value. ✗
- the entropy of the induced Boltzmann policy. ✗
- their immediate reward. ✗
- their TD-error. ✓

2 - True or false

In contrast to vanilla Q-learning, distributional Q-learning can model multi-modalities in value.

60 sec

- True ✓
- False ✗

3 - Quiz

DDPG: What is true?

60 sec

- The actor yields the true argmax_a Q. ✗
- DDPG updates the actor on the MSE. ✗
- DDPG is an on-policy algorithm. ✗
- The actor yields an approximation of argmax_a Q. ✓

4 - Quiz

What is not part of TD3?

60 sec

- Target-policy smoothing ✗
- Clipped Double Q-learning ✗
- Entropy regularization ✓
- Delayed policy update ✗

5 - Quiz

SAC...

60 sec

- penalizes the entropy of the policy. ✗
- augments the reward by the entropy of the policy. ✓
- enforces the policy to be narrow. ✗
- limits changes in the policy between updates. ✗

Lecture 11: Advanced Value-based Methods

Friday, January 28, 2022

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg



Lecture Overview

- 1 Recap
- 2 Prioritized Experience Replay
- 3 Distributional RL
- 4 DDPG
- 5 TD3
- 6 Soft Actor-Critic
- 7 Wrapup

Lecture Overview

1 Recap

2 Prioritized Experience Replay

3 Distributional RL

4 DDPG

5 TD3

6 Soft Actor-Critic

7 Wrapup

Recap: Policy Gradient Methods

- *Policy Gradient Methods* consider a parameterized *policy*

$$\pi(a|s, \theta) = \Pr\{A_t = a | S_t = s, \theta_t = \theta\},$$

with parameters θ

- We update these parameters based on the gradient of some performance measure $J(\theta)$ (expected return) that we want to maximize, i.e. via *gradient ascent*:

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)},$$

where $\widehat{\nabla J(\theta_t)} \in \mathbb{R}^d$ is a stochastic estimate whose expectation approximates the gradient of the performance measure w.r.t. θ_t

Policy Gradient Theorem

For any differentiable policy $\pi(a|s, \theta)$ and any of the introduced policy objective functions, the policy gradient is:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi} [\nabla_{\theta} \log \pi(a|s, \theta) q_{\pi}(s, a)]$$

Recap: Deep Q-Networks (DQN)

Algorithm 1 DQN

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights

for episode $i = 1,..,M$ **do**

for $t = 1,..,T$ **do**

 select action a_t ϵ -greedily

 Store transition (s_t, a_t, s_{t+1}, r_t) in D

 Sample minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D

 Set $y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a', \mathbf{w}^-) & \text{else} \end{cases}$

 Update the parameters of Q according to MSE between y_j and $Q(s_j, a_j, \mathbf{w})$

 Update target network

Lecture Overview

1 Recap

2 Prioritized Experience Replay

3 Distributional RL

4 DDPG

5 TD3

6 Soft Actor-Critic

7 Wrapup

Prioritized Experience Replay

Recall DQN with experience replay:

- Take action a_t according to ϵ -greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D
- Sample random mini-batch of transitions (s, a, r, s') from D
- Optimize MSE between Q-network and Q-learning targets, e.g.

$$L(\mathbf{w}) = \mathbb{E}_{s,a,r,s' \sim D} [(r + \gamma \max_{a'} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}))^2]$$

Problem: In many tasks, the number of "unimportant" samples grows faster than the number of "interesting" transitions, which guide the agent towards the goal (e.g. in sparse reward tasks).

⇒ Sampling mini-batches uniformly can be slow and inefficient!

Prioritized Experience Replay

Prioritized Experience Replay:

- Take action a_t according to ϵ -greedy policy, obtain $(s, a, r, s')_i$
- Measure the "importance" or *priority* p_i of a transition $(s, a, r, s')_i$ by the magnitude of the td-error

$$p_i \leftarrow |r + \gamma \max_{a'} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w})|$$

- Sample transitions with probability relative to their importance, e.g.

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \quad \alpha > 0$$

instead of taking only the samples with highest p_i to avoid sampling only noisy td-samples and forgetting samples with small priority.

- Can be implemented efficiently in a priority queue using a binary heap ($O(1)$ sampling, $O(\log(n))$ for updating priorities)

Lecture Overview

- 1 Recap
- 2 Prioritized Experience Replay
- 3 Distributional RL
- 4 DDPG
- 5 TD3
- 6 Soft Actor-Critic
- 7 Wrapup

Distributional RL

- Standard RL is only concerned with the *expected* long-term value of a transition, e.g. recall the bellman equation

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim p(\cdot|s, a), a' \sim \pi(s')} [r(s, a) + Q^\pi(s', a')]$$

- The Q-function averages over two sources of randomness (i.e. random variables):
 - The transition $p(s'|s, a)$
 - The reward function $r(s, a)$
- In distributional RL, we look at the *distribution* over the long-term value of a transition

$$Z^\pi(s, a) = R(s, a) + \sum_{t=1}^{\infty} \gamma^t R_t,$$

where $Z^\pi(s, a)$ is called the *value distribution*.

- Relationship to Q-function: $Q^\pi(s, a) = \mathbb{E}[Z^\pi(s, a)]$

- We can define a transition operator P with $PZ^\pi(s, a) := Z^\pi(S', A')$, i.e the value distribution that depends on the (random) next state and action $S' \sim p(\cdot|s, a)$, $A' \sim \pi(\cdot|S')$.
- We can define the Bellman operator $\mathcal{T}^\pi Z^\pi(s, a)$ as:

$$\mathcal{T}^\pi Z^\pi(s, a) := R(s, a) + \gamma P^\pi Z^\pi(s, a)$$

and a corresponding distributional Bellman optimality equation exists (similar to the Q-learning update).

- Convergence to an optimal value distribution Z^* is not guaranteed
- A sample-based algorithm similar to DQN can be derived that shows improved performance on most Atari tasks.
- Can model multi-modalities in value distributions and may help with non-stationarities leading to more stable learning.

Distributional RL

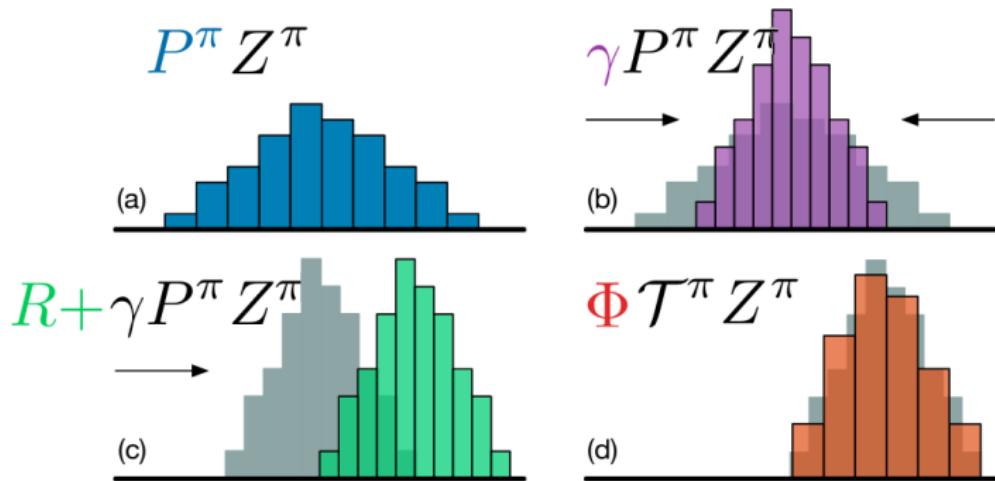


Figure 1. A distributional Bellman operator with a deterministic reward function: (a) Next state distribution under policy π , (b) Discounting shrinks the distribution towards 0, (c) The reward shifts it, and (d) Projection step (Section 4).

Lecture Overview

- 1 Recap
- 2 Prioritized Experience Replay
- 3 Distributional RL
- 4 DDPG
- 5 TD3
- 6 Soft Actor-Critic
- 7 Wrapup

Deep Deterministic Policy Gradient

- DDPG is an actor-critic method (*Continuous DQN*)
- Recall the DQN-target: $y_j = r_j + \gamma \max_a Q(s_{j+1}, a, \mathbf{w}^-)$
- In case of continuous actions, the maximization step is not trivial
- Therefore, we approximate deterministic actor μ representing the $\arg \max_a Q(s_{j+1}, a, \mathbf{w})$ by a neural network and update its parameters following the *Deterministic Policy Gradient Theorem*:

$$\nabla_{\theta} \leftarrow \frac{1}{N} \sum_j \nabla_a Q(s_j, a, \mathbf{w})|_{a=\mu(s_j)} \nabla_{\theta} \mu(s_j, \theta)$$

- Exploration by adding Gaussian noise to the output of μ

Deep Deterministic Policy Gradient

- The Q-function is fitted to the adapted TD-target:

$$y_j = r_j + \gamma Q(s_{j+1}, \mu(s_{j+1}, \boldsymbol{\theta}^-), \mathbf{w}^-)$$

- The parameters of target networks $\mu(\cdot, \boldsymbol{\theta}^-)$ and $Q(\cdot, \cdot, \mathbf{w}^-)$ are then adjusted with a soft update

$$\mathbf{w}^- \leftarrow (1 - \tau)\mathbf{w}^- + \tau\mathbf{w} \text{ and } \boldsymbol{\theta}^- \leftarrow (1 - \tau)\boldsymbol{\theta}^- + \tau\boldsymbol{\theta}$$

with $\tau \in [0, 1]$

- DDPG is very popular and builds the basis for more SOTA actor-critic algorithms
- However, it can be quite unstable and sensitive to its hyperparameters

Deep Deterministic Policy Gradient

Algorithm 2 DDPG

Initialize replay memory D to capacity N

Initialize critic Q and actor μ with random weights

for episode $i = 1, \dots, M$ **do**

for $t = 1, \dots, T$ **do**

 select action $a_t = \mu(s_t, \theta) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma)$

 Store transition (s_t, a_t, s_{t+1}, r_t) in D

 Sample minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D

 Set $y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma Q(s_{j+1}, \mu(s_{j+1}, \theta^-), \mathbf{w}^-) & \text{else} \end{cases}$

 Update the parameters of Q according to the TD-error

 Update the parameters of μ according to:

$$\nabla_{\theta} \leftarrow \frac{1}{N} \sum_j \nabla_a Q(s_j, a, \mathbf{w})|_{a=\mu(s_j)} \nabla_{\theta} \mu(s_j, \theta)$$

Adjust the parameters of the target networks via a soft update

Lecture Overview

- 1 Recap
- 2 Prioritized Experience Replay
- 3 Distributional RL
- 4 DDPG
- 5 TD3
- 6 Soft Actor-Critic
- 7 Wrapup

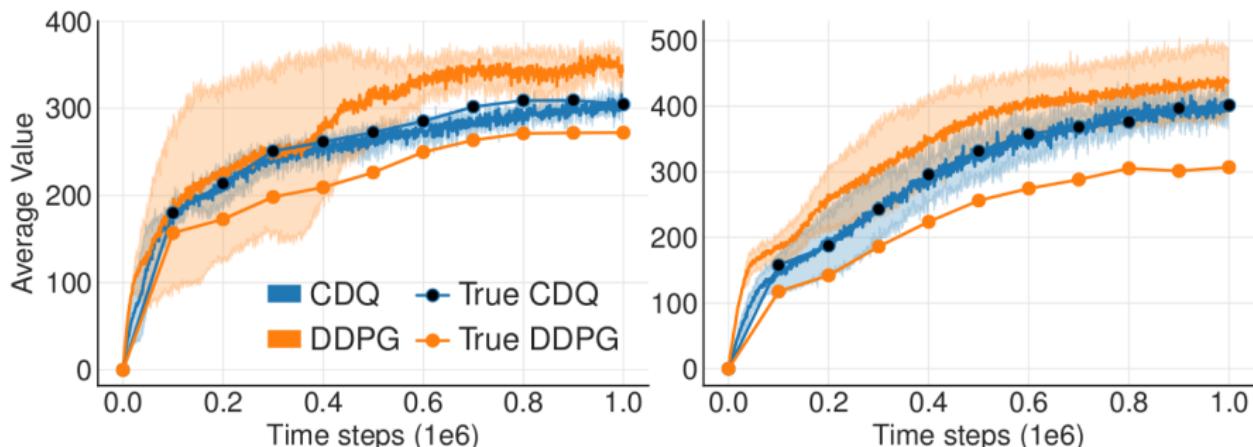
TD3 adds three adjustments to vanilla DDPG

- Clipped Double Q-Learning
- Delayed Policy Updates
- Target-policy smoothing

TD3: Clipped Double Q-Learning

- In order to alleviate the overestimation bias (which is also present in actor-critic methods), TD3 learns two approximations of the action-value function
- It takes the minimum of both predictions as the second part of the TD-target:

$$y_j = r_j + \gamma \min_{i \in \{1,2\}} Q(s_{j+1}, \mu(s_{j+1}), \mathbf{w}_i^-)$$



TD3: Delayed Policy Updates

- Due to the mutual dependency between actor and critic updates...
 - values can diverge when the policy leads to overestimation and
 - the policy will lead to bad regions of the state-action space when the value estimates lack in (relative) accuracy
- Therefore, policy updates on states where the value-function has a high prediction error can cause divergent behaviour
- We already know how to compensate for that: target networks
- Freeze target and policy networks between d updates of the value function
- This is called a *Delayed Policy Update*

TD3: Target-policy Smoothing

- *Target-policy Smoothing* adds Gaussian noise to the next action in target calculation
- It transforms the Q-update towards an Expected SARSA update fitting the value of a small area around the target-action:

$$y_j = r_j + \gamma \min_{i \in \{1,2\}} Q(s_{j+1}, \mu(s_{j+1}) + \text{clip}(\epsilon, -c, c), \mathbf{w}_i^-),$$

where $\epsilon \sim \mathcal{N}(0, \sigma)$

TD3: Ablation

Table 2. Average return over the last 10 evaluations over 10 trials of 1 million time steps, comparing ablation over delayed policy updates (DP), target policy smoothing (TPS), Clipped Double Q-learning (CDQ) and our architecture, hyper-parameters and exploration (AHE). Maximum value for each task is bolded.

Method	HCheetah	Hopper	Walker2d	Ant
TD3	9532.99	3304.75	4565.24	4185.06
DDPG	3162.50	1731.94	1520.90	816.35
AHE	8401.02	1061.77	2362.13	564.07
AHE + DP	7588.64	1465.11	2459.53	896.13
AHE + TPS	9023.40	907.56	2961.36	872.17
AHE + CDQ	6470.20	1134.14	3979.21	3818.71
TD3 - DP	9590.65	2407.42	4695.50	3754.26
TD3 - TPS	8987.69	2392.59	4033.67	4155.24
TD3 - CDQ	9792.80	1837.32	2579.39	849.75

Lecture Overview

- 1 Recap
- 2 Prioritized Experience Replay
- 3 Distributional RL
- 4 DDPG
- 5 TD3
- 6 Soft Actor-Critic
- 7 Wrapup

Soft Actor-Critic

- Soft Actor-Critic: entropy-regularized value-learning
- The policy is trained to maximize a trade-off between expected return and entropy ($H(P) = \mathbb{E}_{x \sim P}[-\log P(x)]$), a measure of randomness in the policy:

$$\pi_* = \arg \max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} R_{t+1} + \alpha H(\pi(\cdot | S_t = s_t)) \right]$$

- The value functions are then defined as:

$$v_{\pi}(s) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} R_{t+1} + \alpha H(\pi(\cdot | S_t = s_t)) \mid S_0 = s, A_0 = a \right]$$

and

$$q_{\pi}(s, a) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} R_{t+1} + \alpha \sum_{t=1}^{\infty} \gamma^t H(\pi(\cdot | S_t = s_t)) \mid S_0 = s, A_0 = a \right]$$

- And their relation as: $v_{\pi}(s) = \mathbb{E}_{\pi}[q_{\pi}(s, a)] + \alpha H(\pi(\cdot | S_t = s))$

Soft Actor-Critic

- The corresponding Bellman equation for q_π is

$$\begin{aligned} q_\pi(s_t, a_t) &= \mathbb{E}_{\pi, p}[R_{t+1} + \gamma(q_\pi(s_{t+1}, a_{t+1}) + \alpha H(\pi(\cdot | S_{t+1} = s_{t+1})))] \\ &= \mathbb{E}_{\pi, p}[R_{t+1} + \gamma v_\pi(s_{t+1})] \end{aligned}$$

- There are initial results that *Maximum Entropy RL*-methods lead to more composable policies

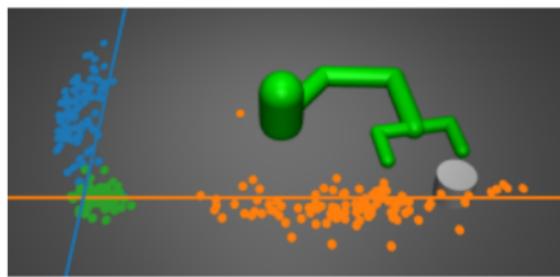


Fig. 2. Two independent policies are trained to push the cylinder to the orange line and blue line, respectively. The colored circles show samples of the final location of the cylinder for the respective policies. When the policies are combined, the resulting policy learns to push the cylinder to the lower intersection of the lines (green circle indicates final location). No additional samples from the environment are used to train the combined policy. The combined policy learns to satisfy both original goals, rather than simply averaging the final cylinder location.

Lecture Overview

- 1 Recap
- 2 Prioritized Experience Replay
- 3 Distributional RL
- 4 DDPG
- 5 TD3
- 6 Soft Actor-Critic
- 7 Wrapup

Summary by Learning Goals

Having heard this lecture, you can now...

- apply more sophisticated value-function methods to problems of higher complexity
- think of ways to improve deep value-function based methods
- provide an overview about the current SOTA

If you want to get an even more detailed overview about the current SOTA, you can have a look at OpenAI SpinningUp:

<https://spinningup.openai.com/en/latest/index.html>