Lecture 05: Temporal-Difference Learning

Oliver Wallscheid

Temporal-difference learning and the previous methods

Temporal-difference (TD) learning combines the previous ideas introduced in DP and MC:

- From Monte Carlo (MC) methods: Learns directly from experience.
- From dynamic programming (DP): Updates estimates based on other learned estimates (bootstrap).

Hence, TD characteristics are:

- Allows model-free prediction and control in unknown MDPs.
- Updates policy evaluation and improvement in an online fashion (i.e., not per episode) by bootstrapping.
- Still assumes finite MDP problems (or problems close to that).

Table of contents

1. Temporal-difference prediction
2. Temporal-difference on-policy control: SARSA
3. Temporal-difference off-policy control: Q-learning
4. Maximization bias and double learning

General TD prediction updates

Recap the every-visit MC update rule (4.3) for non-stationary problems:

$$\hat{v}(x_k) \leftarrow \hat{v}(x_k) + \alpha \left[g_k - \hat{v}(x_k) \right]. \tag{5.1}$$

- $\alpha \in \{\mathbb{R} | 0 < \alpha < 1\}$ is the forgetting factor / step size.
- g_k is the target of the incremental update rule.
- To execute the update (5.1) one has to wait until the episode's termination since only then g_k is available (MC requirement).

One-step TD / TD(0) update

$$\hat{v}(x_k) \leftarrow \hat{v}(x_k) + \alpha \left[r_{k+1} + \gamma \hat{v}(x_{k+1}) - \hat{v}(x_k) \right]. \tag{5.2}$$

- Here, the TD target is $r_{k+1} + \gamma \hat{v}(x_{k+1})$.
- TD is bootstrapping: estimate $\hat{v}(x_k)$ based on $\hat{v}(x_{k+1})$.
- Delay time of one step and no need to wait until the episode's end.

Algorithmic implementation: TD-based prediction

input: a policy π to be evaluated
output: estimate of $\boldsymbol{v}_{\mathcal{X}}^{\pi}$ (i.e., value estimates for all states $x \in \mathcal{X}$)
init: $\hat{v}(x) \, \forall \, x \in \mathcal{X}$ arbitrary except $v_0(x) = 0$ if x is terminal
for $j = 1, \ldots, J$ *episodes* **do**
    Initialize x_0;
    for $k = 0, 1, 2 \ldots$ *time steps* **do**
        $u_k \leftarrow$ apply action from $\pi(x_k)$;
        Observe x_{k+1} and r_{k+1};
        $\hat{v}(x_k) \leftarrow \hat{v}(x_k) + \alpha \left[r_{k+1} + \gamma \hat{v}(x_{k+1}) - \hat{v}(x_k) \right]$;
        Exit loop if x_{k+1} is terminal;

Algo. 5.1: Tabular TD(0) prediction

- Note that the algorithm can be directly adapted to action-value prediction as it will be used for the later TD-based control approaches.

TD error

Fig. 5.1: Back up diagram for TD(0)

- TD as well as MC use sample updates.
- Looking ahead to a sample successor state including its value and the reward along the way to compute a backed up value estimate.

The TD error is:

$$\delta_k = r_{k+1} + \gamma \hat{v}(x_{k+1}) - \hat{v}(x_k). \tag{5.3}$$

- δ_k is available at time step $k+1$.
- Iteratively δ_k converges towards zero.

TD error and its relation to the MC error

Let's assume that the TD(0) estimate $\hat{v}(x)$ is not changing over one episode as it would be for MC prediction:

$$
\begin{aligned}
\underbrace{g_k - \hat{v}(x_k)}_{\text{MC-error}} &= r_{k+1} + \gamma g_{k+1} - \hat{v}(x_k) + \gamma \hat{v}(x_{k+1}) - \gamma \hat{v}(x_{k+1}), \\
&= \delta_k + \gamma(g_{k+1} - \hat{v}(x_{k+1})), \\
&= \delta_k + \gamma \delta_{k+1} + \gamma^2 (g_{k+2} - \hat{v}(x_{k+2})), \\
&= \delta_k + \gamma \delta_{k+1} + \gamma^2 \delta_{k+2} + \gamma^3 (g_{k+3} - \hat{v}(x_{k+3})) = \cdots, \\
&= \sum_{i=k}^{T-1} \gamma^{i-k} \delta_i.
\end{aligned}
\tag{5.4}
$$

- MC error is the discounted sum of TD errors in this simplified case.
- If $\hat{v}(x)$ is updated during an episode (as expected in TD(0)), the above identity only holds approximately.

Overview of the RL methods considered so far

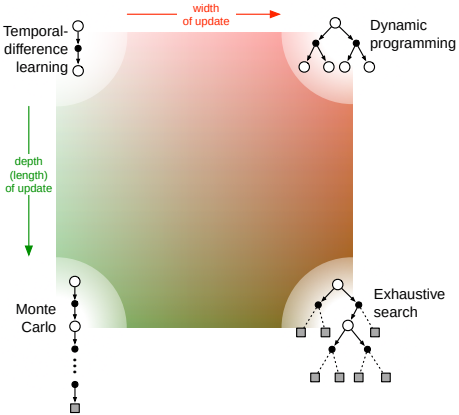

Fig. 5.2: Comparison of the RL methods considered so far with regard to the update rules (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018, CC BY-NC-ND 2.0)

Driving home example

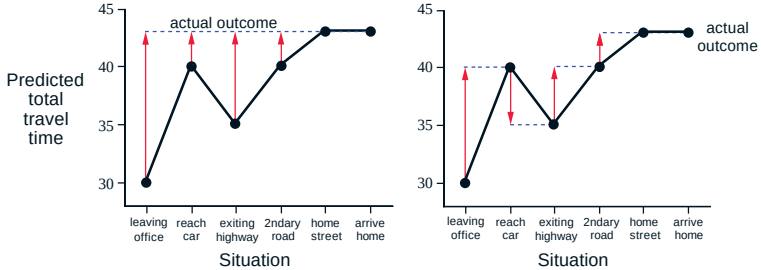

Fig. 5.3: Updates by MC (left) and TD (right) for $\alpha = 1$ (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018, CC BY-NC-ND 2.0)

- TD can learn before knowing the final outcome.
  - TD learns after every step.
  - MC must wait until the episode's end.
- TD could learn without a final outcome.
  - MC is only applicable to episodic tasks.

TD(0) prediction example: forest tree MDP (1)

Let's reuse the forest tree MDP example with *fifty-fifty policy* and discount factor $\gamma = 0.8$ plus disaster probability $\alpha = 0.2$:

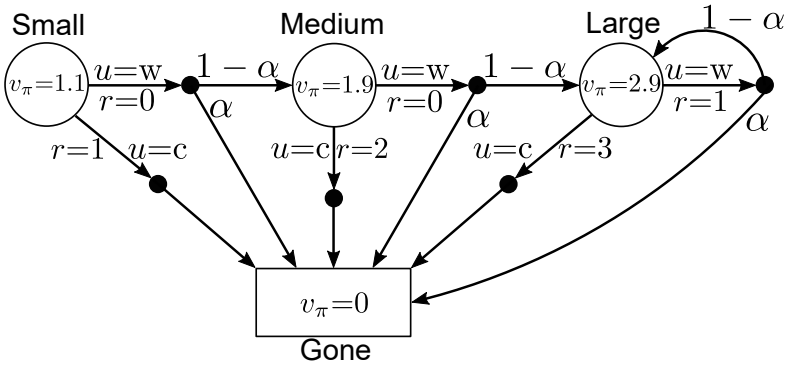

Fig. 5.4: Forest MDP with fifty-fifty-policy including state values

TD(0) prediction example: forest tree MDP (2)

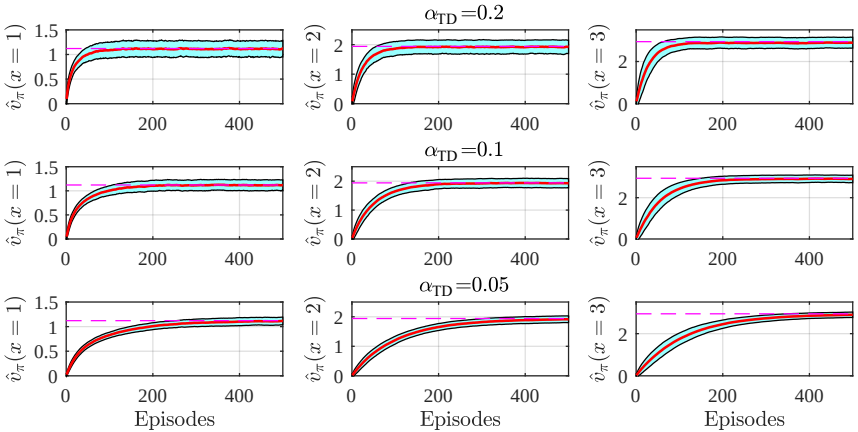

Fig. 5.5: State-value estimate of forest tree MDP using TD(0) prediction over the number of episodes being evaluated (mean and standard deviation are calculated based on 2000 independent runs)

TD(0) vs. MC prediction example: forest tree MDP (1)

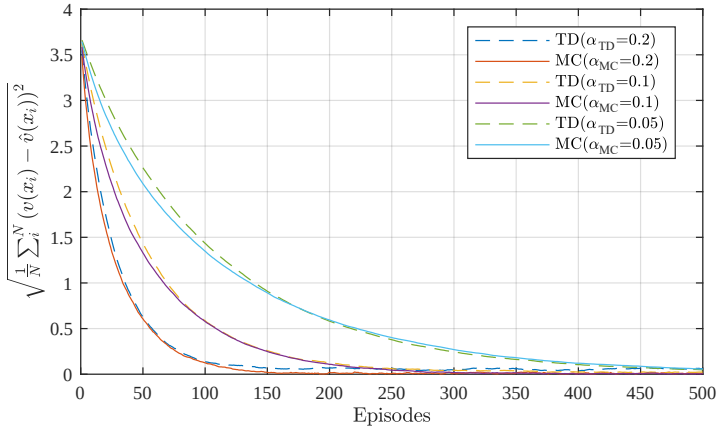

Fig. 5.6: Averaged mean of state-value estimates of forest tree MDP using TD(0) and MC over 1000 independent runs with $\hat{v}_0(x) = 0 \, \forall x \in \mathcal{X}$

TD(0) vs. MC prediction example: forest tree MDP (2)

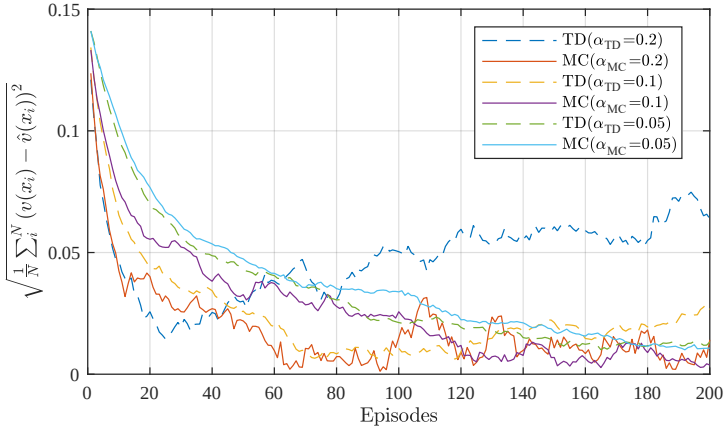

Fig. 5.7: Averaged mean of state-value estimates of forest tree MDP using TD(0) and MC over 1000 independent runs with $\hat{v}_0(x) \approx v(x) \, \forall x \in \mathcal{X}$

Convergence of TD(0)

Theorem 5.1: Convergence of TD(0)

Given a finite MDP and a fixed policy π the state-value estimate of TD(0) converges to the true v_π

- in the mean for a constant but sufficiently small step-size α and
- with probability 1 if the step-size holds the condition

$$\sum_{k=1}^{\infty} \alpha_k = \infty \quad \text{and} \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty. \tag{5.5}$$

Above k is the sample index (i.e., how often the TD update was applied).

- In particular, $\alpha_k = \frac{1}{k}$ meets the condition (5.5).
- Often TD(0) converges faster than MC, but there is no guarantee.
- TD(0) can be more sensitive to bad initializations $\hat{v}_0(x)$ compared to MC.

Batch training

- If experience $\to \infty$ both MC and TD converge $\hat{v}(x) \to v(x)$.
- But how to handle limited experience, i.e., a finite set of episodes

$$
\begin{aligned}
&x_{1,1}, u_{1,1}, r_{2,1}, \ldots, x_{T_1,1},\\
&x_{1,2}, u_{1,2}, r_{2,2}, \ldots, x_{T_2,2},\\
&\vdots\\
&x_{1,j}, u_{1,j}, r_{2,j}, \ldots, x_{T_j,j},\\
&\vdots\\
&x_{1,J}, u_{1,J}, r_{2,J}, \ldots, x_{T_J,J}.
\end{aligned}
$$

Batch training

- Process all available episodes $j \in [1, J]$ repeatedly to MC and TD.
- If the step size α is sufficiently small both will converge to certain steady-state values.

Batch training: AB-example (1)

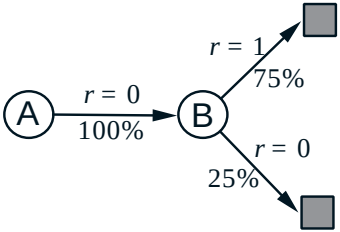

Fig. 5.8: Example environment (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018, CC BY-NC-ND 2.0)

- Only two states: A, B
- No discounting
- 8 episodes of experience available (see Tab. 5.1)
- What is $\hat{v}(A)$ and $\hat{v}(B)$ using batch training TD(0) and MC?

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

Batch training: AB-example (2)

First, recap MC and TD(0) update rules:

$$\begin{aligned}\text{MC}: &\quad \hat{v}(x_k) \leftarrow \hat{v}(x_k) + \alpha\left[g_k - \hat{v}(x_k)\right], \\ \text{TD}: &\quad \hat{v}(x_k) \leftarrow \hat{v}(x_k) + \alpha\left[r_{k+1} + \gamma\hat{v}(x_{k+1}) - \hat{v}(x_k)\right].\end{aligned}$$

Then, in steady state one receives:

$$\begin{aligned}\text{MC}: &\quad 0 = \alpha\left[g_k - \hat{v}(x_k)\right] = g_k - \hat{v}(x_k), \\ \text{TD}: &\quad 0 = \alpha\left[r_{k+1} + \gamma\hat{v}(x_{k+1}) - \hat{v}(x_k)\right] = r_{k+1} + \gamma\hat{v}(x_{k+1}) - \hat{v}(x_k).\end{aligned}$$

Considering a batch learning sweep over $j = 1, \ldots, J$ episodes:

$$\begin{aligned}\text{MC}: &\quad 0 = \sum_{j=1}^{J} g_{k,j} - \hat{v}(x_{k,j}), \\ \text{TD}: &\quad 0 = \sum_{j=1}^{J} r_{k+1,j} + \gamma\hat{v}(x_{k+1,j}) - \hat{v}(x_{k,j}).\end{aligned}$$

Batch training: AB-example (3)

Apply the previous equations first to state B. Since B is a terminal state, $\hat{v}(x_{k+1}) = 0$ and $g_{k,j} = r_{k+1,j}$ apply, i.e., the MC and TD updates are identical for B:

$$\text{MC}|_{x=\text{B}}: \quad 0 = \sum_{j=1}^{J} g_{k,j} - \hat{v}(x_{k,j}) \quad \Leftrightarrow \quad \hat{v}(\text{B}) = \frac{1}{J} \sum_{j=1}^{J} g_{k,j},$$

$$\text{TD}|_{x=\text{B}}: \quad 0 = \sum_{j=1}^{J} r_{k+1,j} - \hat{v}(x_{k,j}) \quad \Leftrightarrow \quad \hat{v}(\text{B}) = \frac{1}{J} \sum_{j=1}^{J} g_{k,j}.$$

This is the average return of the available episodes from Tab. 5.1 , i.e., 6×1 and 2×0:

$$\hat{v}(\text{B})|_{\text{MC}} = \hat{v}(\text{B})|_{\text{TD}} = \frac{6}{8} = 0.75\,. \tag{5.6}$$

Batch training: AB-example (4)

Now consider state A assuming the steady state of batch learning process:

- The instantaneous reward is always $r = 0$.
- The TD bootstrap estimate of B is $\hat{v}(x_{k+1,j}) = \hat{v}(B) = \frac{3}{4}$.

$$\text{MC}: \quad 0 = \sum_{j=1}^{J} g_{k,j} - \hat{v}(x_{k,j}) = \sum_{j=1}^{J} g_{k,j} - \hat{v}(A),$$

$$\text{TD}: \quad 0 = \sum_{j=1}^{J} r_{k+1,j} + \gamma \hat{v}(x_{k+1,j}) - \hat{v}(x_{k,j}) = \sum_{j=1}^{J} \gamma \hat{v}(B) - \hat{v}(A).$$

Looking at Tab. 5.1 there is only one episode visiting state A, where the sample return is $g_{k,j} = 0$. Hence, it follows:

$$\hat{v}(\text{A})|_{\text{MC}} = 0, \quad \hat{v}(\text{A})|_{\text{TD}} = \gamma \hat{v}(B) = \frac{3}{4}.$$

Where does this mismatch between the MC and TD estimates come from?

Certainty equivalence

- MC batch learning converges to the least squares fit of the sampled returns:

$$\sum_{j=1}^{J}\sum_{k=1}^{T_j}\left(g_{k,j}-\hat{v}(x_{k,j})\right)^2. \tag{5.7}$$

- TD batch learning converges to the maximum likelihood estimate such that $\left\langle \mathcal{X},\mathcal{U},\hat{\mathcal{P}},\hat{\mathcal{R}},\gamma \right\rangle$ explains the data with highest probability:

$$\begin{aligned}\hat{p}^u_{xx'} &= \frac{1}{n(x,u)}\sum_{j=1}^{J}\sum_{k=1}^{T_j}1(X_{k+1}=x'|X_k=x,U_k=u),\\ \hat{\mathcal{R}}^u_x &= \frac{1}{n(x,u)}\sum_{j=1}^{J}\sum_{k=1}^{T_j}1(X_k=x|U_k=u)r_{k+1,j}.\end{aligned} \tag{5.8}$$

- Here, TD assumes a MDP problem structure and is absolutely certain that its internal model concept describes the real world perfectly (so-called certainty equivalence).

Table of contents

1. Temporal-difference prediction
2. Temporal-difference on-policy control: SARSA
3. Temporal-difference off-policy control: Q-learning
4. Maximization bias and double learning

Applying generalized policy iteration (GPI) to TD control

GPI concept is directly applied to the TD framework using action values:

$$\pi_0 \to \hat{q}_{\pi_0} \to \pi_1 \to \hat{q}_{\pi_1} \to \cdots \pi^* \to \hat{q}_{\pi^*} \,. \tag{5.9}$$

One-step TD / TD(0) action-value update (SARSA)

The TD(0) action-value update is:

$$\hat{q}(x_k, u_k) \leftarrow \hat{q}(x_k, u_k) + \alpha \left[r_{k+1} + \gamma \hat{q}(x_{k+1}, u_{k+1}) - \hat{q}(x_k, u_k) \right]. \tag{5.10}$$

SARSA: state, action, reward, (next) state, (next) action evaluation

- In contrast to MC: continuous online updates of policy evaluation and improvement.
- On-policy approach requires exploration, e.g., by an ε-greedy policy:

$$\pi_i(u|x) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{U}|, & u = \tilde{u}, \\ \varepsilon/|\mathcal{U}|, & u \neq \tilde{u}. \end{cases} \tag{5.11}$$

TD-based on-policy control (SARSA)

parameter: $\varepsilon \in \{\mathbb{R}|0 < \varepsilon << 1\}$, $\alpha \in \{\mathbb{R}|0 < \alpha < 1\}$
init: $\hat{q}(x, u)$ arbitrarily (except terminal states) $\forall\ \{x \in \mathcal{X}, u \in \mathcal{U}\}$
for $j = 1, 2, \ldots$ *episodes* **do**
    Initialize x_0;
    Choose u_0 from x_0 using a soft policy (e.g., ε-greedy) derived from $\hat{q}(x, u)$;
    $k \leftarrow 0$;
    repeat
        Take action u_k, observe r_{k+1} and x_{k+1};
        Choose u_{k+1} from x_{k+1} using a soft policy derived from $\hat{q}(x, u)$;
        $\hat{q}(x_k, u_k) \leftarrow \hat{q}(x_k, u_k) + \alpha\left[r_{k+1} + \gamma \hat{q}(x_{k+1}, u_{k+1}) - \hat{q}(x_k, u_k)\right]$;
        $k \leftarrow k + 1$;
    until x_k *is terminal*;

Algo. 5.2: TD-based on-policy control (SARSA)

Convergence properties are comparable to MC-based on-policy control:

- Policy improvement theorem Theo. 4.1 holds.
- Greedy in the limit with infinite exploration (GLIE) from Def. 4.1 and step-size requirements in Theo. 5.1 apply.

SARSA example: forest tree MDP (1)

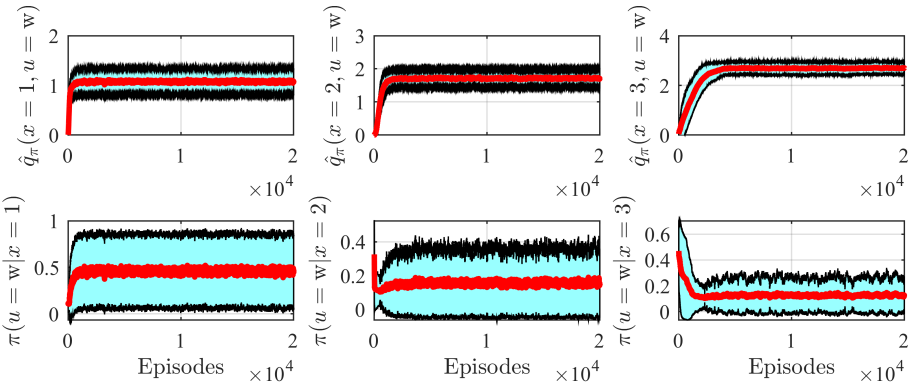

Fig. 5.9: SARSA-based control with $\alpha_{\mathrm{SARSA}} = 0.2$ and ε-greedy policy with $\varepsilon = 0.2$ of forest tree MDP over the number of episodes being evaluated (mean and standard deviation are calculated based on 2000 independent runs)

SARSA example: forest tree MDP (2)

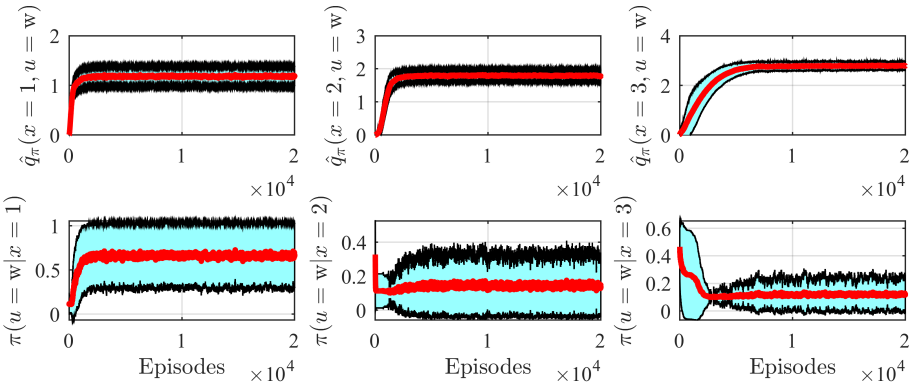

Fig. 5.10: SARSA-based control with $\alpha_{\text{SARSA}} = 0.1$ and ε-greedy policy with $\varepsilon = 0.2$ of forest tree MDP over the number of episodes being evaluated (mean and standard deviation are calculated based on 2000 independent runs)

SARSA example: forest tree MDP (3)

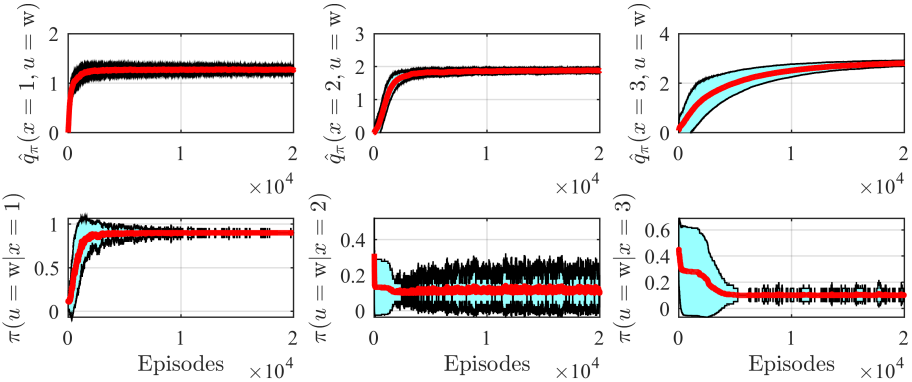

Fig. 5.11: SARSA-based control with adaptive $\alpha_{\mathrm{SARSA}} = \frac{1}{\sqrt{j}}$ (j =episode) and ε-greedy policy with $\varepsilon = 0.2$ of forest tree MDP over the number of episodes being evaluated (mean and standard deviation are calculated based on 2000 independent runs)

Table of contents

1 Temporal-difference prediction

2 Temporal-difference on-policy control: SARSA

3 Temporal-difference off-policy control: Q-learning

4 Maximization bias and double learning

Q-learning approach

Similar to SARSA updates, but Q-learning directly estimates q^*:

Q-learning action-value update

The Q-learning action-value update is:

$$\hat{q}(x_k, u_k) \leftarrow \hat{q}(x_k, u_k) + \alpha \left[r_{k+1} + \gamma \max_u \hat{q}(x_{k+1}, u) - \hat{q}(x_k, u_k) \right]. \tag{5.12}$$

This is an off-policy update, since the optimal action-value function is updated independent of a given behavior policy.

Requirement for Q-learning control:

- Coverage: behavior policy b has nonzero probability of selecting actions that might be taken by the target policy π.
- Consequence: behavior policy b is soft (e.g., ε-soft).
- Step-size requirements (5.5) regarding α apply.

TD-based off-policy control (Q-learning)

parameter: $\varepsilon \in \{\mathbb{R}|0 < \varepsilon << 1\}$, $\alpha \in \{\mathbb{R}|0 < \alpha < 1\}$
init: $\hat{q}(x, u)$ arbitrarily (except terminal states) $\forall \{x \in \mathcal{X}, u \in \mathcal{U}\}$
for $j = 1, 2, \ldots$ *episodes* **do**
    Initialize x_0;
    $k \leftarrow 0$;
    repeat
        Choose u_k from x_k using a soft behavior policy;
        Take action u_k, observe r_{k+1} and x_{k+1};
        $\hat{q}(x_k, u_k) \leftarrow \hat{q}(x_k, u_k) + \alpha \left[r_{k+1} + \gamma \max_u \hat{q}(x_{k+1}, u) - \hat{q}(x_k, u_k)\right]$;
        $k \leftarrow k + 1$;
    until x_k *is terminal*;

Algo. 5.3: TD-based off-policy control (Q-learning)

- As discussed with MC-based off-policy control: avoidance of the exploration-optimality trade-off for on-policy methods.
- No importance sampling required as for off-policy MC-based control.

Q-learning control example: cliff walking

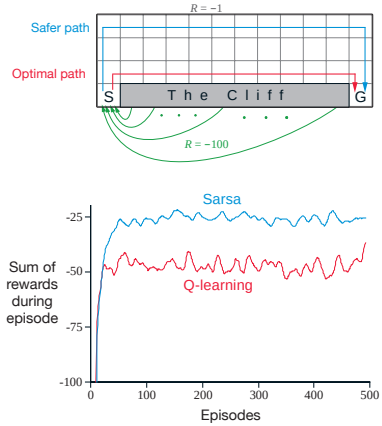

Fig. 5.12: Cliff walking environment (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018, CC BY-NC-ND 2.0)

- $r = -1$ per time step
- Large penalty if you fall off the cliff
- No discounting
- $\varepsilon = 0.1$

- Why is SARSA better in this example?
- And what policy's performance is shown here in particular?

Table of contents

1. Temporal-difference prediction
2. Temporal-difference on-policy control: SARSA
3. Temporal-difference off-policy control: Q-learning
4. Maximization bias and double learning

Maximization bias

All control algorithms discussed so far involve maximization operations:

- Q-learning: target policy is greedy and directly uses $\max$ operator for action-value updates.
- SARSA: typically uses an ε-greedy framework, which also involves $\max$ updates during policy improvement.

This can lead to a significant positive bias:

- Maximization over sampled values is used implicitly as an estimate of the maximum value.
- This issue is called maximization bias.

Small example:

- Consider a single state x with multiple possible actions u.
- The true action values are all $q(x, u) = 0$.
- The sampled estimates $\hat{q}(x, u)$ are uncertain, i.e., randomly distributed. Some samples are above and below zero.
- Consequence: The maximum of the estimate is positive.

Double learning approach

Split the learning process:

- Divide sampled experience into two sets.
- Use sets to estimate independent estimates $\hat{q}_1(x, u)$ and $\hat{q}_2(x, u)$.

Assign specific tasks to each estimate:

- Estimate the maximizing action:

$$u^* = \arg\max_u \hat{q}_1(x, u). \tag{5.13}$$

- Estimate corresponding action value:

$$q(x, u^*) \approx \hat{q}_2(x, u^*) = \hat{q}_2(x, \arg\max_u \hat{q}_1(x, u)). \tag{5.14}$$

Double Q-learning algorithm

parameter: $\varepsilon \in \{\mathbb{R}|0 < \varepsilon << 1\}$, $\alpha \in \{\mathbb{R}|0 < \alpha < 1\}$
init: $\hat{q}_1(x,u)$, $\hat{q}_2(x,u)$ arbitrarily (except terminal states) $\forall\ \{x \in \mathcal{X}, u \in \mathcal{U}\}$
for $j = 1, 2, \ldots$ *episodes* **do**
    Initialize x_0;
    $k \leftarrow 0$;
    repeat
        Choose u_k from x_k using the policy ε-greedy based on $\hat{q}_1(x,u) + \hat{q}_2(x,u)$;
        Take action u_k, observe r_{k+1} and x_{k+1};
        if $n \sim \mathcal{N}(\mu = 0, \sigma) > 0$ **then**
            $\hat{q}_1(x_k,u_k) \leftarrow \hat{q}_1(x_k,u_k) + \alpha\left[r_{k+1} + \gamma \hat{q}_2(x_{k+1}, \arg\max_u \hat{q}_1(x_{k+1},u)) - \hat{q}_1(x_k,u_k)\right]$;
        else
            $\hat{q}_2(x_k,u_k) \leftarrow \hat{q}_2(x_k,u_k) + \alpha\left[r_{k+1} + \gamma \hat{q}_1(x_{k+1}, \arg\max_u \hat{q}_2(x_{k+1},u)) - \hat{q}_2(x_k,u_k)\right]$;
        $k \leftarrow k + 1$;
    until x_k *is terminal*;

Algo. 5.4: TD-based off-policy control with double learning

- Doubles memory demand while computational demand per episode is remains unchanged
- Less sample efficient than regular Q-learning (samples are split between two estimators)

Maximization bias example

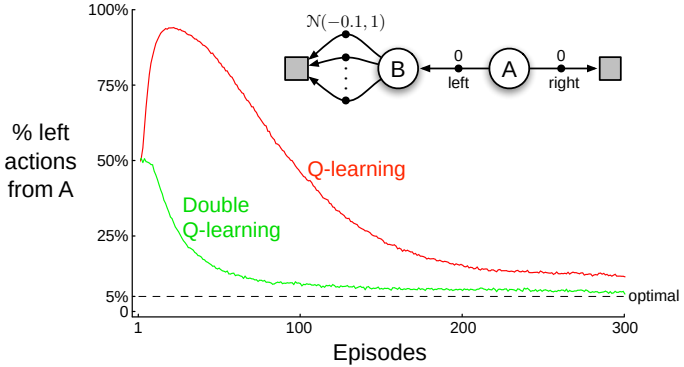

Fig. 5.13: Comparison of Q-learning and double Q-learning on a simple episodic MDP. Q-learning initially learns to take the left action much more often than the right action, and always takes it significantly more often than the 5% minimum probability enforced by ε-greedy action selection with $\varepsilon = 0.1$. In contrast, double Q-learning is essentially unaffected by maximization bias. These data are averaged over 10,000 runs. The initial action-value estimates were zero. (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018, CC BY-NC-ND 2.0)

Summary: what you've learned today

- TD unites two key characteristics from DP and MC:
  - From MC: Sample-based updates (i.e., operating in unknown MDPs).
  - From DP: Update estimates based on other estimates (bootstrapping).
- TD allows certain simplifications and improvements compared to MC:
  - Updates are available after each step and not after each episode.
  - Off-policy learning comes without importance sampling.
  - Exploits MDP formalism by maximum likelihood estimates.
  - Hence, TD prediction and control exhibit a high applicability for many problems.
- Batch training can be used when only limited experience is available, i.e., the available samples are re-processed again and again.
- Greedy policy improvements can lead to maximization biases and, therefore, slow down the learning process.
- TD requires careful tuning of learning parameters:
  - Step size α: how to tune convergence rate vs. uncertainty / accuracy?
  - Exploration vs. exploitation: how to visit all state-action pairs?