Markov Decision Processes

Definition

A **Markov decision process** (MDP) $M = (S, A, P, R, T)$ is a 5-tuple containing the following components:

- S is a set of **states** in the world;
- A is a set of **actions** that the agent can take;
- P is a **state-transition function**: $P(s, a, s')$ denotes the probability of ending up in state s' if action a is taken in state s;
- R is a **reward function**: $R(s, a, s')$ is the one-step reward obtained if the agent enters state s' after taking action a in state s;
- T is a **time horizon** (i.e., a limit on the number of steps that can be taken).

Revisiting the Bellman Equations

Recall the **Bellman equations** for the **state-value** and **action-value** functions:

$$V^\pi(s) = \sum_{s' \in S} P(s, \pi(s), s') \cdot \left[R(s, \pi(s), s') + \gamma V^\pi(s') \right]$$

$$Q^\pi(s, a) = \sum_{s' \in S} P(s, a, s') \cdot \left[R(s, a, s') + \gamma V^\pi(s') \right]$$

We have $V^\pi(s) = Q^\pi(s, \pi(s))$, so we can write $Q^\pi(s, a)$ as:

$$Q^\pi(s, a) = \sum_{s' \in S} P(s, a, s') \cdot \left[R(s, a, s') + \gamma Q^\pi(s', \pi(s')) \right].$$

During **learning**, when **the agent** takes action a in state s and ends up in state s' with one-step reward $r = R(s, a, s')$, it can assume that $P(s, a, s') = 1$ and approximate the **Bellman equation** as:

$$Q^\pi(s, a) \approx r + \gamma Q^\pi(s', \pi(s')).$$

TD Updates for Action-Values

The **TD(0) update** for **action-values** is given by:

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right].$$

TD Updates for Action-Values: Evaluating a Policy

---

procedure $\textsc{TDPolicyEvaluation}$(MDP M; Policy π; Discount factor γ; Step size α)
  for each $(s,a) \in S \times A$ **do**
    $Q(s,a) \leftarrow 0$ ▷ $Q(s,a)$ is current estimate of $Q^{\pi}(s,a)$
  for each episode **do**
    $s \leftarrow s_0$ ▷ s is current state, s_0 is fixed start state of M
    $a \leftarrow \pi(s)$ ▷ a is next action to take
    while episode has not ended **do**
      Execute action a
      Observe next state s' and one-step reward $r = R(s,a,s')$
      $a' \leftarrow \pi(s')$ ▷ a' is what the agent will do next
      $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
      $s \leftarrow s'$, $a \leftarrow a'$ ▷ Update current state and next action
  return Q ▷ $Q(s,a) \approx Q^{\pi}(s,a)$ for all $(s,a) \in S \times A$

Towards an Optimal Policy

The agent can use the following process to find an **optimal** policy:

1. Start with an initial policy π (possibly random)
2. Learn Q^π via TD(0) updates (**policy evaluation**)
3. Construct π' from Q^π values (**policy improvement**)
4. Update policy $\pi \leftarrow \pi'$ and repeat until no changes

This is potentially **slow**, though: **the agent** has to wait until it has learned Q^π before it can improve its policy.

Generalized policy iteration refers to the general idea of combining **policy evaluation** and **policy improvement** within a single process.

In this particular case, we'd like **the agent** to try to learn an **optimal** policy from the get-go, by potentially **updating** its policy after each step.

Greedy Policy Improvement

A **greedy policy** always picks the action that looks best at each step:

$$\pi^{\text{greedy}}(s) \leftarrow \arg\max_{a \in A} Q(s,a) \qquad \text{(ties broken at random)}$$

Greedy Policy Improvement

---

procedure $\textsc{GreedyPolicyImprovement}$(MDP M; Discount factor γ; Step size α)
    for each $(s,a) \in S \times A$ **do**
        $Q(s,a) \leftarrow 0$

    for each episode **do**
        $s \leftarrow s_0$ $\quad\triangleright$ s is current state, s_0 is fixed start state of M
        $a \leftarrow \pi^{\text{greedy}}(s)$ $\quad\triangleright$ a is next action to take, chosen greedily
        while episode has not ended **do**
            Execute action a
            Observe next state s' and one-step reward $r = R(s,a,s')$
            $a' \leftarrow \pi^{\text{greedy}}(s')$ $\quad\triangleright$ a' is what the agent will do next
            $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
            $s \leftarrow s'$, $a \leftarrow a'$ $\quad\triangleright$ Update current state and next action

    return learned policy π, with actions chosen greedily for each state

---

Almost-Greedy Policies

The **greedy policy** focuses **almost exclusively** on **exploitation** (aside from breaking ties at random).

A relatively simple way to incorporate **exploration** is to adjust the policy to be **mostly greedy**, but **not always**.

An **epsilon-greedy** (ϵ-greedy) policy $\pi^{\epsilon\text{-greedy}}$, where $\epsilon \in [0, 1]$, chooses the next action in any state s using the following process:

1. Generate a random number $R \in [0, 1]$
2. If $R \leq \epsilon$, choose an action **at random**
3. If $R > \epsilon$, choose an action **greedily** via $\arg\max_{a \in A} Q(s, a)$

By using an ϵ-**greedy policy** in the policy improvement process, the **learned** **action-values** $Q(s, a)$ can be made to converge to the **optimal action-values** $Q^*(s, a)$.

SARSA Learning

ϵ-Greedy Policy Improvement: SARSA

procedure SARSALEARNING(MDP M; γ; α; ϵ)
  for each $(s, a) \in S \times A$ **do**
    $Q(s, a) \leftarrow 0$ ▷ $Q(s, a)$ is current estimate of $Q^*(s, a)$
  for each episode **do**
    $s \leftarrow s_0$ ▷ s is current state, s_0 is fixed start state of M
    $a \leftarrow \pi^{\epsilon\text{-greedy}}(s)$ ▷ a is next action to take, chosen ϵ-greedily
    while episode has not ended **do**
      Execute action a
      Observe next state s' and one-step reward $r = R(s, a, s')$
      $a' \leftarrow \pi^{\epsilon\text{-greedy}}(s')$ ▷ a' is what the agent will do next
      $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a)\right]$
      $s \leftarrow s'$, $a \leftarrow a'$ ▷ Update current state and next action
  return learned policy π, with actions chosen greedily for each state

- This algorithm is called **SARSA** for s, a, r, s', a'
- By decreasing ϵ over time, **the agent** can stabilize the policy and ensure convergence

SARSA Learning: On-Policy Updates

ϵ-Greedy Policy Improvement: SARSA

---

procedure SARSALEARNING(MDP M; γ; α; ϵ)
  for each $(s, a) \in S \times A$ **do**
    $Q(s, a) \leftarrow 0$ ▷ $Q(s, a)$ is current estimate of $Q^*(s, a)$
  for each episode **do**
    $s \leftarrow s_0$ ▷ s is current state, s_0 is fixed start state
    $a \leftarrow \pi^{\epsilon\text{-greedy}}(s)$ ▷ a is next action to take, chosen ϵ-greedily
    while episode has not ended **do**
      Execute action a
      Observe next state s' and one-step reward $r = R(s, a, s')$
      Determine $a' \leftarrow \pi^{\epsilon\text{-greedy}}(s')$ ▷ a' is what we're going to do next
      $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a)\right]$
      $s \leftarrow s'$, $a \leftarrow a'$ ▷ Update current state and next action
  return learned policy π, with actions chosen greedily for each state

---

SARSA is an **on-policy** update method: The **value** of the next state s' is based on the next action a', which is picked **according to the policy**.

The Impact of On-Policy Updates

An **on-policy** update of $Q(s,a)$ uses the action value $Q(s',a')$ where action a' is chosen according to the ϵ-**greedy policy**:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$$

- With probability $(1-\epsilon)$, a' **is the best possible action** in state s'
- With probability ϵ, a' is a **random** action

This means that the TD(0) update of $Q(s,a)$ is based on a **potentially suboptimal** action a' in the next state:

$$Q(s',a') \leq \max_{a'' \in A} Q(s',a'').$$

Q-Learning: Off-Policy Updates

Q-Learning with ϵ-Greedy and Off-Policy Updates

procedure $\textsc{QLearning}$(MDP M; γ; α; ϵ)
  for each $(s, a) \in S \times A$ **do**
    $Q(s, a) \leftarrow 0$ $\triangleright$ $Q(s, a)$ is current estimate of $Q^*(s, a)$
  for each episode **do**
    $s \leftarrow s_0$ $\triangleright$ s is current state, s_0 is fixed start state of M
    $a \leftarrow \pi^{\epsilon\text{-greedy}}(s)$ $\triangleright$ a is next action to take, chosen ϵ-greedily
    while episode has not ended **do**
      Execute action a
      Observe next state s' and one-step reward $r = R(s, a, s')$
      $a' \leftarrow \pi^{\epsilon\text{-greedy}}(s')$ $\triangleright$ a' is what the agent will do next
      $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'' \in A} \{Q(s', a'')\} - Q(s, a)\right]$
      $s \leftarrow s'$, $a \leftarrow a'$ $\triangleright$ Update current state and next action
  return learned policy π, with actions chosen greedily for each state

Q-Learning is an **off-policy** update method.

- **The agent** still picks actions a and a' according to policy (e.g., ϵ-greedy)
- **The agent** updates $Q(s, a)$ using the value of the **optimal action** in state s', instead of the action a' that it will take (which **may be suboptimal**)