10703 Deep Reinforcement Learning

Solving known MDPs

Tom Mitchell

September 10, 2018

Many slides borrowed from
Katerina Fragkiadaki
Russ Salakhutdinov

Markov Decision Process (MDP)

A **Markov Decision Process** is a tuple $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- T is a state transition probability function

$$T(s'|s,a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$$

- r is a reward function

$$r(s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$$

- γ is a discount factor $\gamma \in [0,1]$

Outline

Previous lecture:

- Policy evaluation

This lecture:

- Policy iteration
- Value iteration
- Asynchronous DP

Policy Evaluation

Policy evaluation: for a given policy π, compute the state value function

$$\mathrm{v}_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s\right]$$

where $\mathrm{v}_\pi(s)$ is implicitly given by the **Bellman equation**

$$\mathrm{v}_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s)\left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a)\mathrm{v}_\pi(s')\right)$$

a system of $|\mathcal{S}|$ simultaneous equations.

Iterative Policy Evaluation

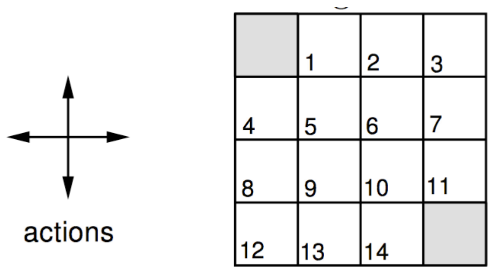
(Synchronous) Iterative Policy Evaluation for given policy π

- Initialize V(s) to anything
- Do until change in $\max_s[V_{[k+1]}(s) - V_k(s)]$ is below desired threshold
  - for every state s, update:

$$\mathrm{v}_{[k+1]}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a) \mathrm{v}_{[k]}(s') \right)$$

Iterative Policy Evaluation

Policy π, choose an equiprobable random action



- An undiscounted episodic task
- Nonterminal states: 1, 2, … , 14
- Terminal states: two, shown in shaded squares
- Actions that would take the agent off the grid leave the state unchanged
- Reward is -1 until the terminal state is reached

$\mathbf{V}[k]$ for the random policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Is Iterative Policy Evaluation Guaranteed to Converge?

Contraction Mapping Theorem

Definition:

An operator F on a normed vector space $\mathcal{X}$ is a γ**-contraction**, for $0 < \gamma < 1$, provided for all $x, y \in \mathcal{X}$

$$||F(x) - F(y)|| \leq \gamma ||x - y||$$

Contraction Mapping Theorem

Definition:

An operator F on a normed vector space $\mathcal{X}$ is a γ**-contraction**, for $0 < \gamma < 1$, provided for all $x, y \in \mathcal{X}$

$$||F(x) - F(y)|| \leq \gamma ||x - y||$$

Theorem (Contraction mapping)
For a γ-contraction F in a complete normed vector space $\mathcal{X}$

- Iterative application of F converges to a unique fixed point in $\mathcal{X}$ independent of the starting point
- at a linear convergence rate determined by γ

Value Function Sapce

- Consider the vector space V over value functions
- There are $|\mathcal{S}|$ dimensions
- Each point in this space fully specifies a value function $\mathbf{v}(s)$
- Bellman backup is a contraction operator that brings value functions closer in this space (we will prove this)
- And therefore the backup must converge to a unique solution

Value Function ∞-Norm

- We will measure distance between state-value functions $\mathbf{u}$ and $\mathbf{v}$ by the ∞-norm
- i.e. the largest difference between state values:

$$||\mathbf{u} - \mathbf{v}||_\infty = \max_{s \in \mathcal{S}} |\mathbf{u}(s) - \mathbf{v}(s)|$$

Bellman Expectation Backup is a Contraction

- Define the Bellman expectation backup operator

$$F^\pi(\mathrm{v}) = r^\pi + \gamma T^\pi \mathrm{v}$$

- This operator is a γ-contraction, i.e. it makes value functions closer by at least γ,

$$\begin{aligned}
||F^\pi(\mathrm{u}) - F^\pi(\mathrm{v})||_\infty &= ||(r^\pi + \gamma T^\pi \mathrm{u}) - (r^\pi + \gamma T^\pi \mathrm{v})||_\infty \\
&= ||\gamma T^\pi (\mathrm{u} - \mathrm{v})||_\infty \\
&\leq ||\gamma T^\pi (\mathbb{1} \cdot ||\mathrm{u} - \mathrm{v}||_\infty)||_\infty \\
&\leq ||\gamma (T^\pi \mathbb{1}) \cdot ||\mathrm{u} - \mathrm{v}||_\infty)||_\infty \\
&\leq \gamma ||\mathrm{u} - \mathrm{v}||_\infty
\end{aligned}$$

Note we define $\mathbb{1} \equiv [1,1,\ldots,1]^T$

Note $T^\pi \mathbb{1} = \mathbb{1}$

Matrix Form

The Bellman expectation equation can be written concisely using the induced matrix form:

$$\mathrm{v}_\pi = r^\pi + \gamma T^\pi \mathrm{v}_\pi$$

with direct solution

$$\mathrm{v}_\pi = (I - \gamma T^\pi)^{-1} r^\pi$$

of complexity $O(|S|^3)$

here T^{π} is an |S|x|S| matrix, whose (j,k) entry gives $P(s_k \mid s_j, a=\pi(s_j))$

r^{π} is an |S|-dim vector whose j^{th} entry gives $E[r \mid s_j, a=\pi(s_j)]$

v_π is an |S|-dim vector whose j^{th} entry gives $V_\pi(s_j)$

where |S| is the number of distinct states

Convergence of Iterative Policy Evaluation

- The Bellman expectation operator F^π has a unique fixed point
- $\mathbf{v}_\pi$ is a fixed point of F^π (by Bellman expectation equation)
- By contraction mapping theorem: Iterative policy evaluation converges on $\mathbf{v}_\pi$

Given that we know how to evaluate a policy,

how can we discover the optimal policy?

Policy Iteration

$$\pi_0 \xrightarrow{\mathrm{E}} \mathrm{v}_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} \mathrm{v}_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} \mathrm{v}_*$$

policy evaluation

policy improvement
"greedification"

Policy Improvement

- Suppose we have computed v_π for a deterministic policy π
- For a given state s, would it be better to do an action $a \neq \pi(s)$?
- It is better to switch to action a for state s if and only if

$$q_\pi(s,a) > \mathrm{v}_\pi(s)$$

- And we can compute $q_\pi(s,a)$ from v_π by:

$$\begin{aligned} q_\pi(s,a) &\equiv \mathbb{E}_\pi[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \mathrm{v}_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a) \mathrm{v}_\pi(s') \end{aligned}$$

Policy Improvement Cont.

- Do this for all states to get a new policy $\pi' \geq \pi$ that is greedy with respect to v_π:

$$\begin{aligned}
\pi'(s) &= \arg\max_a q_\pi(s,a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma \mathrm{v}_\pi(s') | S_t = s, A_t = a] \\
&= \arg\max r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a) \mathrm{v}_\pi(s')
\end{aligned}$$

- What if the policy is unchanged by this?
    - Then the policy must be optimal.

Policy Iteration

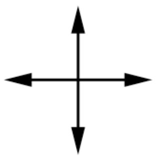
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \boxed{\Sigma_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \Sigma_{s' \in \mathcal{S}} T(s'|s,a) V(s')\right)}$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

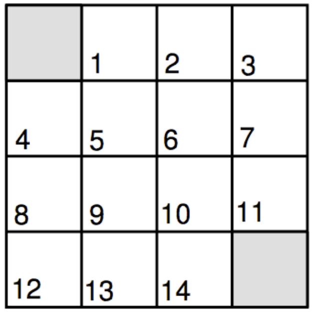
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $a \leftarrow \pi(s)$
   $\pi(s) \leftarrow \boxed{\arg\max r(s,a) + \gamma \Sigma_{s' \in \mathcal{S}} T(s'|s,a) \mathrm{v}_\pi(s')}$
   If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

Iterative Policy Eval for the Small Gridworld

Policy π, an equiprobable random action



actions



$R = -1$
on all transitions

$\gamma = 1$

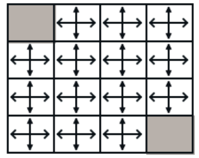
- An undiscounted episodic task
- Nonterminal states: 1, 2, … , 14
- Terminal state: one, shown in shaded square
- Actions that take the agent off the grid leave the state unchanged
- Reward is -1 until the terminal state is reached

V_k for the Random Policy

Greedy Policy w.r.t. V_k

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

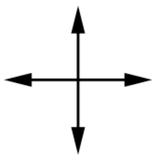
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

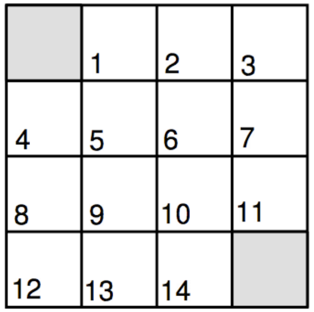
$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Iterative Policy Eval for the Small Gridworld

Initial policy π: equiprobable random action





$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, … , 14
- Terminal state: two, shown in shaded squares
- Actions that take the agent off the grid leave the state unchanged
- Reward is -1 until the terminal state is reached

V_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

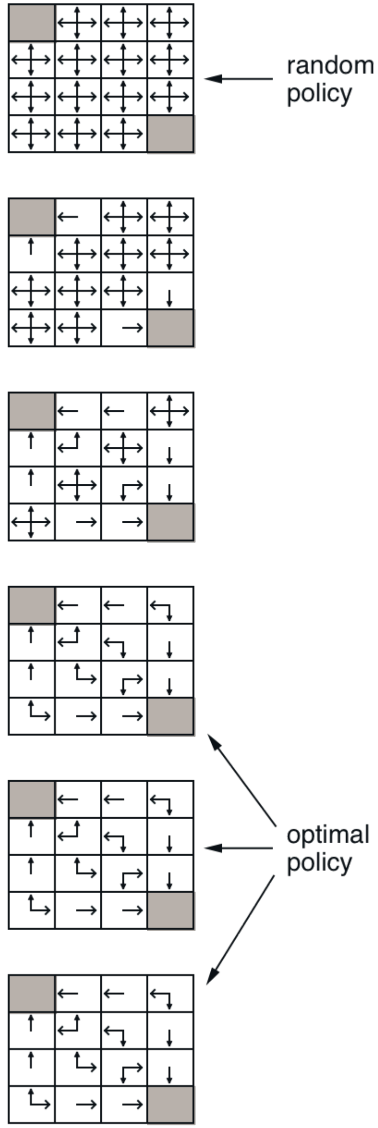
$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

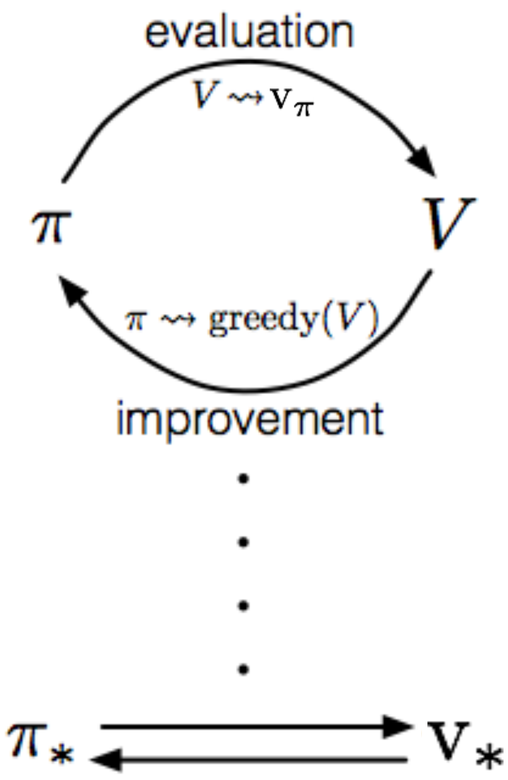
0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Greedy Policy w.r.t. V_k

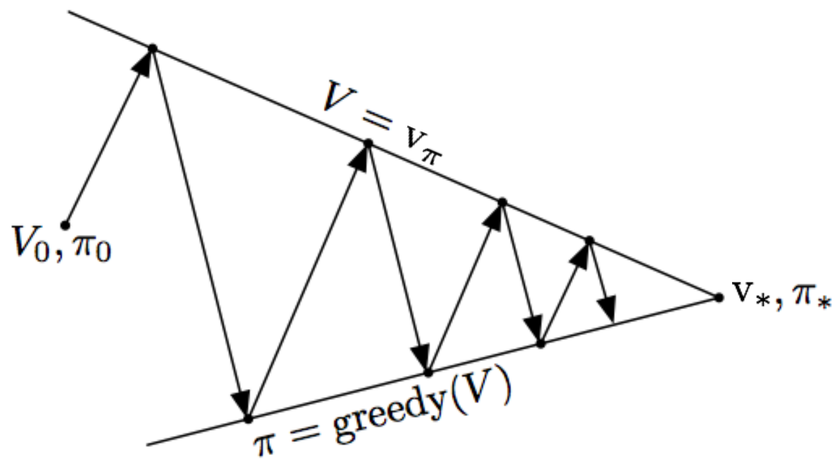


Generalized Policy Iteration

Generalized Policy Iteration (GPI): any interleaving of policy evaluation and policy improvement, independent of their granularity.



A geometric metaphor for convergence of GPI:



Generalized Policy Iteration

- Does policy evaluation need to converge to $\mathbf{v}_\pi$?
- Or should we introduce a stopping condition
  - e.g. ϵ-convergence of value function
- Or simply stop after ***k*** iterations of iterative policy evaluation?
- For example, in the small grid world ***k*** = 3 was sufficient to achieve optimal policy
- Why not update policy every iteration? i.e. stop after ***k*** = 1
  - This is equivalent to value iteration (next section)

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

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

Principle of Optimality

- Any optimal policy can be subdivided into two components:
    - An optimal first action A_*
    - Followed by an optimal policy from successor state S'

- Theorem (Principle of Optimality)
    - A policy $\pi(a|s)$ achieves the optimal value from state s, $v_\pi(s) = v_*(s)$, if and only if
    - For any state s' reachable from s, π achieves the optimal value from state s', $v_\pi(s') = v_*(s')$

Example: Shortest Path

r(s,a)= -1 except for actions entering terminal state

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Bellman Optimality Backup is a Contraction

- Define the Bellman optimality backup operator F^*,

$$F^*(\mathrm{v}) = \max_{a \in \mathcal{A}} r(a) + \gamma T(a)\mathrm{v}$$

- This operator is a γ-contraction, i.e. it makes value functions closer by at least γ (similar to previous proof)

$$||F^*(\mathrm{u}) - F^*(\mathrm{v})||_\infty \leq \gamma ||\mathrm{u} - \mathrm{v}||_\infty$$

Value Iteration Converges to V_*

- The Bellman optimality operator F^* has a unique fixed point
- $\mathbf{v}_*$ is a fixed point of F^* (by Bellman optimality equation)
- By contraction mapping theorem, value iteration converges on $\mathbf{v}_*$

Synchronous Dynamic Programming Algorithms

"Synchronous" here means we
- sweep through every state s in S for each update
- don't update V or π until the full sweep in completed

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s, a)$ or $q_*(s, a)$

Asynchronous DP

- Synchronous DP methods described so far require
  - exhaustive sweeps of the entire state set.
  - updates to V or Q only after a full sweep
- Asynchronous DP does not use sweeps. Instead it works like this:
  - Repeat until convergence criterion is met:
    - Pick a state at random and apply the appropriate backup
- Still need lots of computation, but does not get locked into hopelessly long sweeps
- Guaranteed to converge if all states continue to be selected
- Can you select states to backup intelligently? YES: an agent's experience can act as a guide.

Asynchronous Dynamic Programming

- Three simple ideas for asynchronous dynamic programming:
  - In-place dynamic programming
  - Prioritized sweeping
  - Real-time dynamic programming

In-Place Dynamic Programming

- Multi-copy synchronous value iteration stores two copies of value function
  - for all s in $\mathcal{S}$

$$\mathrm{v}_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a) \mathrm{v}_{old}(s') \right)$$

$$\mathrm{v}_{old} \leftarrow \mathrm{v}_{new}$$

- In-place value iteration only stores one copy of value function
  - for all s in $\mathcal{S}$

$$\mathrm{v}(s) \leftarrow \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a) \mathrm{v}(s') \right)$$

Prioritized Sweeping

- Use magnitude of Bellman error to guide state selection, e.g.

$$\left| \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a) \mathrm{v}(s') \right) - \mathrm{v}(s) \right|$$

- Backup the state with the largest remaining Bellman error
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority queue

Real-time Dynamic Programming

- Idea: update only states that the agent experiences in real world
- After each time-step $\mathcal{S}_t, \mathcal{A}_t, r_{t+1}$
- Backup the state $\mathcal{S}_t$

$$\mathrm{v}(\mathcal{S}_t) \leftarrow \max_{a \in \mathcal{A}} \left(r(\mathcal{S}_t, a) + \gamma \sum_{s' \in \mathcal{S}} T(s' | \mathcal{S}_t, a) \mathrm{v}(s') \right)$$

Sample Backups

- In subsequent lectures we will consider sample backups
- Using sample rewards and sample transitions $(\mathcal{S}, \mathcal{A}, r, \mathcal{S}')$

- Advantages:
  - Model-free: no advance knowledge of T or r(s,a) required
  - Breaks the curse of dimensionality through sampling
  - Cost of backup is constant, independent of $n = |\mathcal{S}|$

Approximate Dynamic Programming

- Approximate the value function
- Using function approximation (e.g., neural net) $\hat{\mathrm{v}}(s, \mathrm{w})$
- Apply dynamic programming to $\hat{\mathrm{v}}(\cdot\,, \mathrm{w})$
- e.g. Fitted Value Iteration repeats at each iteration ***k***,
  - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$
  - For each state $s \in \tilde{\mathcal{S}}$, estimate target value using Bellman optimality equation,

$$\tilde{\mathrm{v}}_k(s) = \max_{a \in \mathcal{A}} \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s, a)\hat{\mathrm{v}}(s', \mathrm{w}_k) \right)$$

  - Train next value function $\hat{\mathrm{v}}(\cdot\,, \mathrm{w}_{k+1})$ using targets $\{\langle s, \tilde{\mathrm{v}}_k(s)\rangle\}$