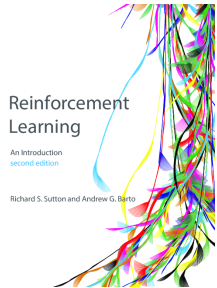
Reinforcement Learning: An Introduction

R. Sutton and A. Barto (2018)

M2 DS - Fall 2021

Outline

The Book



Reinforcement Learning, an Introduction

- R. Sutton and A. Barto
- 2nd edition!
- Available at `http://incompleteideas.net/book/the-book-2nd.html`

Outline

1 Introduction

1.5 An Extended Example: Tic-Tac-Toe - Figure 1.1

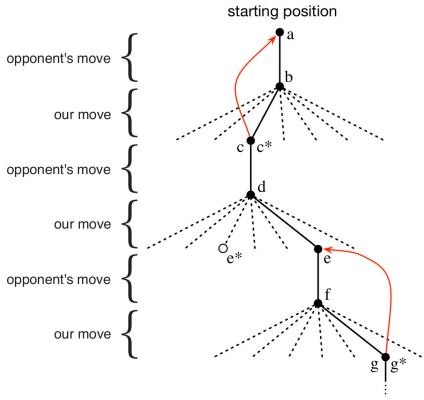


Figure 1.1: A sequence of tic-tac-toe moves. The solid black lines represent the moves taken during a game; the dashed lines represent moves that we (our reinforcement learning player) considered but did not make. Our second move was an exploratory move, meaning that it was taken even though another sibling move, the one leading to e^*, was ranked higher. Exploratory moves do not result in any learning, but each of our other moves does, causing updates as suggested by the red arrows in which estimated values are moved up the tree from later nodes to earlier nodes as detailed in the text.

Outline

2 Multi-armed Bandits

2.1 A k-armed Bandit Problem

- k-armed bandit:
  - $s = \emptyset$
  - $q_*(a) = \mathbb{E}[R_t | A_a] = r_a$
- Optimal policy: $a_* = \text{argmax}\, r_a$
- Estimate $Q_t(a) \sim \mathbb{E}[R_t | A_a]$.
- Next step:
  - Exploitation: $A_t = \text{argmax}\, Q_t(a)$, bet on the current winner.
  - Exploration: $A_t \neq \text{argmax}\, Q_t(a)$, verify that no other arm are better.
- Conflict between exploration and exploitation.
- Theoretical results under strong assumptions.

2.2 Action-value Methods

- Action-value:
$$Q_t(a) = \frac{\sum_{i=1}^{t-1} \mathbf{1}_{A_i=a} R_i}{\sum_{i=1}^{t-1} \mathbf{1} A_i = a}$$
- Sample average that converges to $q_*(a)$ provided $\sum_{i=1}^{t-1} \mathbf{1}_{A_i=a} \to \infty$
- Greedy action:
$$A_t = \text{argmax}\, Q_t(a)$$
- ϵ-greedy action:
$$A_t = \begin{cases} \text{argmax}\, Q_t(a) & \text{with probability } 1-\epsilon \\ A' & \text{with A' uniform on the arm otherwise} \end{cases}$$
- ϵ-greedy forces exploration and guarantees that $Q_t(a) \to a_*(a)$

2.3 The 10-armed Testbed

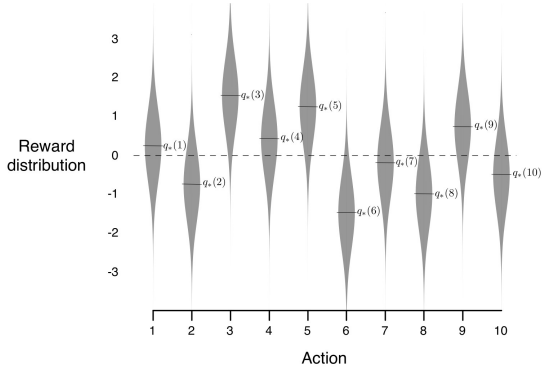


Figure 2.1: An example bandit problem from the 10-armed testbed. The true value $q_*(a)$ of each of the ten actions was selected according to a normal distribution with mean zero and unit variance, and then the actual rewards were selected according to a mean $q_*(a)$ unit variance normal distribution, as suggested by these gray distributions.

- $q_*(a) \sim \mathcal{N}(0, 1)$
- $R_t | A_t = a \sim \mathcal{N}(q_*(a), 1)$

2.3 The 10-armed Testbed

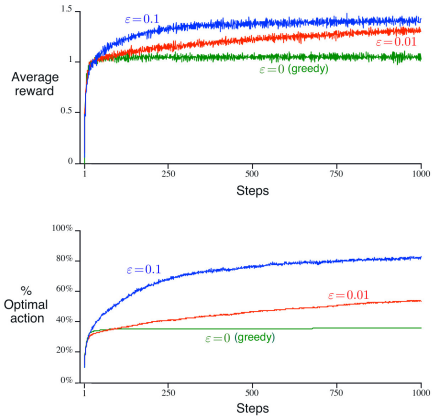


Figure 2.2: Average performance of ε-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 runs with different bandit problems. All methods used sample averages as their action-value estimates.

- Greedy policy may fail in a stationary context.
- Even more risky in a nonstationary one.

2.4 Incremental Implementation

- Mean for a single action:

$$Q_n = \frac{R_1 + \cdots + R_{n-1}}{n-1}$$

- Incremental implementation:

$$\begin{aligned} Q_{n+1} &= \frac{R_1 + \cdots + R_n}{n} \\ &= \frac{1}{n}\left((n-1)Q_n + R_n\right) \\ &= Q_n + \frac{1}{n}\left(R_n - Q_n\right) \end{aligned}$$

- General update rule:

$$\mathit{NewEstimate} = \mathit{OldEstimate} + \mathit{StepSize}\left(\mathit{Target} - \mathit{OldEstimate}\right)$$

where

  - *Target* is a noisy estimate of the true target,
  - *StepSize* may depends on t and a.

2.4 Incremental Implementation

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$

2.5 Tracking a Nonstationary Problem

- Nonstationary setting often present in reinforcement learning.
- Incremental update:

$$Q_{n+1} = Q_n + \alpha\,(R_n - Q_n)$$

- If $\alpha \in (0, 1]$ is constant,

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha\,(R_n - Q_n) \\ &= (1-\alpha)Q_{n-1} + \alpha R_n \\ &= (1-\alpha)^2 Q_{n-1} + \alpha(1-\alpha)R_{n-1} + \alpha R_n \\ &= (1-\alpha)^n Q_1 + \sum_{k=1}^{n} \alpha(1-\alpha)^{n-i} R_i \end{aligned}$$

- Weighted average with more emphasis on later values.

2.5 Tracking a Nonstationary Problem

- Incremental update:

$$Q_{n+1} = Q_n + \alpha_n \left(R_n - Q_n\right)$$

- Convergence toward the expectation of R requires some assumptions on α:

$$\sum_{n=1}^{\infty} \alpha_n = +\infty \quad \sum_{n=1}^{\infty} \alpha_n^2 < +\infty$$

  - First condition guarantees that one can escape any initial condition.
  - Second condition that the iterations converges.

- If $\alpha_n = \alpha$ no convergence but track any nonstationarity.

2.6 Optimistic Initial Values

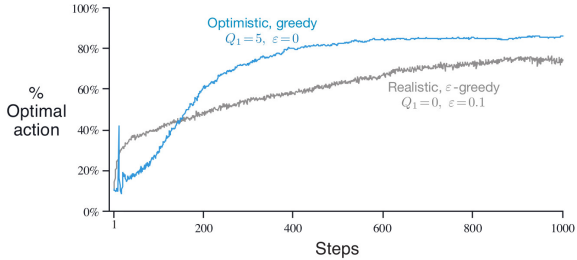


Figure 2.3: The effect of optimistic initial action-value estimates on the 10-armed testbed. Both methods used a constant step-size parameter, $\alpha = 0.1$.

- Estimate depends on initial values (except for the case where $\alpha_1 = 1$).
- Way of supplying prior knowledge about the level of rewards expected.
- Optimistic initialization leads to exploration at the beginning.
- Fails to help in a nonstationary setting.

2.7 Upper-Confidence-Bound Action Selection

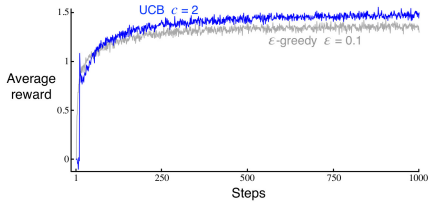


Figure 2.4: Average performance of UCB action selection on the 10-armed testbed. As shown, UCB generally performs better than ε-greedy action selection, except in the first k steps, when it selects randomly among the as-yet-untried actions.

- ϵ-greedy fails to discriminate between good/bad actions or certain/uncertain actions.
- Upper-Confidence-Bound:

$$A_t = \text{argmax} \left(Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right)$$

- Arm with lower values estimates will be selected with decreasing frequency over time.
- Bandit proof hard to extend to reinforcement setting.

2.8 Gradient Bandit Algorithm

- Numerical preference associated to action a: $H_t(a)$.
- Induced soft-max policy:

$$\mathbb{P}(A_t = a) = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} = \pi_t(a)$$

- Natural learning algorithm with update

$$H_{t+1}(a) = \begin{cases} H_t(a) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(a)) & \text{if } a = A_t \\ H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a) & \text{otherwise} \end{cases}$$

  with $\bar{R}_t = (\sum_{i=1}^{t-1} R_i)$
- Baseline $\bar{R}_t$ accelerates convergence.

2.8 Gradient Bandit Algorithm - Figure 2.5

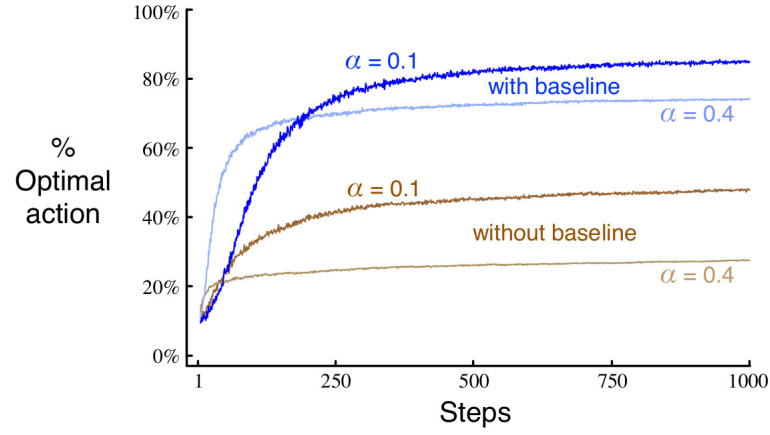


Figure 2.5: Average performance of the gradient bandit algorithm with and without a reward baseline on the 10-armed testbed when the $q_*(a)$ are chosen to be near $+4$ rather than near zero.

2.8 Gradient Bandit Algorithm

- Ideal gradient ascent:

$$
\begin{aligned}
H_{t+1}(a) &= H_t(a) + \alpha \frac{\partial \mathbb{E}\left[R_t\right]}{\partial H_t(a)} \\
&= H_t(a) + \alpha \sum_b \frac{\partial \pi_t(b)}{\partial H_t(a)} q_*(b) \\
&= H_t(a) + \alpha \sum_b \frac{\partial \pi_t(b)}{\partial H_t(a)} \left(q_*(b) - B_t\right) \\
&= H_t(a) + \alpha \sum_b \pi_t(b) \frac{\partial \ln \pi_t(b)}{\partial H_t(a)} \left(q_*(b) - B_t\right) \\
&= H_t(a) + \alpha \mathbb{E}_{\pi_t} \left[\frac{\partial \ln \pi_t(A)}{\partial H_t(a)} \left(q_*(A) - B_t\right)\right]
\end{aligned}
$$

- Stochastic gradient descent:

$$
H_{t+1}(a) = H_t(a) + \alpha \frac{\partial \ln \pi_t(A_t)}{\partial H_t(a)} \left(q_*(A_t) - B_t\right)
$$

2.8 Gradient Bandit Algorithm

- Policy gradient:

$$\begin{aligned}\frac{\partial \ln \pi_t(b)}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \left(H_t(b) - \ln \left(\sum_{b'} e^{H_t(b')} \right) \right) \\ &= \mathbf{1}_{a=b} - \frac{e^{H_t(a)}}{\sum_{b'} e^{H_t(b')}} \\ &= \mathbf{1}_{a=b} - \pi_t(a)\end{aligned}$$

- Stochastic gradient descent:

$$H_{t+1}(a) = H_t(a) + \alpha \left(q_*(A_t) - B_t \right) \left(\mathbf{1}_{a=b} - \pi_t(a) \right)$$

2.9 Associative Search (Contextual Bandits)

- Associative search: reward depends on the arm and on the situation.
- Often call contextual bandits.
- In between bandits and reinforcement learning, the action only impact the next reward

2 Multi-armed bandits - Figure 2.6

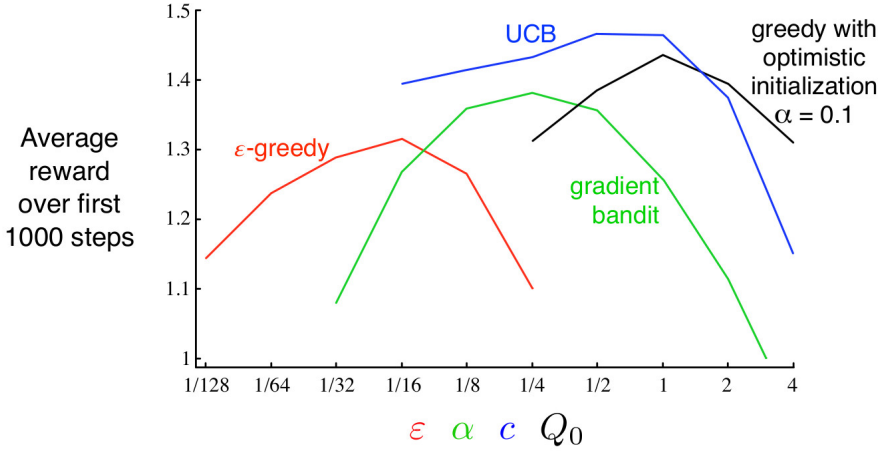


Figure 2.6: A parameter study of the various bandit algorithms presented in this chapter. Each point is the average reward obtained over 1000 steps with a particular algorithm at a particular setting of its parameter.

Outline

3 Finite Markov Decision Processes

- 3.1 The Agent-Environment Interface
- 3.2 Goals and Rewards
- 3.3 Returns and Episodes
- 3.4 Unified Notation for Episodic and Continuing Tasks
- 3.5 Policies and Value Functions
- 3.6 Optimal Policies and Optimal Value Functions
- 3.7 Optimality and Approximation

3.1 The Agent-Environment Interface

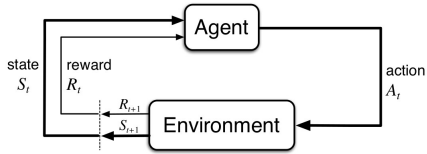


Figure 3.1: The agent–environment interaction in a Markov decision process.

- At time step $t \in \mathcal{N}$:
  - State $S_t \in \mathcal{S}$: representation of the environment
  - Action $A_t \in \mathcal{A}(S_t)$: action chosen
  - Reward $R_{t+1} \in \mathcal{R}$: instantaneous reward
  - New state S_{t+1}
- Finite MDP:
  - $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ are finite.
  - Dynamic entirely defined by

$$\mathbb{P}\left(S_t = s', R_r = r | S_{t-1} = s, A_{t-1} = a\right) = p(s', r|s, a)$$

3.1 The Agent-Environment Interface - Figure 3.1

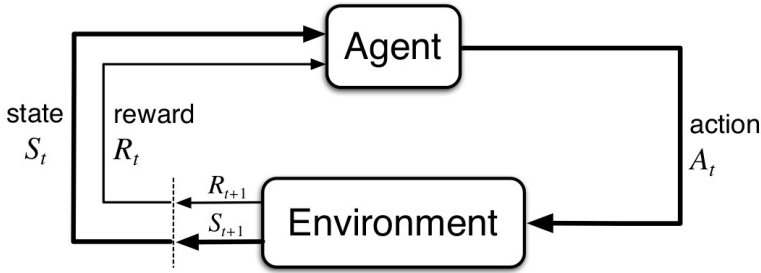


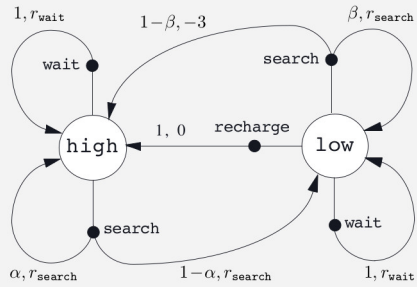
Figure 3.1: The agent–environment interaction in a Markov decision process.

3.1 The Agent-Environment Interface

- State-transition probabilities:
$$p(s'|s,a) = \mathbb{P}\left(S_t = s' | S_{t-1} = s, A_{t-1} = a\right) = \sum_r p(s', r|s, a)$$
- Expected reward for a state-action:
$$r(s,a) = \mathbb{E}\left[R_t | S_{t-1} = s, A_{t-1} = a\right] = \sum_r r \sum_{s'} p(s', r|s, a)$$
- Expected reward for a state-action-state:
$$r(s,a,s') = \mathbb{E}\left[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'\right] = \sum_r r \frac{p(s', r|s, a)}{p(s'|s, a)}$$

3.1 The Agent-Environment Interface

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1-\alpha$	$r_{\texttt{search}}$
low	search	high	$1-\beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	-



- Examples:
  - Bioreactor
  - Pick-and-Place Robots
  - Recycling Robot

3.2 Goals and Rewards

That 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).

- The reward signal is your way of communicating to the robot what you want it to achieve, not how you want it achieved.

3.3 Returns and Episodes

- Episodic: Final time step T and
$$G_t = \sum_{t'=t+1}^{T} R_{t'}$$
- Continuous tasks: undiscounted reward
$$G_t = \sum_{t'=t+1}^{+\infty} R_{t'} \quad \text{may not exist!}$$
- Continuous tasks: discounted reward
$$G_t = \sum_{0}^{+\infty} \gamma^k R_{t+1+k}$$
with $0 \leq \gamma < 1$.

3.3 Returns and Episodes

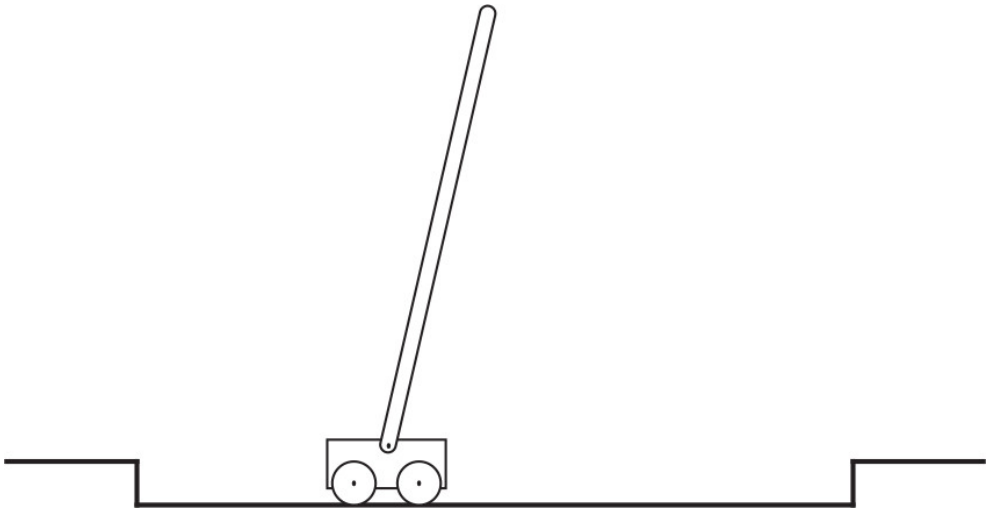
- Recursive property

$$G_t = R_{t+1} + \gamma G_{t+1}$$

- Finiteness if $|R| \leq M$

$$|G_t| \leq \begin{cases} (T - t - +1)M & \text{if } T < \infty \\ M\frac{1}{1-\gamma} & \text{otherwise} \end{cases}$$

3.3 Returns and Episodes - Example 3.4



Unified Notation for Episodic and Continuing Tasks

- Episodic case: several episodes instead of a single trajectory ($S_{t,i} \dots$)
- Absorbing state $\tilde{s}$ such $p(\tilde{s}|\tilde{s}, a) = 1$ and $R_t = 0$ when $S_t = \tilde{s}$.
- Convert episodic case into a continuing one.
- Alternative: notation

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

- Undefined if $T = \infty$ and $\gamma = 1 \dots$

3.5 Policies and Value Functions

- Policy: $\pi(a|s)$
- Value function:

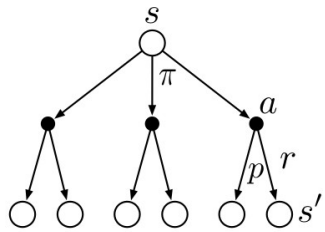
$$v_\pi(s) = \mathbb{E}_\pi\left[G_t|S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s\right]$$

- Action value function:

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

- *Implicit stationary assumption on π!*

3.5 Policies and Value Functions

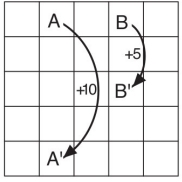


Backup diagram for v_π

- Bellman Equation

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

3.5 Policies and Value Functions - Figure 3.2



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

Figure 3.2: Gridworld example: exceptional reward dynamics (left) and state-value function for the equiprobable random policy (right).

3.5 Policies and Value Functions - Figure 3.3

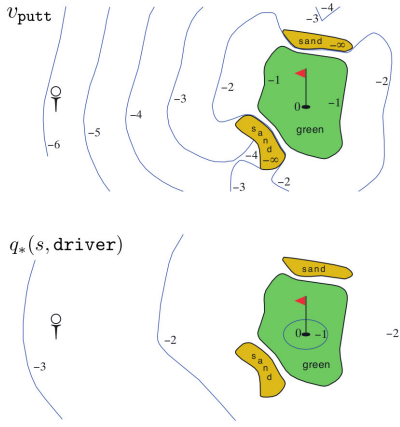


Figure 3.3: A golf example: the state-value function for putting (upper) and the optimal action-value function for using the driver (lower). ■

3.6 Optimal Policies and Optimal Value Functions

- Optimal policies: $v_{\pi_*}(s) \geq v_\pi(s)$ (not necessarily unique)
- Optimal state-value function :

$$v_*(s) = \max_\pi v_\pi(s) \quad \text{Uniqueness}$$

- Optimal state-action-value function:

$$q_*(s,a) = \max_\pi q_\pi(s,a) \quad \text{Uniqueness}$$

- Link:

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right]$$

3.6 Optimal Policies and Optimal Value Functions

- Bellman optimality equation:

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

- Bellman optimality equation for q:

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

3.6 Optimal Policies and Optimal Value Functions - Figure 3.4

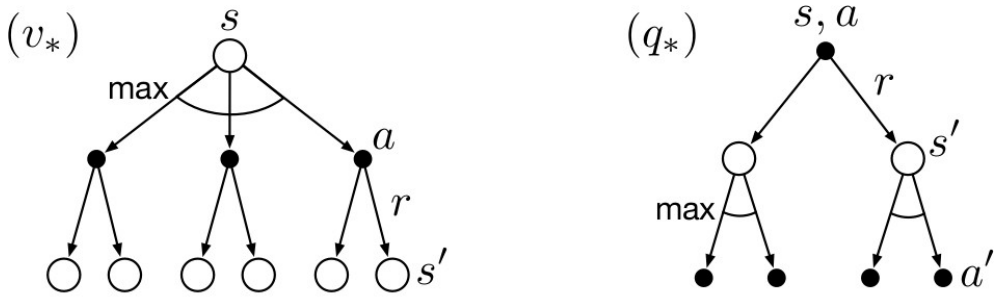
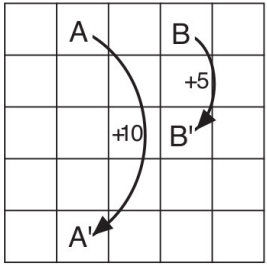


Figure 3.4: Backup diagrams for v_* and q_*

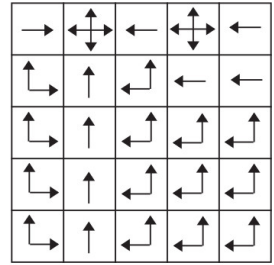
3.6 Optimal Policies and Optimal Value Functions - Figure 3.5



Gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*



π_*

Figure 3.5: Optimal solutions to the gridworld example.

3.7 Optimality and Approximation

- Very difficult to learn the optimal policy.
- Knowing the environment helps but is not sufficient. (Chap. 4)
- Computational challenges even in the finite case! (Chap. 5-8)
- Need to resort to approximation! (Chap. 9-12)

Outline

4 Dynamic Programming

4.1 Policy Evaluation

- Policy Evaluation or Prediction
- Bellman Equation

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

- Linear system that can be solved!
- Bellman iteration:

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

- v_π is a fixed point.
- Iterative policy evaluation: iterative algorithm that can be proved to converge.

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 $\Delta < \theta$

Example 4.1 Consider the 4×4 gridworld shown below.

actions

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

$R_t = -1$
on all transitions

4.2 Policy Improvement

- If π' is such that $\forall s, q_\pi(s, \pi'(s)) \geq v_\pi(s)$ then $v_{\pi'} \geq v_\pi$.
- Sketch of proof:

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)\right] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})\right] | S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \middle| S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \middle| S_t = s\right] \\
&\leq v_{\pi'}(s)
\end{aligned}
$$

- Greedy update:

$$
\begin{aligned}
\pi'(s) &= \operatorname*{argmax}_a q_\pi(s, a) \\
&= \operatorname*{argmax}_a \sum_{s', r} p(s', r | s, a)\left(r + \gamma v_\pi(s')\right)
\end{aligned}
$$

- If $\pi' = \pi$ after a greedy update $v_{\pi'} = v_\pi = v_*$.

4.2 Policy Improvement - Figure 4.1

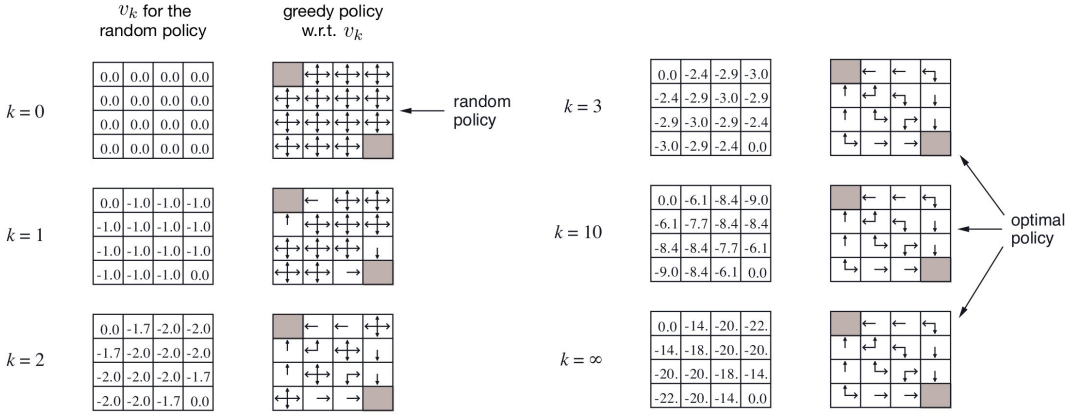


Figure 4.1: Convergence of iterative policy evaluation on a small gridworld. The left column is the sequence of approximations of the state-value function for the random policy (all actions equally likely). The right column is the sequence of greedy policies corresponding to the value function estimates (arrows are shown for all actions achieving the maximum, and the numbers shown are rounded to two significant digits). The last policy is guaranteed only to be an improvement over the random policy, but in this case it, and all policies after the third iteration, are optimal.

4.3 Policy Iteration

- Policy iteration: sequence of policy evaluation and policy improvement

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

- In a finite states/actions setting, converges in finite time.

4.3 Policy Iteration - 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 $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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 $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$
   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

4.3 Policy Iteration - Figure 4.2

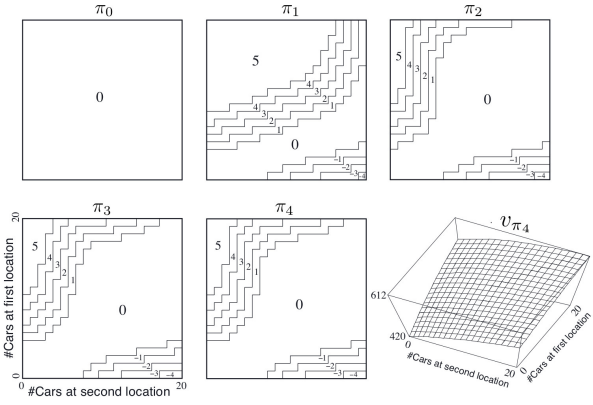


Figure 4.2: The sequence of policies found by policy iteration on Jack's car rental problem, and the final state-value function. The first five diagrams show, for each number of cars at each location at the end of the day, the number of cars to be moved from the first location to the second (negative numbers indicate transfers from the second location to the first). Each successive policy is a strict improvement over the previous policy, and the last policy is optimal. ■

4.4 Value Iteration

- Policy evaluation can be time consuming.
- Value iteration: improve policy after only one step of policy evaluation.
- Bellman iteration:

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

- Update corresponds to the Bellman optimality equation.
- Variation possible on the number of steps in the policy evaluation.

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

4.4 Value Iteration - Figure 4.3

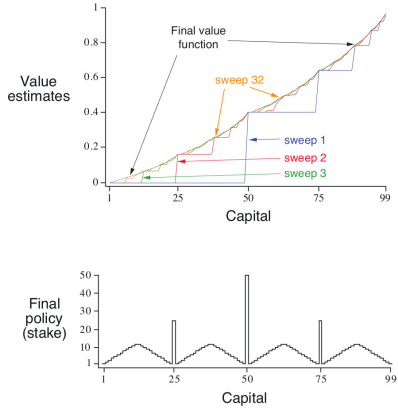
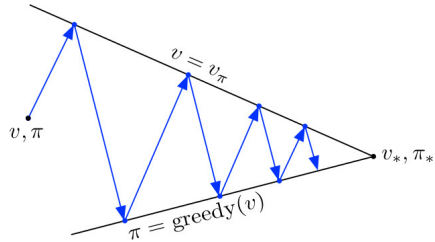


Figure 4.3: The solution to the gambler's problem for $p_h = 0.4$. The upper graph shows the value function found by successive sweeps of value iteration. The lower graph shows the final policy.

4.5 Asynchronous Dynamic Programming

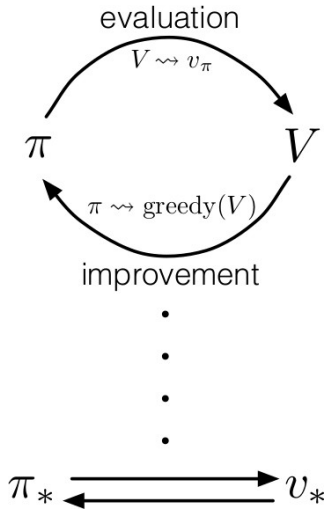
- Synchronous Dynamic Programming: update all states at each step.
- Asynchronous Dynamic Programming: update only a few states at each step.
- No systematic sweeps of the state set!
- Only need to update every state infinitely often!
- One possibility is to update the states seen by an agent experiencing the MDP.

4.6 Generalized Policy Iteration



- Policy iteration consists of two simultaneous interacting processes:
  - one making a value function consistent with the current policy (policy evaluation)
  - one making the policy greedy with respect to the current value function (policy improvement)
- Generalized Policy Iteration: any similar scheme.
- Stabilizes only if one reaches the optimal value/policy pair.

4.6 Generalized Policy Iteration - Generalized Policy Iteration



4.7 Efficiency of Dynamic Programming

- DP quite efficient: polynomial in $|\mathcal{S}|$ and $|\mathcal{A}|$.
- Linear programming alternative also possible.
- Curse of dimensionality if the as the number of states grows exponentially with the number of state variables.
- Asynchronous DP methods are preferred.

Outline

5 Monte Carlo Methods

5.1 Monte Carlo Prediction

- Estimate $v_\pi(s)$ by the average gain following π after passing through s.
- Two variants:
  - First-visit: use only first visit of s in each episode
  - Every-visit: use every visit of s in each episode
- First-visit is easier to analyze due to independence of each episode.
- Every-visit works. . . but not necessarily better!

5.1 Monte Carlo Prediction - First Visit Monte-Carlo Policy Evaluation

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 $Returns(S_t)$

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

5.1 Monte Carlo Prediction

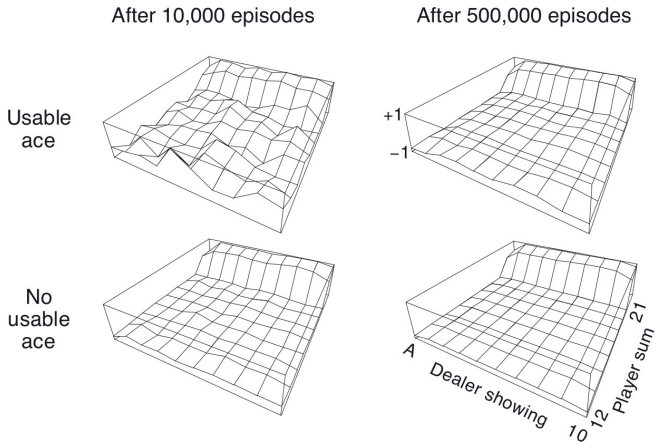
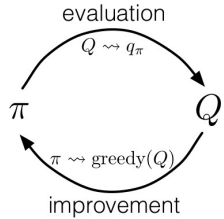


Figure 5.1: Approximate state-value functions for the blackjack policy that sticks only on 20 or 21, computed by Monte Carlo policy evaluation. ■

5.2 Monte Carlo Estimation of Action Values

- Without a model, $v_\pi(s)$ is not sufficient to do a policy enhancement step.
- Need to estimate $q_\pi(s, a)$ directly.
- **Issue:** require that any state-action pair is visited.
- Impossible with a deterministic policy.
- Instance of problem of *maintaining exploration* seen with the bandits.
- Exploring starts: start the game from a random stat-action pair in a way that every stat-action pair has a nonzero probability.
- Alternative: impose condition on policy itself.

5.3 Monte Carlo Control



- Generalized Policy Iteration can be implemented with MC.
- Scheme:

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

- Infinite number of MC simulations to compute q_π.
- Easy policy improvement:

$$\pi(s) = \text{argmax}\, q(s, a)$$

- Improvement at each step hence convergence...

5.3 Monte Carlo Control

- Two strong assumptions:
  - Every state-action pair is visited infinitely often (Exploring Starts)
  - Infinite number of MC simulations
- Approximate policy iteration required.
- First approach: if the number of MC simulations is large enough then the approximation is good enough...
- The required number of simulations can be very large.
- Second approach: use the current MC estimation to update a current Q-value estimate (GPI).

5.3 Monte Carlo Control

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$

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

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

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0

Generate an episode from S_0, A_0, 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 $Returns(S_t, A_t)$

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

$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

- Cannot converge to any suboptimal policy.
- Convergence still an open question. . .

5.3 Monte Carlo Control - Figure 5.2

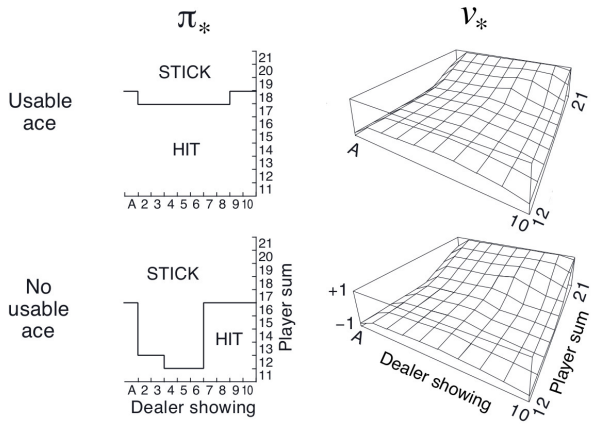


Figure 5.2: The optimal policy and state-value function for blackjack, found by Monte Carlo ES. The state-value function shown was computed from the action-value function found by Monte Carlo ES. ■

5.4 Monte Carlo Control without Exploring Starts

- Exploring starts assumption can be removed if one guarantees that the agent selects every action infinitely often.
- Two approaches:
  - on-policy, where the policy is constrained to explore.
  - off-policy, where the agent use a different policy than the one we want to estimate
- On-policy control: use *soft* policy such that $\pi(a|s) > 0$ but gradually shift closer and closer to a deterministic optimal policy.
- Impossible to use the classical policy improvement step.

5.4 Monte Carlo Control without Exploring Starts

- Use of ϵ-greedy rules:

$$\pi(s) = \begin{cases} \text{argmax}\, q(s,a) & \text{with probability } 1-\epsilon+\epsilon/|\mathcal{A}(s)| \\ a' \neq \text{argmax}\, q(s,a) & \text{with probability } \epsilon/|\mathcal{A}(s)| \end{cases}$$

- Improvement over any ϵ-greedy policy:

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

- Fixed point should be an optimal ϵ-greedy policy.

5.4 Monte Carlo Control without Exploring Starts

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

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

- In practice, one reduces ϵ during the iterations.

5.5 Off-policy Prediction via Importance Sampling

- Two approaches:
  - on-policy, where the policy is constrained to explore.
  - off-policy, where the agent use a different policy than the one we want to estimate
- target policy (policy we want to estimate) vs behavior policy (policy we use to explore)
- on-policy: simpler
- off-policy: more complex but more powerful and general.
- Example: off-policy can be used to learn from observation.
- Focus now on prediction: estimation of v_π or q_π while having episode following policy $b \neq \pi$.
- Minimum (coverage) requirement: if $\pi(a|s) > 0$ then $b(a|s) > 0$.
- We may have $\pi(a|s) = 0$ and $b(a|s) > 0 \ldots$
- Typically, b is an ϵ-greedy policy.

5.5 Off-policy Prediction via Importance Sampling

- Most off-policy methods are base on importance sampling:

$$\mathbb{E}_\pi [f(X)] = \mathbb{E}_q \left[\frac{\pi(X)}{q(X)} f(X) \right]$$

- By construction,

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

- Relative probability ratio:

$$\rho_{t:T-1} = \frac{\mathbb{P}(A_t, S_{t+1}, A_{t+1}, \dots, S_T | S_t, \pi)}{\mathbb{P}(A_t, S_{t+1}, A_{t+1}, \dots, S_T | S_t, b)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

- Depends only on the policy and not on the MDP.

- Value function using importance sampling:
$$v_\pi(s) = \mathbb{E}\left[\rho_{t:T-1} G_t | S_t = s\right] \neq \mathbb{E}\left[G_t | S_t = s\right] = v_b(s)$$
- Natural estimate for v_π (ordinary importance sampling):
$$V(s) = \frac{\sum_{t \in \mathcal{I}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{I}(s)|}$$
where $\mathcal{I}(s)$ are the time step where s is visited (only for the first time for a first-visit method)
- Alternative (weighted importance sampling):
$$V(s) = \frac{\sum_{t \in \mathcal{I}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{I}(s)} \rho_{t:T(t)-1}}$$
- Rk: $\mathbb{E}\left[\sum_{t \in \mathcal{I}(s)} \rho_{t:T(t)-1}\right] = |\mathcal{I}(s)|$

5.5 Off-policy Prediction via Importance Sampling

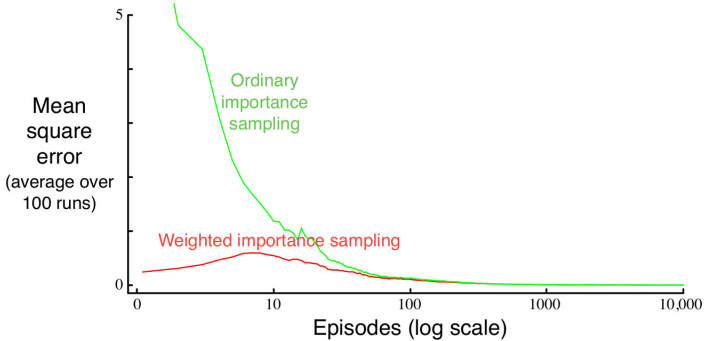


Figure 5.3: Weighted importance sampling produces lower error estimates of the value of a single blackjack state from off-policy episodes. ■

- ordinary importance sampling is unbiased (for the first-visit method) but may have a large variance
- weighted importance sampling is biased but may have a smaller variance.
- No asymptotic bias.
- ordinary importance sampling is nevertheless simpler to extend...

5.5 Off-policy Prediction via Importance Sampling

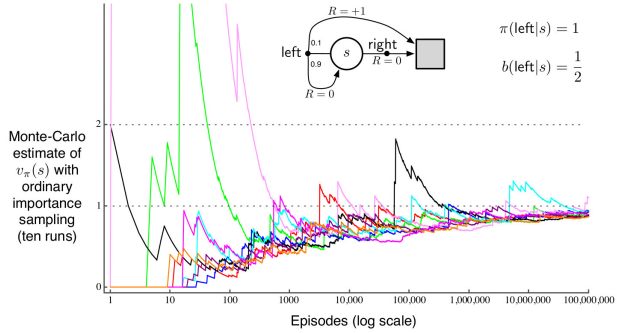


Figure 5.4: Ordinary importance sampling produces surprisingly unstable estimates on the one-state MDP shown inset (Example 5.5). The correct estimate here is 1 ($\gamma = 1$), and, even though this is the expected value of a sample return (after importance sampling), the variance of the samples is infinite, and the estimates do not converge to this value. These results are for off-policy first-visit MC.

- Very large variance terms lead to convergence issues.

5.6 Incremental Implementation

- Incremental implementation avoids to store all the returns.
- Observation: if

$$V_n = \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}$$

then

$$V_{n+1} = V_n + \frac{W_n}{C_n}(G_n - V_n)$$

with $C_{n+1} = C_n + W_{n+1}$.

- Rk: if

$$V_n = \frac{\sum_{k=1}^{n-1} W_k G_k}{n-1}$$

then

$$V_{n+1} = V_n + \frac{1}{n}(W_n G_n - V_n)$$

- Leads to a better implementation.

5.6 Incremental Implementation

Off-policy MC prediction (policy evaluation) for estimating $Q \approx q_\pi$

Input: an arbitrary target policy π
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \in \mathbb{R}$ (arbitrarily)
$C(s,a) \leftarrow 0$

Loop forever (for each episode):
$b \leftarrow$ any policy with coverage of π
Generate an episode following b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
$W \leftarrow 1$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$, while $W \neq 0$:
$G \leftarrow \gamma G + R_{t+1}$
$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
$W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

5.7 Off-policy Monte Carlo Control

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

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \in \mathbb{R}$ (arbitrarily)
$C(s,a) \leftarrow 0$
$\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
$b \leftarrow$ any soft policy
Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
$W \leftarrow 1$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
$\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)
If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
$W \leftarrow W \frac{1}{b(A_t|S_t)}$

- GPI principle
- Require a exploratory target policy

Outline

6 Temporal-Difference Learning

6.1 TD Prediction

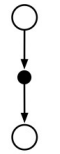
- constant α Monte Carlo update:
$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t)\right)$$
- Target G_t ($\sim v_\pi(S_t)$) requires to wait until the episode end.
- Simplest TD method:
$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$
- Target $R_{t+1} + \gamma V(S_{t+1})$ ($\sim v_\pi(S_t)$) is available immediately.
- Estimate based on a previous estimate: *bootstrapping* method (like DP).
- Underlying expectations:
$$\begin{aligned} v_\pi &= \mathbb{E}_\pi \left[G_t | S_t = s\right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right] \end{aligned}$$
- Estimate:
  - Expectation (MC / TD)
  - Value function (DP /TD)

6.1 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 $V(terminal) = 0$

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



TD(0)

- TD error:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

- MC error:

$$\begin{aligned} G_t - V(S_t) &= \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\ &= \sum_{k=T}^{T-1} \gamma^{k-t} \delta_k \end{aligned}$$

if V is kept frozen during each episode.

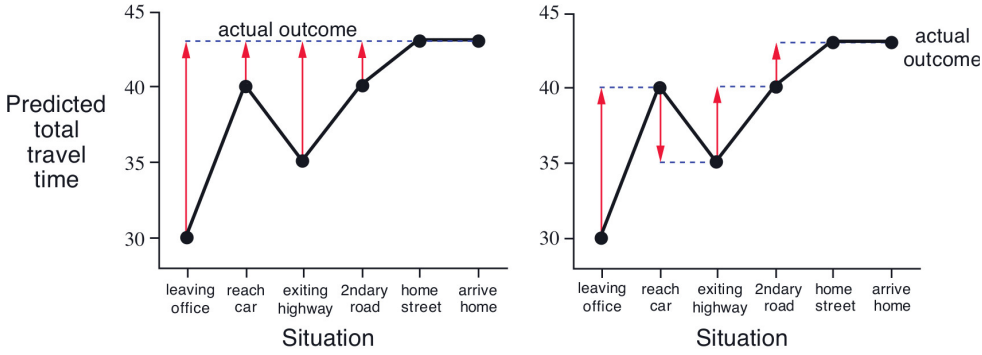
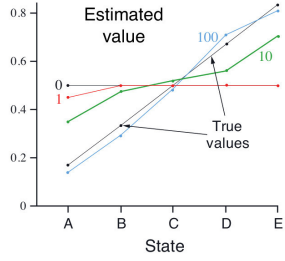
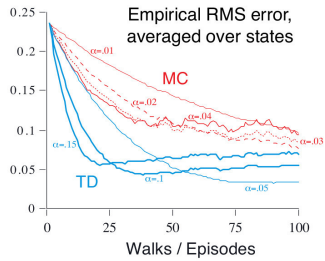


Figure 6.1: Changes recommended in the driving home example by Monte Carlo methods (left) and TD methods (right).

6.2 Advantages of TD Prediction Methods





- Obvious advantages:
  - No need for a model (cf DP)
  - No need to wait until the episode end (cf MC)
- Theoretical guarantee on the convergence!
- No theoretical winner between TD and MC...
- In practice, TD is often faster.

6.3 Optimality of TD(0)

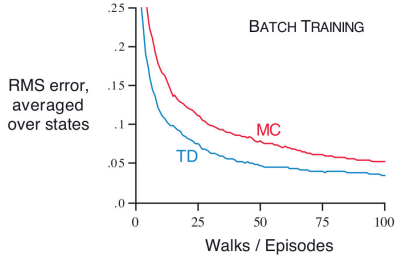
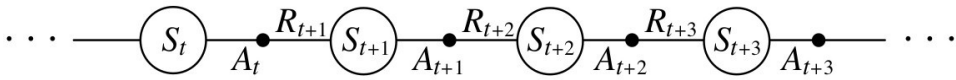


Figure 6.2: Performance of TD(0) and constant-α MC under batch training on the random walk task.

- Batch updating setting: several passes on the same data.
- MC and TD converges (provided α is small enough).
- Different limits:
  - MC: sample average of the return
  - TD: value function if one replaces the true MDP by the maximum likelihood one. (*certainty-equivalence estimate*)
- Rk: no need to compute the ML estimate!

6.4 Sarsa: On-policy TD Control



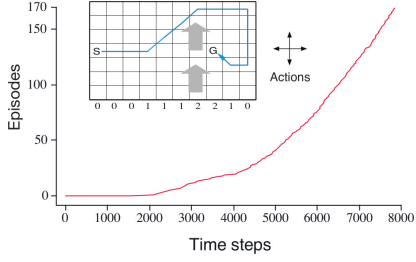
- GPI setting:
  - Update Q using the current policy with
    $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right)$$
  - Update π by policy improvement
- May not converge if one use a greedy policy update!
- Convergence results if ϵ_t greedy update with $\epsilon_t \to 0$.

6.4 Sarsa: On-policy TD Control

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 $Q(terminal, \cdot) = 0$

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



6.5 Q-Learning: Off-policy TD Control

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

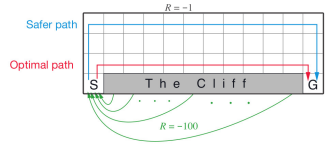
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

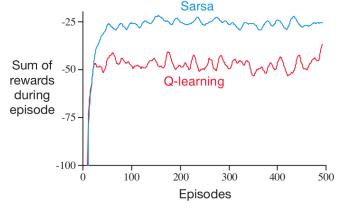
- Q-learning update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q_t(S_{t+1}, a) - Q(S_t, A_t) \right)$$

- Update independent from the behavior policy!
- Convergence provided the policy visit each state-action infinitely often.

6.5 Q-Learning: Off-policy TD Control





- Q-learning takes more risk...

6.6 Expected Sarsa

Figure 6.4: The backup diagrams for Q-learning and Expected Sarsa.

- Idea: replace the action sampling in Sarsa by an expectation

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

$$\leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a) - Q(S_t, A_t) \right)$$

- More complex but variance reduction.
- Off-policy as π can be different from b

6.6 Expected Sarsa

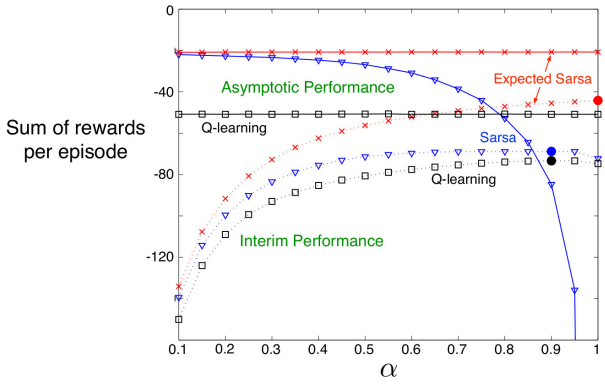


Figure 6.3: Interim and asymptotic performance of TD control methods on the cliff-walking task as a function of α. All algorithms used an ε-greedy policy with $\varepsilon = 0.1$. Asymptotic performance is an average over 100,000 episodes whereas interim performance is an average over the first 100 episodes. These data are averages of over 50,000 and 10 runs for the interim and asymptotic cases respectively. The solid circles mark the best interim performance of each method. Adapted from van Seijen et al. (2009).

6.7 Maximization Bias and Double Learning

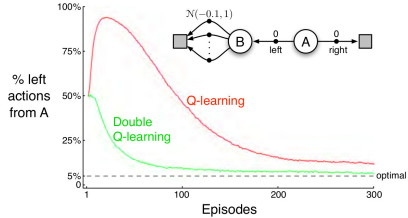


Figure 6.5: Comparison of Q-learning and Double Q-learning on a simple episodic MDP (shown inset). Q-learning initially learns to take the left action much more often than the right action, and always takes it significantly more often than the 5% minimum probability enforced by ε-greedy action selection 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. Any ties in ε-greedy action selection were broken randomly.

- Maximization bias issue: $\mathbb{E}\left[\max\right] \geq \max \mathbb{E}$!
- Double learning:
  - Maintain two *independent* estimates of q: Q_1 and Q_2
  - Maintain two estimates of the best action $A_{1,*} = \text{argmax}\, Q_1(., a)$ and $A_{2,*} = \text{argmax}\, Q_2(., a)$
  - Maintain two *unbiased* estimates $q(A_{1,*}) = Q_2(A_{1,*})$ and $q(A_{2,*}) = Q_1(A_{2,*})$

6.7 Maximization Bias and Double 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 $Q(\mathit{terminal}, \cdot) = 0$

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 probabilility:

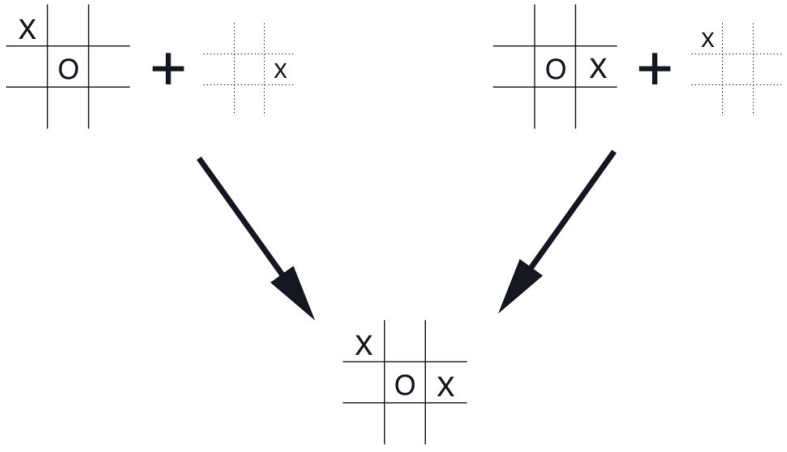
$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{arg\,max}_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

    else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{arg\,max}_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

    $S \leftarrow S'$
  until S is terminal

6.8 Games, Afterstates, and Other Special Cases



- Book focuses on state-action value function.
- Other approach possible: afterstates value function
- Interesting in games in particular...

Outline

7 n-step Bootstrapping

- 7.1 n-step TD Prediction
- 7.2 n-step Sarsa
- 7.3 n-step Off-policy Learning
- 7.4 Per-decision Methods with Control Variates
- 7.5 Off-policy Learning Without Importance Sampling: The n-step Tree Backup Algorithm
- 7.6 A Unifying Algorithm: n-step $Q(\sigma)$

7.1 n-step TD Prediction

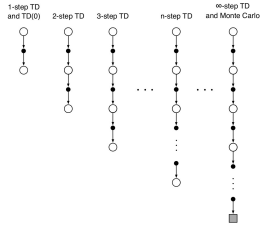


Figure 7.1: The backup diagrams of n-step methods. These methods form a spectrum ranging from one-step TD methods to Monte Carlo methods.

- MC:

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

- One-step return:

$$G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1})$$

- n-step return:

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

7.1 *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 $t = 0, 1, 2, \ldots$:
  | 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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

- *n*-step TD:

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha \left(G_{t:t+n} - V_{t+n-1}(S_t)\right)$$

- Contraction property:

$$\|\mathbb{E}_\pi \left[G_{t:t+n} | S_t = s\right] - v_\pi(s)\|_\infty \leq \gamma^n \|V_{t+n-1}(s) - v_\pi(s)\|_\infty$$

7.1 *n*-step TD Prediction

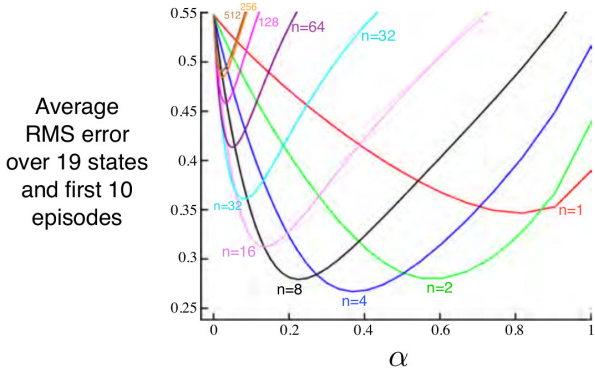


Figure 7.2: Performance of n-step TD methods as a function of α, for various values of n, on a 19-state random walk task (Example 7.1). ■

- Optimum for intermediate n.

7.2 *n*-step Sarsa

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

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, 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 $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
    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 $A_{t+1} \sim \pi(\cdot|S_{t+1})$
    $\tau \leftarrow t - n + 1$ (τ is the time whose 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 $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\left[G - Q(S_\tau, A_\tau)\right]$
      If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

- *n*-step return:

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

- *n*-step Sarsa:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha \left(G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right)$$

7.2 *n*-step Sarsa

Figure 7.3: The backup diagrams for the spectrum of n-step methods for state–action values. They range from the one-step update of Sarsa(0) to the up-until-termination update of the Monte Carlo method. In between are the n-step updates, based on n steps of real rewards and the estimated value of the nth next state–action pair, all appropriately discounted. On the far right is the backup diagram for n-step Expected Sarsa.

- Expected Sarsa possible. . .

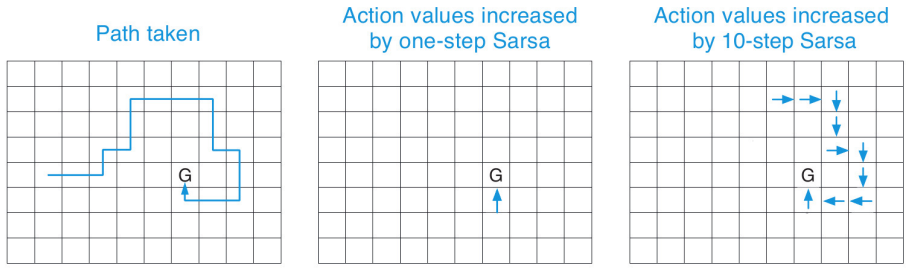


Figure 7.4: Gridworld example of the speedup of policy learning due to the use of n-step methods. The first panel shows the path taken by an agent in a single episode, ending at a location of high reward, marked by the G. In this example the values were all initially 0, and all rewards were zero except for a positive reward at G. The arrows in the other two panels show which action values were strengthened as a result of this path by one-step and n-step Sarsa methods. The one-step method strengthens only the last action of the sequence of actions that led to the high reward, whereas the n-step method strengthens the last n actions of the sequence, so that much more is learned from the one episode.

7.3 *n*-step Off-policy Learning

- Need to take into account the exploratory policy b.
- Importance sampling correction:

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

- Off-policy n-step TD:

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha\rho_{t:t+n-1}\left(G_{t:t+n} - V_{t+n-1}(S_t)\right)$$

- Off-policy n-step Sarsa:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, Q_t) + \alpha\rho_{t:\mathbf{t+n}}\left(G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right)$$

- Expected Sarsa possible...

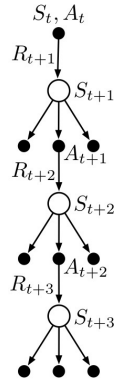
7.3 *n*-step Off-policy Learning

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 $t = 0, 1, 2, \ldots$:
    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 $\tau \geq 0$:
      $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1, T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ $(\rho_{\tau+1:t+n-1})$
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$
      If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\left[G - Q(S_\tau, A_\tau)\right]$
      If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

7.5 Off-policy Learning Without Importance Sampling: The n-step Tree Backup Algorithm



the 3-step tree-backup update

- Use reward for action taken and bootstrap for the others.
- Weight each branch by $\pi(a|S_t)$.

7.5 Off-policy Learning Without Importance Sampling: The n-step Tree Backup Algorithm

- 1-step return (Expected Sarsa)

$$G_{t:t+1} = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a)$$

- 2-step return:

$$\begin{aligned} G_{t:t+2} &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) \\ &\quad + \gamma \pi(A_{t+1}|S_{t+1}) \left(R_{t+2} + \gamma \sum_a \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a) \right) \\ &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+2} \end{aligned}$$

- Recursive definition of n-step return:

$$G_{t:t+n} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n}$$

7.5 Off-policy Learning Without Importance Sampling: The n-step Tree Backup Algorithm

n-step Tree Backup for estimating $Q \approx q_*$ or q_π

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 can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Choose an action A_0 arbitrarily as a function of S_0; Store A_0
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$:
  | Take action A_t; observe and store the next reward and state as R_{t+1}, S_{t+1}
  | If S_{t+1} is terminal:
  | $T \leftarrow t+1$
  | else:
  | Choose an action A_{t+1} arbitrarily as a function of S_{t+1}; Store A_{t+1}
  | $\tau \leftarrow t+1-n$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | If $t+1 \geq T$:
  | $G \leftarrow R_T$
  | else
  | $G \leftarrow R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a)$
  | Loop for $k = \min(t, T-1)$ down through $\tau+1$:
  | $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi(a|S_k) Q(S_k, a) + \gamma \pi(A_k|S_k) G$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau) \right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T-1$

7.6 A Unifying Algorithm: n-step $Q(\sigma)$

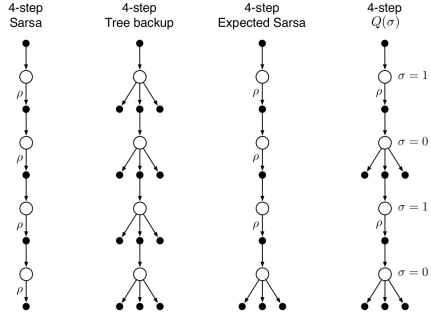


Figure 7.5: The backup diagrams of the three kinds of n-step action-value updates considered so far in this chapter (4-step case) plus the backup diagram of a fourth kind of update that unifies them all. The 'ρ's indicate half transitions on which importance sampling is required in the off-policy case. The fourth kind of update unifies all the others by choosing on a state-by-state basis whether to sample ($\sigma_t = 1$) or not ($\sigma_t = 0$).

- Different strategy at each node:
  - $\sigma = 1$: action sampling
  - $\sigma = 0$: action averaging

7.6 A Unifying Algorithm: n-step $Q(\sigma)$

Off-policy n-step $Q(\sigma)$ 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 ε-greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Choose and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
    If $t < T$:
      Take action A_t; observe and store the next reward and state as R_{t+1}, S_{t+1}
      If S_{t+1} is terminal:
        $T \leftarrow t+1$
      else:
        Choose and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
        Select and store σ_{t+1}
        Store $\frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t+1}|S_{t+1})}$ as ρ_{t+1}
    $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
    If $\tau \geq 0$:
      $G \leftarrow 0$:
      Loop for $k = \min(t+1, T)$ down through $\tau + 1$:
        if $k = T$:
          $G \leftarrow R_T$
        else:
          $\bar{V} \leftarrow \sum_a \pi(a|S_k) Q(S_k, a)$
          $G \leftarrow R_k + \gamma\big(\sigma_k \rho_k + (1 - \sigma_k)\pi(A_k|S_k)\big)\big(G - Q(S_k, A_k)\big) + \gamma \bar{V}$
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\,[G - Q(S_\tau, A_\tau)]$
      If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

- Generalization to $\sigma_t \in [0, 1]$

Outline

8 Planning and Learning with Tabular Methods

8.1 Models and Planning

- *Model*: anything that can be used to predict the environment response.
- Two different model families:
  - Distribution models: explicitly learn the MDP transitions
  - Sample models: learn to simulate the MDP
- Second type is easier to obtain.
- Planning:

$$\text{model} \xrightarrow{\text{planning}} \text{policy}$$

- In the book, *state-space planning* by opposition of *plan-space planning* which works on the plans.
- Common structure of *state-space planning*

$$\text{model} \longrightarrow \text{simulated exp.} \xrightarrow{\text{backups}} \text{values} \longrightarrow \text{policy}$$

- Learning methods use real experiences instead of simulated ones

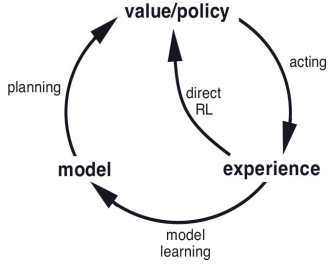
Random-sample one-step tabular Q-planning

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

- Similar algorithm than Q-learning!
- Only difference is the source of experience.
- Rk: we have used this algorithm in the section 6.3 Optimality of TD(0).

8.2 Dyna: Integrated Planning, Acting and Learning



- Dyna-Q: architecture combining both planning and learning.
- Two uses of experience:
  - model-learning (to improve the models)
  - reinforcement-learning (direct RL) (to improve the value/policy)
- Indirect methods (model based) can use an a priori model but can be mislead by a false model.
- Direct methods do not require a model but may require a lot of experience.

8.2 Dyna: Integrated Planning, Acting and Learning

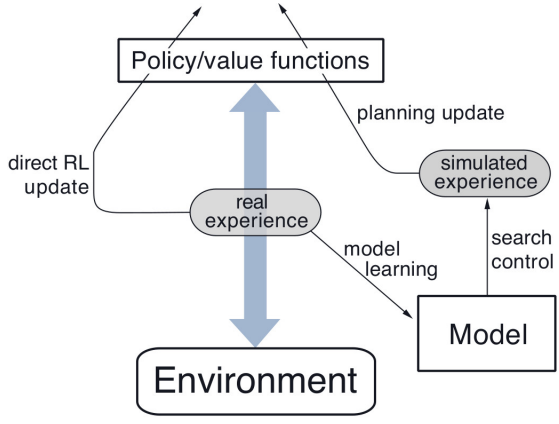


Figure 8.1: The general Dyna Architecture. Real experience, passing back and forth between the environment and the policy, affects policy and value functions in much the same way as does simulated experience generated by the model of the environment.

- Planning, acting, model-learning and direct RL are conceptually simultaneous.
- Need to deal with the scheduling in practice.

8.2 Dyna: Integrated Planning, Acting and Learning

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) $A \leftarrow \varepsilon$-greedy(S,Q)
- (c) Take action A; observe resultant reward, R, and state, S'
- (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
- (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
  - $R, S' \leftarrow Model(S,A)$
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

- Learning, model-learning and planning are present.
- Extension of Q-learning.

8.2 Dyna: Integrated Planning, Acting and Learning

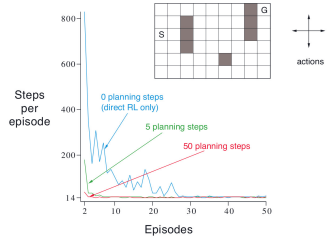


Figure 8.2: A simple maze (inset) and the average learning curves for Dyna-Q agents varying in their number of planning steps (n) per real step. The task is to travel from S to G as quickly as possible.

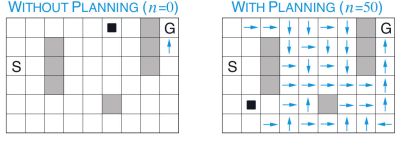


Figure 8.3: Policies found by planning and nonplanning Dyna-Q agents halfway through the second episode. The arrows indicate the greedy action in each state; if no arrow is shown for a state, then all of its action values were equal. The black square indicates the location of the agent. ■

8.3 When the Model is Wrong

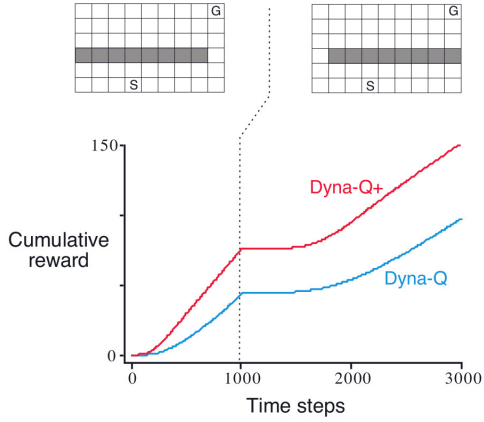


Figure 8.4: Average performance of Dyna agents on a blocking task. The left environment was used for the first 1000 steps, the right environment for the rest. Dyna-Q+ is Dyna-Q with an exploration bonus that encourages exploration. ■

- If the model is wrong, it may eventually be corrected...

8.3 When the Model is Wrong

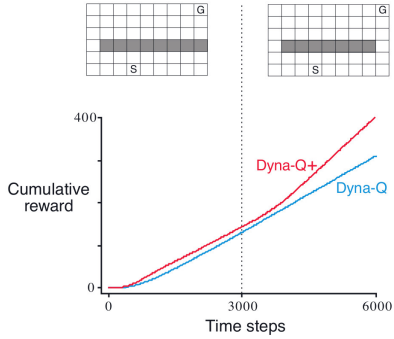


Figure 8.5: Average performance of Dyna agents on a shortcut task. The left environment was used for the first 3000 steps, the right environment for the rest.

- but this may be complicated if the model was pessimistic...
- Dyna-Q+ forces exploration by increasing the rewards of non explored-lately state-action.

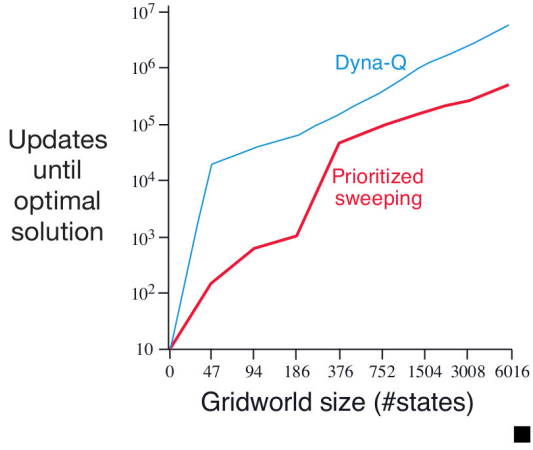
8.4 Prioritized Sweeping

Prioritized sweeping for a deterministic environment

Initialize $Q(s,a)$, $Model(s,a)$, for all s, a, and $PQueue$ to empty
Loop forever:
- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow policy(S,Q)$
- (c) Take action A; observe resultant reward, R, and state, S'
- (d) $Model(S,A) \leftarrow R, S'$
- (e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$.
- (f) if $P > \theta$, then insert S, A into $PQueue$ with priority P
- (g) Loop repeat n times, while $PQueue$ is not empty:
  - $S, A \leftarrow first(PQueue)$
  - $R, S' \leftarrow Model(S,A)$
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
  - 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

- Freedom in the order of the state/action during planning.
- Intuition says that one should work *backward* from the *goal*.
- Prioritized sweeping: order state-action by a predicted value difference.

8.4 Prioritized Sweeping



- Prioritized sweeping leads to faster convergence.

8.5 Expected vs. Sample Updates

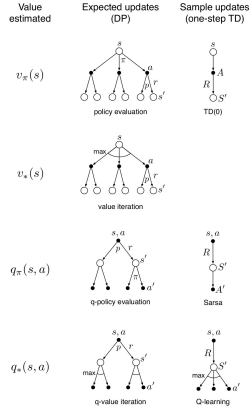


Figure 8.6: Backup diagrams for all the one-step updates considered in this book.

- If the transition probability are available, should we use expectation or samples? DP or RL?

8.5 Expected vs. Sample Updates

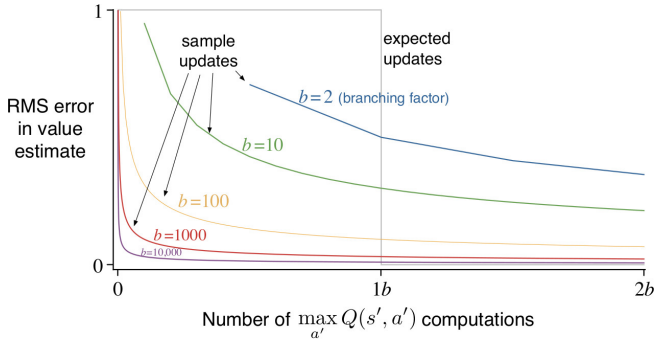


Figure 8.7: Comparison of efficiency of expected and sample updates.

- Expectations are more stable but require more computation.
- Cost depends heavily on the *branching factor*.
- In practice, sampling seems interesting for large branching factors!

8.6 Trajectory Sampling

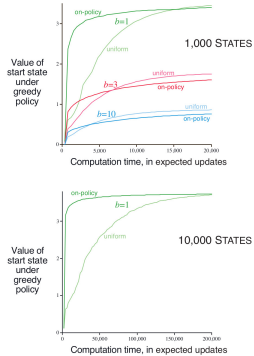
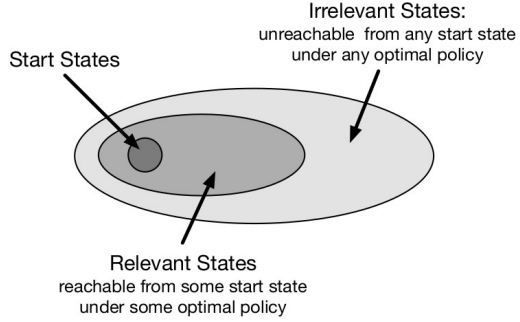


Figure 8.8: Relative efficiency of updates distributed uniformly across the state space versus focused on simulated on-policy trajectories, each starting in the same state. Results are for randomly generated tasks of two sizes and various branching factors, b.

- Trajectory sampling: sample states-actions by interacting with the model...
- Initial gain but may be slow in the long run.

8.7 Real-time Dynamic Programming



- Classical DP but with a trajectory sampling.
- Convergence holds with exploratory policy.
- Optimal policy does not require to specify the action in irrelevant states.
- Convergence holds even without full exploration in some specific cases!
- In practice, seems to be computation efficient.

8.8 Planning at Decision Time

- Background planning: planning works on *all* states.
- Decision-time planning: planning starts from the current state.
- Extension of the one-step lookahead but often without memory.
- Combination looks interesting...

8.9 Heuristic Search

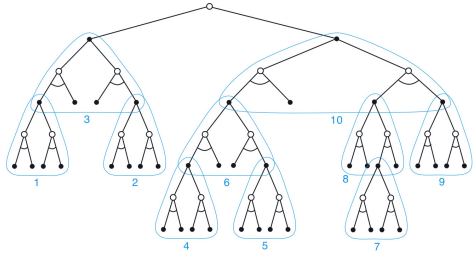


Figure 8.9: Heuristic search can be implemented as a sequence of one-step updates (shown here outlined in blue) backing up values from the leaf nodes toward the root. The ordering shown here is for a selective depth-first search.

- Heuristic search: most classical decision time planning method.
- At each step,
  - a tree of possible continuations is grown
  - an approximate value function is used at the leaves.
  - those values are backed-up to the root
- Often value function is fixed.
- Can be seen as an extension of greedy policy beyond a single step.

8.9 Heuristic Search

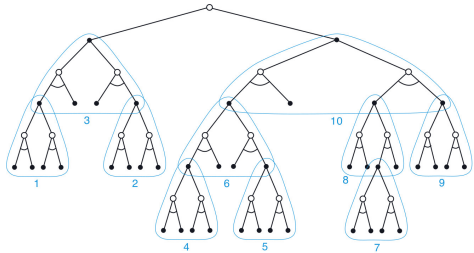


Figure 8.9: Heuristic search can be implemented as a sequence of one-step updates (shown here outlined in blue) backing up values from the leaf nodes toward the root. The ordering shown here is for a selective depth-first search.

- The deeper the tree the less influence the value function.
- The better the value function the better the policy.
- Computational trade-off.
- Focus on the states that are accessible from the current state.

8.10 Rollout Algorithms

- Use MC simulation with a policy π to choose the next action.
- Simulation equivalent of the policy improvement idea.
- State-action value function estimated only for the current state.
- Computation time influenced by the complexity of the policy and the number of simulations.
- Early stopping possible using an approximate value function!

8.11 Monte Carlo Tree Search

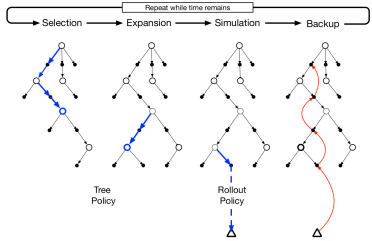


Figure 8.10: Monte Carlo Tree Search. When the environment changes to a new state, MCTS executes as many iterations as possible before an action needs to be selected, incrementally building a tree whose root node represents the current state. Each iteration consists of the four operations **Selection**, **Expansion** (though possibly skipped on some iterations), **Simulation**, and **Backup**, as explained in the text and illustrated by the bold arrows in the trees. Adapted from Chaslot, Bakkes, Szita, and Spronck (2008).

- Rollout algorithm combined with backup-ideas.
- Repeat 4 steps:
  - Selection of a sequence of actions according to the current values with a tree policy.
  - Expansion of the tree at the last node without values.
  - Simulation with a rollout policy to estimate the values at this node.
  - Backup of the value by empirical averaging.
- MCTS focuses on promising path.

Tabular Reinforcement

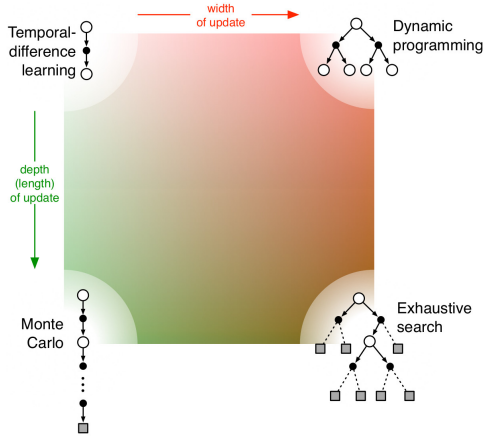


Figure 8.11: A slice through the space of reinforcement learning methods, highlighting the two of the most important dimensions explored in Part I of this book: the depth and width of the updates.

Other Raised Issues

- **Definition of return** Is the task episodic or continuing, discounted or undiscounted?
- **Action values vs. state values vs. afterstate values** What kind of values should be estimated? If only state values are estimated, then either a model or a separate policy (as in actor–critic methods) is required for action selection.
- **Action selection/exploration** How are actions selected to ensure a suitable trade-off between exploration and exploitation? (ϵ-greedy, optimistic initialization of values, soft-max, and UCB. . .)
- **Synchronous vs. asynchronous** Are the updates for all states performed simultaneously or one by one in some order?
- **Real vs. simulated** Should one update based on real experience or simulated experience? If both, how much of each?
- **Location of updates** What states or state–action pairs should be updated? Model-free methods can choose only among the states and state–action pairs actually encountered, but model-based methods can choose arbitrarily. There are many possibilities here.
- **Timing of updates** Should updates be done as part of selecting actions, or only after- ward?
- **Memory for updates** How long should updated values be retained? Should they be retained permanently, or only while computing an action selection, as in heuristic search?

Outline

9 On-policy Prediction with Approximation

- 9.1 Value-function Approximation
- 9.2 The Prediction Objective ($\overline{VE}$)
- 9.3 Stochastic-gradient and Semi-gradient Methods
- 9.4 Linear Methods
- 9.5 Feature Construction for Linear Methods
- 9.6 Selecting Step-Size Parameters Manually
- 9.7 Nonlinear Function Approximation: Artificial Neural Networks
- 9.8 Least-Squares TD
- 9.9 Memory-based Function Approximation
- 9.10 Kernel-based Function Approximation
- 9.11 Looking Deeper at On-policy Learning: Interest and Emphasis

9.1 Value-function Approximation

- Prediction methods covered based on *backed-up values* to which the current value is shifted.
- Examples:
  - MC with G_t
  - TD(0) with $R_{t+1} + \gamma V(S_{t+1})$
  - n-step TD $G_{t:t+n}$
- V is defined by its value at each state.
- When the number of states is large this may be intractable.
- Idea: replace $V(s)$ by an approximation $\hat{v}(s, \boldsymbol{w})$ where $\boldsymbol{w}$ are weights (parameters) defining the function.
- Goal: find $\boldsymbol{w}$ such that $\hat{v}(s, \boldsymbol{w}) \sim v_\pi(s)$ from the backed-up values.
- Similar to supervised learning!
- Function approximation (or regression) setting.
- Reinforcement Learning requires on-line methods rather than batch.
- Often non-stationary target functions...

9.2 The Prediction Objective ($\overline{VE}$)

- How to measure the quality of $\hat{v}(s, \boldsymbol{w})$?
- So far, we have use implicitly a $\|\cdot\|_\infty$ norm.
- Prediction Objective ($\overline{VE}$):
$$\overline{VE}(\boldsymbol{w}) = \sum_s \mu(s) \left(v_\pi(s) - \hat{v}(s, \boldsymbol{w})\right)^2$$
where μ is a state distribution.
- Most classical choice: $\mu(s)$ is the fraction of time spent in s.
- Under on-policy training, such a choice is called on-policy distribution
  - Stationary distribution under π for the continuing tasks.
  - Depends on the law of initial state for episodic tasks.
- More difference between episodic and continuing that without approximation.
- Rk: Prediction Objective not linked to corresponding policy performance!
- Often only local optimal in $\boldsymbol{w}$...

9.3 Stochastic-gradient and Semi-gradient Methods

- Prediction Objective ($\overline{VE}$):
$$\overline{VE}(\boldsymbol{w}) = \sum_s \mu(s) \left(v_\pi(s) - \hat{v}(s, \boldsymbol{w})\right)^2$$
- Prediction Objective ($\overline{VE}$) gradient:
$$\nabla \overline{VE}(\boldsymbol{w}) = -2 \sum_s \mu(s) \left(v_\pi(s) - \hat{v}(s, \boldsymbol{w})\right) \nabla \hat{v}(s, \boldsymbol{w})$$
- Prediction Objective ($\overline{VE}$) stochastic gradient:
$$\widehat{\nabla} \overline{VE}(\boldsymbol{w}) = -2 \left(v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{w})\right) \nabla \hat{v}(S_t, \boldsymbol{w})$$
- SGD algorithm:
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{w})\right) \nabla \hat{v}(S_t, \boldsymbol{w})$$
- Issue: v_π is unknown!

9.3 Stochastic-gradient and Semi-gradient Methods

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[G_t - \hat{v}(S_t, \mathbf{w})\big] \nabla \hat{v}(S_t, \mathbf{w})$

- Monte Carlo: replace $v_\pi(S_t)$ by G_t.
- Algorithm:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(G_t - \hat{v}(S_t, \boldsymbol{w})\right) \nabla \hat{v}(S_t, \boldsymbol{w})$$

- Convergence guarantees because $\mathbb{E}\left[G_t | S_t = s\right] = v_\pi(s)$.
- Stochastic-gradient setting!

9.3 Stochastic-gradient and Semi-gradient Methods

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
    $S \leftarrow S'$
  until S is terminal

- TD: replace $v_\pi(S_t)$ by $R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}_t)$.
- Algorithm:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\left(R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{w})\right)\nabla\hat{v}(S_t, \boldsymbol{w})$$

- Not a stochastic-gradient setting anymore!
- Effect of the change of $\boldsymbol{w}$ to the target is ignored, hence the name semi-gradient
- Less stable but converges for linear approximation.

9.3 Stochastic-gradient and Semi-gradient Methods

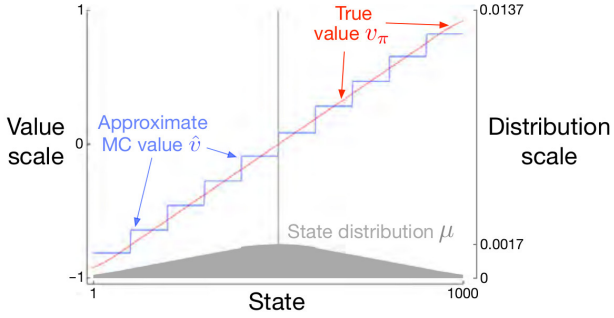


Figure 9.1: Function approximation by state aggregation on the 1000-state random walk task, using the gradient Monte Carlo algorithm (page 202).

- Example with state aggregation: several states are using the same value for the value function.
- MC stochastic gradient algorithm.

9.4 Linear Methods

- Linear method: $\hat{s}(s, \boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{x}(s)$ with $\boldsymbol{x}(s)$ a feature vector representing state s.
- $\boldsymbol{x}_i$ are basis functions of the linear space of possible $\hat{s}$.
- Linear method gradient:

$$\nabla \hat{s}(s, \boldsymbol{w}) = \boldsymbol{x}(s)$$

- Generic *SGD* algorithm:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(U_t - \boldsymbol{w}_t^\top \boldsymbol{x}(S_t)) \right) \boldsymbol{x}(S_t)$$

- MC stochastic gradient:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(G_t - \boldsymbol{w}_t^\top \boldsymbol{x}(S_t)) \right) \boldsymbol{x}(S_t)$$

- TD(0) semi-gradient:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(R_{t+1} + \gamma \boldsymbol{w}_t^\top \boldsymbol{x}(S_{t+1}) - \boldsymbol{w}_t^\top \boldsymbol{x}(S_t)) \right) \boldsymbol{x}(S_t)$$

- Simple form allows mathematical analysis!

9.4 Linear Methods

- Using $\boldsymbol{x}(S_t) = \boldsymbol{x}_t$, TD becomes

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

- Assume we are in the steady state regime,

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

  with $b = \mathbb{E}\left[R_{t+1}\boldsymbol{x}_t\right]$ and $A = \mathbb{E}\left[\boldsymbol{x}_t(\boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1})^\top\right]$.
- If the algorithm converges to $\boldsymbol{w}_{TD}$, then

$$\boldsymbol{b} - \boldsymbol{A}\boldsymbol{w}_{TD} = 0$$

- If $\boldsymbol{A}$ is invertible,

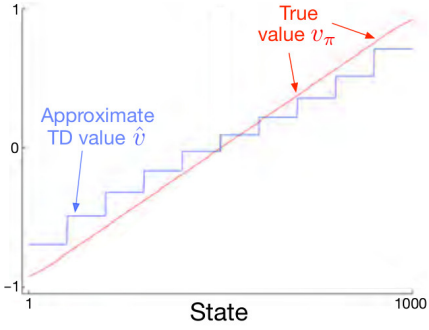
$$\boldsymbol{w}_{TD} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

  and

$$\mathbb{E}\left[\boldsymbol{w}_{t+1} - \boldsymbol{w}_{TD} | \boldsymbol{w}_t\right] = (\mathrm{Id} - \alpha \boldsymbol{A})(\boldsymbol{w}_t - \boldsymbol{w}_{TD})$$

- $\boldsymbol{A}$ is definite positive thanks to the stationarity of the measure μ for the policy π.
- Complete proof much more involved.

- Error bound for MC:
$$\overline{VE}(\boldsymbol{w}_{MC}) = \min_{\boldsymbol{w}} \overline{VE}(\boldsymbol{w})$$
- Error bound for TD:
$$\overline{VE}(\boldsymbol{w}_{MC}) \leq \frac{1}{1-\gamma} \min_{\boldsymbol{w}} \overline{VE}(\boldsymbol{w})$$
- Possible asymptotic loss for TD... but often faster convergence and lower variance!
- Similar results for other on-policy prediction algorithms.
- Similar results for episodic tasks.



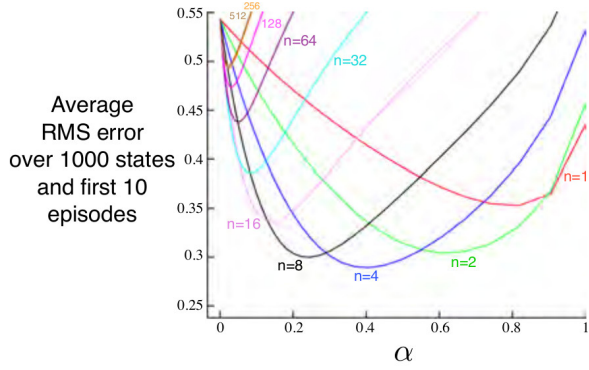


Figure 9.2: Bootstrapping with state aggregation on the 1000-state random walk task. *Left*: Asymptotic values of semi-gradient TD are worse than the asymptotic Monte Carlo values in Figure 9.1. *Right*: Performance of n-step methods with state-aggregation are strikingly similar to those with tabular representations (cf. Figure 7.2). These data are averages over 100 runs.

9.4 Linear Methods

n-step semi-gradient TD for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameters: step size $\alpha > 0$, a positive integer n
Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
All store and access operations (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 $t = 0, 1, 2, \ldots$:
  | 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: $G \leftarrow G + \gamma^n \hat{v}(S_{\tau+n},\mathbf{w})$ $(G_{\tau:\tau+n})$
  | $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G - \hat{v}(S_\tau,\mathbf{w})\right] \nabla \hat{v}(S_\tau,\mathbf{w})$
  Until $\tau = T - 1$

- Algorithm:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(G_{t:t+n} - \boldsymbol{w}_t^\top \boldsymbol{x}(S_t) \right) \boldsymbol{x}(S_t)$$

9.5 Feature Construction for Linear Methods

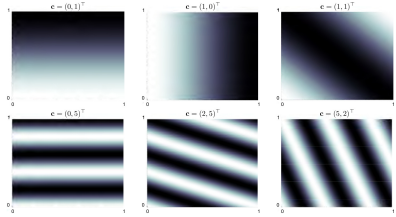


Figure 9.4: A selection of six two-dimensional Fourier cosine features, each labeled by the vector $\mathbf{c}^i$ that defines it (s_1 is the horizontal axis, and $\mathbf{c}^i$ is shown with the index i omitted). After Konidaris et al. (2011).

- Polynomials: simplest possible basis

$$\boldsymbol{x}(s)_i = \prod_{j=1}^{k} s_j^{c_{i,j}}$$

- Fourier Basis: useful when dealing with periodic functions

$$\boldsymbol{x}(s)_i = \cos(\pi s^\top c_i) \quad \text{or} \quad \sin(\pi s^\top c_i)$$

- Renormalization may be required.
- Selection of the order can be useful.

9.5 Feature Construction for Linear Methods

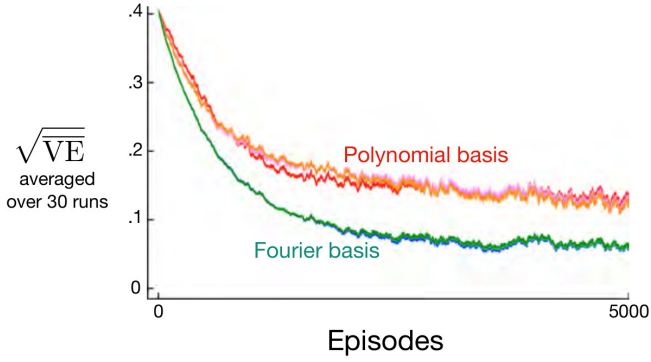


Figure 9.5: Fourier basis vs polynomials on the 1000-state random walk. Shown are learning curves for the gradient Monte Carlo method with Fourier and polynomial bases of order 5, 10, and 20. The step-size parameters were roughly optimized for each case: $\alpha = 0.0001$ for the polynomial basis and $\alpha = 0.00005$ for the Fourier basis. The performance measure (y-axis) is the root mean squared value error (9.1).

- Fourier well adapted to smooth functions.

9.5 Feature Construction for Linear Methods

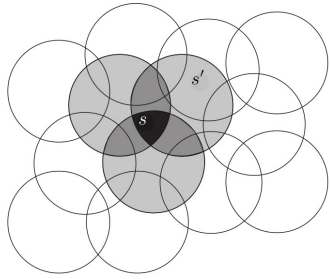


Figure 9.6: Coarse coding. Generalization from state s to state s' depends on the number of their features whose receptive fields (in this case, circles) overlap. These states have one feature in common, so there will be slight generalization between them.

- Coarse coding: extension of state aggregation.
- Large freedom in the design of the cells.

9.5 Feature Construction for Linear Methods

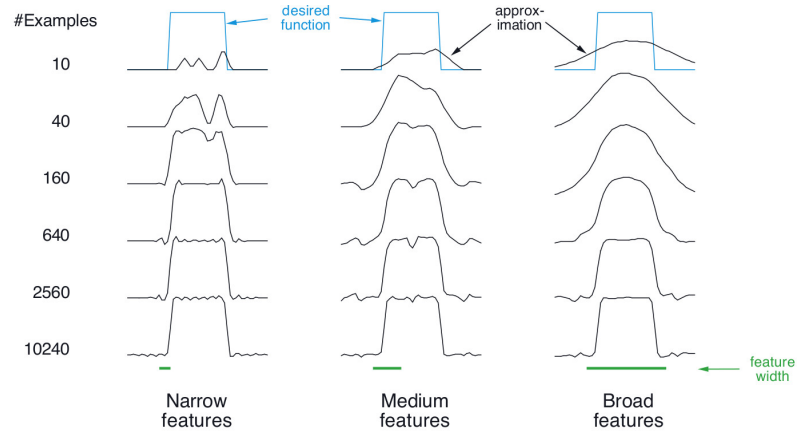


Figure 9.8: Example of feature width's strong effect on initial generalization (first row) and weak effect on asymptotic accuracy (last row). ■

- Bandwidth issue...

9.5 Feature Construction for Linear Methods

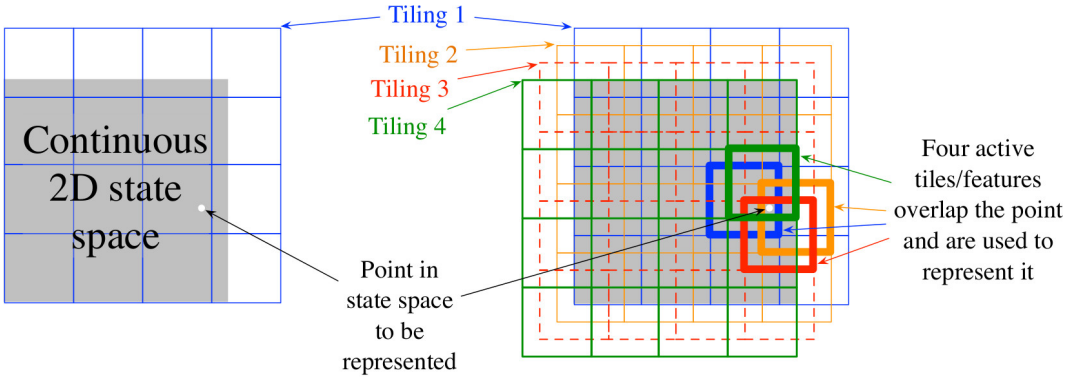


Figure 9.9: Multiple, overlapping grid-tilings on a limited two-dimensional space. These tilings are offset from one another by a uniform amount in each dimension.

- Systematic way of construction coarse coding: grid plus overlap.

9.5 Feature Construction for Linear Methods

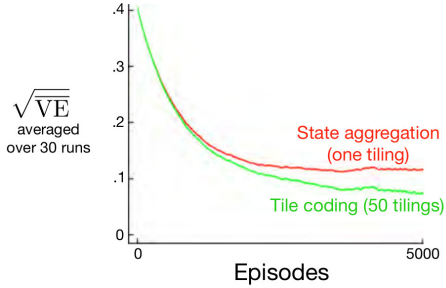


Figure 9.10: Why we use coarse coding. Shown are learning curves on the 1000-state random walk example for the gradient Monte Carlo algorithm with a single tiling and with multiple tilings. The space of 1000 states was treated as a single continuous dimension, covered with tiles each 200 states wide. The multiple tilings were offset from each other by 4 states. The step-size parameter was set so that the initial learning rate in the two cases was the same, $\alpha = 0.0001$ for the single tiling and $\alpha = 0.0001/50$ for the 50 tilings.

- Same size for all cells: easier choice of α.
- Easy computation.

9.5 Feature Construction for Linear Methods

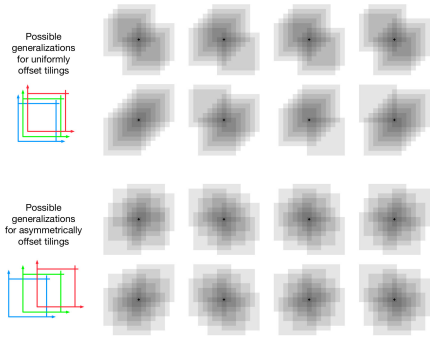


Figure 9.11: Why tile asymmetrical offsets are preferred in tile coding. Shown is the strength of generalization from a trained state, indicated by the small black plus, to nearby states, for the case of eight tilings. If the tilings are uniformly offset (above), then there are diagonal artifacts and substantial variations in the generalization, whereas with asymmetrically offset tilings the generalization is more spherical and homogeneous.

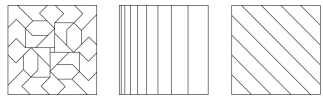


Figure 9.12: Tilings need not be grids. They can be arbitrarily shaped and non-uniform, while still in many cases being computationally efficient to compute.

- Different offset leads to different approximations.
- Large freedom on the tiling.
- Hashing can also be used

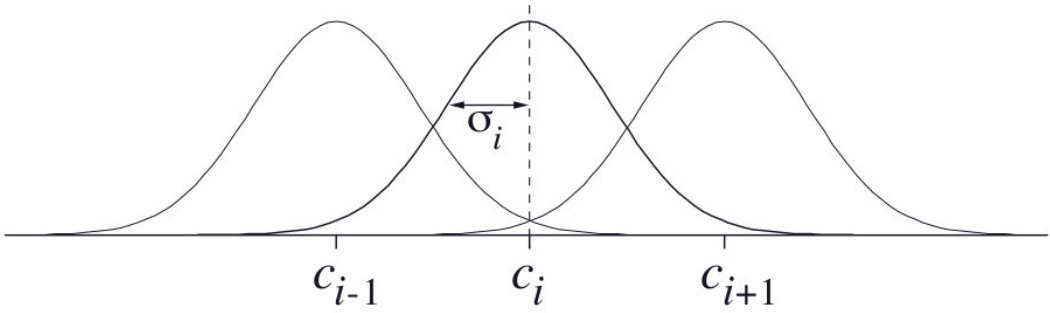


Figure 9.13: One-dimensional radial basis functions.

- Radial Basis Function:

$$\boldsymbol{x}(s)_i = \Phi(\|s - c_i\|^2/2\sigma^2)$$

- Smoothed version of tiling...

9.6 Selecting Step-Size Parameters Manually

- SGD requires the selection of an appropriate step-size.
- Theory proposes a slowly decreasing step-size ($O(1/t)$)
- Often very slow convergence and not adapted to non-stationary target.
- Intuition from tabular case: $\alpha = 1/\tau$ with τ the number of experience required to converge approximately.
- Rule of thumb for linear SGD

$$\alpha = (\tau \mathbb{E}\left[\boldsymbol{x}^\top \boldsymbol{x}\right])^{-1}$$

9.7 Nonlinear Function Approximation: Artificial Neural Networks

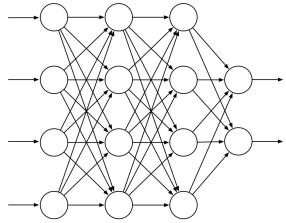


Figure 9.14: A generic feedforward ANN with four input units, two output units, and two hidden layers.

- Artificial Neural Networks are widely used for nonlinear function approximations.
- Complex architecture trained with gradient descent (backprop)
- Several clever tools: initialization, activation function, dropout, residual networks, batch normalization...
- See lecture on Deep Learning...
- Can be used in stochastic-gradient or semi-gradient method.

9.7 Nonlinear Function Approximation: Artificial Neural Networks

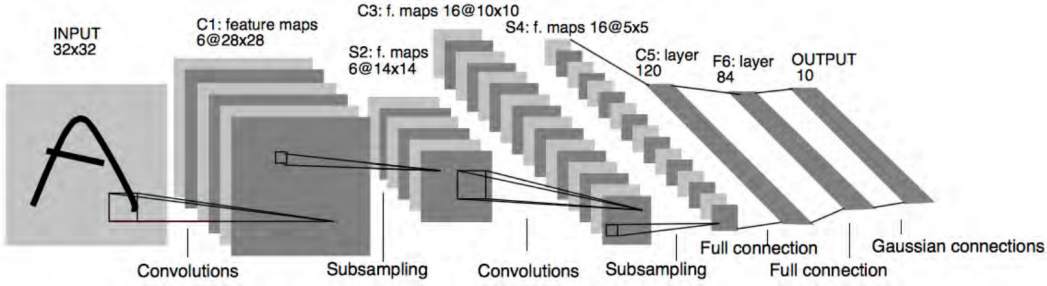


Figure 9.15: Deep Convolutional Network. Republished with permission of Proceedings of the IEEE, from Gradient-based learning applied to document recognition, LeCun, Bottou, Bengio, and Haffner, volume 86, 1998; permission conveyed through Copyright Clearance Center, Inc.

- Use of Convolutional Network to capitalize on partial translation invariance.

9.8 Least-Squares TD

- Better sample accuracy can be obtained through a different direction.
- The TD fixed point satisfy:

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

  with $\boldsymbol{A} = \mathbb{E}\left[\boldsymbol{x}_t(\boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1})^\top\right]$ and $\boldsymbol{b} = \mathbb{E}\left[R_{t+1}\boldsymbol{x}_t\right]$
- TD algorithm: iterative update of a $\boldsymbol{w}_t$.
- Least-Squares TD algorithm: iterative update of $\boldsymbol{A}_t$ and $\boldsymbol{b}_t$ and final application of the inverse formula.
- Natural estimates:

$$\boldsymbol{A}_t = \frac{1}{t}\sum_{k=0}^{t-1} \boldsymbol{x}_t(\boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1})^\top \quad \text{and} \quad \boldsymbol{b}_t = \frac{1}{t}\sum_{k=0}^{t-1} R_{t+1}\boldsymbol{x}_t$$

- To avoid some invertibility issue:

$$\widehat{\boldsymbol{A}_t} = \sum_{k=0}^{t-1} \boldsymbol{x}_t(\boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1})^\top + \epsilon \mathrm{Id} \quad \text{and} \quad \widehat{\boldsymbol{b}_t} = \sum_{k=0}^{t-1} R_{t+1}\boldsymbol{x}_t$$

- Missing $1/t$ factor cancels...

9.8 Least-Squares TD

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

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

- $O(d^2)$ per step but inversion required in the end...
- Even better algorithm by maintaining an estimate of $(t\boldsymbol{A})^{-1}$:

$$\widehat{\boldsymbol{A}_t}^{-1} = \left(\widehat{\boldsymbol{A}_{t-1}} + \boldsymbol{x}_t(\boldsymbol{x}_t - \gamma\boldsymbol{x}_{t+1})^\top\right)^{-1}$$

$$= \widehat{\boldsymbol{A}_{t-1}}^{-1} - \frac{\widehat{\boldsymbol{A}_{t-1}}^{-1}\boldsymbol{x}_t(\boldsymbol{x}_t - \gamma\boldsymbol{x}_{t+1})^\top \widehat{\boldsymbol{A}_{t-1}}^{-1}}{1 + (\boldsymbol{x}_t - \gamma\boldsymbol{x}_{t+1})^\top \widehat{\boldsymbol{A}_{t-1}}^{-1}\boldsymbol{x}_t}$$

- still $O(d^2)$ per step but no inversion in the end.
- No stepsize... but thus no forgetting...

9.9 Memory-based Function Approximation

- Non-parametric approach using memory.
- Requires an estimate of the target for a set of states.
- After (lazy) learning, each time a new query state arrives, one retrieve a set of close examples in the training dataset and deduces an estimate for the query state.
- Examples: nearest-neighbors, weighted average, locally weighted regression...
- Focus on states that are close to observed states.
- No need to be accurate far from typical states.
- Computational issue: need to find (approximate) nearest neighbors...

9.10 Kernel-based Function Approximation

- Kernel-based approximation is a weighted average strategy where the weights are defined by a kernel k measuring the similarity between states.
- Kernel-based approximation:

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

- Similar to RBF but kernel centered on the examples.
- Large freedom in the choice of kernel...
- Kernel tricks!

9.11 Looking Deeper at On-policy Learning: Interest and Emphasis

- Prediction Objective ($\overline{VE}$):
$$\overline{VE}(\boldsymbol{w}) = \sum_s \mu(s) \left(v_\pi(s) - \hat{v}(s, \boldsymbol{w})\right)^2$$
where μ is a state distribution.
- Not necessarily the most interesting goal!
- For instance, one may be more interested by earlier states than later states.
- Interest I_t defined for each state.
- General algorithm:
$$\boldsymbol{w}_{t+n} = \boldsymbol{w}_{t+n-1} + \alpha M_t (G_{t:t+n} - \hat{v}(S_t, \boldsymbol{w}_{t+n-1})) \nabla \hat{v}(S_t, \boldsymbol{w}_{t+n-1})$$
with
$$M_t = I_t + \gamma^n M_{t-n}$$
- Proof of convergence similar to TD(0).

Outline

10 On-policy Control with Approximation

- 10.1 Episodic Semi-gradient Control
- 10.2 Semi-gradient n-step Sarsa
- 10.3 Average Reward: A New Problem Setting for Continuing Tasks
- 10.4 Deprecating the Discounted Setting
- 10.5 Differential Semi-gradient n-step Sarsa

10.1 Episodic Semi-gradient Control

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

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
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:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

- Straightforward extension to $q_\pi(s, a) \sim \hat{q}(s, a, \boldsymbol{w})$.
- Prediction algorithm:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\left(U_t - \hat{q}(S_t, A_t, \boldsymbol{w}_t)\right)\nabla\hat{q}(S_t, A_t, \boldsymbol{w})$$

- On-policy control by adding a (ϵ-greedy) policy improvement step.

10.1 Episodic Semi-gradient Control

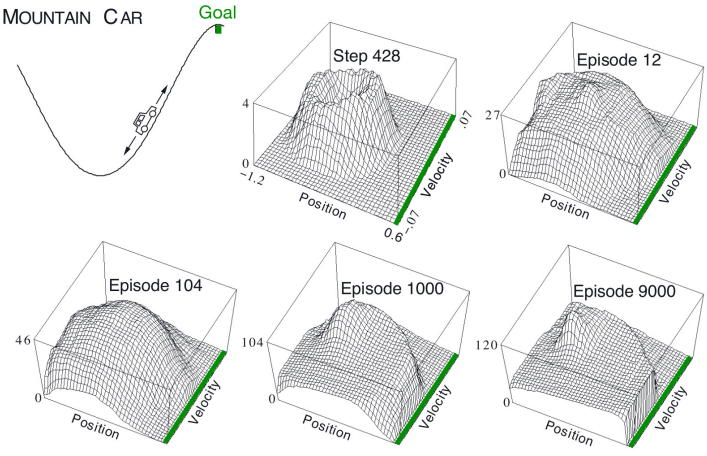


Figure 10.1: The Mountain Car task (upper left panel) and the cost-to-go function ($-\max_a \hat{q}(s, a, \mathbf{w})$) learned during one run.

- Grid tiling with 8 cells and asymmetrical offsets.

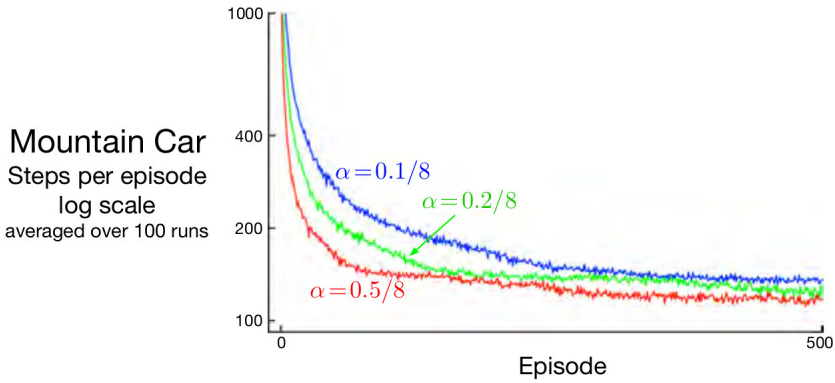


Figure 10.2: Mountain Car learning curves for the semi-gradient Sarsa method with tile-coding function approximation and ε-greedy action selection. ■

10.2 Semi-gradient n-step Sarsa

Episodic semi-gradient n-step Sarsa for estimating $\hat{q} \approx q_*$ or q_π

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$, a positive integer n
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
All store and access operations (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 $A_0 \sim \pi(\cdot|S_0)$ or ε-greedy wrt $\hat{q}(S_0, \cdot, \mathbf{w})$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
    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 $A_{t+1} \sim \pi(\cdot|S_{t+1})$ or ε-greedy wrt $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$
    $\tau \leftarrow t - n + 1$ (τ is the time whose 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 $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w})$ $(G_{\tau:\tau+n})$
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G - \hat{q}(S_\tau, A_\tau, \mathbf{w}) \right] \nabla \hat{q}(S_\tau, A_\tau, \mathbf{w})$
  Until $\tau = T - 1$

- Natural extension of tabular methods:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, \boldsymbol{w}_{t+n-1})$$

$$\boldsymbol{w}_{t+n} = \boldsymbol{w}_{t+n-1} + \alpha \left(G_{t:t+n} - \hat{q}(S_t, A_t, \boldsymbol{w}_{t+n-1}) \right) \nabla \hat{q}(S_t, A_t, \boldsymbol{w}_{t+n-1})$$

10.2 Semi-gradient *n*-step Sarsa

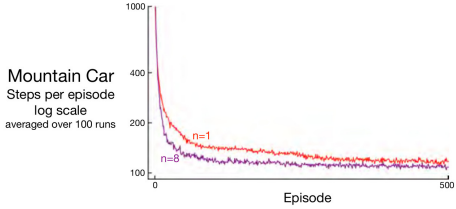


Figure 10.3: Performance of one-step vs 8-step semi-gradient Sarsa on the Mountain Car task. Good step sizes were used: $\alpha = 0.5/8$ for $n = 1$ and $\alpha = 0.3/8$ for $n = 8$.

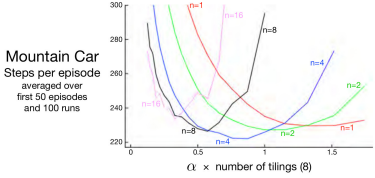


Figure 10.4: Effect of the α and n on early performance of n-step semi-gradient Sarsa and tile-coding function approximation on the Mountain Car task. As usual, an intermediate level of bootstrapping ($n = 4$) performed best. These results are for selected α values, on a log scale, and then connected by straight lines. The standard errors ranged from 0.5 (less than the line width) for $n = 1$ to about 4 for $n = 16$, so the main effects are all statistically significant.

10.3 Average Reward: A New Problem Setting for Continuing Tasks

- Continuous task without discounting.
- Average reward:

$$\begin{aligned} r(\pi) &= \lim_{h \to +\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}\left[R_t | S_0, A_{0:t-1} \sim \pi\right] \\ &= \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) r \end{aligned}$$

- *Ergodicity* assumption on the MDP: existence of

$$\mu_\pi(s) = \lim_{t \to +\infty} \mathbb{P}\left(S_t = s | A_{0:t-1} \sim \pi\right)$$

  which is independent of S_0 for any π.
- By construction,

$$\sum_s \mu_\pi \sum_a \pi(a|s) p(s'|s, a) = \mu_\pi(s')$$

10.3 Average Reward: A New Problem Setting for Continuing Tasks

- Differential returns:
$$G_t = R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + \cdots$$
- Differential value functions:
$$v_\pi(s) = \mathbb{E}_\pi\left[G_t | S_t = s\right] \quad \text{and} \quad q_\pi(s,a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right]$$
- Bellman:
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(s', r|s, a)\left(r - r(\pi) + v_\pi(s')\right)$$
$$q_\pi(s,a) = \sum_{r,s'} p(s', r|s, a)\left(r - r(\pi) + \sum_{a'} \pi(a'|s') q_\pi(s', a')\right)$$
- Optimality:
$$v_*(s) = \max_a \sum_{r,s'} p(s', r|s, a)\left(r - \max_\pi r(\pi) + v_*(s')\right)$$
$$q_\pi(s,a) = \sum_{r,s'} p(s', r|s, a)\left(r - \max_\pi r(\pi) + \max_{a'} \; q_\pi(s', a')\right)$$
- Rk: True derivation much more involved!

10.3 Average Reward: A New Problem Setting for Continuing Tasks

Differential semi-gradient Sarsa for estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step sizes $\alpha, \beta > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Initialize average reward estimate $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., $\bar{R} = 0$)

Initialize state S, and action A
Loop for each step:
  Take action A, observe R, S'
  Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
  $\delta \leftarrow R - \bar{R} + \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})$
  $\bar{R} \leftarrow \bar{R} + \beta\delta$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S, A, \mathbf{w})$
  $S \leftarrow S'$
  $A \leftarrow A'$

- Differential TD errors:

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

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

- Differential algorithm: for instance semi-gradient Sarsa

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\delta_t \nabla \hat{q}(S_t, A_t, \boldsymbol{w}_t)$$

10.3 Average Reward: A New Problem Setting for Continuing Tasks

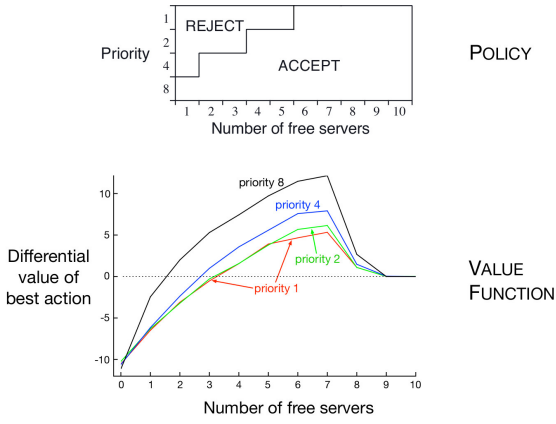


Figure 10.5: The policy and value function found by differential semi-gradient one-step Sarsa on the access-control queuing task after 2 million steps. The drop on the right of the graph is probably due to insufficient data; many of these states were never experienced. The value learned for $\bar{R}$ was about 2.31. ■

10.4 Deprecating the Discounted Setting

- In the ergodic setting, discounting is equivalent to averaging!

$$
\begin{aligned}
J(\pi) &= \sum_s \mu_\pi(s) v_{\pi,\gamma}(s) \\
&= \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_{\pi,\gamma}(s')) \\
&= r(\pi) + \gamma \sum_{s'} v_{\pi,\gamma}(s') \sum_s \mu_\pi(s) \sum_a p(s'|s, a)\pi(a|s) \\
&= r(\pi) + \gamma \sum_{s'} v_{\pi,\gamma}(s')\mu_\pi(s') \\
&= r(\pi) + \gamma J(\pi) \\
&= \frac{1}{1-\gamma} r(\pi)
\end{aligned}
$$

- Same optimal policies.
- Issue: with approximation, policy improvement theorem is lost (in finite, discounted and average cases).

10.5 Differential Semi-gradient n-step Sarsa

Differential semi-gradient n-step Sarsa for estimating $\hat{q} \approx q_\pi$ or q_*

Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$, a policy π
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Initialize average-reward estimate $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., $\bar{R} = 0$)
Algorithm parameters: step size $\alpha, \beta > 0$, a positive integer n
All store and access operations (S_t, A_t, and R_t) can take their index mod $n+1$

Initialize and store S_0 and A_0
Loop for each step, $t = 0, 1, 2, \ldots$:
  Take action A_t
  Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$, or ε-greedy wrt $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$
  $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  If $\tau \geq 0$:
    $\delta \leftarrow \sum_{i=\tau+1}^{\tau+n} (R_i - \bar{R}) + \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w}) - \hat{q}(S_\tau, A_\tau, \mathbf{w})$
    $\bar{R} \leftarrow \bar{R} + \beta\delta$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S_\tau, A_\tau, \mathbf{w})$

- Differential n-returns:

$$G_{t:t+n} = R_{t+1} - \overline{R}_{t+n-1} + \cdots + R_{t+n} - \overline{R}_{t+n-1} + \hat{q}(S_{t+n}, A_{t+}, \boldsymbol{w}_{t+n-1})$$

where $\overline{R}_{t+n-1}$ is an estimate of $r(\pi)$.

Outline

11 Off-policy Methods with Approximation

11.1 Semi-gradient Methods

- Off-policy learning: two challenges
  - dealing with the target update
  - dealing with the distributions of update
- First-part is easier...
- Importance sampling ratio:

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

- Semi-gradient off-policy TD(0):

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\rho_t\delta_t\nabla\hat{v}(S_t, \boldsymbol{w}_t)$$

with

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{W}_t) \quad \text{(episodic)} \\ &= R_{t+1} - \overline{R}_t + \hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{W}_t) \quad \text{(continuing)}\end{aligned}$$

- Semi-gradient Expected Sarsa:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\delta_t \nabla \hat{q}(S_t, A_t, \boldsymbol{w}_t)$$

with

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma \sum \pi(a|S_t+1)\hat{q}(S_{t+1}, a, \boldsymbol{w}_t) \\ &\quad - \hat{q}(S_t, A_t, \boldsymbol{W}_t) \quad \text{(episodic)} \\ &= R_{t+1} - \overline{R}_t + \sum \pi(a|S_t+1)\hat{q}(S_{t+1}, a, \boldsymbol{w}_t) \\ &\quad - \hat{q}(S_t, A_t, \boldsymbol{W}_t) \quad \text{(continuing)}\end{aligned}$$

11.1 Semi-gradient Methods

- Multi-step Semi-gradient Sarsa:

$$\boldsymbol{w}_{t+n} = \boldsymbol{w}_{t+n-1} + \alpha\rho_{t+1} \cdot \rho_{t+n-1}\delta_t \nabla \hat{q}(S_t, A_t, \boldsymbol{w}_t)$$

with

$$\begin{aligned}\delta_t &= R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, \boldsymbol{w}_{t+n-1}) \\ &\quad - \hat{q}(S_t, A_t, \boldsymbol{W}_t) \quad \text{(episodic)} \\ &= R_{t+1} - \overline{R}_t + \cdots + R_{t+n} - \overline{R}_{t+n-1} + \hat{q}(S_{t+n}, A_{t+n}, \boldsymbol{w}_{t+n-1}) \\ &\quad - \hat{q}(S_t, A_t, \boldsymbol{W}_t) \quad \text{(continuing)}\end{aligned}$$

- Tree-backup:

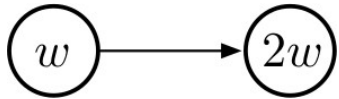
$$\boldsymbol{w}_{t+n} = \boldsymbol{w}_{t+n-1} + \alpha \left(G_{t:t+n} - \hat{q}(S_t, A_t, w_{t+n-1})\right) \nabla \hat{q}(S_t, A_t, \boldsymbol{w}_t)$$

with

$$G_{t:t+n} = \hat{q}(S_t, A_t, w_{t-1}) + \sum_{k=t}^{t+n-1} \delta_k \prod_{i=t+1}^{k} \gamma\pi(A_i|S_i)$$

and δ_k as in Expected Sarsa.

11.2 Examples of Off-policy Divergence



- Simple transition with a reward 0.
- TD error:

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{w}_t) \\ &= 0 + \gamma 2 w_t - w_t = (2\gamma - 1) w_t\end{aligned}$$

- Off-policy semi-gradient TD(0) update:

$$\begin{aligned}w_{t+1} &= w_t + \alpha \rho_t \delta_t \nabla \hat{v}(S_{t+1}, \boldsymbol{w}_t) \\ &= w_t + \alpha \times 1 \times (2\gamma - 1) w_t = (1 + \alpha(2\gamma - 1)) w_t\end{aligned}$$

- Explosion if this transition is explored without $\boldsymbol{w}$ being update on other transitions as soon as $\gamma > 1/2$.

11.2 Examples of Off-policy Divergence

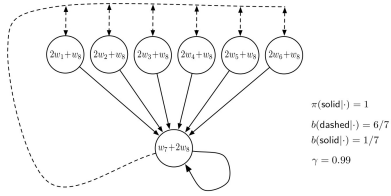


Figure 11.1: Baird's counterexample. The approximate state-value function for this Markov process is of the form shown by the linear expressions inside each state. The solid action usually results in the seventh state, and the dashed action usually results in one of the other six states, each with equal probability. The reward is always zero.

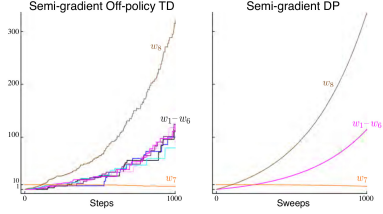
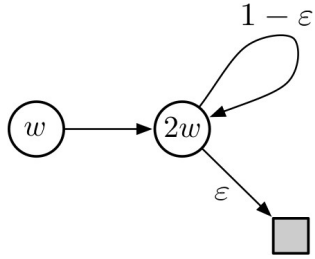


Figure 11.2: Demonstration of instability on Baird's counterexample. Shown are the evolution of the components of the parameter vector $\mathbf{w}$ of the two semi-gradient algorithms. The step size was $\alpha = 0.01$, and the initial weights were $\mathbf{w} = (1, 1, 1, 1, 1, 1, 10, 1)^\top$.

- Divergence of off-policy algorithm.

11.2 Examples of Off-policy Divergence



- Exact minimization of bootstrapped $\overline{VE}$ at each step:

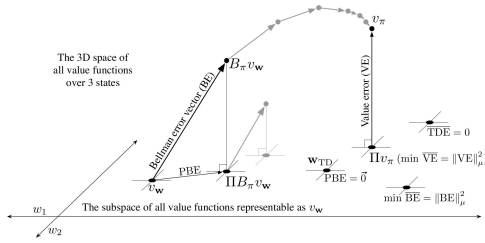
$$\begin{aligned} w_{k+1} &= \operatorname*{argmin}_w \sum_s \left(\hat{v}(s,w) - \mathbb{E}_\pi \left[R_{t+1} + \gamma \hat{v}(S_{t+1}, w_k) | S_t = s\right]\right)^2 \\ &= \operatorname*{argmin}_w (w - \gamma 2 w_k)^2 + (2w - (1-\epsilon)\gamma 2 w_k)^2 \\ &= \frac{6-4\epsilon}{5}\gamma w_k \end{aligned}$$

- Divergence if $\gamma > 5/(6-4\epsilon)$.

11.3 The Deadly Triad

- Deadly triad:
  - **Function approximation**
  - **Bootstrapping**
  - **Off-policy training**
- Instabilities as soon as the three are present!
- Issue:
  - Function approximation is unavoidable.
  - Bootstrap is much more computational and data efficient.
  - Off-policy may be avoided... but essential when dealing with extended setting (learn from others or learn several tasks)

11.4 Linear Value-function Geometry



- Natural weighted squared norm:

$$\|v\|^2_\mu = \sum_s \mu(s) v(s)^2$$

- Prediction objective $\overline{VE}$:

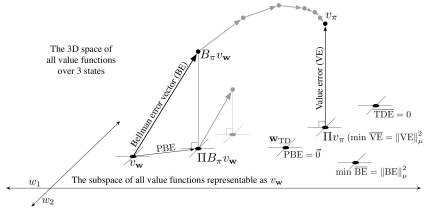
$$\overline{VE}(\boldsymbol{w}) = \|v_{\boldsymbol{w}} - v_\pi\|^2_\mu$$

- Projection operator Π:

$$\Pi v = v_{\boldsymbol{w}} \quad \text{where} \quad \boldsymbol{w} = \operatorname*{argmin}_{\boldsymbol{w}} \|v - v_{\boldsymbol{w}}\|^2$$

- Target for MC methods

11.4 Linear Value-function Geometry



- Bellman error:

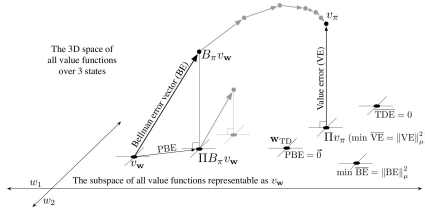
$$\delta_\mathbf{w}(s) = \left(\sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_\mathbf{w}(s')) \right) - v_\mathbf{w}(s)$$
$$= \mathbb{E}_\pi \left[R_{t+1} - \gamma v_\mathbf{w}(S_{t+1}) - v_\mathbf{w}(S_t) | S_t = s, A_t \sim \pi \right]$$

- Expectation of TD error.
- Mean Squared Bellmann Error:

$$\overline{BE}(\mathbf{w}) = \|\delta_\mathbf{w}(s)\|^2_\mu$$

- Unique minimizer in linear case
- Different from minimizer of $\overline{VE}$.

11.4 Linear Value-function Geometry



- Bellman operator:
$$B_\pi(v)(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_{\mathbf{w}}(s'))$$
- $B_\pi(v_{\mathbf{w}})(s)$ is in general not representable.
- Projected Bellman operator: $\Pi B_\pi v$
- Projected Bellman iteration: $v_{\mathbf{w}_{t+1}} = \Pi B_\pi v_{\mathbf{w}_t} \dots$
- Mean Square Projected Bellman error:
$$\overline{PBE}(\mathbf{w}) = \|\Pi B_\pi v_{\mathbf{w}} - v_{\mathbf{w}}\|^2_\mu = \|\Pi \delta_{\mathbf{w}}\|^2_\mu$$

11.5 Gradient Descent in the Bellman Error

- With approximation, only MC so far is a SGD method...
- TD error:

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

- Mean Squared TD Error:

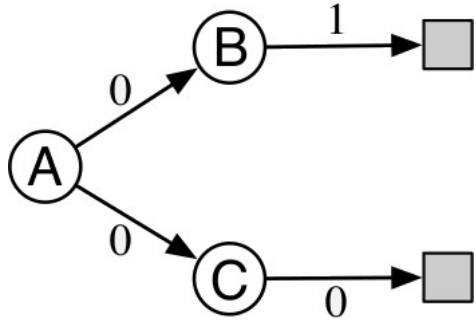
$$\begin{aligned}\overline{TDE}(\boldsymbol{w}) &= \sum_s \mu(s) \mathbb{E}\left[\delta_t^2 | S_t = s, A_t \sim \pi\right] \\ &= \mathbb{E}_b\left[\rho_t \delta_t^2\right]\end{aligned}$$

  with μ linked to b.
- Minimization of an expectation leads to SGD.
- SGD algorithm:

$$\begin{aligned}\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - 1/2 \alpha \nabla(\rho_t \delta_t^2) \\ &= \boldsymbol{w}_t + \alpha \rho_t \delta_t (\nabla \hat{v}(S_t, \boldsymbol{w}_t) - \gamma \nabla \hat{v}(S_{t+1}, \boldsymbol{w}_t))\end{aligned}$$

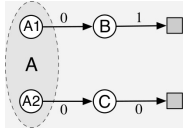
- Convergence guarantees for this naïve residual-gradient algorithm.

11.5 Gradient Descent in the Bellman Error



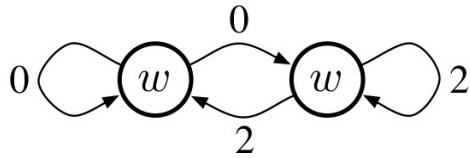
- Solution not necessarily interesting!
- True value function is easy: $v_\pi(A) = 1/2,\ v_\pi(B) = 1,\ v_\pi(C) = 0.$
- Optimal $\overline{TDE}$ attained for: $v_\pi(A) = 1/2,\ v_\pi(B) = 3/4,\ v_\pi(C) = 1/4!$

11.5 Gradient Descent in the Bellman Error



- Bellman error can also be measured by
$$\left(\mathbb{E}_\pi\left[\delta_t\right]\right)^2$$
- SGD type algorithm:
$$\begin{aligned}\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - 1/2\alpha\nabla\left(\mathbb{E}_\pi\left[\delta_t\right]\right)^2 \\ &= \boldsymbol{w}_t - \alpha\mathbb{E}_b\left[\rho_t\delta_t\right]\nabla\mathbb{E}_b\left[\rho_t\delta_t\right] \\ &= \boldsymbol{w}_t + \alpha\left(\mathbb{E}_b\left[\rho_t(R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}_t))\right] - \hat{v}(S_t, \boldsymbol{w}_t)\right) \\ &\qquad \times \left(\nabla\hat{v}(S_t, \boldsymbol{w}_t) - \gamma\mathbb{E}_b\left[\rho_t\nabla\hat{v}(S_{t+1}, \boldsymbol{w}_t)\right]\right)\end{aligned}$$
- Requires two independent samples for S_{t+1}!
- Converges toward the true value in the tabular case.
- Can be slow. . . and the solution with approximation may be unsatisfactory.

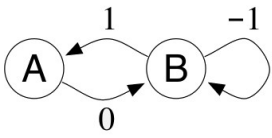
11.6 The Bellman Error is not Learnable

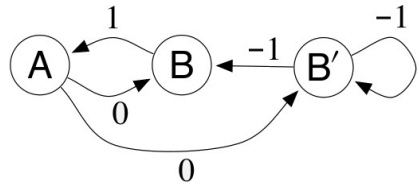


- Two MRP with the same outputs (because of approximation).
- but different $\overline{VE}$.
- Impossibility to learn $\overline{VE}$.
- Minimizer however is learnable:

$$\begin{aligned}\overline{RE}(\boldsymbol{w}) &= \mathbb{E}\left[(G_t - \hat{v}(S_t, \boldsymbol{w}))^2\right] \\ &= \overline{VE}(\boldsymbol{w}) + \mathbb{E}\left[(G_t - v_\pi(S_t))^2\right]\end{aligned}$$

11.6 The Bellman Error is not Learnable





- Two MRP with the same outputs (because of approximation).
- Different $\overline{BE}$.
- Different minimizer!
- $\overline{BE}$ is not learnable!

11.6 The Bellman Error is not Learnable

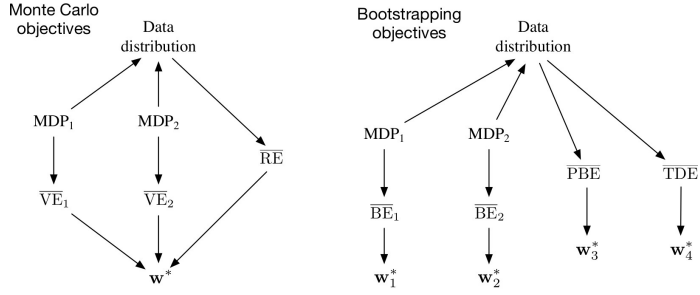


Figure 11.4: Causal relationships among the data distribution, MDPs, and various objectives. **Left, Monte Carlo objectives:** Two different MDPs can produce the same data distribution yet also produce different $\overline{\text{VE}}$s, proving that the $\overline{\text{VE}}$ objective cannot be determined from data and is not learnable. However, all such $\overline{\text{VE}}$s must have the same optimal parameter vector, $\mathbf{w}^*$! Moreover, this same $\mathbf{w}^*$ can be determined from another objective, the $\overline{\text{RE}}$, which *is* uniquely determined from the data distribution. Thus $\mathbf{w}^*$ and the $\overline{\text{RE}}$ are learnable even though the $\overline{\text{VE}}$s are not. **Right, Bootstrapping objectives:** Two different MDPs can produce the same data distribution yet also produce different $\overline{\text{BE}}$s *and* have different minimizing parameter vectors; these are not learnable from the data distribution. The $\overline{\text{PBE}}$ and $\overline{\text{TDE}}$ objectives and their (different) minima can be directly determined from data and thus are learnable.

- $\overline{PBE}$ and $\overline{TDE}$ are learnable.

11.7 Gradient-TD Methods

- SGD Methods for minimizing $\overline{PBE}$.

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

- Gradient:

$$\overline{PBE}(\boldsymbol{w}) = 2\nabla(\boldsymbol{X}^\top D\delta_{\boldsymbol{w}})^\top (\boldsymbol{X}^\top D\boldsymbol{X})^{-1}(\boldsymbol{X}^\top D\delta_{\boldsymbol{w}})$$

- Expectations:

$$\begin{aligned}\boldsymbol{X}^\top D\delta_{\boldsymbol{w}} &= \mathbb{E}\left[\rho_t \delta_t \boldsymbol{x}_t\right] \\ \nabla(\boldsymbol{X}^\top D\delta_{\boldsymbol{w}})^\top &= \mathbb{E}\left[\rho_t(\gamma \boldsymbol{x}_{t+1} - \boldsymbol{x}_t)\boldsymbol{x}^\top\right] \\ \boldsymbol{X}^\top D\boldsymbol{X} &= \mathbb{E}\left[\boldsymbol{x}_t \boldsymbol{x}_t^\top\right]\end{aligned}$$

- Not yet a SGD...

11.7 Gradient-TD Methods

- Expectations:

$$\nabla(\boldsymbol{X}^\top D\delta_{\boldsymbol{w}})^\top = \mathbb{E}\left[\rho_t(\gamma \boldsymbol{x}_{t+1} - \boldsymbol{x}_t)\boldsymbol{x}^\top\right]$$

$$\boldsymbol{X}^\top D\boldsymbol{X} = \mathbb{E}\left[\boldsymbol{x}_t \boldsymbol{x}_t^\top\right]$$

$$\boldsymbol{X}^\top D\delta_{\boldsymbol{w}} = \mathbb{E}\left[\rho_t \delta_t \boldsymbol{x}_t\right]$$

- Learn the product $\boldsymbol{v}$ of last two terms:

$$\boldsymbol{v} \simeq \mathbb{E}\left[\boldsymbol{x}_t \boldsymbol{x}_t^\top\right]^{-1} \mathbb{E}\left[\rho_t \delta_t \boldsymbol{x}_t\right]$$

- Solution of a least square
- Iterative algorithm:

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t + \beta\rho_t(\delta_t - \boldsymbol{v}_t^\top \boldsymbol{x}_t)\boldsymbol{x}_y$$

- Insert expression in the gradient to obtain GTD

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\rho_t(\gamma \boldsymbol{x}_{t+1} - \boldsymbol{x}_t)\boldsymbol{x}^\top \boldsymbol{v}_t$$

  or GTD2

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\rho_t(\delta \boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1}\boldsymbol{x}_t \boldsymbol{x}^\top \boldsymbol{v}_t)$$

- Convergence with $\beta \to 0$ and $\alpha/\beta \to 0$ (two scales)

11.7 Gradient-TD Methods

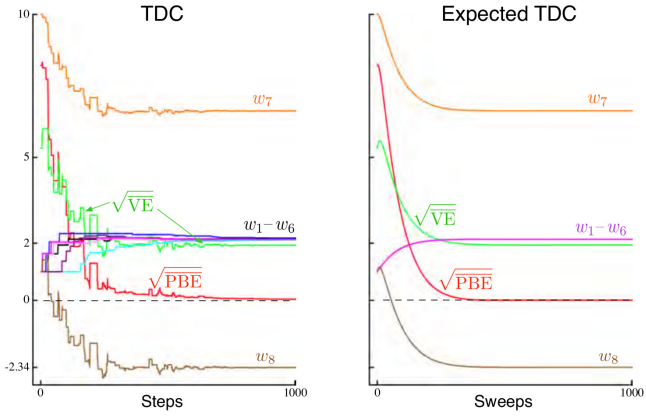


Figure 11.5: The behavior of the TDC algorithm on Baird's counterexample. On the left is shown a typical single run, and on the right is shown the expected behavior of this algorithm if the updates are done synchronously (analogous to (11.9), except for the two TDC parameter vectors). The step sizes were $\alpha = 0.005$ and $\beta = 0.05$.

11.8 Emphatic-TD Methods

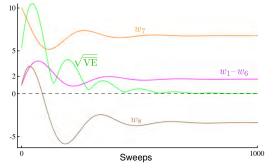


Figure 11.6: The behavior of the one-step Emphatic-TD algorithm in expectation on Baird's counterexample. The step size was $\alpha = 0.03$.

- Linear semi-gradient TD methods convergence due to a match between the on-policy distribution and the transition prob.
- Emphatic-TD: clever manner to weight the examples in off-policy that maintains the compatibility.
- One-step Emphatic-RD:

$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{w}_t)$$
$$M_t = \gamma \rho_{t-1} M_{t-1} + I_t$$
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha M_t \rho_t \delta_t \nabla \hat{v}(S_t, \boldsymbol{w}_t)$$

where I_t is an arbitrary interest and M_t the emphasis ($M_0 = 0$).

11.9 Reducing Variance

- Off-policy methods have more variance than off-policy ones.
- High variability of importance sampling ratio.
- Tree Backup does not have importance ratio.
- Other variants such as Retrace.
- Variance can also be reduced by more strongly linking the behavior and target policies.

Outline

12 Eligibility Traces

12.1 The λ-return

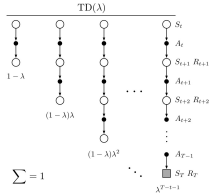


Figure 12.1: The backup digram for TD(λ). If $\lambda = 0$, then the overall update reduces to its first component, the one-step TD update, whereas if $\lambda = 1$, then the overall update reduces to its last component, the Monte Carlo update.

- n-step return:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, \boldsymbol{w}_{t+n-1})$$

- Averaged n-step return: (compound update)

$$G_t^\omega = \sum_{n=1}^{\infty} \omega_n G_{t:t+n} \quad \text{with} \sum_{i=1}^{\infty} \omega_n = 1$$

- TD(λ): specific averaging

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

12.1 The λ-return

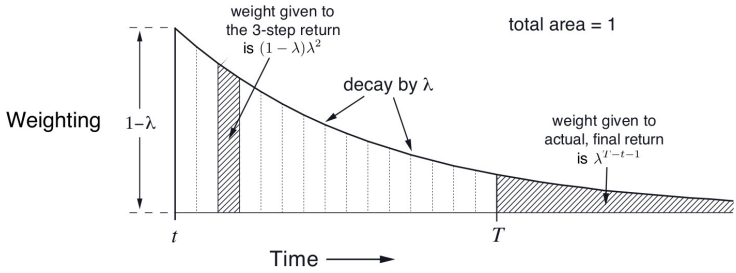


Figure 12.2: Weighting given in the λ-return to each of the n-step returns.

- TD(λ): interpolation between TD(0) and MC

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

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

12.1 The λ-return

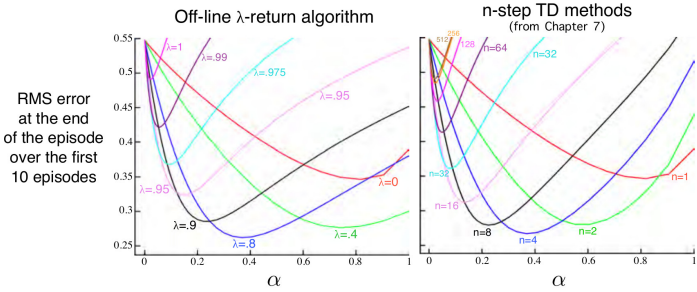


Figure 12.3: 19-state Random walk results (Example 7.1): Performance of the offline λ-return algorithm alongside that of the n-step TD methods. In both case, intermediate values of the bootstrapping parameter (λ or n) performed best. The results with the offline λ-return algorithm are slightly better at the best values of α and λ, and at high α.

- offline λ-return algorithm:
  - Play an episode according to the policy
  - Afterwards, modify $\boldsymbol{w}$ according to

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(G_t^\lambda - \hat{v}(S_t, \boldsymbol{w}_t) \right) \nabla \hat{v}(S_t, \boldsymbol{w}_t), \quad t = 0, \ldots, T$$

12.1 The λ-return

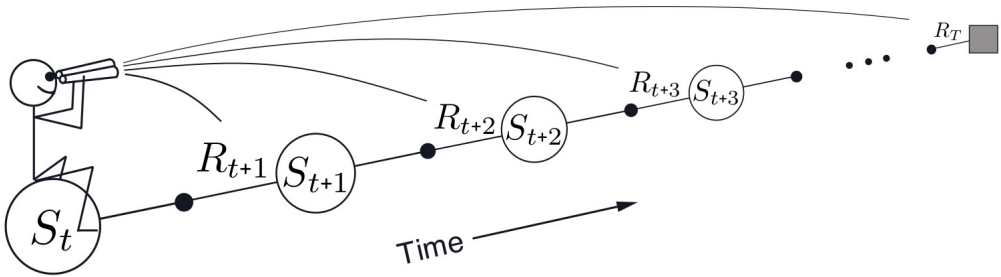


Figure 12.4: The forward view. We decide how to update each state by looking forward to future rewards and states.

- Forward view!
- Need to wait till the end before any update.

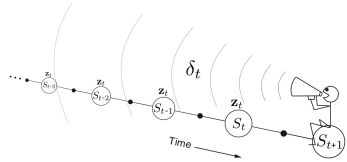


Figure 12.5: The backward or mechanistic view of TD(λ). Each update depends on the current TD error combined with the current eligibility traces of past events.

- Backward view!
- Eligibility trace $\boldsymbol{z}_t$ keeping track of the components of $\boldsymbol{w}$ that have contributed to state valuations:

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

- TD Error:

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

- Update:

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

12.2 TD(λ)

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

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ 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

- $TD(\lambda)$:

$$\boldsymbol{z}_t = \gamma\lambda\boldsymbol{z}_{t-1} + \nabla\hat{v}(S_t, \boldsymbol{w}_t)$$
$$\delta_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{w}_t)$$
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\delta_t\boldsymbol{z}_t$$

- $TD(0)$ is the classical TD algorithm.
- $TD(1)$ is similar to a MC algorithm but with parameter update before the end of the episode.

12.2 TD(λ)

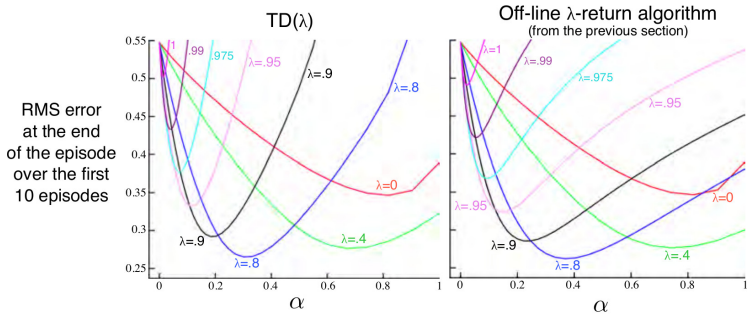


Figure 12.6: 19-state Random walk results (Example 7.1): Performance of TD(λ) alongside that of the offline λ-return algorithm. The two algorithms performed virtually identically at low (less than optimal) α values, but TD(λ) was worse at high α values.

- Not the same as using λ-return...
- Convergence guarantees in the linear case:

$$\overline{VE}(\boldsymbol{w}_\infty) \leq \frac{1-\gamma\lambda}{1-\gamma} \min_{\boldsymbol{w}} \overline{VE}(\boldsymbol{w})$$

12.3 n-step Truncated λ-return Methods

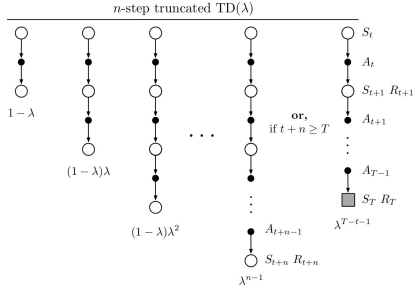


Figure 12.7: The backup diagram for truncated TD(λ).

- λ-return is technically never known in the continuing case!
- Truncated λ-return:

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

- Similar to an episodic setting of length h.

12.3 n-step Truncated λ-return Methods

- Family of n-step λ-return...
- TTD(λ):

$$\boldsymbol{w}_{t+n} = \boldsymbol{w}_{t+n-1} + \alpha \left(G^{\lambda}_{t:t+n} - \hat{v}(S_t, \boldsymbol{w}_{t+n-1}) \right) \nabla \hat{v}(S_t, \boldsymbol{w}_{t+n-1})$$

- Efficient implementation relying on:

$$G^{\lambda}_{t:t+k} = \hat{v}(S_t, \boldsymbol{w}_{t+1}) + \sum_{i=t}^{t+k-1} (\gamma\lambda)^{i-1} \delta_i$$

with

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

- Forward view but strong connection with backward one.

12.4 Redoing Updates: Online λ-return

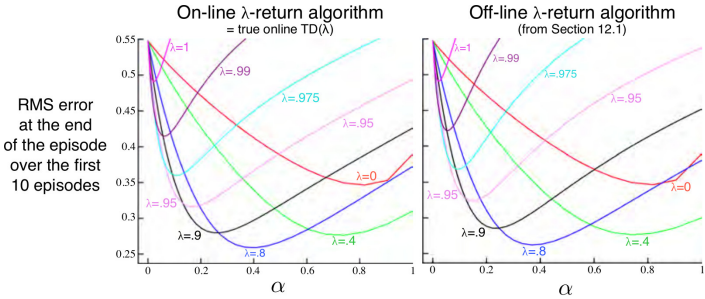


Figure 12.8: 19-state Random walk results (Example 7.1): Performance of online and offline λ-return algorithms. The performance measure here is the $\overline{\text{VE}}$ at the end of the episode, which should be the best case for the offline algorithm. Nevertheless, the online algorithm performs subtly better. For comparison, the $\lambda = 0$ line is the same for both methods.

- Conceptual algorithm which computes a weight at any time t using only the returns known at this time.
- Pass on all the data at each time step using the largest possible horizon h:

$$\boldsymbol{w}_{t+1}^h = \boldsymbol{w}_t^h + \alpha \left(G_{t:h}^\lambda - \hat{v}(S_t, \boldsymbol{w}_t^h) \right) \nabla \hat{v}(S_t, \boldsymbol{w}_t), \quad 0 \leq t < h \leq T$$

12.5 True Online TD(λ)

$$
\begin{array}{ccccc}
\boldsymbol{w}_0^0 & & & & \\
\boldsymbol{w}_0^1 & \boldsymbol{w}_1^1 & & & \\
\boldsymbol{w}_0^2 & \boldsymbol{w}_1^2 & \boldsymbol{w}_2^2 & & \\
\vdots & \vdots & \vdots & \ddots & \\
\boldsymbol{w}_0^T & \boldsymbol{w}_1^T & \boldsymbol{w}_2^T & \cdots & \boldsymbol{w}_T^T
\end{array}
$$

- Online λ-return algorithm has a triangular structure.
- True online TD(λ) computes $\boldsymbol{w}_{t+1}^{t+1}$ directly from $\boldsymbol{w}_t^t$.
- Linear case:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\delta_t \boldsymbol{z}_t + \alpha(\boldsymbol{w}_t^\top \boldsymbol{x}_t - \boldsymbol{w}_{t-1}^\top \boldsymbol{x}_t)(\boldsymbol{z}_t - \boldsymbol{x}_t)$$

with

$$\boldsymbol{z}_t = \gamma\lambda \boldsymbol{z}_{t-1} + (1 - \alpha\gamma\lambda \boldsymbol{z}_{t-1}^\top \boldsymbol{x}_t)\boldsymbol{x}_t$$

12.5 True Online TD(λ)

True online TD(λ) for estimating $\mathbf{w}^\top \mathbf{x} \approx v_\pi$

Input: the policy π to be evaluated
Input: a feature function $\mathbf{x} : \mathcal{S}^+ \to \mathbb{R}^d$ such that $\mathbf{x}(terminal, \cdot) = \mathbf{0}$
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize state and obtain initial feature vector $\mathbf{x}$
  $\mathbf{z} \leftarrow \mathbf{0}$ (a d-dimensional vector)
  $V_{old} \leftarrow 0$ (a temporary scalar variable)
  Loop for each step of episode:
    Choose $A \sim \pi$
    Take action A, observe R, $\mathbf{x}'$ (feature vector of the next state)
    $V \leftarrow \mathbf{w}^\top \mathbf{x}$
    $V' \leftarrow \mathbf{w}^\top \mathbf{x}'$
    $\delta \leftarrow R + \gamma V' - V$
    $\mathbf{z} \leftarrow \gamma\lambda \mathbf{z} + \big(1 - \alpha\gamma\lambda \mathbf{z}^\top \mathbf{x}\big)\, \mathbf{x}$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + V - V_{old})\mathbf{z} - \alpha(V - V_{old})\mathbf{x}$
    $V_{old} \leftarrow V'$
    $\mathbf{x} \leftarrow \mathbf{x}'$
  until $\mathbf{x}' = \mathbf{0}$ (signaling arrival at a terminal state)

12.5 True Online TD(λ)

- Dutch trace:

$$\boldsymbol{z}_t = \gamma\lambda\boldsymbol{z}_{t-1} + (1 - \alpha\gamma\lambda\boldsymbol{z}_{t-1}^\top\boldsymbol{x}_t)\boldsymbol{x}_t$$

- Accumulating trace TD(λ):

$$\boldsymbol{z}_t = \gamma\lambda\boldsymbol{z}_{t-1} + \boldsymbol{x}_t$$

- Replacing trace:

$$\boldsymbol{z}_t = \begin{cases} 1 & \text{if } x_{i,t} = 1 \\ \gamma\lambda z_{i,t-1} & \text{otherwise} \end{cases}$$

was a precursor of Dutch traces for binary features.

12.6 Dutch Traces in Monte Carlo Learning

- Linear MC update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(G - \boldsymbol{w}_t^\top \boldsymbol{x}_t \right) \boldsymbol{x}_t$$

- Here G is a single reward known at time T.
- All the computation have to be done at the end to obtain $\boldsymbol{w}_T$.
- More efficient implementation using:

$$\begin{aligned}
\boldsymbol{w}_T &= \boldsymbol{w}_{T+1} + \alpha \left(G - \boldsymbol{w}_{T-1}^\top \boldsymbol{x}_{T-1} \right) \boldsymbol{x}_{T-1} \\
&= (\mathrm{Id} - \alpha \boldsymbol{x}_{T-1} \boldsymbol{x}_{T-1}^\top) \boldsymbol{w}_{T-1} + \alpha G \boldsymbol{x}_{T-1} \\
&= \boldsymbol{F}_{T-1} \boldsymbol{w}_{T-1} + \alpha G \boldsymbol{x}_{T-1} \\
&= \boldsymbol{F}_{T-1} \boldsymbol{F}_{T-2} \boldsymbol{w}_{T-2} + \alpha G (\boldsymbol{F}_{T-2} \boldsymbol{x}_{T-2} + \boldsymbol{x}_{T-1}) \\
&= \underbrace{\boldsymbol{F}_{T-1} \boldsymbol{F}_{T-2} \cdots \boldsymbol{F}_0 \boldsymbol{w}_0}_{\boldsymbol{a}_{T-1}} + \alpha G \underbrace{\sum_{k=0}^{T-1} \boldsymbol{F}_{T-1} \cdots \boldsymbol{F}_{k+1} \boldsymbol{x}_k}_{\boldsymbol{z}_{T-1}}
\end{aligned}$$

  with $\boldsymbol{F}_t = \mathrm{Id} - \alpha \boldsymbol{x}_t (\boldsymbol{x}_t)^\top$.
- Recursive online implementation possible.

12.6 Dutch Traces in Monte Carlo Learning

- Furthermore,

$$\begin{aligned} \boldsymbol{z}_t &= \sum_{k=0}^{t} \boldsymbol{F}_t \cdots \boldsymbol{F}_{k+1} \boldsymbol{x}_k \\ &= F_t \boldsymbol{z}_{t-1} + \boldsymbol{x}_t \\ &= \boldsymbol{z}_{t-1} + (1 - \alpha \boldsymbol{z}_{t-1}^\top \boldsymbol{x}_t) \boldsymbol{x}_t \end{aligned}$$

  and

$$\boldsymbol{a}_t = \boldsymbol{F}_t \cdot \boldsymbol{F}_0 \boldsymbol{w}_0 = \boldsymbol{F}_t \boldsymbol{a}_{t-1} = a_{t-1} - \alpha \boldsymbol{x}_t \boldsymbol{x}_t^\top a_{t-1}$$

- Efficient computation with only $O(d)$ operation per step.
- Eligibility traces arise whenever one tries to learn a long-term prediction in an efficient manner.

12.7 Sarsa(λ)

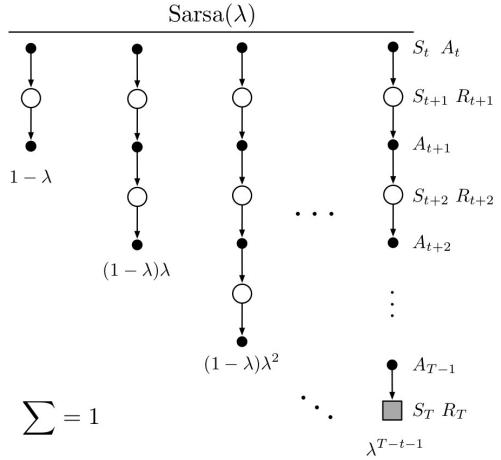
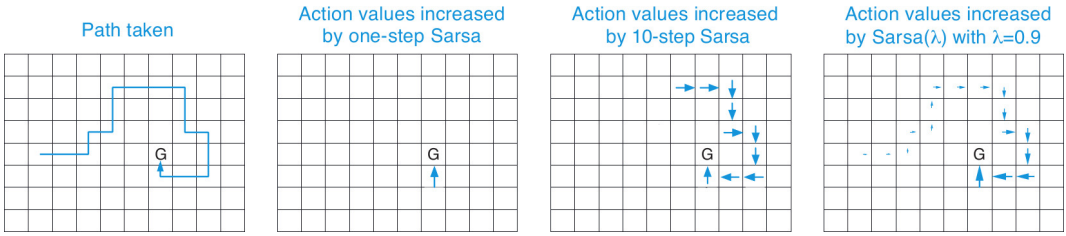


Figure 12.9: Sarsa(λ)'s backup diagram. Compare with Figure 12.1.

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

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

12.7 Sarsa(λ)



- Backward view:

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

with

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

and

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

$$\boldsymbol{z}_t = \gamma \lambda \boldsymbol{z}_{t-1} + \nabla \hat{q}(S_t, A_t, \boldsymbol{w}_t)$$

12.7 Sarsa(λ)

Sarsa(λ) with binary features and linear function approximation for estimating $\mathbf{w}^\top \mathbf{x} \approx q_\pi$ or q_*

Input: a function $\mathcal{F}(s,a)$ returning the set of (indices of) active features for s,a
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0,1]$
Initialize: $\mathbf{w} = (w_1, \ldots, w_d)^\top \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$), $\mathbf{z} = (z_1, \ldots, z_d)^\top \in \mathbb{R}^d$

Loop for each episode:
  Initialize S
  Choose $A \sim \pi(\cdot|S)$ or ε-greedy according to $\hat{q}(S, \cdot, \mathbf{w})$
  $\mathbf{z} \leftarrow \mathbf{0}$
  Loop for each step of episode:
    Take action A, observe R, S'
    $\delta \leftarrow R$
    Loop for i in $\mathcal{F}(S,A)$:
      $\delta \leftarrow \delta - w_i$
      $z_i \leftarrow z_i + 1$ (accumulating traces)
      or $z_i \leftarrow 1$ (replacing traces)
    If S' is terminal then:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
      Go to next episode
    Choose $A' \sim \pi(\cdot|S')$ or near greedily $\sim \hat{q}(S', \cdot, \mathbf{w})$
    Loop for i in $\mathcal{F}(S', A')$: $\delta \leftarrow \delta + \gamma w_i$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
    $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z}$
    $S \leftarrow S'; A \leftarrow A'$

12.7 Sarsa(λ)

True online Sarsa(λ) for estimating $\mathbf{w}^\top \mathbf{x} \approx q_\pi$ or q_*

Input: a feature function $\mathbf{x} : \mathcal{S}^+ \times \mathcal{A} \to \mathbb{R}^d$ such that $\mathbf{x}(terminal, \cdot) = \mathbf{0}$
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$
Initialize: $\mathbf{w} \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Choose $A \sim \pi(\cdot|S)$ or near greedily from S using $\mathbf{w}$
  $\mathbf{x} \leftarrow \mathbf{x}(S, A)$
  $\mathbf{z} \leftarrow \mathbf{0}$
  $Q_{old} \leftarrow 0$
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose $A' \sim \pi(\cdot|S')$ or near greedily from S' using $\mathbf{w}$
    $\mathbf{x}' \leftarrow \mathbf{x}(S', A')$
    $Q \leftarrow \mathbf{w}^\top \mathbf{x}$
    $Q' \leftarrow \mathbf{w}^\top \mathbf{x}'$
    $\delta \leftarrow R + \gamma Q' - Q$
    $\mathbf{z} \leftarrow \gamma\lambda \mathbf{z} + \big(1 - \alpha\gamma\lambda \mathbf{z}^\top \mathbf{x}\big)\, \mathbf{x}$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + Q - Q_{old})\mathbf{z} - \alpha(Q - Q_{old})\mathbf{x}$
    $Q_{old} \leftarrow Q'$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    $A \leftarrow A'$
  until S' is terminal

12.7 Sarsa(λ)

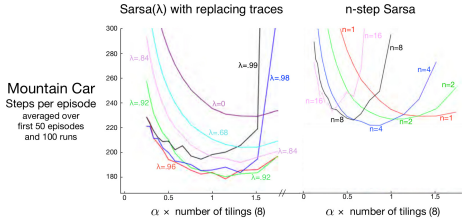


Figure 12.10: Early performance on the Mountain Car task of Sarsa(λ) with replacing traces and n-step Sarsa (copied from Figure 10.4) as a function of the step size, α.

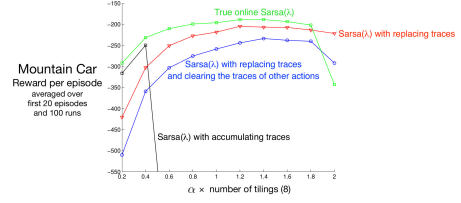


Figure 12.11: Summary comparison of Sarsa(λ) algorithms on the Mountain Car task. True online Sarsa(λ) performed better than regular Sarsa(λ) with both accumulating and replacing traces. Also included is a version of Sarsa(λ) with replacing traces in which, on each time step, the traces for the state and the actions not selected were set to zero.

12.8 Variable λ and γ

- Generalization to non constant discount and non constant λ.
- γ termination function:

$$\begin{aligned} G_t &= R_{t+1} + \gamma_{t+1} G_{t+1} \\ &= \sum_{k=t}^{\infty} \left(\prod_{i=t+1}^{k} \gamma_i \right) R_{k+1} \end{aligned}$$

- Using $\gamma_t = 0$ at the transition allows to see a single stream in episodic setting.
- Recursive definition of the λ-return for time dependent λ_t:

$$G_t^{\lambda s} = R_{t+1} + \gamma_{t+1} \left((1 - \lambda_{t+1}) \hat{v}(S_{t+1}, \boldsymbol{w}_t) + \lambda_{t+1} G_{t+1}^{\lambda s} \right)$$

$$G_t^{\lambda a} = R_{t+1} + \gamma_{t+1} \left((1 - \lambda_{t+1}) \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}_t) + \lambda_{t+1} G_{t+1}^{\lambda a} \right)$$

$$G_t^{\lambda a} = R_{t+1} + \gamma_{t+1} \left((1 - \lambda_{t+1}) \sum_a \pi(a|S_{t+1}) \hat{q}(S_{t+1}, a, \boldsymbol{w}_t) + \lambda_{t+1} G_{t+1}^{\lambda a} \right)$$

12.9 Off-policy Traces with Control Variates

- Off-policy λ-return:

$$G_t^{\lambda s} = \rho_t \left(R_{t+1} + \gamma_{t+1} \left((1 - \lambda_{t+1}) \hat{v}(S_{t+1}, \boldsymbol{w}_t) + \lambda_{t+1} G_{t+1}^{\lambda s} \right) \right) + \underbrace{(1 - \rho_t) \hat{v}(S_t, \boldsymbol{w}_t)}_{\text{Control variate}}$$

- TD error:

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

- Approximation of off-policy λ-return:

$$G_t^{\lambda s} \simeq \hat{v}(S_t, \boldsymbol{w}_t) + \rho_t \sum_{k=t}^{\infty} \delta_k \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i$$

- Not an approximation if $\boldsymbol{w}$ is not updated!
- Forward-view update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left(G_t^{\lambda s} - \hat{v}(S_t, \boldsymbol{w}_t) \right) \nabla \hat{v}(S_t, \boldsymbol{w}_t)$$

$$\simeq \boldsymbol{w}_t + \alpha \rho_t \left(\sum_{k=t}^{\infty} \delta_k \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i \right) \nabla \hat{v}(S_t, \boldsymbol{w}_t)$$

12.9 Off-policy Traces with Control Variates

- Link between forward and backward views:

$$\boldsymbol{w}_{t+1} - \boldsymbol{w}_t \simeq \left(\sum_{k=t}^{\infty} \alpha \rho_t \delta_k \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i \right) \nabla \hat{v}(S_t, \boldsymbol{w}_t)$$

- Sum over time:

$$\begin{aligned}
\sum_{t=1}^{\infty} (\boldsymbol{w}_{t+1} - \boldsymbol{w}_t) &\simeq \sum_{t=1}^{\infty} \sum_{k=t}^{\infty} \alpha \rho_t \delta_k \left(\prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i \right) \nabla \hat{v}(S_t, \boldsymbol{w}_t) \\
&= \sum_{k=1}^{\infty} \sum_{t=1}^{k} \alpha \rho_t \delta_k \left(\prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i \right) \nabla \hat{v}(S_t, \boldsymbol{w}_t) \\
&= \sum_{k=1}^{\infty} \alpha \delta_k \sum_{t=1}^{k} \rho_t \left(\prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i \right) \nabla \hat{v}(S_t, \boldsymbol{w}_t)
\end{aligned}$$

- Eligibility trace (accumulating):

$$\begin{aligned}
\boldsymbol{z}_t &= \sum_{t=1}^{k} \rho_t \left(\prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i \right) \nabla \hat{v}(S_t, \boldsymbol{w}_t) \\
&= \rho_k (\gamma_k \lambda_k \boldsymbol{z}_{k-1} + \nabla \hat{v}(S_k, \boldsymbol{w}_k))
\end{aligned}$$

12.9 Off-policy Traces with Control Variates

- Similar results for action-value methods.
- Off-policy λ return:

$$G_t^{\lambda s} = R_{t+1} + \gamma_{t+1}\big((1-\lambda_{t+1})\bar{V}_t(S_{t+1}, \boldsymbol{w}_t) + \lambda_{t+1}\left(\rho_{t+1} G_{t+1}^{\lambda s} + \bar{V}_t(S_{t+1}, \boldsymbol{w}_t) - \rho_{t+1}\widehat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}_t)\right)\big)$$

$$\simeq \widehat{q}(S_t, A_t, \boldsymbol{w}_t) + \sum_{k=t}^{\infty} \delta_k \prod_{i=t+1}^{k} \gamma_i \lambda_i \rho_i$$

with

$$\delta_t = R_{t+1} + \gamma_{t+1}\bar{V}(S_{t+1}) - \hat{q}(S_t, A_t, \boldsymbol{w}_t)$$

- Eligibility trace:

$$\boldsymbol{z}_t = \gamma_t \lambda_t \rho_t z_{t-1} + \nabla \hat{q}(S_t, A_t, \boldsymbol{w}_t)$$

- $\lambda = 1$ close from MC but subtly distinct...
- Theoretical guarantees still investigated.

12.10 Watkins's Q(λ) to Tree-Backup(λ)

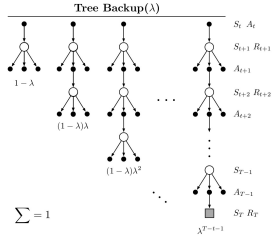


Figure 12.13: The backup diagram for the λ version of the Tree Backup algorithm.

- Tree backup return:

$$G_t^{\lambda a} = R_{t+1} + \gamma_{t+1}\big((1-\lambda_{t+1})\bar{V}S_{t+1} + \lambda_{t+1}\Big(\bar{V}(S_{t+1}) - \pi(A_{t+1}|S_{t+1})(\widehat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}_t) - G_{t+1}^{\lambda a})\Big)\big)$$

$$\simeq \widehat{q}(S_t, A_t, \boldsymbol{w}_t) + \sum_{k=t}^{+\infty} \delta_k \prod_{i=t+1}^{k} \gamma_i \lambda_i \pi(A_i|S_i)$$

- Tree Backup eligibility trace:

$$\boldsymbol{z}_t = \gamma_t \lambda_t \pi(A_t|S_t) + \nabla \hat{q}(S_t, A_t, \boldsymbol{w})$$

12.10 Watkins's Q(λ) to Tree-Backup(λ)

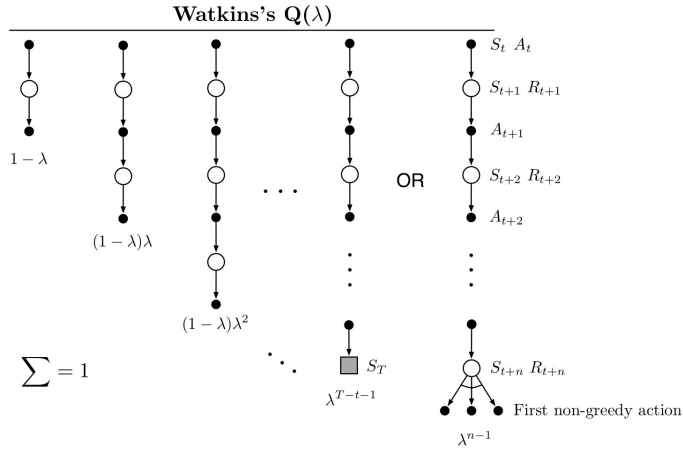


Figure 12.12: The backup diagram for Watkins's Q(λ). The series of component updates ends either with the end of the episode or with the first nongreedy action, whichever comes first.

- Watkins's Q(λ) cuts the trace to zero after the first non greedy update.

12.11 Stable Off-policy Methods with Traces

- Four of the most important examples of stable off-policy using eligibility traces.
- GTD(λ): Gradient-TD (TDC)
- GQ(λ): Gradient-TD for action-value
- HTD(λ): Hybrid between GTD(λ) and TD(λ)
- Emphatic TD(λ): Extension of Emphatic-TD.
- All of them maintain an eligibility trace used in some TD error updates

12.11 Stable Off-policy Methods with Traces

- GTD(λ): Gradient-TD (TDC)

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\delta_t^s \boldsymbol{z}_t - \alpha\gamma_{t+1}(1-\lambda_{t+1})(\boldsymbol{z}_t^\top \boldsymbol{v}_t)\boldsymbol{x}_{t+1}$$

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t + \beta\delta \boldsymbol{z}_t - \beta(\boldsymbol{v}_t^\top \boldsymbol{x}_t)\boldsymbol{x}_t$$

- GQ(λ): Gradient-TD for action-value

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\delta_t^a \boldsymbol{z}_t - \alpha\gamma_{t+1}(1-\lambda_{t+1})(\boldsymbol{z}_t^\top \boldsymbol{v}_t)\overline{x}_{t+1}$$

where

$$\overline{x_t} = \sum_a \pi(a|S_t)\boldsymbol{x}(S_t, a)$$

$$\delta_t^a = R_{t+1} + \gamma_{t+1}\boldsymbol{w}_t^\top \overline{x}_{t+1} - \boldsymbol{w}_t^\top \boldsymbol{x}_t$$

and the rest as in GTD(λ)

12.11 Stable Off-policy Methods with Traces

- HTD(λ): Hybrid between GTD(λ) and TD(λ)

$$\begin{aligned}
\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \alpha\delta_t^s \boldsymbol{z}_t + \alpha((\boldsymbol{z}_t - \boldsymbol{z}_t^b)^\top \boldsymbol{v}_t)(\boldsymbol{x}_t - \gamma_{t+1}\boldsymbol{x}_{t+1}) \\
\boldsymbol{v}_{t+1} &= \boldsymbol{v}_t + \beta\delta_t^s \boldsymbol{z}^\top - \beta((\boldsymbol{z}_t^b)^\top \boldsymbol{v}_t)(\boldsymbol{x}_t - \gamma_{t+1}\boldsymbol{x}_{t+1}) \\
\boldsymbol{z}_t &= \rho_t(\gamma_t\lambda_t \boldsymbol{z}_{t-1} + \boldsymbol{x}_t) \\
\boldsymbol{z}_t^b &= \gamma_t\lambda_t \boldsymbol{z}_{t-1}^b + \boldsymbol{x}_t
\end{aligned}$$

- Emphatic TD(λ): Extension of Emphatic-TD

$$\begin{aligned}
\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \alpha\delta_t \boldsymbol{z}_t \\
\delta_t &= R_{t+1} + \gamma_{t+1}\boldsymbol{w}_t^\top \boldsymbol{x}_{t+1} - \boldsymbol{w}_t^\top \boldsymbol{x}_t \\
\boldsymbol{z}_t &= \rho_t(\gamma_t\lambda_t \boldsymbol{z}_{t-1} + M_t \boldsymbol{x}_t) \\
M_t &= \lambda_t I_t + (1-\lambda_t)F_t \\
F_t &= \rho_{t-1}\gamma_t F_{t-1} + I_t.
\end{aligned}$$

12.12 Implementation Issues

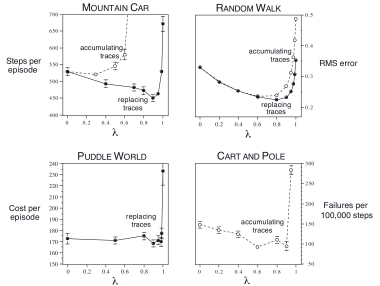


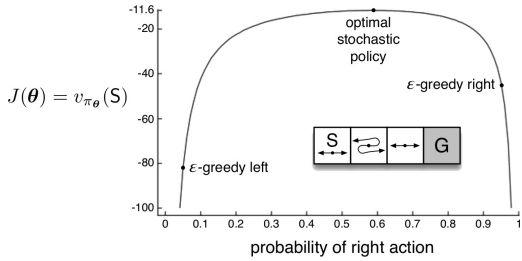
Figure 12.14: The effect of λ on reinforcement learning performance in four different test problems. In all cases, performance is generally best (a *lower* number in the graph) at an intermediate value of λ. The two left panels are applications to simple continuous-state control tasks using the Sarsa(λ) algorithm and tile coding, with either replacing or accumulating traces (Sutton, 1996). The upper-right panel is for policy evaluation on a random walk task using TD(λ) (Singh and Sutton, 1996). The lower right panel is unpublished data for the pole-balancing task (Example 3.4) from an earlier study (Sutton, 1984).

- Trace is of the dimension of $\boldsymbol{w}$.
- Eligibility trace seems unusable for tabular methods.
- Trick: list the small numbers of non-zero coordinates in the trace, update them and kill them if they are too small.

Outline

13 Policy Gradient Methods

13.1 Policy Approximation and its Advantages



- Parametric policy $\pi(a|s, \boldsymbol{\theta})$ differentiable with respect to θ.
- Soft-max in action preferences:
  - numerical preferences $h(a, s, \boldsymbol{\theta})$.
  - probability:

$$\pi(a|s, \boldsymbol{\theta}) = \frac{e^{h(a,s,\boldsymbol{\theta})}}{\sum_b e^{h(b,s,\boldsymbol{\theta})}}$$

- Lots of freedom in the parameterization of h: from linear to deep network...
- Can approach a deterministic policy or stochastic ones...
- May be simpler than value-based method.

13.2 The Policy Gradient Theorem

- Performance measure $J(\boldsymbol{\theta})$ depends smoothly on $\boldsymbol{\theta}$.
- Easier to obtain convergence guarantees.
- Episodic case: performance
$$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(s_0)$$
where s_0 is the initial state.
- Policy gradient theorem:
$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta})$$

13.2 The Policy Gradient Theorem

- Gradient of $v_\pi(s)$:

$$\begin{aligned}\nabla v_\pi(s) &= \nabla \left(\sum_a \pi(a|s) q_\pi(s,a) \right) \\ &= \sum_a \left(\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla q_\pi(s,a) \right) \\ &= \sum_a \left(\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) \nabla v_\pi(s') \right) \\ &= \sum_x \sum_{k=0}^{\infty} \mathbb{P}\left(s \to x, k, \pi\right) \sum_a \nabla \pi(a|x) q_\pi(x,a)\end{aligned}$$

- Stationary measure $\mu(s)$:

$$\mu(s) = \frac{\sum_k \mathbb{P}\left(s_0 \to s, k, \pi\right)}{\sum_{s'} \sum_k \mathbb{P}\left(s_0 \to s, k, \pi\right)}$$

- Gradient of $J(\boldsymbol{\theta})$:

$$\nabla J(\boldsymbol{\theta}) = \left(\sum_{s'} \sum_k \mathbb{P}\left(s_0 \to s, k, \pi\right) \right) \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a)$$

13.3 REINFORCE: Monte Carlo Policy Gradient

- Gradient of $J(\boldsymbol{\theta})$:

$$\begin{aligned}\nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a) \\ &= \mathbb{E}_\pi \left[\sum_a \nabla \pi(a|S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \right]\end{aligned}$$

- All-actions SGD:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1} + \alpha \sum_a \nabla \pi(a|S_t, \boldsymbol{\theta}) \hat{q}(S_t, a, \boldsymbol{w})$$

- REINFORCE use a single action:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1} + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t)}$$

where G_t is the usual MC return.

13.3 REINFORCE: Monte Carlo Policy Gradient

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

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\quad (G_t)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

- REINFORCE derivation:

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &\propto \mathbb{E}_\pi \left[\sum_a \nabla \pi(a|S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \right] \\
&= \mathbb{E}_\pi \left[\sum_a \pi(a|S_t, \boldsymbol{\theta}) \frac{\nabla \pi(a|S_t, \boldsymbol{\theta})}{\pi(a|S_t, \boldsymbol{\theta})} q_\pi(S_t, a) \right] \\
&= \mathbb{E}_\pi \left[\frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_|S_t, \boldsymbol{\theta})} q_\pi(S_t, A_t) \right] \\
&= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right] = \mathbb{E}_\pi \left[G_t \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) \right]
\end{aligned}$$

13.3 REINFORCE: Monte Carlo Policy Gradient

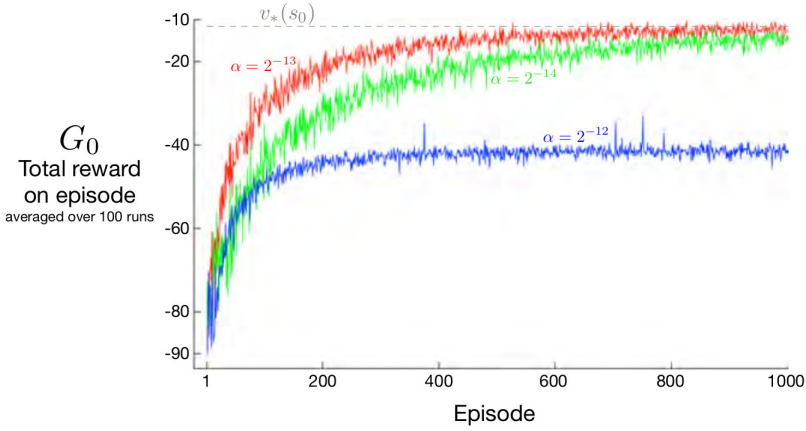


Figure 13.1: REINFORCE on the short-corridor gridworld (Example 13.1). With a good step size, the total reward per episode approaches the optimal value of the start state.

- SGD algorithm hence good theoretical convergence properties.
- Variance may be high.

13.4 REINFORCE with Baseline

- Any baseline $b(s)$ satisfies
$$\sum_a b(s)\nabla\pi(a|s,\boldsymbol{\theta}) = 0$$
- Generalized policy gradient theorem:
$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a (q_\pi(s,a) - b(s))\nabla\pi(a|s,\boldsymbol{\theta})$$
- REINFORCE algorithm with baseline:
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1} + \alpha(G_t - b(S_t))\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t)}$$
- Allows variance reduction.
- Natural choice for $b(S_t)$: $\hat{v}(S_t, \boldsymbol{w})$.

13.4 REINFORCE with Baseline

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

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\qquad (G_t)$
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

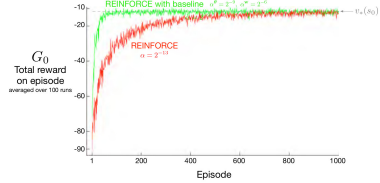


Figure 13.2: Adding a baseline to REINFORCE can make it learn much faster, as illustrated here on the short-corridor gridworld (Example 13.1). The step size used here for plain REINFORCE is that at which it performs best (to the nearest power of two; see Figure 13.1).

13.5 Actor-Critic Methods

- REINFORCE learns a policy and a state value function, but does not use it to bootstrap.
- Actor-Critic methods: replace G_t by a bootstrap estimate:

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

- Same tradeoff with respect to MC...

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

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (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):
    $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

13.5 Actor-Critic Methods

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

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: trace-decay rates $\lambda^{\boldsymbol{\theta}} \in [0,1]$, $\lambda^{\mathbf{w}} \in [0,1]$; step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ (d'-component eligibility trace vector)
  $\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$ (d-component eligibility trace vector)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma\, \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{z}^{\mathbf{w}} \leftarrow \gamma \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + \nabla \hat{v}(S,\mathbf{w})$
    $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \gamma \lambda^{\boldsymbol{\theta}} \mathbf{z}^{\boldsymbol{\theta}} + I \nabla \ln \pi(A|S, \boldsymbol{\theta})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\, \delta\, \mathbf{z}^{\mathbf{w}}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\, \delta\, \mathbf{z}^{\boldsymbol{\theta}}$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

13.6 Policy Gradient for Continuing Tasks

- Most natural performance measure:

$$J(\boldsymbol{\theta}) = r(\pi) = \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}\left[R_t | S_0, A_{0:t-1} \sim \pi\right]$$
$$= \sum_s \mu(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) r$$

- μ is such that

$$\sum_s \mu(s) \sum_a \pi(a|s) p(s'|s, a) = \mu(s')$$

- Gradient of $J(\boldsymbol{\theta})$:

$$\nabla J(\boldsymbol{\theta}) = \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s, a)$$

13.6 Policy Gradient for Continuing Tasks

Actor–Critic with Eligibility Traces (continuing), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: $\lambda^{\mathbf{w}} \in [0,1]$, $\lambda^{\boldsymbol{\theta}} \in [0,1]$, $\alpha^{\mathbf{w}} > 0$, $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\bar{R}} > 0$
Initialize $\bar{R} \in \mathbb{R}$ (e.g., to 0)
Initialize state-value weights $\mathbf{w} \in \mathbb{R}^d$ and policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)
Initialize $S \in \mathcal{S}$ (e.g., to s_0)

$\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$ (d-component eligibility trace vector)
$\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ (d'-component eligibility trace vector)
Loop forever (for each time step):
  $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
  Take action A, observe S', R
  $\delta \leftarrow R - \bar{R} + \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$
  $\bar{R} \leftarrow \bar{R} + \alpha^{\bar{R}}\delta$
  $\mathbf{z}^{\mathbf{w}} \leftarrow \lambda^{\mathbf{w}}\mathbf{z}^{\mathbf{w}} + \nabla\hat{v}(S,\mathbf{w})$
  $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \lambda^{\boldsymbol{\theta}}\mathbf{z}^{\boldsymbol{\theta}} + \nabla \ln \pi(A|S,\boldsymbol{\theta})$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\,\delta\,\mathbf{z}^{\mathbf{w}}$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\,\delta\,\mathbf{z}^{\boldsymbol{\theta}}$
  $S \leftarrow S'$

13.6 Policy Gradient for Continuing Tasks

- Gradient of the state-value function:

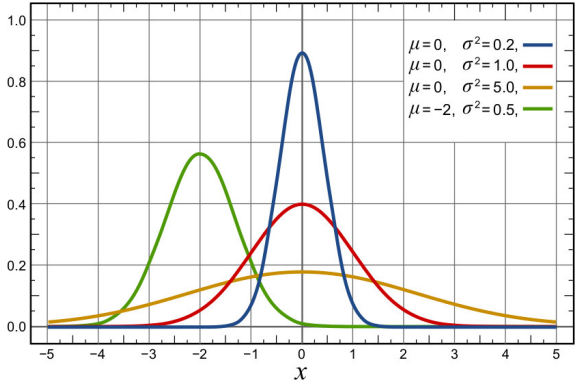
$$
\begin{aligned}
\nabla v_\pi(s) &= \nabla \left(\sum_a \pi(a|s) q_\pi(s,a) \right) \\
&= \sum_a \left(\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla q_\pi(s,a) \right) \\
&= \sum_a \left(\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) \nabla (-r(\boldsymbol{\theta}) + v_\pi(s')) \right) \\
&= \sum_a \left(\nabla \pi(a|s) q_\pi(s,a) \right. \\
&\qquad \left. + \pi(a|s) \left(-\nabla r(\boldsymbol{\theta}) + \sum_{s'} p(s'|s,a) \nabla v_\pi(s') \right) \right)
\end{aligned}
$$

13.6 Policy Gradient for Continuing Tasks

- Gradient of $J(\boldsymbol{\theta}) = r(\boldsymbol{\theta})$:

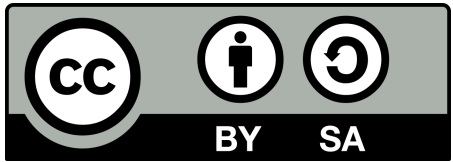
$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \sum_a \left(\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) \nabla v_\pi(s') \right) - \nabla v_\pi(s) \\
&= \sum_s \mu(s) \left(\sum_a \left(\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) \nabla v_\pi(s') \right) - \nabla v_\pi(s) \right) \\
&= \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&\quad + \sum_{s'} \underbrace{\sum_s \mu(s) \pi(a|s) p(s'|s,a)}_{\mu(s')} \nabla v_\pi(s') - \sum_s \mu(s) \nabla v_\pi(s) \\
&= \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a)
\end{aligned}
$$

13.7 Policy Parameterization for Continuous Actions



- Policy-based methods offer a way to deal with continuous action space.
- Only requirement is a parametric policy that can be sampled.
- For instance, a Gaussian parameterization with linear mean and linear log variance.

Licence and Contributors



Contributors

- Main source: R. Sutton and A. Barto
- Main contributor: E. Le Pennec
- Contributors: S. Boucheron, A. Dieuleveut, A.K. Fermin, S. Gadat, S. Gaiffas, A. Guilloux, Ch. Keribin, E. Matzner, M. Sangnier, E. Scornet.