Model Predictive Control and Reinforcement Learning
– Monte Carlo RL, Temporal Difference and Q-Learning –

Joschka Boedecker and Moritz Diehl

University Freiburg

July 27, 2021

Lecture Overview

Acknowledgement

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

Monte Carlo Reinforcement Learning

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

Monte Carlo Prediction

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

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

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

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

- 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: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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

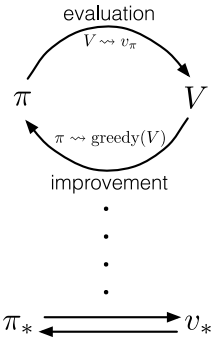
$G \leftarrow \gamma G + R_{t+1}$

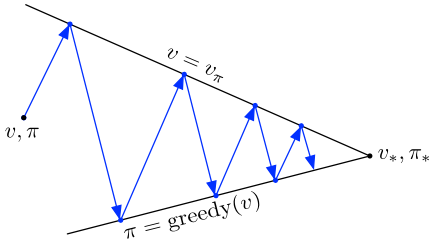
Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

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

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.

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

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: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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

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

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

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

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

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

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

Temporal Difference Learning

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)

TD Prediction

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 $V(\mathit{terminal}) = 0$

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

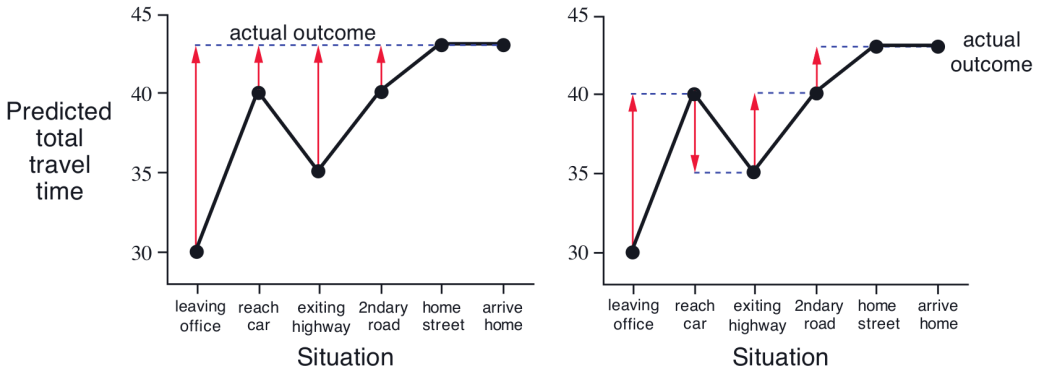


Figure 6.1: Changes recommended in the driving home example by Monte Carlo methods (left) and TD methods (right).

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?

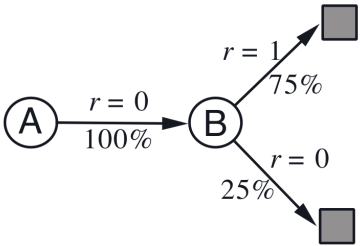
MC vs TD

- 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 to B in all cases. 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.



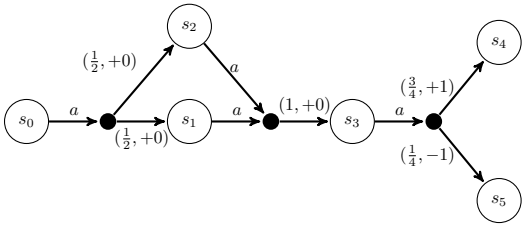
MC vs TD

- 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: Example

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

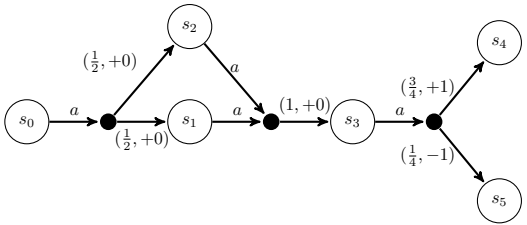
$$
\begin{array}{rccccccc}
t_i: & s_0 & \to & s_1 & \to & s_3 & \to & s_4 \\
t_{i+1}: & s_0 & \to & s_1 & \to & s_3 & \to & s_4 \\
t_{i+2}: & s_0 & \to & s_1 & \to & s_3 & \to & s_4 \\
t_{i+3}: & s_0 & \to & s_2 & \to & s_3 & \to & s_5
\end{array}
$$



Description

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?

MC vs 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.

SARSA

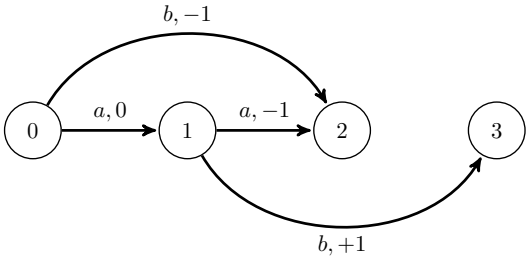
- SARSA: State, Action, Reward, State, Action
- Why is it considered an on-policy method?

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 $Q(\mathit{terminal}, \cdot) = 0$

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)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

SARSA Example



$$
\begin{array}{lccccc}
\text{traj}_1: & 0 & \to & 1 & \to & 2 \\
\text{traj}_2: & 0 & \to & 1 & \to & 3 \\
\text{traj}_3: & 0 & \to & 1 & \to & 2
\end{array}
$$

Q-learning

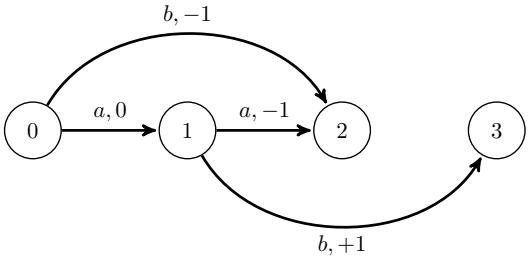
- Why is it considered an off-policy method?

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 $Q(\mathit{terminal}, \cdot) = 0$

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

Q-learning Example



$$
\begin{aligned}
\text{traj}_1: &\quad 0 \to 1 \to 2 \\
\text{traj}_2: &\quad 0 \to 1 \to 3 \\
\text{traj}_3: &\quad 0 \to 1 \to 2
\end{aligned}
$$

Summary

- Monte Carlo RL methods average sample returns from episodes of experience interacting with the environment, making it possible to learn without a given transition model
- Temporal Differenc methods learn from incomplete episodes by bootstrapping, combining ideas from of Monte Carlo and dynamic programming
- TD can learn online after every step, MC has to wait for the final outcome/return
- TD target has lower variance than the MC return, but is biased due to bootstrapping with wrong initial values
- Q-Learning can learn to approximate q_* even while gathering data with a different policy (off-policy learning)