Lecture 1.2

# Reinforcement Learning: Dynamic Programming Methods

Alexey Gruzdev
*alexey.s.gruzdev@gmail.com*
HSE, Winter 2019

# Recap: Markov Decision Process

- A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

- A Markov reward process is a Markov chain with values.

- *A Markov Decision Process* is a tuple $(\boldsymbol{S}, A, \boldsymbol{P}, \boldsymbol{R}, \boldsymbol{\gamma})$
  - $\boldsymbol{S}$ is a (finite) set of states
  - $\boldsymbol{A}$ is a finite set of actions
  - $\boldsymbol{P}^{a}_{ss'}$ is a state transition probability matrix
  - $\boldsymbol{P}^{a}_{ss'} = \mathrm{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
  - $\boldsymbol{R}$ is a reward function, $\boldsymbol{R}^{a}_{s} = \mathbf{E}\,[R_{t+1} \mid S_t = s, A_t = a]$
  - $\boldsymbol{\gamma}$ is a discount factor, $\gamma \in [0, 1]$

# Ergodic MDPs

*Definition:*

An MDP is ergodic if the Markov Chain induced by any policy is ergodic.

For any policy $\pi$, an ergodic MDP has an *average reward per time-step* $\rho^{\pi}$ that is *independent* of start state.

$$\rho^{\pi} = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}\left[\sum_{t=1}^{T} R_t\right]$$

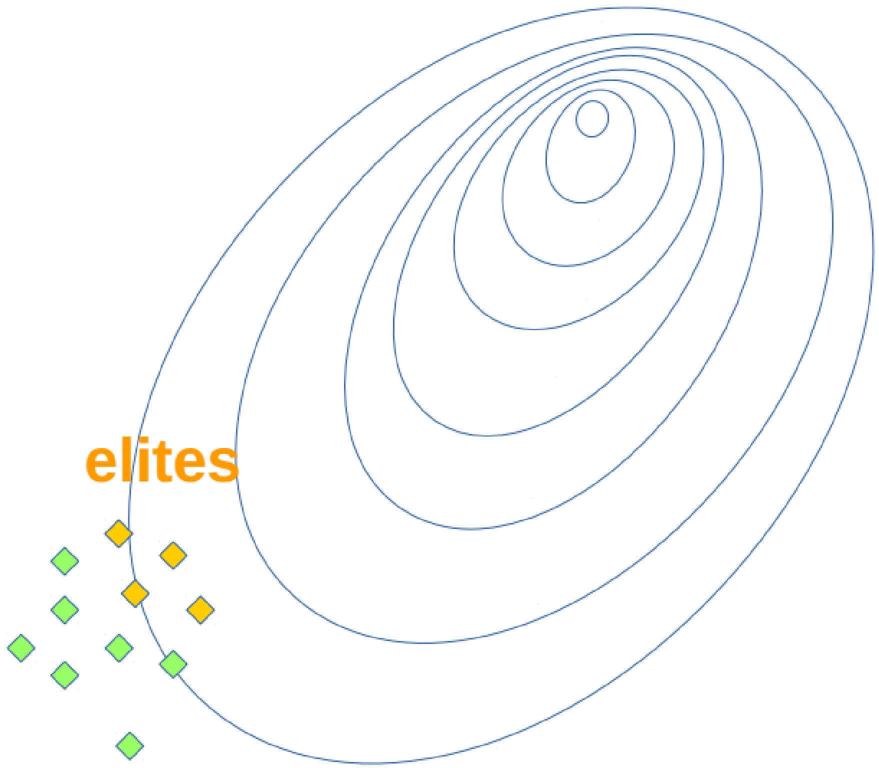
# Recap: Tabular Cross Entropy method

- Policy is matrix *A*
  - $\pi(a \mid s) = P[A_t = a \mid S_t = s] = A_{s,a}$
- Sample N sessions with that policy
- Get M best sessions (elites)
- Elite = $[(s_1,a_1), (s_2,a_2), \ldots, (s_k,a_k)]$
- Update policy:

$$\pi(a \mid s) = \frac{took\ a\ at\ s\ state}{was\ at\ s\ state} = \frac{\sum[s_t=s][a_t=a]}{\sum[s_t=s]}$$

# Recap: Cross Entropy Method

- CEM:
  - Evolutionary
  - Go in the direction where elite goes.
  - Easy to implement
  - Black Box
  - Need to play full episode to start learning



# What is Dynamic Programming?

- *Dynamic* sequential or temporal component to the problem

- *Programming* optimizing a `program`, i.e. a policy

  - A method for solving complex problems
  - By breaking them down into sub problems
  - Solve the sub problems
  - Combine solutions to sub problems

# Requirements for Dynamic Programming

- Dynamic Programming is a very general solution method for problems which have two properties:

- Optimal substructure
  - *Principle of optimality* applies
  - Optimal solution can be decomposed into sub problems

- Overlapping sub problems
  - Sub problems recur many times
  - Solutions can be cached and reused
  - Markov decision processes satisfy both properties
  - Bellman equation gives recursive decomposition
    Value function stores and reuses solutions

# Planning by Dynamic Programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for *planning* in an MDP

- For prediction:
  - *Input:* **MDP (S, A, P, R, γ)** and policy $\boldsymbol{\pi}$
  - or: **MRP (S, P, R, γ)**
  - *Output:* value function $\boldsymbol{v_\pi}$

- For control:
  - *Input:* **MDP - (S, A, P, R, γ)**
  - *Output:* optimal value function $\boldsymbol{v_*}$ and optimal policy $\boldsymbol{\pi_*}$

# A bunch of applications for Dynamic Programming

- Dynamic programming is used to solve many other problems:
  - Scheduling algorithms
  - String algorithms (e.g. sequence alignment)
  - Graph algorithms (e.g. shortest path algorithms)
  - Graphical models (e.g. Viterbi algorithm)
  - Bioinformatics (e.g. lattice models)

- Q: Which algorithms do you know for finding shortest path problems in graphs?

# State-value function

- The *return* $G_t$ is the total discounted reward from time-step $t$.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{t=0}^{\infty} \gamma^t R_{t+k+1}$$

- The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state $s$, and then following policy $\pi$

$$v_\pi(s) = \mathrm{E}_\pi[G_t \mid S_t = s] = \mathrm{E}_\pi[R_t + \gamma G_{t+1} \mid S_t = s] =$$

$$= \sum_a \pi(a \mid s) \sum_{r,s'} p(r, s' \mid s, a)[r + \gamma \mathrm{E}_\pi[G_{t+1} \mid S_{t+1} = s']]$$

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

$\pi(a \mid s)$ – policy stochasticity
$p(r, s' \mid s, a)$ – environment stochasticity

- Intuition: value of following policy $\boldsymbol{\pi}$ from state s

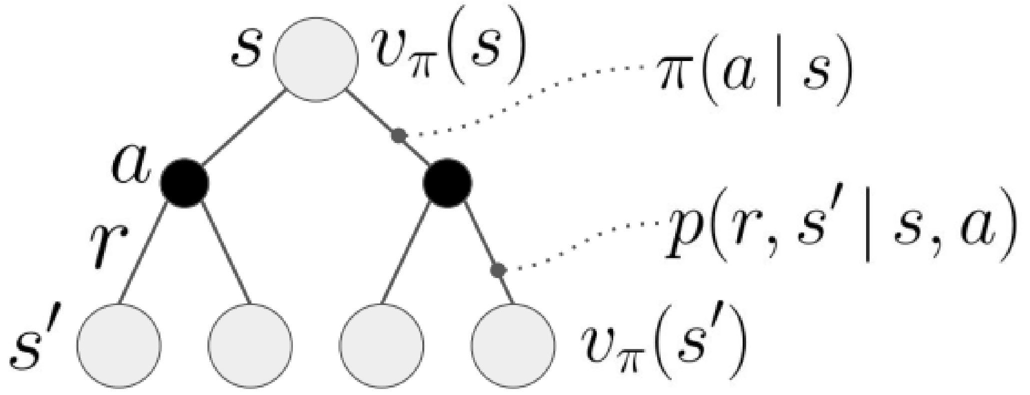
# Bellman Expectation Equation #1

- Recursive definition of $\mathbf{v_\pi(s)}$ is an important concept in RL

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

$$= E_\pi[R_t + \gamma v_\pi(s_{t+1}) \mid S_t = s]$$

Backup diagram



# Action-value function

- The *action-value function* $q_\pi(s, a)$ is the expected return starting from state *s*, taking action *a*, and then following policy $\pi$

$$
\begin{aligned}
q_\pi(s, a) &= E_\pi[G_t \mid S_t = s, A_t = a] \\
&= E_\pi[R_t + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \sum_{r,s'} p(r, s' \mid s, a)[r + \gamma E_\pi[G_{t+1} \mid S_{t+1} = s']] \\
&= \sum_{r,s'} p(r, s' \mid s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

- <u>Intuition:</u> value of following policy $\boldsymbol{\pi}$ from state **s**, after committing action **a**

# Relations between Action/state value functions

- We know relationship $q_\pi(s, a)$ in terms of $v_\pi(s)$

$$q_\pi(s, a) = \sum_{r,s'} p(r, s' \mid s, a)[r + \gamma v_\pi(s')]$$

- What about $v_\pi(s)$ in terms of $q_\pi(s, a)$

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

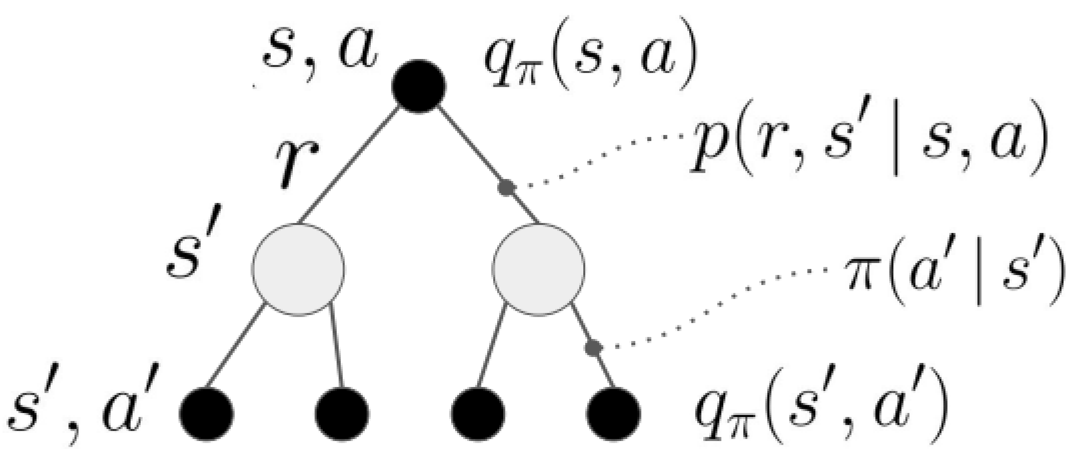
$$v_\pi(s) = \sum_a \pi(a \mid s) \; q_\pi(s, a)$$

# Bellman Expectation Equation #2

- Recursive definition of $\boldsymbol{q_\pi(s, a)}$ in term of $\boldsymbol{q_\pi(s, a)}$

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

Backup diagram for q(s, a)



# Recap: optimal policy

- Define a partial ordering over policies:

$$\boldsymbol{\pi \geq \pi' \text{ if } v_{\pi}(s) \geq v_{\pi'}(s) \;\; \forall s}$$

***Theorem:*** *For any Markov Decision Process*

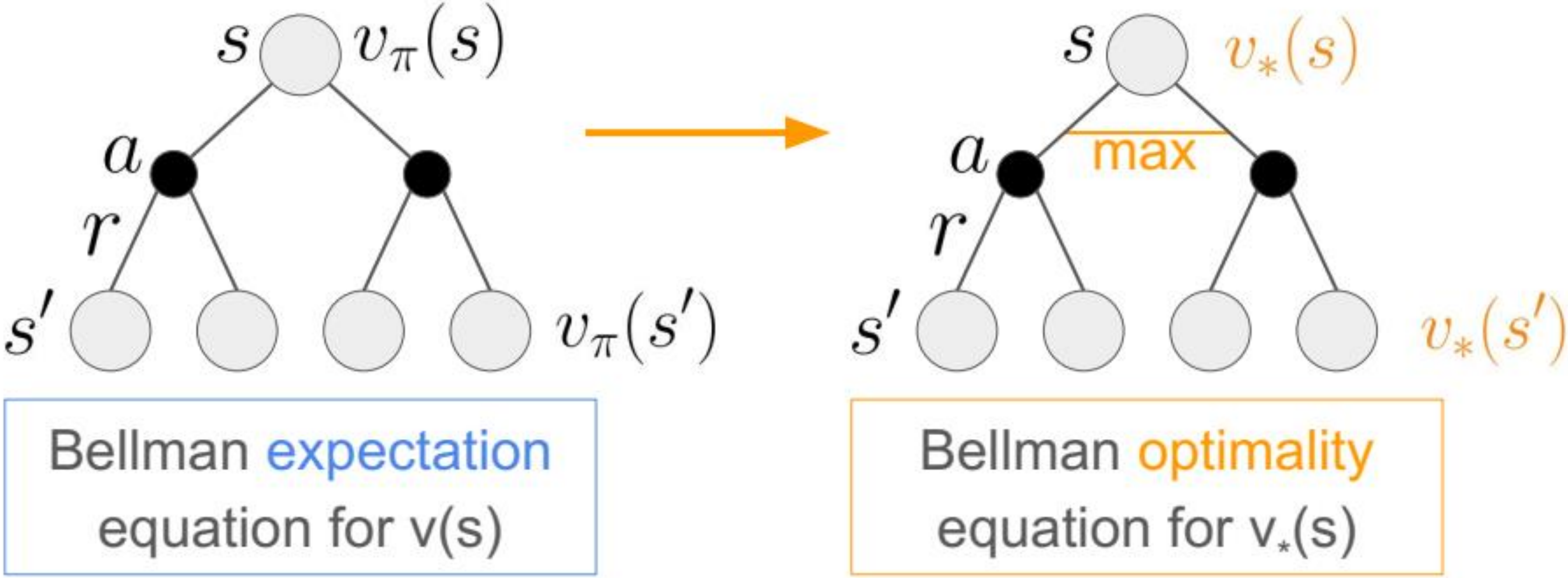
- *Best policy* $\boldsymbol{\pi_*}$ *performs better or equal than any other policy* $\boldsymbol{\pi_* \geq \pi \;\; \forall \pi}$
- *Optimal state-value function:*

$$\boldsymbol{v_*(s) = \max_{\pi}(v_{\pi}(s))}$$

- *Optimal action-value function:*

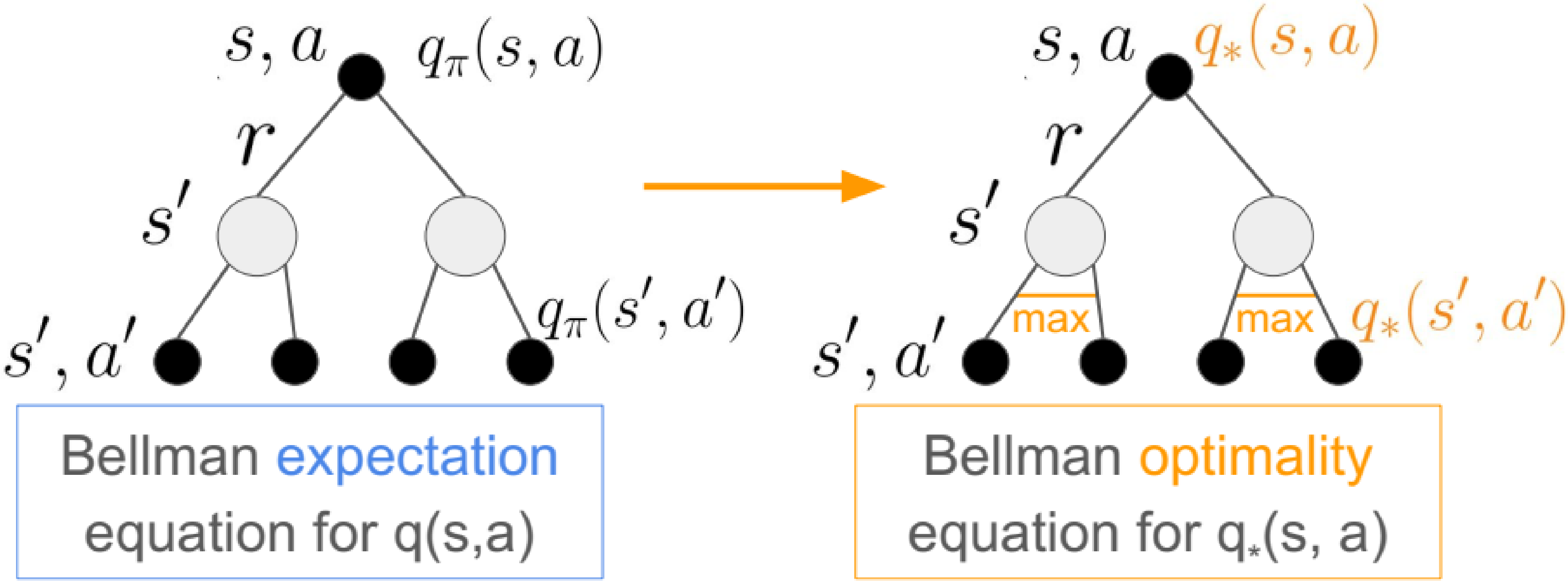
$$\boldsymbol{q_{\pi*}(s, a) = \max_{\pi}(q_{\pi}(s, a))}$$

# Bellman Optimality Equation #1




$$v_*(s) = \max_a \sum_{r,s'} p(r, s' \mid s, a)[r + \gamma v_*(s')]$$

# Bellman Optimality Equation #2




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

# Iterative Policy Evaluation

- <u>Problem:</u> evaluate a given policy $\boldsymbol{\pi}$
- Solution: iterative application of Bellman expectation backup
- $\boldsymbol{v_1 \text{ -> } v_2 \text{ -> } .... \text{ -> } v_\pi}$
- Using *synchronous* backups:
  - At each iteration $t + 1$
  - For all states $\boldsymbol{s} \in \boldsymbol{S}$
  - Update $\boldsymbol{v_{t+1}(s)}$ from $\boldsymbol{v_t(s')}$, where $s'$ is a successor state of $s$
- Convergence is guaranteed but is not subject of this lecture.

# Policy Evaluation

**Iterative Policy Evaluation, for estimating $V \approx v_\pi$**

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

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

# Policy Improvement

- Given a policy $\pi$:
  - Evaluate the policy $\pi$

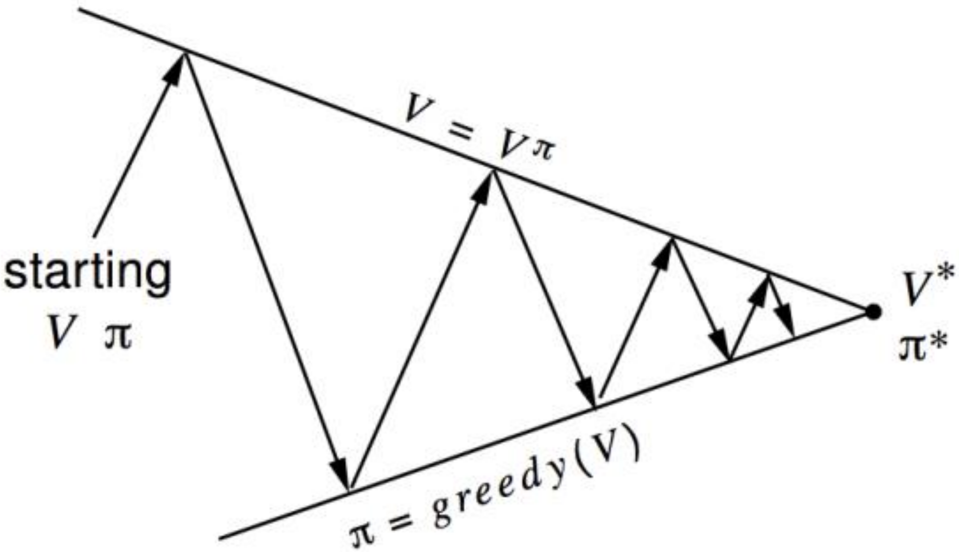
$$\boldsymbol{v_\pi(s) = \mathrm{E}\,[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s]}$$

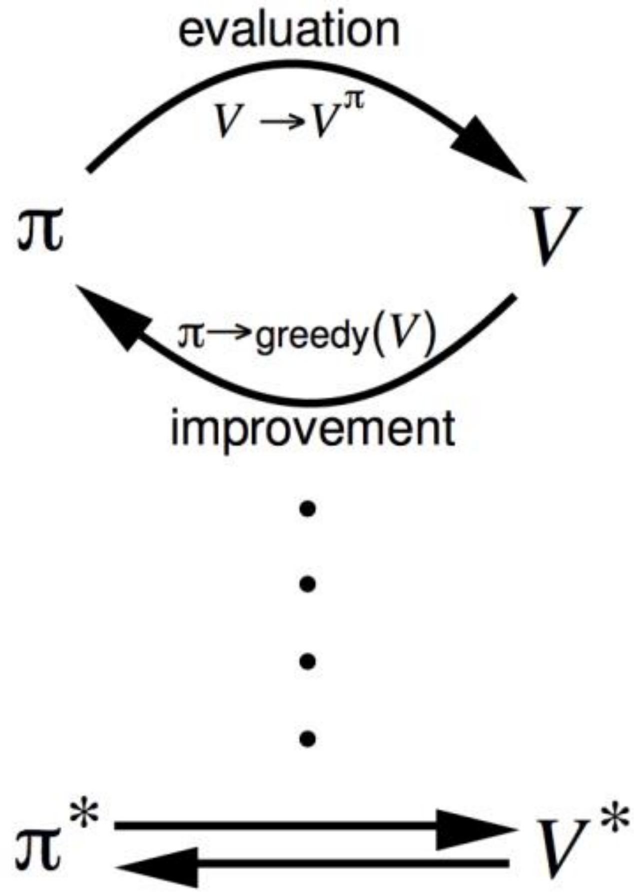
  - Improve the policy by acting greedily with respect to $v_\pi$

$$\boldsymbol{\pi' = \mathrm{greedy}(v_\pi)}$$

  - In general, need more iterations of improvement / evaluation
  - But this process of policy iteration always converges to $\pi *$

# Policy Improvement




Policy evaluation Estimate $v_\pi$

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

# 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:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[ r + \gamma V(s') \big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

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

# Modified Policy Iteration

- Does policy evaluation need to converge to $v_\pi$?
- Or should we introduce a stopping condition
  - e.g. $\varepsilon$-convergence of value function
- Or simply stop after $k$ iterations of iterative policy evaluation?
- Why not update policy every iteration? i.e. stop after $k = 1$
  - This is equivalent to *value iteration* (next section)

# Value Iteration

- Problem: find optimal policy $\pi$
- Solution  iterative application of Bellman optimality backup
- $\boldsymbol{v_1} \textbf{ -> } \boldsymbol{v_2} \textbf{ -> .... -> } \boldsymbol{v_\pi}$
- Using synchronous backups
  - At each iteration $t + 1$
  - For all states $\boldsymbol{s} \in \boldsymbol{S}$
  - Update $\boldsymbol{v_{t+1}(s)}$ from $\boldsymbol{v_t(s')}$, where $s'$ is a successor state of $s$
  - Unlike policy iteration, there is no explicit policy
  - Intermediate value functions may not correspond to any policy

# Value Iteration

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

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

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

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

# Value vs Policy Iteration

- VI is faster per iteration – $O(|A||S|^2)$
- VI requires many iterations

- PI is slower per iteration – $O(|A||S|^2 + |S|^3)$
- PI requires few iterations

- No silver bullet $\longrightarrow$ experiment with # of steps spent in policy evaluation phase to find the best

# Resume

| 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 |

Let’s do something real?!

# Questions?