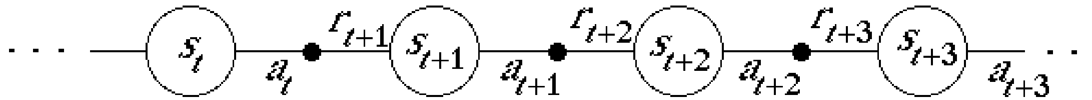
COMP 652: Machine Learning

Lecture 19

Today

- Estimating value functions for large state spaces
  (Via Monte Carlo and supervised learning)
- Approximate policy iteration
- Relating state values – Bellman equations, Bellman optimality equations
- Model-based reinforcement learning
- Model-free reinforcement learning

Recall: Markov Decision Process (MDP)



An MDP is defined by:

- Set of *states* S
- Set of *actions* $A(s)$ available in each state s
- *Rewards*:

$$r_{ss'}^{a} = E\left\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\right\}$$

- *Transition probabilities*

$$p_{ss'}^{a} = P\left(s_{t+1} = s' | s_t = s, a_t = a\right)$$

Recall: Policy and action-value functions

- A deterministic policy maps every state to an action: $\pi : S \mapsto A$.

- The value function of a policy is the expected return the agent receives if following π, conditioned on the state in which the environment starts:

$$
\begin{aligned}
V^\pi(s) &= E\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s, a_{t'} = \pi(s_{t'}) \text{ for all } t' \geq t\} \\
&= E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s\}
\end{aligned}
$$

- The action-value function of a policy the expected return of choosing an action and following the policy afterwards, conditioned on the state in which the environment starts:

$$Q^\pi(s,a) = E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s, a_t = a\}$$

- (These are for $\gamma \in (0,1]$, though finiteness for $\gamma = 1$ requires extra conditions.)

Policy iteration

- Last time, we describe Monte Carlo methods for estimating V^π and Q^π
- We showed the policy iteration algorithm, which alternates:
  - Estimating Q^π
  - Updating π to be "greedy" with respect to Q^π
- Policy iteration terminates at a globally optimal policy, π^*
  - For all s and all π', $V^{\pi^*}(s) \geq V^{\pi'}(s)$
  - For all s, all a, and all π', $Q^{\pi^*}(s,a) \geq Q^{\pi'}(s,a)$
- The Monte Carlo approach uses data quite inefficiently. . . a problem to which we return shortly.
- What if S is too large, even infinite, so that we cannot represent V^π explicitly?

Dealing with large/continuous state spaces

- Consider the problem of V^π estimation when $|S|$ is too large / continuous, so that explicit, tabular representation is not possible.
- What can we do?

Estimating V^π by supervised learning

- Consider a sample trajectory $s_0, a_0, r_1, s_1, a_1, r_2, \ldots$ taken under policy π
- Suppose we represent states by some feature mapping $\phi(s)$
- We can formulate the supervised learning (regression) problem:

input	output
$\phi(s_0)$	$r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots$
$\phi(s_1)$	$r_2 + \gamma r_3 + \gamma^2 r_4 + \ldots$
$\phi(s_2)$	$r_3 + \gamma r_4 + \gamma^2 r_5 + \ldots$
$\phi(s_3)$	$r_4 + \gamma r_5 + \gamma^2 r_6 + \ldots$
$\vdots$	$\vdots$

- We can fit this data with any regressor: linear/polynomial regression, neural network, nearest-neighbor, regression tree, ...
- If we have multiple trajectories, each one contributes data similarly.

Estimating Q^π by supervised learning

- Estimating Q^π is slightly trickier. How can we do it?

Estimating Q^π by supervised learning

- Can define a feature mapping $\phi(s, a)$
- One way (Monte Carlo exploring starts):
  - Generate multiple trajectories, τ_1, τ_2, ..., τ_m
  - Trajectory $\tau_i =< s_0^i, a_0^i, r_1^i, s_1^i, a_1^i, r_2^i, \ldots >$ generated by choosing random initial state s_0^i, random initial action a_0^i, and following π afterwards
  - Supervised learning data is:

input	output
$\phi(s_0^1, a_0^1)$	$r_1^1 + \gamma r_2^1 + \gamma^2 r_3^1 + \ldots$
$\phi(s_0^2, a_0^2)$	$r_1^2 + \gamma r_2^2 + \gamma^2 r_3^2 + \ldots$
$\phi(s_0^3, a_0^3)$	$r_1^3 + \gamma r_2^3 + \gamma^2 r_3^3 + \ldots$
$\vdots$	$\vdots$

- Alternatively, one can define a state-feature mapping, $\phi(s)$, and learn a separate approximator for each action's value (as a function of state).

Approximate policy iteration

- This enables approximate policy iteration, one version of which is:
  - Start with an arbitrary initial policy π_0
  - Repeat the following steps:
    1. Approximate Q^{π_i} for the current policy, π_i, by generated Monte Carlo Exploring Starts (MCES) data, and applying some supervised learning method.
    2. Define $\pi_{i+1}(s)$ implicitly as $\arg\max_a Q^{\pi_i}(s,a)$
- Does it converge?
- Does it converge to an optimal policy?

Approximate policy iteration

- This enables approximate policy iteration, one version of which is:
  - Start with an arbitrary initial policy π_0
  - Repeat the following steps:
    1. Approximate Q^{π_i} for the current policy, π_i, by generated Monte Carlo Exploring Starts (MCES) data, and applying some supervised learning method.
    2. Define $\pi_{i+1}(s)$ implicitly as $\arg\max_a Q^{\pi_i}(s, a)$

- Does it converge?

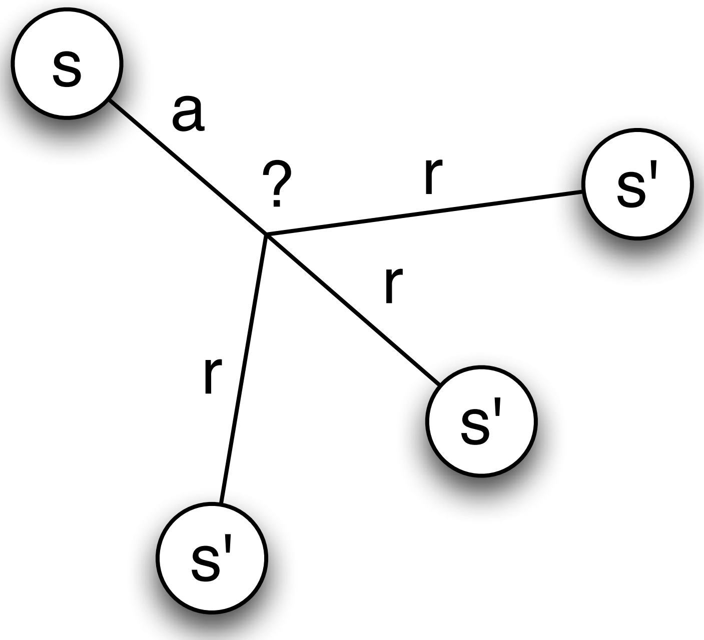
$\Rightarrow$ In general, no. For certain kinds of MDPs and/or function approximators, though, it does.

- Does it converge to an optimal policy?

$\Rightarrow$ In general, no. Though one can bound degree of suboptimality in terms of error in Q^{π} approximation.

Relationships between state values

- $V^{\pi}(s)$ an expected sum of future rewards
- But intuitively, a state's value must by related to the states to which it leads



Relationships between state values (II)

$$\begin{aligned}
V^\pi(s) &= E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s\} \\
&= \sum_{s'} P(s_{t+1} = s' | s_t = s) E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s, s_{t+1} = \\
&= \sum_{s'} p_{ss'}^{\pi(s)} [r_{ss'}^{\pi(s)} + \gamma E_\pi\{r_{t+2} + \gamma r_{t+3} + \ldots | s_t = s, s_{t+1} = s'\}] \\
&= \sum_{s'} p_{ss'}^{\pi(s)} [r_{ss'}^{\pi(s)} + \gamma V^\pi(s')]
\end{aligned}$$

- □ These are called the *Bellman equations* for policy value
- □ Treating the $V^\pi(s)$ as a set of $|S|$ variables, the above gives a *linear* system of $|S|$ equations with $|S|$ unknowns.
- ⇒ We can use linear system solvers to find $V^\pi(s)$, if we know $p_{ss'}^a$ and $r_{ss'}^a$. (There's other ways to do it too, though.)
- □ Are solutions unique?

Uniqueness of solutions for V^π

- Let r^π be a the expected immediate reward upon following policy π in state s:

$$r^\pi(s) = \sum_{s'} p_{ss'}^{\pi(s)} r_{ss'}^{\pi(s)}$$

  We will treat r^π as a length $|S|$ column vector

- Let $P^\pi(s, s') = p_{ss'}^{\pi(s)}$ be the probability that state s' follows state s, when agent acts according to π.

  We will treat this as an $|S| \times |S|$ matrix.

- Then considering V^π as a length $|S|$ column vector, the equation of the previous slide says that:

$$V^\pi = r^\pi + \gamma P^\pi V^\pi$$

Uniqueness of solutions for V^π (I)

- Rearranging, we find that:

$$(I - \gamma P^\pi)V^\pi = r^\pi$$

- P^π is a Markov matrix. If it describes a non-terminating chain, its largest eigenvalue is 1. If it describe a terminating chain, its largest eigenvalue is < 1.

- If P^π is non-terminating and $\gamma < 1$, or if P^π is terminating, then the largest eigenvalue of $(I - \gamma P^\pi)$ is < 0.

- Thus, it is invertible, and V^π is uniquely determined as:

$$V^\pi = (I - \gamma P^\pi)^{-1}V^\pi$$

An alternative approach to computing V^π

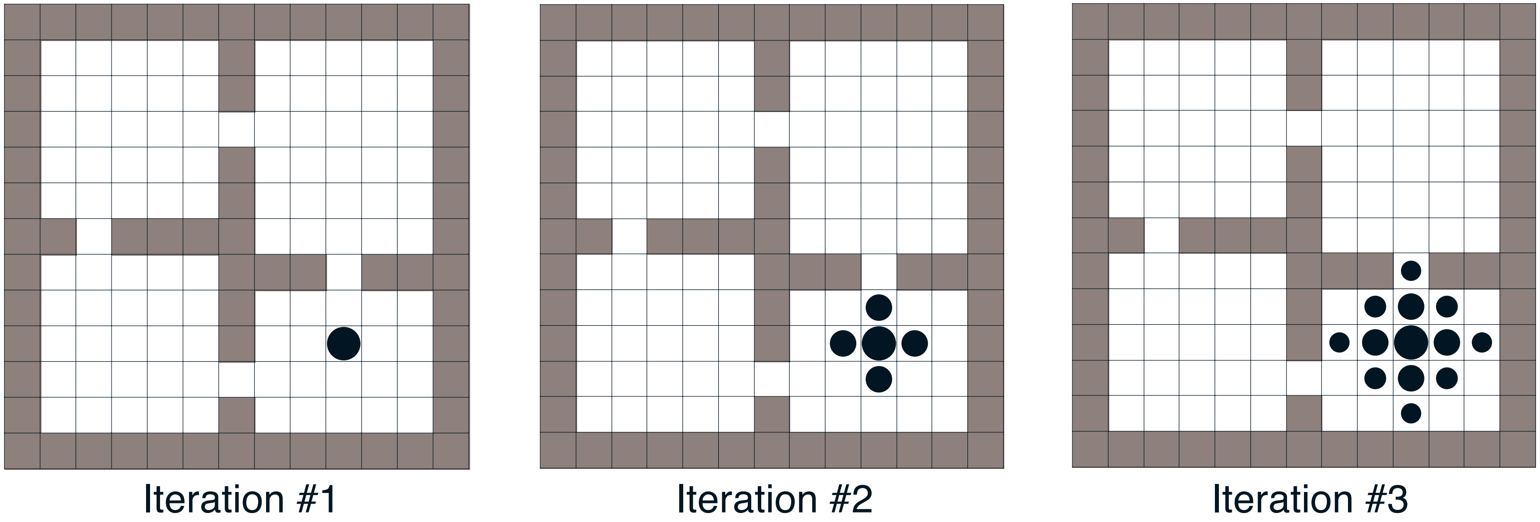
- Matrix inversion is computationally expensive
- Iterative approaches are more commonly used for computing value functions, e.g.:
  - Initialize $V^\pi(s)$ arbitrarily
  - Repeat, until changes are sufficiently small:

$$V^\pi(s) \leftarrow \sum_{s'} p_{ss'}^{\pi(s)}(r_{ss'}^{\pi(s)} + \gamma V^\pi(s')) \text{ for all } s$$

- This can be shown to converge exponentially quickly to the correct values. (Aside: What does "exponentially quickly" mean?)

Example

- Imagine the simple deterministic “maze” below
- Goal is to get into the dotted square in the lower right room
- Reward of $+1$ upon arrival to goal
- Discount factor $\gamma = 0.9$
- π is correct, optimal policy
- Initialize $V^{\pi}(s) = 0$ for all s, except goal, which is set to $+1$
- Figure shows values by dots of radius proportional to value



Action-value functions

- Similar reasoning holds for action-value functions.
- First, note that $V^\pi(s) = Q^\pi(s, \pi(s))$. Then:

$$
\begin{aligned}
Q^\pi(s,a) &= E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s, a_t = a\} \\
&= \sum_{s'} p^a_{ss'}[r^a_{ss'} + \gamma E_\pi\{r_{t+2} + \gamma r_{t+3} + \ldots | s_t = s, a_t = a, s_{t+1} = s' \\
&= \sum_{s'} p^a_{ss'}[r^a_{ss'} + \gamma V^\pi(s')] \\
&= \sum_{s'} p^a_{ss'}[r^a_{ss'} + \gamma Q^\pi(s', \pi(s'))]
\end{aligned}
$$

- We have $\sum_s |A(s)|$ linear equations in the same number of unknowns.
- A similar matrix argument shows the solution, Q^π, is unique.
- An iterative-style algorithm can be used to compute it:

$$Q^\pi(s,a) \leftarrow \sum_{s'} p^a_{ss'}(r^a_{ss'} + \gamma Q^\pi(s', \pi(s')))$$

Finding optimal policies

- The previous techniques can be used to implement exact policy iteration
- However, there is another approach based on *Bellman optimality equations* (see below)
- Recall that there is at least one optimal policy π^*.
  (It satisfies $V^{\pi^*}(s) \geq V^{\pi}(s)$ and $Q^{\pi^*}(s,a) \geq Q^{\pi}(s,a)$ for all π, s, a.)
- For an optimal policy π^*, we must have:

$$
\begin{aligned}
V^{\pi^*}(s) &= \max_a Q^{\pi^*}(s,a) \\
&= \max_a \left[\sum_{s'} p^a_{ss'} (r^a_{ss'} + \gamma V^{\pi^*}(s')) \right] \\
Q^{\pi^*}(s,a) &= \sum_{s'} p^a_{ss'} (r^a_{ss'} + \gamma V^{\pi^*}(s')) \\
&= \sum_{s'} p^a_{ss'} (r^a_{ss'} + \gamma \max_{a'} Q^{\pi^*}(s',a')
\end{aligned}
$$

- These are <u>*nonlinear*</u> systems of equations

Solving the Bellman optimality equations

- Solutions can be found by linear programming
- More common, however, is *value iteration*:

$$V(s) \leftarrow \max_a \left[\sum_{s'} p_{ss'}^a (r_{ss'}^a + \gamma V^{\pi^*}(s')) \right]$$

- Or action-value iteration:

$$Q(s,a) \leftarrow \sum_{s'} p_{ss'}^a (r_{ss'}^a + \gamma \max_{a'} Q^{\pi^*}(s', a')$$

- These approaches converge exponentially quickly to the optimal value function V^* or action-value function Q^* (under the same conditions needed for well-definedness)

Back to reinforcement learning

- The previous discussion assumes that we know $p_{ss'}^a$ and $r_{ss'}^a$
- What if we don't?

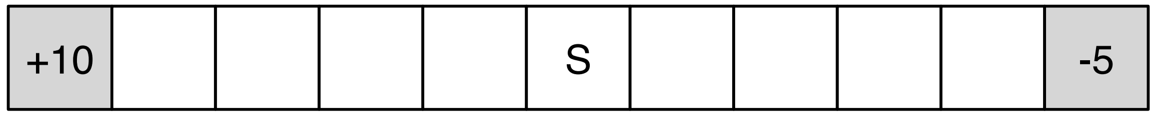
Back to reinforcement learning

- The previous discussion assumes that we know $p^a_{ss'}$ and $r^a_{ss'}$
- What if we don't?
  - Model-based value function learning
  - Model-free value function learning
  - Value function-free learning

Model-based reinforcement learning

- Model-based learning algorithms use experience from the environment to build an <u>*approximate model*</u> $\hat{r}^a_{ss'}$, $\hat{p}^a_{ss'}$
- Then we pretend the approximate model is correct and use it to compute the optimal value function/policy as above
- How do we estimate the rewards and transition probabilities?
  - If $|S|$ is small enough for tabular representation, $\hat{r}^a_{ss'}$ can be estimated as the empirical mean reward from every $s - a - r - s'$ quadruple in the data. $\hat{p}^a_{ss'}$ can be taken to be the empirical probability that s' follows $s - a$ in the data.
  - Otherwise, solutions are ad hoc. For low-dimensional continuous state spaces, state aggregation or interpolators are often used.

Example



- Followed Rand policy (equal chance of left or right action)
- 10 trajectories
- Estimated rewards and transition probabilities

Results

- Reward function exactly correct
- Transition probabilities somewhat correct:

state s	2	3	4	5	6	7	8	9	10
$p_{s,s-1}^{left}$	0.6	0.85714	0.64706	0.95238	0.72	0.78571	0.875	0.8	
$p_{s,s+1}^{left}$	0.4	0.14286	0.35294	0.047619	0.28	0.21429	0.125	0.2	
$p_{s,s-1}^{right}$	0.25	0.2	0.27273	0.26087	0.23529	0.21429	0.083333	0.13333	
$p_{s,s+1}^{right}$	0.75	0.8	0.72727	0.73913	0.76471	0.78571	0.91667	0.86667	

- Q^* estimate a bit off, but π^* estimate correct:

state s	2	3	4	5	6	7	8	9	10
$Q^*(s, left)$	10	9.9999	9.9998	9.9994	9.9926	9.975	9.9104	9.4586	
$Q^*(s, right)$	10	9.9998	9.9996	9.9944	9.9807	9.928	9.5016	7.9525	
$\pi^*(s)$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	

Pros and cons of model-based RL

- \+ Tend to be very data-efficient
- \+ The models may be of independent interest
- \+ If reward function changes, or dynamics of only small part of the environment changes, lots of information can be reused
- − If state set S or action set A are very large or infinite, it will be very hard to estimate the model from data—especially for the transition probabilities; this can lead to poor performance.
- − Even if model is accurate, solving for π^* can be nontrivial.
(It's polynomial in $|S|$ and $|A|$, but if large, can be problematic.)

Model-based RL with state-features

- Suppose that we represented every state s with a *feature vector* ϕ_s, of size $k \leq |S|$
- We can represent all the feature vectors, for all the state, in a *feature matrix* $\mathbf{\Phi}$, of size $k \times |S|$, where the sth column is ϕ_s
- Important special case: if each column has exactly one element equal to 1 and all the others are 0, the matrix represents a *state partition*, where the state space has been partitioned in k disjoint subsets.
- In general, the features (also called *basis functions*) can be anything (Gaussian, sine-cosine, etc)

Bellman equations with features

- Consider the Bellman equations:

$$\mathbf{V}^\pi = \mathbf{r}^\pi + \gamma \mathbf{P}^\pi \mathbf{V}^\pi$$

where $\mathbf{V}^\pi$ is a column vector repesenting $V^\pi(s)$, $\mathbf{r}^\pi$ is the expected rewards following each state, and $\mathbf{P}^\pi$ is the matrix containing $p^\pi_{ss'}(s)$.

- We multiply at the left by $\mathbf{\Phi}$:

$$\mathbf{\Phi}\mathbf{V}^\pi = \mathbf{\Phi}\mathbf{r}^\pi + \gamma \mathbf{\Phi}\mathbf{P}^\pi \mathbf{V}^\pi$$

- We make $\mathbf{\Phi}\mathbf{V}^\pi$ appear on the right hand side as well:

$$\mathbf{P}^\pi\mathbf{V}^\pi = \mathbf{P}^\pi\mathbf{I}\mathbf{V}^\pi = \mathbf{P}^\pi\mathbf{\Phi}^T\mathbf{\Phi}\mathbf{V}^\pi$$

- Now we can re-write the Bellman equations:

$$\mathbf{\Phi}\mathbf{V}^\pi = \mathbf{\Phi}\mathbf{r}^\pi + \gamma \mathbf{\Phi}\mathbf{P}^\pi\mathbf{\Phi}^T\mathbf{\Phi}\mathbf{V}^\pi \Rightarrow (\mathbf{I} - \gamma\mathbf{\Phi}\mathbf{P}^\pi\mathbf{\Phi}^T)\mathbf{\Phi}\mathbf{V}^\pi = \mathbf{\Phi}\mathbf{r}^\pi$$

Approximate models

- We re-write the above equation as:

$$\mathbf{\Phi V}^\pi = (\mathbf{I} - \gamma \mathbf{\Phi P}^\pi \mathbf{\Phi}^T)^{-1} \mathbf{\Phi r}^\pi$$

  (I'm glossing over assumption needed to ensure the inverse exists.)

- Let $\hat{\mathbf{r}}^\pi = \mathbf{\Phi r}^\pi$; this is a vector of size k, representing the reward for every feature
- E.g., in the special case of state partitioning, the reward associated with a partition will be the sum of the rewards for the states in that partition (why?)
- Let $\hat{\mathbf{P}}^\pi = \mathbf{\Phi P}^\pi \mathbf{\Phi}^T$; this is a $k \times k$ matrix showing transitions between features
- Since this is typically much smaller than the original matrix, it can be estimated more accurately with less data

Approximate value function

$$\mathbf{\Phi V}^\pi = (\mathbf{I} - \gamma \mathbf{\Phi P}^\pi \mathbf{\Phi}^T)^{-1} \mathbf{\Phi r}^\pi$$

- Let $\hat{\mathbf{V}}^\pi = \mathbf{\Phi V}^\pi$; this is the approximation of the value function using the features
- E.g., in the case of a state partition, each partition will have a value associated with it, and all states in the partition share the same value
- Obviously, not all value functions can be represented correctly anymore.
- The Bellman equations for approximate values become:

$$\hat{\mathbf{V}}^\pi = (\mathbf{I} - \gamma \hat{\mathbf{P}}^\pi) \hat{\mathbf{r}}^\pi$$

Trade-off

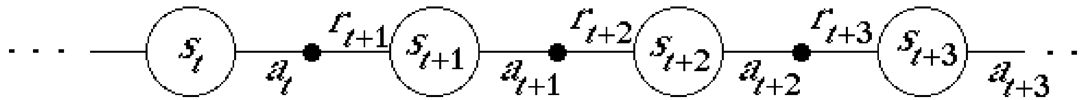
- The above systems has k equations with k unknowns
- Model-based approximate methods will estimate $\hat{\mathbf{r}}^\pi$ and $\hat{\mathbf{P}}^\pi$ from data
- The smaller k is, the less data we need to do this estimation, and the easier it is to solve the system
- But the smaller k is, the less accurate will the value function be
- Instead of estimating V^π, Q^π can be estimated, leading to an alternative approximate policy iteration algorithm

Modeling value, not dynamics

- In model-free value function-based RL, we directly estimate value function, but not $r^a_{ss'}$ or $p^a_{ss'}$
- The Monte Carlo methods describe last lecture and at the start of this lecture are one way to do so
- However, the iterative (dynamic programming) approaches to value function computation provide inspiration for another class of approaches

Modeling value, not dynamics (II)

- Consider a trajectory, with actions selected according to policy π:



- The Bellman equation is:

$$V^\pi(s_t) = E_\pi\left[r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t\right]$$

which suggested the dynamic programming update:

$$V(s_t) \leftarrow E_\pi\left[r_{t+1} + \gamma V(s_{t+1}) | s_t\right]$$

- In general, we do not know this expected value, but we do have an possibly-biased sample estimate of it, $r_{t+1} + \gamma V(s_{t+1})$
- We can make an update ***towards*** the sample value, with step size α:

$$V(s_t) \leftarrow (1 - \alpha)V(s_t) + \alpha\left(r_{t+1} + \gamma V(s_{t+1})\right)$$

Temporal-Difference (TD) Learning

- We can rewrite the previous as:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

- The term after the α is called the "temporal difference" – it is the difference between what our estimate $V(s_t)$ suggested we would see, and $r_{t+1} + \gamma V(s_{t+1})$
- Does it converge to V^π?

Temporal-Difference (TD) Learning

- We can rewrite the previous as:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

- The term after the α is called the "temporal difference" – it is the difference between what our estimate $V(s_t)$ suggested we would see, and $r_{t+1} + \gamma V(s_{t+1})$
- Does it converge to V^π? Yes! If:
  - We have infinitely much data collected under policy π
  - If all states s appear infinitely often in the data
  - If learning rate(s) α decrease towards zero at an appropriate rate (Robbins-Monroe conditions)

Q-Learning

- That's fine for learning V^π. A similar procedure can be designed to learn Q^π.
- What about learning optimal policies?
- Suppose we generate experience $(s_0, a_0, r_1, s_1, a_1, r_2, \ldots)$ from the environment, and update an action-value function as:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'))$$

- This is motivated by the Bellman optimality equation:

$$Q^*(s,a) = \sum_{s'} p^a_{ss'}(r^a_{ss'} + \gamma \max_a{}' Q^*(s', a'))$$

- This is called <u>*Q-Learning*</u>

Q-Learning convergence

- Does Q-Learning converge? Yes!
- The experience can be generated under any policy π at all – or not even according to a policy, strictly speaking
- We need infinitely much data
- Every possible state-action pair must occur infinitely many times
- Learning rates need to be scheduled appropriately