CSCI E-82a
Probabilistic Programming and AI
Lecture 9
Dynamic Programming

Steve Elston

HARVARD
Extension School

Introduction to the Bellman Equations and Dynamic Programming

- What is Dynamic programming?
- Policy evaluation
- State-value policy evaluation
- Action-value policy evaluation
- Control: Policy-improvement
- Policy improvement with state-values
- Policy improvement with action-values
- Policy iteration
- Value iteration

Introduction to Dynamic Programming

- Sequential and single decision processes both built on **Markov process theory**
- We have explored single decision processes
  - Each decision is made based on **MAP values**
  - **State updates** are used to make subsequent decisions
- Can a sequential decision process uses a policy
  - Policy determines **probabilities of action** given state
  - **Sequence of actions** determined by policy and state transitions
  - Use a Markov model to find policy

Introduction to Dynamic Programming

- **Dynamic programming** was developed by the mathematician Richard Bellman in the early 1950s
- Dynamic programming is a **optimal sequential planning method**
  - Planning methods use a **model of the environment**
  - Environment model is a **Markov process**
  - Maximize **total reward** given a **model of the environment**
- Planning methods enable an intelligent agent to **gain autonomy**
  - Perform a **sequence** of **optimal actions**
  - Follow **plan** or **policy**

Introduction to Dynamic Programming

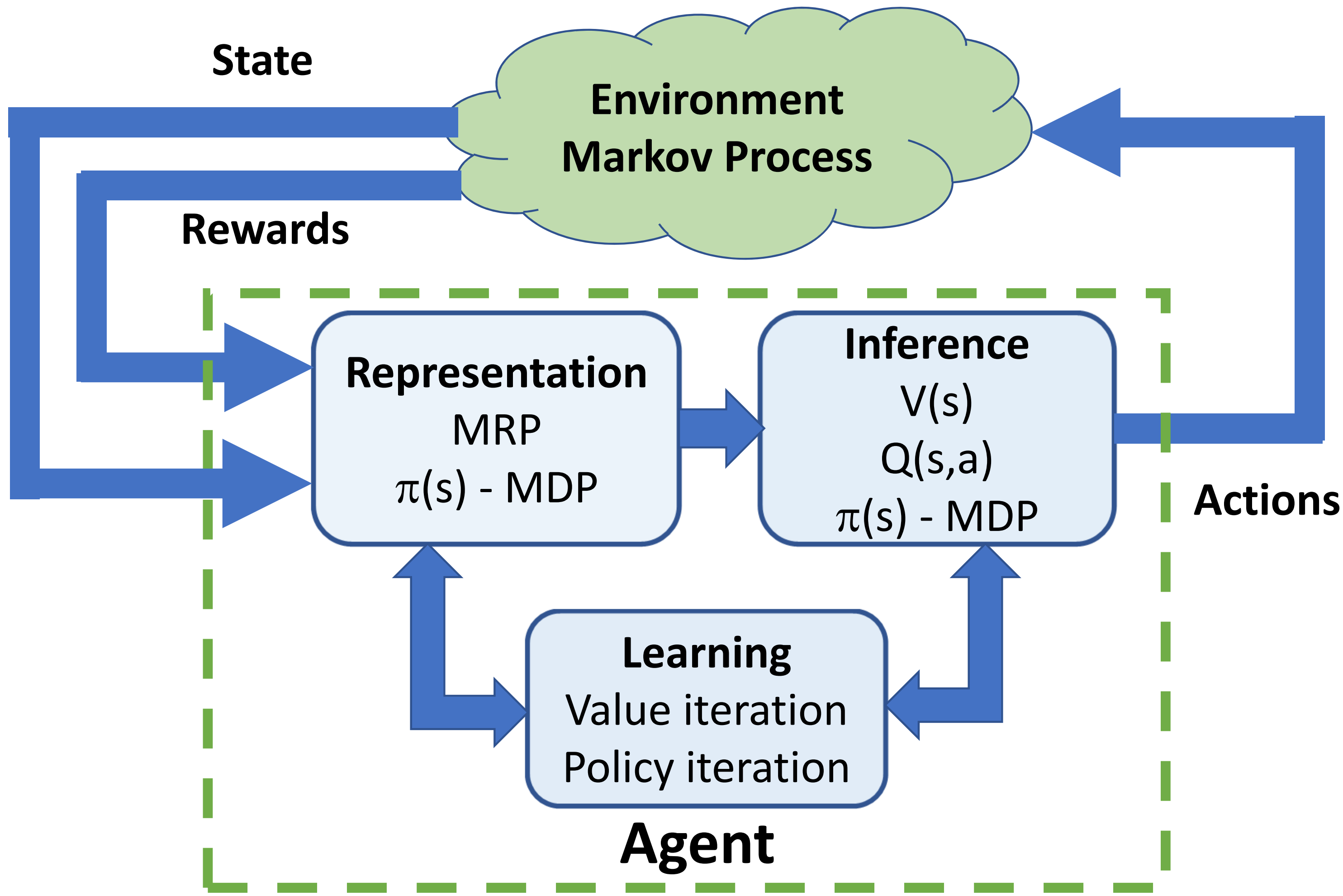
- Why did Bellman give this method its name?
  - **Programming** is a computer algorithm which creates a **plan of actions** to optimize the **utility** or **total reward**
  - A **Dynamic** algorithm solves the problem recursively, operating on **smaller and simpler overlapping sub-problems**
- DP methods are scalable, practical and widely used
  - Schedule optimization
  - Optimal routing
  - Optimal control
  - Motion planning
  - Etc.

Introduction to Dynamic Programming

Reinforcement learning vs. dynamic programming

- Like DP, **reinforcement learning** is a class of optimization algorithms to optimize utility in a system represented by a Markov processes
- For many problems intelligent agents can use either DP or RL
    - DP requires **model specification**
    - RL is **model free**
    - Both DP and RL use **bootstrapping algorithms**

The Dynamic Learning Agent



Policy Evaluation

- We want our intelligent agent to follow an **optimal policy**
- **Policy evaluation** is needed to compare policy
- Can evaluate policy by value:
  - **State value**: expected value of being in a state
  - **Action value**: expected value of taking an action in a given state

Policy Evaluation

- Recall the definition of **discounted return** from the current time t

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

- Where, R_{t+1} is the **expected reward or change in utility** over the possible state transitions from the **current state**, *s*, to all possible **successor states**, *s'*

$$R_{t+1} = E\left[\mathcal{R}_{ss'} \mid S_t = s\right]$$

And,

$\mathcal{R}_{ss'}$ = the reward for the transition from state *s* to *s'*

γ = discount factor

- In words, the gain is the sum of the probabilistic expectation of rewards for all future possible state transitions

State Value Policy Evaluation

- **Bellman value equations** are fundamental to computing expected state-values
- Recall that a first order Markov process depends only on the current state, *s*
- The **probability of a state transition** and therefore the expected reward is determined, in part, by the **policy**, π
- We can find the state-value of state s, given a policy π, as the expected value of the gain from the Bellman value equation:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

Where $\mathbb{E}_\pi[] =$ Expectation given policy π

State Value Policy Evaluation

Expand the Bellman value equations to find a **recursion**

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

Since **gain equals the reward plus the gain for the next step**:

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

This recursion leads to **one-step bootstrap approximation of gain using state-value** for the next step:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

The **bootstrap** approximation uses the current estimate of $v_\pi(S_{t+1})$ to update the estimate of $v_\pi(s)$

State Value Policy Evaluation

Compute the expected value for the Bellman value equations

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

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

Where

$a =$ an action by the agent

$\pi(a|s) =$ the policy specifying the probability of action, a, given state, s,

$p(s', r \mid s, a) =$ probability of successor state, s', and reward, r, given state, s, and action a

$\Big[r + \gamma v_\pi(s')\Big] =$ the bootstrapped state value

State Value Policy Evaluation

- There is one Bellman value equation for each possible state, s, or n equations for a n-state system
- In theory this system of equations can be solved directly
  But requires $O(n^3)$ computations
- Or, can use the recursion relationship using the last estimate of $v_\pi(s')$

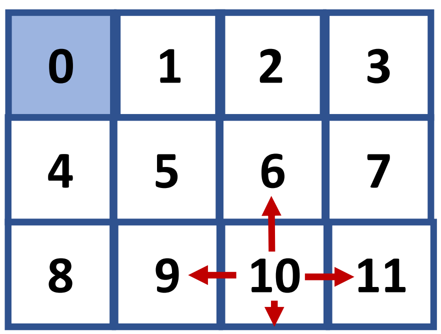
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|s, a)\Big[r + \gamma v_\pi(s')\Big], \ \forall a$$

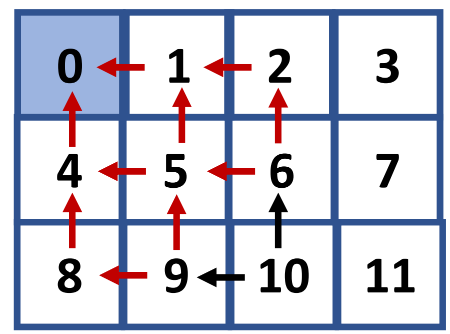
- Using the estimated state-value to compute a better estimate is called **bootstrapping**
- Recursion continues until **convergence criteria** is achieved

State Value Policy Evaluation

How to interpret the Bellman state-value equations

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





$\pi(a|s)$ = {u:0.5,d:0.0,l:0.5,r:0.0}

$p(s', r \mid s, a)$ = {(6,$r_{10\text{-}6}$|10,u):1.0,
(9,$r_{10\text{-}9}$|10,l):1.0}

$$V_\pi(10) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]$$

Action Value Policy Evaluation

- **Bellman action-value equations** are fundamental to computing expected action-values
- Recall the definition of state-value:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

- We can find the action-value, of taking action a in state s, given a policy π as:

$$q_\pi(s,\ a) = \mathbb{E}_\pi[G_t \mid S_t = s,\ A_t = a]$$

Where: $\mathbb{E}_\pi[] =$ Expectation given policy π

Action Value Policy Evaluation

Expand the Bellman action-value equations to find a **recursion**

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t \,\middle|\, S_t = s, A_t = a\right]$$

Expanding the **gain as the sum of rewards** gives:

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

Then using the transition probability and the **bootstrapped action-values** gives:

$$q_\pi(s,a) = \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \, q_\pi(S_{t+1}, a')\right]$$

Where,

$$A_t = \text{the action taken at step } t$$

$$a' = \text{the action taken from the successor state, } s'$$

Control: Policy Improvement

- There are two approaches to policy improvement algorithms
- Iterate between **improvement** and **evaluation**
- **Evaluation** measures progress at improving policy
- **Policy iteration** finds improvement with state value method
  - Policy seeks the action with the greatest value improvement given state
- **Value iteration** finds improvement with action-value method
  - Policy seeks the action with the greatest action-value improvement given state

Control: Policy Improvement

- We want algorithms which **improve value of a policy, π,** at each iteration to find an improved policy **,** π', such that:

$$v_{\pi'}(s) \geq v_\pi(s)$$

- Ideally want an **optimal policy**
- **Policy improvement theorem** says an optimal policy has **the highest value of any possible policy**
- For **state-values**, the **policy improvement theorem** is:

$$v_*(s) \geq v_\pi(s) \; \forall \pi$$

  Where $v_*(s)$ is the optimal policy
- But, $v_*(s)$ is **not necessarily unique**!

Control: Policy Improvement

- Can also measure policy with **action-values**
- For **action-values**, the **policy improvement theorem** is:

$$q_*(s,\ a) \geq q_\pi(s,\ a)\ \forall\ \pi$$

Where

$$q_*(s, a) = max_\pi q(s, a)$$

- Again, the optimal policy may **not be unique**

Policy Improvement with State-Values

- Need to find an algorithm to compute a policy π_* such that:

$$v_{\pi'}(s) \geq v_\pi(s)$$

- The **Bellman state-value equations** are a **bootstrap** formulation for computing optimal policy

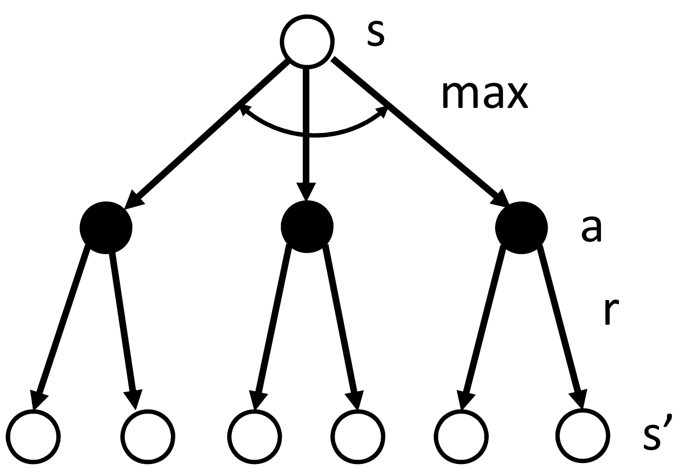
$$\begin{aligned} v_*(s) &= max_\pi v_\pi(s) \\ &= max_a \ \mathbb{E}[G_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= max_a \ \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_*(s')\Big] \end{aligned}$$

Policy Improvement with State-Values

- How to understand the **bootstrap Bellman state-value equations**?

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

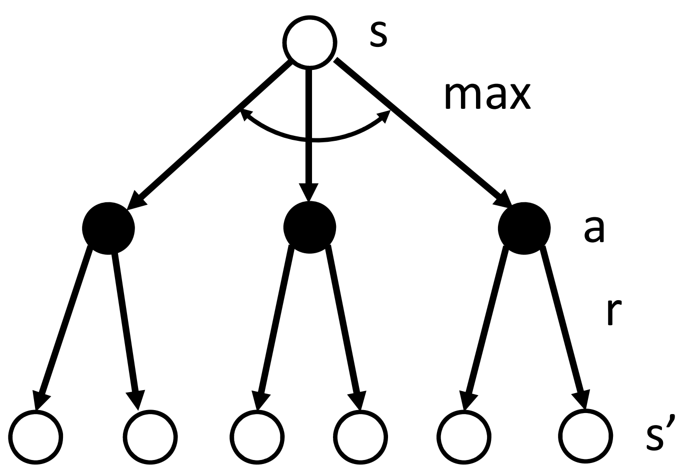
- Use a **backup diagram**:



1. Start Markov process in state s:
   state = Open circle
2. Take action a that maximizes state-value:
   action = Filled circle
3. Leads to successor states s’ with reward r

Policy Improvement with State-Values

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



- Each iteration **backs up** into a better estimates of state-value by looking one step ahead
- Since the reward for all actions must be computed, the algorithm is said to use a **full backup**
- The computations for the full backup grow with the number of actions and successor states
- Bellman called this property the **curse of dimensionality**

Policy Iteration

Policy iteration alternately evaluates and then improves policy, π

- Algorithm converges toward optimal policy, π_* and state-value v_*

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

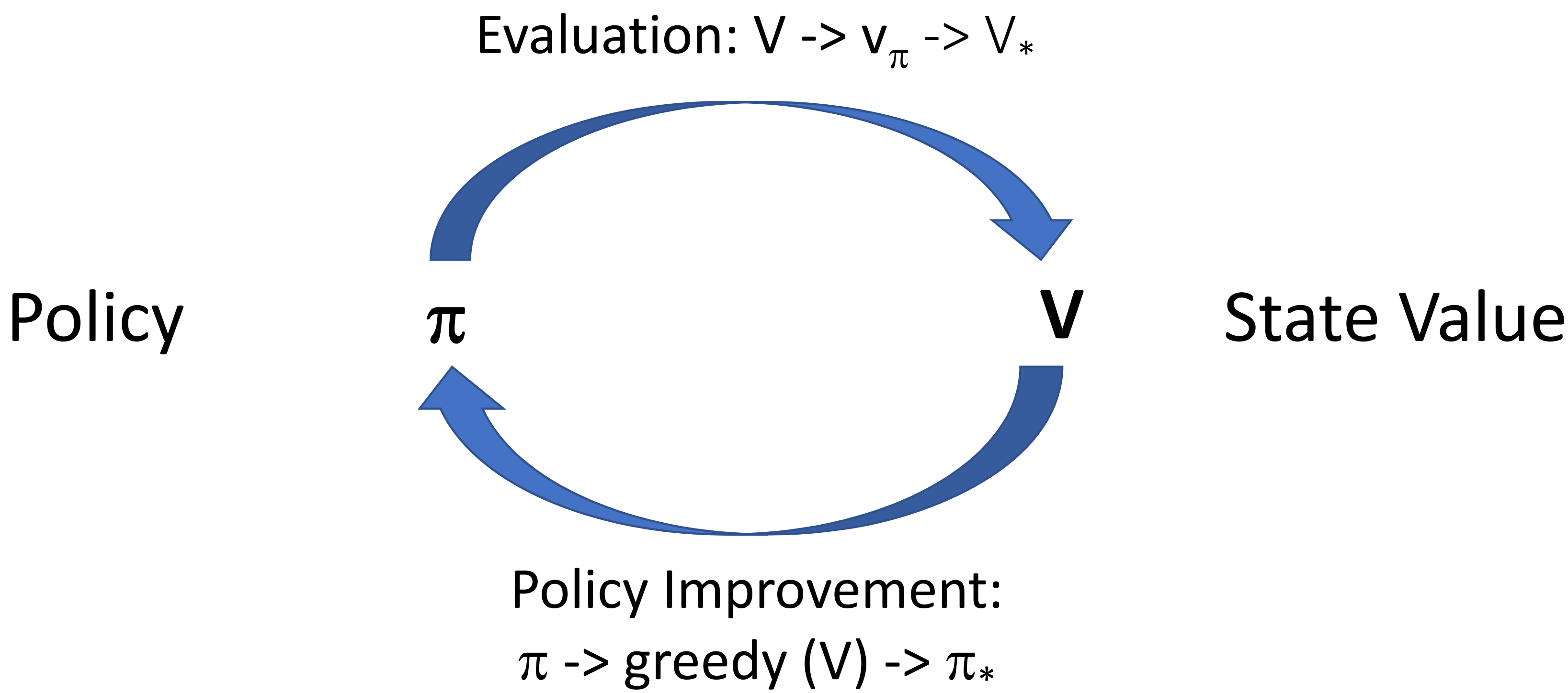
Where

$\xrightarrow{E}$ is the **evaluation** by state-value estimation

$\xrightarrow{I}$ is the **greedy policy improvement** update

Policy Iteration

Iterative algorithm alternates **state value policy evaluation** and **state value policy improvement**:



Policy Iteration

Policy iteration algorithm iterates two steps

1. Perform state-value policy iteration **until convergence**
   - Convergence occurs when state value improvement is less that threshold value
2. Find optimal policy for each state in environment, *s*:

$$v(s) \leftarrow \max_{a} \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma\, v_k(s')\right]$$

Step 1 is **full policy evaluation**

Policy Improvement with Action-Values

- Need to find an algorithm to compute a policy π_* such that:

$$q_*(s,\ a) \geq q_\pi(s,\ a)\ \forall\ \pi$$

- The Bellman **action-value equations** are a **bootstrap** formulation for computing optimal policy

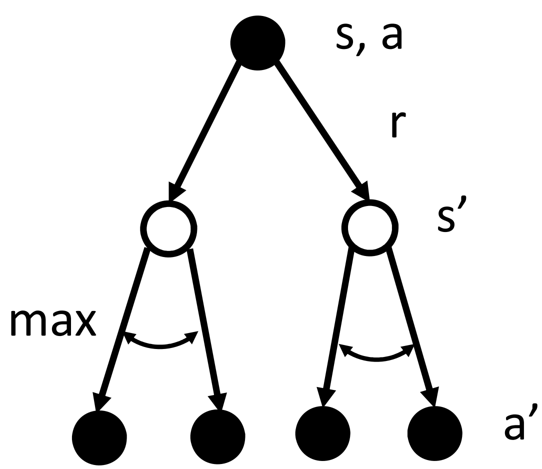
$$\begin{aligned} q_*(s,a) &= \max_a q(s,a) \\ &= \mathbb{E}\Big[R_{t+1} + \gamma \max_a q_\pi(S_{t+1}, a')\ \Big|\ S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_a q_\pi(S_{t+1}, a)\Big] \end{aligned}$$

Policy Improvement with Action-Values

- How to understand the Bellman **bootstrap action-value equations**?

$$q_*(s,a) = \sum_{s',r} p(s',r \mid s,a)\Big[r + \gamma \max_a q_\pi(S_{t+1},a)\Big]$$

- Use a **backup diagram** to understand this method:



1. Start Markov process with state action tuple (s,a)

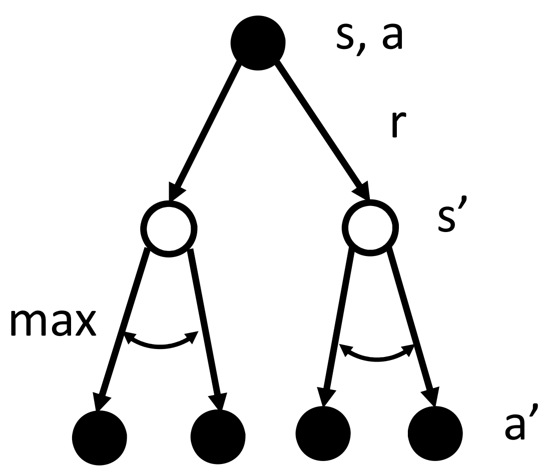
2. Leads to successor states, s’, with reward r

3. Successor action, a’, that maximizes expected value

4. Maximum of successor action-value is used to bootstrap next optimal action-value

Policy Improvement with Action-Values

$$q_*(s,a) = \sum_{s',r} p(s',r \mid s,a)\Big[r + \gamma \max_a q_\pi(S_{t+1},a)\Big]$$



- Each iteration **backs up** into better estimates of action-value by looking one step ahead
- Since the reward for all successor state action pairs is computed, the algorithm is said to use a **full backup**
- The computations for the full backup grow with the number of actions and successor states
- Same **curse of dimensionality** as policy iteration

Value Iteration

Value iteration uses the recursion relation of the **Bellman optimal state-value equations:**

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

- The next state-value, $v_{k+1}(s)$, is the **maximum over all possible actions** for that state
- Algorithm converges using **truncated policy evaluation**

Value Iteration

Value iteration algorithm has 2 steps**:**

1. Update state-values using the maximum over possible actions
2. If state-value change is greater than **convergence criteria**, repeat step 1

- Convergence occurs when change in action value between iterations is less than threshold
- Step 1 **need not run to convergence!**

Definitions

- Dynamic programming is a **optimal sequential planning method** with Markov model and a policy
- **Policy** determines probabilities of action given state
- **Sequence of actions** determined by policy and state transitions
- A **Dynamic** algorithm solves the problem recursively, operating on **smaller and simpler overlapping sub-problems**
- **Reinforcement learning** is a class of optimization algorithms to optimize utility in a system represented by a Markov processes
- **Bellman value equations** are fundamental to computing expected state-values
- The state-value, of state s, given a policy π as the expected value of the gain:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

Definitions

- The **expected value for the Bellman value equations**

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

- Recursion continues until **convergence criteria** is achieved
- **discounted return** from the current time t

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

- Using the estimated state-value to compute a better estimate is called **bootstrapping**

Definitions

- For **state-values**, the **policy improvement theorem** is:

$$v_{\pi'}(s) \geq v_{\pi}(s)$$

  Where $v_*(s)$ is the optimal policy

- **Value iteration**: The next state-value, vk+1(s), is the maximum over all possible actions for that state