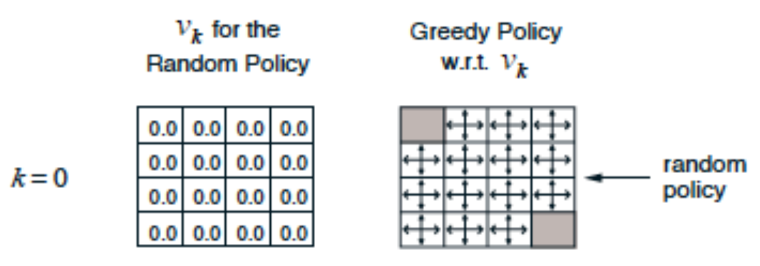
Summary



First step of policy iteration in gridworld example (Sutton and Barto, 2017)

Introduction

- In the **dynamic programming** setting, the agent has full knowledge of the MDP. (This is much easier than the **reinforcement learning** setting, where the agent initially knows nothing about how the environment decides state and reward and must learn entirely from interaction how to select actions.)

An Iterative Method

- In order to obtain the state-value function v_π corresponding to a policy π, we need only solve the system of equations corresponding to the Bellman expectation equation for v_π.
- While it is possible to analytically solve the system, we will focus on an iterative solution approach.

Iterative Policy Evaluation

- **Iterative policy evaluation** is an algorithm used in the dynamic programming setting to estimate the state-value function v_π corresponding to a policy π. In this approach, a Bellman update is applied to the value function estimate until the changes to the estimate are nearly imperceptible.

Iterative Policy Evaluation

Input: MDP, policy π, small positive number θ
Output: $V \approx v_\pi$
Initialize V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)
repeat
  $\Delta \leftarrow 0$
  for $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma V(s'))$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
until $\Delta < \theta$;
return V

Estimation of Action Values

- In the dynamic programming setting, it is possible to quickly obtain the action-value function q_π from the state-value function v_π with the equation: $q_\pi(s, a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma v_\pi(s'))$.

Estimation of Action Values

Input: state-value function V
Output: action-value function Q
for $s \in \mathcal{S}$ **do**
  for $a \in \mathcal{A}(s)$ **do**
    $Q(s, a) \leftarrow \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma V(s'))$
  end
end
return Q

Policy Improvement

- **Policy improvement** takes an estimate V of the action-value function v_π corresponding to a policy π, and returns an improved (or equivalent) policy π', where $\pi' \geq \pi$. The algorithm first constructs the action-value function estimate Q. Then, for each state $s \in \mathcal{S}$, you need only select the action a that maximizes $Q(s, a)$. In other words, $\pi'(s) = \arg\max_{a \in \mathcal{A}(s)} Q(s, a)$ for all $s \in \mathcal{S}$.

Policy Improvement

Input: MDP, value function V
Output: policy π'
for $s \in \mathcal{S}$ **do**
  for $a \in \mathcal{A}(s)$ **do**
    $Q(s, a) \leftarrow \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma V(s'))$
  end
  $\pi'(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)} Q(s, a)$
end
return π'

Policy Iteration

- **Policy iteration** is an algorithm that can solve an MDP in the dynamic programming setting. It proceeds as a sequence of policy evaluation and improvement steps, and is guaranteed to converge to the optimal policy (for an arbitrary *finite* MDP).

Policy Iteration

Input: MDP, small positive number θ
Output: policy $\pi \approx \pi_*$
Initialize π arbitrarily (e.g., $\pi(a|s) = \frac{1}{|\mathcal{A}(s)|}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$)
policy-stable $\leftarrow$ *false*
repeat
  $V \leftarrow$ **Policy_Evaluation**$(\text{MDP}, \pi, \theta)$
  $\pi' \leftarrow$ **Policy_Improvement**(MDP, V)
  if $\pi = \pi'$ **then**
    policy-stable $\leftarrow$ *true*
  end
  $\pi \leftarrow \pi'$
until *policy-stable* $=$ *true*;
return π

Truncated Policy Iteration

- **Truncated policy iteration** is an algorithm used in the dynamic programming setting to estimate the state-value function v_π corresponding to a policy π. In this approach, the evaluation step is stopped after a fixed number of sweeps through the state space. We refer to the algorithm in the evaluation step as **truncated policy evaluation**.

Truncated Policy Evaluation

Input: MDP, policy π, value function V, positive integer *max_iterations*
Output: $V \approx v_\pi$ (if *max_iterations* is large enough)
counter $\leftarrow 0$
while *counter* $<$ *max_iterations* **do**
  for $s \in \mathcal{S}$ **do**
    $V(s) \leftarrow \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma V(s'))$
  end
  counter $\leftarrow$ *counter* $+ 1$
end
return V

Truncated Policy Iteration

Input: MDP, positive integer *max_iterations*, small positive number θ
Output: policy $\pi \approx \pi_*$
Initialize V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)
Initialize π arbitrarily (e.g., $\pi(a|s) = \frac{1}{|\mathcal{A}(s)|}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$)
repeat
  $\pi \leftarrow$ **Policy_Improvement**(MDP, V)
  $V_{old} \leftarrow V$
  $V \leftarrow$ **Truncated_Policy_Evaluation**$(\text{MDP}, \pi, V, \textit{max\_iterations})$
until $\max_{s \in \mathcal{S}} |V(s) - V_{old}(s)| < \theta$;
return π

Value Iteration

- **Value iteration** is an algorithm used in the dynamic programming setting to estimate the state-value function v_π corresponding to a policy π. In this approach, each sweep over the state space simultaneously performs policy evaluation and policy improvement.

Value Iteration

Input: MDP, small positive number θ
Output: policy $\pi \approx \pi_*$
Initialize V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)
repeat
  $\Delta \leftarrow 0$
  for $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma V(s'))$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
until $\Delta < \theta$;
$\pi \leftarrow$ **Policy_Improvement**(MDP, V)
return π

NEXT