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

Joschka Boedecker and Moritz Diehl

University Freiburg

October 6, 2023

universität freiburg

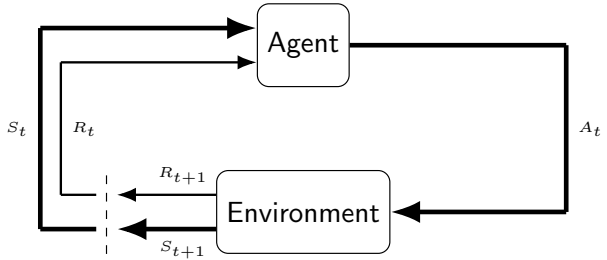
Lecture Overview

Acknowledgement

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

Lecture Overview

Agent and Environment



Time steps t: $0, 1, 2, \ldots$
States: $S_0, S_1, S_2, \ldots$
Actions: $A_0, A_1, A_2, \ldots$
Rewards: $R_1, R_2, R_3, \ldots$

Finite 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 set of states,
- $\mathcal{A}$ is a finite set 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:

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

The future is independent of the past given the present.

Finite 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 set of states,
- $\mathcal{A}$ is a finite set 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 $r(s, a, s') = \mathbb{E}[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s']$.

A deterministic system is a special case of an MDP:

$$p(s_{t+1} | s_t, u_t) = \begin{cases} 1 & s_{t+1} = f(s_t, a_t) \\ 0 & \text{otherwise} \end{cases}$$

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/minimization of the expected value of the cumulative sum of a received scalar signal (called reward/cost).

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: $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$
- Discounted: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
- 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

Description

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.

MDP: Example

Solution

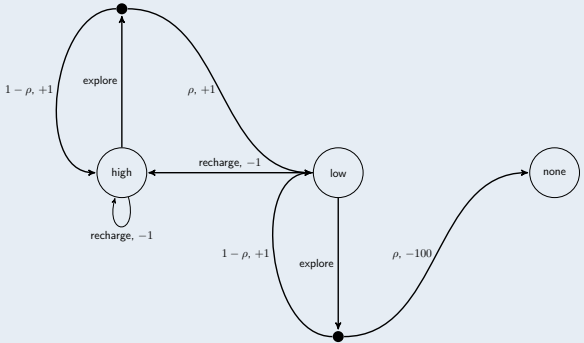
For the given problem, we set:

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

MDP: Example

Solution

The transition graph therefore is:



Lecture Overview

Policies

- 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

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

Bellman Equation for v_π

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

Bellman Equation

For a deterministic system and a deterministic policy, the Bellman Equation simplifies to:

Bellman equation for value-function v_π for a deterministic system and policy

$$v_\pi(s) = r + \gamma v_\pi(f(s, \pi(s))).$$

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

Bellman Equation for action-value function q_π

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

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 $s \in S$:

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

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

For a deterministic system and a deterministic policy, the Bellman Optimality Equation simplifies to:

Bellman equation for the optimal value-function v_* for a deterministic system and policy

$$v_*(s) = \max_a [r + \gamma v_*(f(s,a))].$$

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

Bellman equation for the optimal action-value function q_*

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

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

Lecture Overview

Policy Iteration: Overview

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

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

Policy Evaluation

Compute the state-value function v_π for an arbitrary policy π.
$\forall s \in S$:

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

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:

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

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 $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

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:

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

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:

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

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 $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s)) \big[r + \gamma V(s') \big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s') \big]$
   If *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

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:

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

Value Iteration

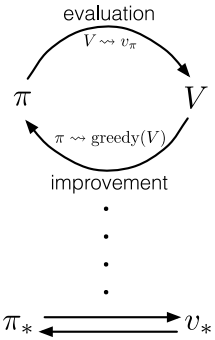
Value Iteration, for estimating $\pi \approx \pi_*$

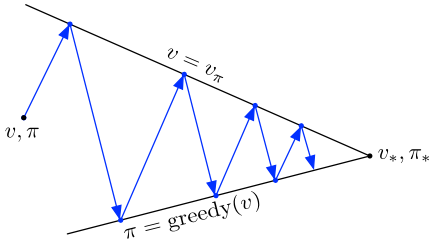
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$

Generalized Policy Iteration





- Policy Evaluation: estimate v_π
- Policy Improvement: greedy

Summary

- MDPs allow us to formalize RL (and more generally, stochastic optimal control) problems, 4-tuple $\langle \mathcal{S}, \mathcal{A}, p, \mathcal{R} \rangle$, assume Markov Property holds
- Bellman Equations express a relationship between the value of a state and the values of its successor states, provide structure to search for an optimal policy *intelligently*
- Policy Iteration and Value iteration use the structure of the Bellman Equations and turn them into iterative algorithms for finding optimal policies given an MDP (with and without explicit representation of the policy)

Lecture Overview

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

Lecture Overview

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 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, $t = T-1, T-2, \ldots, 0$:

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

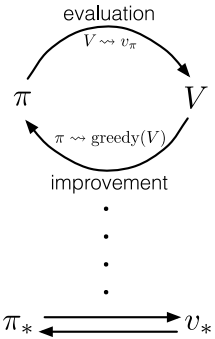
Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

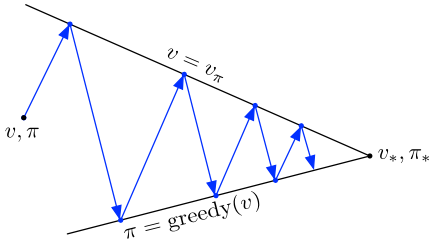
Append G to $Returns(S_t)$

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

Lecture Overview

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)

Lecture Overview

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*

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

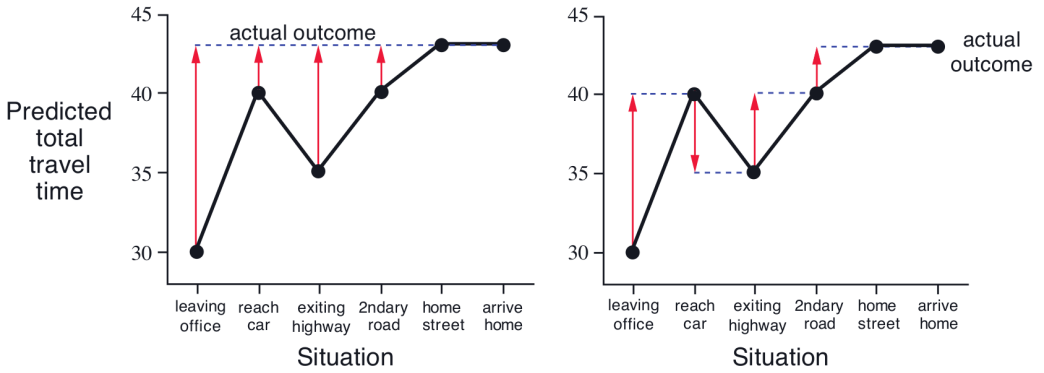


Figure 6.1: Changes recommended in the driving home example by Monte Carlo methods (left) and TD methods (right).

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?

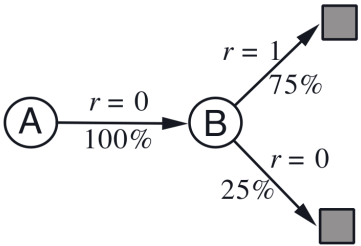
MC vs TD

- 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.



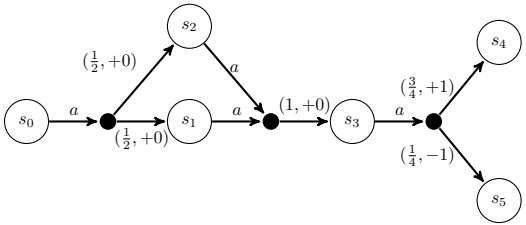
MC vs TD

- 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 $i \mod 4 = 0$):

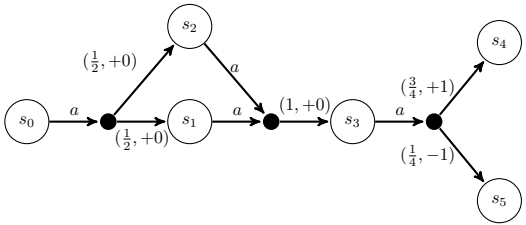
$$
\begin{array}{rccccccc}
t_i: & s_0 & \rightarrow & s_1 & \rightarrow & s_3 & \rightarrow & s_4 \\
t_{i+1}: & s_0 & \rightarrow & s_1 & \rightarrow & s_3 & \rightarrow & s_4 \\
t_{i+2}: & s_0 & \rightarrow & s_1 & \rightarrow & s_3 & \rightarrow & s_4 \\
t_{i+3}: & s_0 & \rightarrow & s_2 & \rightarrow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



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: $v(s_0) = v(s_3) = 0.5$, $v(s_1) = +1$ and $v(s_2) = -1$. 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

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

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

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)