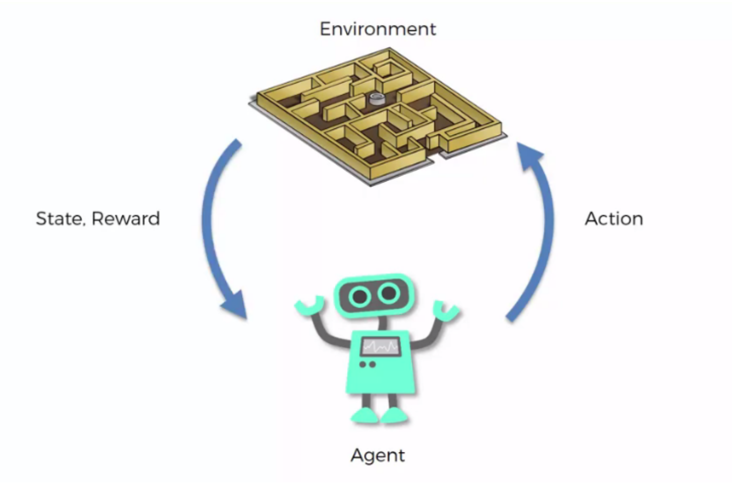
Q-Learning

Insoon Yang

Department of Electrical and Computer Engineering
Seoul National University

Recap



Review of value-based methods

MDP problem:

$$\max_{\pi \in \Pi} \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

Review of value-based methods

MDP problem:

$$\max_{\pi \in \Pi} \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

Known model:

Review of value-based methods

MDP problem:

$$\max_{\pi \in \Pi} \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

Known model:

- Policy iteration

Review of value-based methods

MDP problem:

$$\max_{\pi \in \Pi} \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

Known model:

- Policy iteration
- Value iteration

Review of value-based methods

MDP problem:

$$\max_{\pi \in \Pi} \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

Known model:

- Policy iteration
- Value iteration

Unknown model:

Review of value-based methods

MDP problem:

$$\max_{\pi \in \Pi} \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

Known model: Know reward & transition probability

- Policy iteration
- Value iteration

Unknown model: Unknow reward & transition probability

- Temporal-difference learning
- Q-learning

Review: Policy Iteration

Initialize a policy π_0;

Review: Policy Iteration

Initialize a policy π_0;

1. (Policy Evaluation) Compute the value v^{π_k} of π_k by solving the Bellman equation $v^{\pi_k} = T^{\pi_k} v^{\pi_k}$;

Review: Policy Iteration

Initialize a policy π_0;

1. (Policy Evaluation) Compute the value v^{π_k} of π_k
   by solving the Bellman equation $v^{\pi_k} = T^{\pi_k} v^{\pi_k}$;

2. (Policy Improvement) Update the policy to π_{k+1} so that
   $\pi_{k+1}(s) \in \arg\max_a \{r(s,a) + \gamma \mathbb{E}_{s'}[v^{\pi_k}(s')]\}$

Review: Policy Iteration

Initialize a policy π_0;

1. (Policy Evaluation) Compute the value v^{π_k} of π_k by solving the Bellman equation $v^{\pi_k} = T^{\pi_k} v^{\pi_k}$;
2. (Policy Improvement) Update the policy to π_{k+1} so that $\pi_{k+1}(s) \in \arg\max_a \{r(s,a) + \gamma \mathbb{E}_{s'}[v^{\pi_k}(s')]\}$
3. Set $k \leftarrow k+1$; Repeat until convergence;

Review: Policy Iteration

Initialize a policy π_0;

1. (Policy Evaluation) Compute the value v^{π_k} of π_k by solving the Bellman equation $v^{\pi_k} = T^{\pi_k} v^{\pi_k}$;
2. (Policy Improvement) Update the policy to π_{k+1} so that $\pi_{k+1}(s) \in \arg\max_a \{r(s,a) + \gamma \mathbb{E}_{s'}[v^{\pi_k}(s')]\}$
3. Set $k \leftarrow k+1$; Repeat until convergence;

- Converged policy is optimal!

Review: Policy Iteration

Initialize a policy π_0;

1. (Policy Evaluation) Compute the value v^{π_k} of π_k by solving the Bellman equation $v^{\pi_k} = T^{\pi_k} v^{\pi_k}$;
2. (Policy Improvement) Update the policy to π_{k+1} so that $\pi_{k+1}(s) \in \arg\max_a \{r(s,a) + \gamma \mathbb{E}_{s'}[v^{\pi_k}(s')]\}$
3. Set $k \leftarrow k+1$; Repeat until convergence;

- Converged policy is optimal!
- Q) Can we do something even simpler?

Review: Value Iteration

Initialize a value function v_0;

Review: Value Iteration

Initialize a value function v_0;

1. Update the value function by

$$v_{k+1} \leftarrow T v_k;$$

Review: Value Iteration

Initialize a value function v_0;

1. Update the value function by

$$v_{k+1} \leftarrow T v_k;$$

2. Set $k \leftarrow k + 1$; Repeat until convergence;

Review: Value Iteration

Initialize a value function v_0;

1. Update the value function by

$$v_{k+1} \leftarrow T v_k;$$

2. Set $k \leftarrow k + 1$; Repeat until convergence;

- Converged value function is optimal!

Review: Value Iteration

Initialize a value function v_0;

1. Update the value function by

$$v_{k+1} \leftarrow T v_k;$$

2. Set $k \leftarrow k + 1$; Repeat until convergence;

- Converged value function is optimal!
- Q) Can we do something similar even when we do not know model?

State-Action Value Functions (Q-Functions)

state & action function

- Another very useful concept in MDP and RL is the state-action value functions (often called the Q-functions).

Definition (Q-function)

The optimal Q-function $Q^*(\boldsymbol{s}, \boldsymbol{a})$ is the maximum expected return starting from state $\boldsymbol{s}$, taking action $\boldsymbol{a}$:

$$Q^*(\boldsymbol{s}, \boldsymbol{a}) := \max_{\pi} Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \max_{\pi} \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = \boldsymbol{s}, a_0 = \boldsymbol{a}\right]$$

- By definition, we have

$$v^*(\boldsymbol{s}) = \max_{\boldsymbol{a} \in A} Q^*(\boldsymbol{s}, \boldsymbol{a}).$$

Bellman Equation for Q-functions

$$
\begin{aligned}
Q^*(s,a) &= \underbrace{r(s,a)}_{\text{immediate reward}} + \gamma \underbrace{\sum_{s' \in S} p(s'|s,a) v^*(s')}_{\text{optimal value of next state}} \\
&= r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} Q^*(s',a')
\end{aligned}
$$

- Define the Bellman operator $\mathcal{T}$ for Q-functions by

$$
(\mathcal{T}Q)(\boldsymbol{s},\boldsymbol{a}) := r(\boldsymbol{s},\boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in S} p(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a}) \max_{\boldsymbol{a}' \in A} Q(\boldsymbol{s}',\boldsymbol{a}').
$$

  Then, it is a monotone contraction mapping.

- Bellman equation:

$$
Q = \mathcal{T}Q.
$$

Idea: Using Q-Function

When the model is known...

Idea: Using Q-Function

When the model is known...

Initialize a value function v_0;

Idea: Using Q-Function

When the model is known...

Initialize a value function v_0;

1. Set $Q_{k+1}(s, a) \leftarrow r(s, a) + \gamma \mathbb{E}_{s'}[v_k(s')]$;

Idea: Using Q-Function

When the model is known...

Initialize a value function v_0;

1. Set $Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{s'}[v_k(s')]$;
2. Set $v_{k+1}(s) \leftarrow \max_a Q_{k+1}(s,a)$;

Idea: Using Q-Function

When the model is known...

Initialize a value function v_0;

1. Set $Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{s'}[v_k(s')]$;
2. Set $v_{k+1}(s) \leftarrow \max_a Q_{k+1}(s,a)$;
3. Set $k \leftarrow k+1$; Repeat until convergence;

Idea: Using Q-Function

When the model is known...

Initialize a value function v_0;

1. Set $Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{s'}[v_k(s')]$;
2. Set $v_{k+1}(s) \leftarrow \max_a Q_{k+1}(s,a)$;
3. Set $k \leftarrow k+1$; Repeat until convergence;

- Converged value functions are optimal!

Idea: Using Q-Function

When the model is known...

Initialize a value function v_0;

1. Set $Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{s'}[v_k(s')]$;
2. Set $v_{k+1}(s) \leftarrow \max_a Q_{k+1}(s,a)$;
3. Set $k \leftarrow k+1$; Repeat until convergence;

- Converged value functions are optimal!
- Key observation: Step 2 is model-free!

Idea: Using Q-Function

When the model is known...

Initialize a value function v_0;

1. Set $Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{s'}[v_k(s')]$;
2. Set $v_{k+1}(s) \leftarrow \max_a Q_{k+1}(s,a)$;
3. Set $k \leftarrow k+1$; Repeat until convergence;

- Converged value functions are optimal!
- Key observation: Step 2 is model-free!
- Q) How can we perform Step 1 using samples?

Idea: Using samples to approximate transition probability (Stochastic Approximation)

When the model is unknown...

Idea: Using samples to approximate transition probability (Stochastic Approximation)

When the model is unknown...

Initialize a value function v_0;

Idea: Using samples to approximate transition probability (Stochastic Approximation)

When the model is unknown...

Initialize a value function v_0;

1. Sample (s_t, a_t, s_{t+1}, r_t) by running some policy;

Idea: Using samples to approximate transition probability (Stochastic Approximation)

When the model is unknown...

Initialize a value function v_0;

1. Sample (s_t, a_t, s_{t+1}, r_t) by running some policy;
2. Set $Q_{k+1}(s_t, a_t) \leftarrow (1-\alpha)Q_k(s_t, a_t) + \alpha\big[r_t + \gamma v_k(s_{t+1})\big]$;

Idea: Using samples to approximate transition probability (Stochastic Approximation)

When the model is unknown...

Initialize a value function v_0;

1. Sample (s_t, a_t, s_{t+1}, r_t) by running some policy;
2. Set $Q_{k+1}(s_t, a_t) \leftarrow (1-\alpha)Q_k(s_t, a_t) + \alpha\big[r_t + \gamma v_k(s_{t+1})\big]$;
3. Set $v_{k+1}(s) \leftarrow \max_a Q_{k+1}(s, a)$;

Idea: Using samples to approximate transition probability (Stochastic Approximation)

When the model is unknown...

Initialize a value function v_0;

1. Sample (s_t, a_t, s_{t+1}, r_t) by running some policy;
2. Set $Q_{k+1}(s_t, a_t) \leftarrow (1-\alpha)Q_k(s_t, a_t) + \alpha\big[r_t + \gamma v_k(s_{t+1})\big]$;
3. Set $v_{k+1}(s) \leftarrow \max_a Q_{k+1}(s, a)$;
4. Set $k \leftarrow k+1$; Repeat until convergence;

Idea: Using samples to approximate transition probability (Stochastic Approximation)

When the model is unknown...

Initialize a value function v_0;

1. Sample (s_t, a_t, s_{t+1}, r_t) by running some policy;
2. Set $Q_{k+1}(s_t, a_t) \leftarrow (1-\alpha)Q_k(s_t, a_t) + \alpha\big[r_t + \gamma v_k(s_{t+1})\big]$;
3. Set $v_{k+1}(s) \leftarrow \max_a Q_{k+1}(s, a)$;
4. Set $k \leftarrow k+1$; Repeat until convergence;

- Key observation: All steps are model-free!

Idea: Using samples to approximate transition probability (Stochastic Approximation)

When the model is unknown...

Initialize a value function v_0;

1. Sample (s_t, a_t, s_{t+1}, r_t) by running some policy;
2. Set $Q_{k+1}(s_t, a_t) \leftarrow (1-\alpha)Q_k(s_t, a_t) + \alpha\big[r_t + \gamma v_k(s_{t+1})\big]$;
3. Set $v_{k+1}(s) \leftarrow \max_a Q_{k+1}(s, a)$;
4. Set $k \leftarrow k+1$; Repeat until convergence;

- Key observation: All steps are model-free!
- Q) Can we merge Steps 2 and 3?

Q-Learning

Initialize a Q-function Q_0;

Q-Learning

Initialize a Q-function Q_0;

1. Sample (s_t, a_t, r_t, s_{t+1}) by executing some policy;

Q-Learning

Initialize a Q-function Q_0;

1. Sample (s_t, a_t, r_t, s_{t+1}) by executing some policy;
2. Set $Q_{k+1}(s_t, a_t) \leftarrow (1-\alpha)Q_k(s_t, a_t) + \alpha\big[r_t + \gamma \max_a Q_k(s_{t+1}, a)\big]$;

Q-Learning

Initialize a Q-function Q_0;

1. Sample (s_t, a_t, r_t, s_{t+1}) by executing some policy;
2. Set $Q_{k+1}(s_t, a_t) \leftarrow (1-\alpha)Q_k(s_t, a_t) + \alpha\big[r_t + \gamma \max_a Q_k(s_{t+1}, a)\big]$;
3. Set $k \leftarrow k+1$; Repeat until convergence;

Q-Learning

Initialize a Q-function Q_0;

1. Sample (s_t, a_t, r_t, s_{t+1}) by executing some policy;
2. Set $Q_{k+1}(s_t, a_t) \leftarrow (1-\alpha)Q_k(s_t, a_t) + \alpha\left[r_t + \gamma \max_a Q_k(s_{t+1}, a)\right]$;
3. Set $k \leftarrow k+1$; Repeat until convergence;

- Only updates Q-function (no need to use v_k)

Q-Learning

Initialize a Q-function Q_0;

1. Sample (s_t, a_t, r_t, s_{t+1}) by executing some policy;
2. Set $Q_{k+1}(s_t, a_t) \leftarrow (1-\alpha)Q_k(s_t, a_t) + \alpha\left[r_t + \gamma \max_a Q_k(s_{t+1}, a)\right]$;
3. Set $k \leftarrow k+1$; Repeat until convergence;

- Only updates Q-function (no need to use v_k)
- Q) How to obtain a policy from Q-function?

Q* => argmax_a(Q*), a=>pi(s)
we don't have to know transition
probability and formular of reward

VI or PI

$$\max_{\boldsymbol{a} \in A} \left[r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in S} p(\boldsymbol{s}' | \boldsymbol{s}, \boldsymbol{a}) v(\boldsymbol{s}') \right].$$

Advantages of Q-Learning

Advantages of Q-Learning

- Very simple

Advantages of Q-Learning

- Very simple
- Off-policy:
  Can use any policy to generate samples

Advantages of Q-Learning

- Very simple
- Off-policy:
  Can use any policy to generate samples
- Some useful theory:
  Converges when all (s, a)'s are visited infinitely many times

Disadvantages of Q-Learning

Disadvantages of Q-Learning

- Large-scale problems?

Disadvantages of Q-Learning

- Large-scale problems?
- Correlation between samples

Disadvantages of Q-Learning

- Large-scale problems?
- Correlation between samples
- Overestimation:
  Max-operator

Disadvantages of Q-Learning

- Large-scale problems?
- Correlation between samples
- Overestimation:
  Max-operator
- Exploration issue:
  ϵ-greedy

Approximate Q-Learning

Scalability Issue

"Curse of dimensionality"

- Rapid increase of the required computation and memory storage as the size of problems increases
- Suboptimal (approximation) methods with a reasonable balance between convenient implementation and adequate performance?

Two Approximation Approaches

1. Approximation in value space (parameters: θ)

$$v(\boldsymbol{s}) \approx v_\theta(\boldsymbol{s}) \quad \text{or} \quad Q(\boldsymbol{s}, \boldsymbol{a}) \approx Q_\theta(\boldsymbol{s}, \boldsymbol{a})$$

   Goal: Learning θ so that the approximate value function is close to the optimal one.

2. Approximation in policy space (parameters: θ)

$$\pi(\boldsymbol{s}) \approx \pi_\theta(\boldsymbol{s}) \quad \text{or} \quad \pi(\boldsymbol{a}|\boldsymbol{s}) \approx \pi_\theta(\boldsymbol{a}|\boldsymbol{s})$$

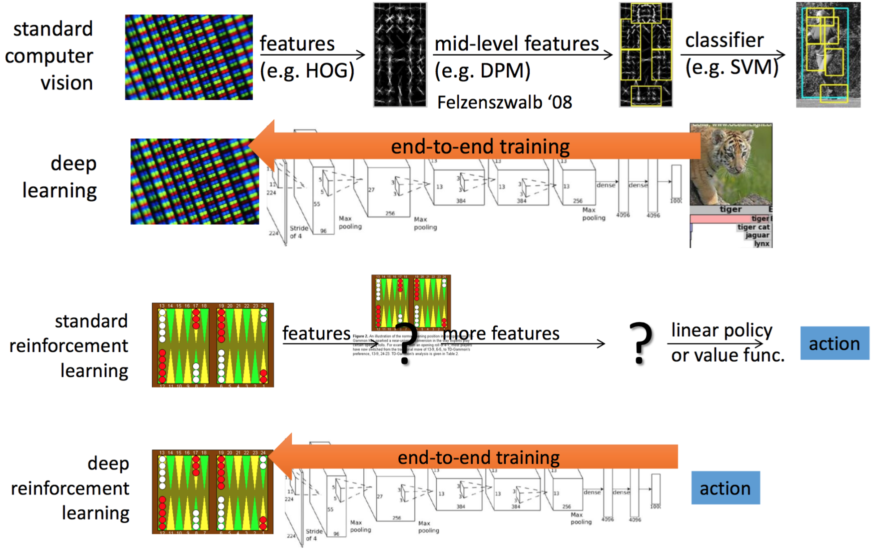
   Goal: Learning θ so that the approximate policy is close to the optimal one.

Approximation Architectures

- Linear (and nonlinear) feature-based architecture
  Two stages:
  1. feature extraction $\boldsymbol{s} \to \phi_\ell(\boldsymbol{s})$, and
  2. linear mapping $\phi_\ell(\boldsymbol{s}) \to \sum_\ell \theta_\ell \phi_\ell(\boldsymbol{s}) \approx v(\boldsymbol{s})$

- Neural network-based architecture
  End-to-end

Feature-based vs End-to-end



Example I: Piecewise Constant Approximation

- Partition the state space into $S_1, \ldots, S_m$
- Define the ℓth feature be defined by membership to S_ℓ:

$$\phi_\ell(\boldsymbol{s}) := \begin{cases} 1 & \text{if } \boldsymbol{s} \in S_\ell \\ 0 & \text{if } \boldsymbol{s} \notin S_\ell. \end{cases}$$

- Consider the architecture:

$$v_\theta(\boldsymbol{s}) := \sum_{\ell=1}^{m} \theta_\ell \phi_\ell(\boldsymbol{s})$$

Example II: Polynomial Approximation

- Suppose $S := \{\boldsymbol{s}_1, \dots, \boldsymbol{s}_n\}$.
- Let

$$\phi_0(\boldsymbol{s}) = 1, \quad \phi_k(\boldsymbol{s}) = \boldsymbol{s}_k, \quad \phi_{k\ell}(\boldsymbol{s}) = \boldsymbol{s}_k \boldsymbol{s}_\ell, \quad k, \ell = 1, \dots, n.$$

- Linear architecture:

$$v_\theta(\boldsymbol{s}) := \theta_0 + \sum_{k=1}^{n} \theta_k \boldsymbol{s}_k + \sum_{k=1}^{n} \sum_{k=1}^{n} \theta_{k\ell} \boldsymbol{s}_k \boldsymbol{s}_\ell.$$

Example III: Feature Extraction from Data

- In many cases, we do not have enough prior knowledge to handcraft features.
- Suppose with some preliminary calculation using data, we have identified some suitable states $\boldsymbol{s}_\ell$ that can serve as "anchors" for the construction of Gaussian basis functions of the form

$$\phi_\ell(\boldsymbol{s}) := e^{-\frac{\|\boldsymbol{s}-\boldsymbol{s}_\ell\|^2}{2\sigma^2}}$$

General Version of Model-Free PI

- Assume a nonlinear architecture: $Q(\boldsymbol{s}, \boldsymbol{a}) \approx Q_\theta(\boldsymbol{s}, \boldsymbol{a})$

General Version of Model-Free PI

- Assume a nonlinear architecture: $Q(\boldsymbol{s}, \boldsymbol{a}) \approx Q_\theta(\boldsymbol{s}, \boldsymbol{a})$
- Idea:
  1. (Approximate policy evaluation)
     Evaluate approximate Q-function Q_θ^π of current policy π

General Version of Model-Free PI

- Assume a nonlinear architecture: $Q(\boldsymbol{s}, \boldsymbol{a}) \approx Q_\theta(\boldsymbol{s}, \boldsymbol{a})$
- Idea:
  1. (Approximate policy evaluation)
     Evaluate approximate Q-function Q_θ^π of current policy π
  2. (Policy improvement)
     Generate improved policy π':

$$\pi'(\boldsymbol{s}) \in \arg\max_{\boldsymbol{a} \in A} Q_\theta^\pi(\boldsymbol{x}, \boldsymbol{a})$$

Approximate Policy Evaluation in General Cases

- Given a pair $(\boldsymbol{s}^i, \boldsymbol{a}^i)$, using the policy π, collect M sample trajectories starting from $\boldsymbol{s}^i$ with initial action $\boldsymbol{a}^i$.
- Estimate $Q^\pi(\boldsymbol{s}^i, \boldsymbol{a}^i)$ as the sample mean y^i.
- Determine θ using a least-squares fit:

$$\theta \in \arg\min_{\theta} \sum_{i=1}^{N} (Q_\theta^\pi(\boldsymbol{s}^i, \boldsymbol{a}^i) - y^i)^2$$

Q) What's the issue in this approach?

Several Issues

- Architectural issue
- Exploration issue
- Convergence issue

Q-Learning for Policy Evaluation

1. Initialize $Q \equiv 0$; Set $t \leftarrow 0$;
2. Given state s_t in stage t, choose an *arbitrary action* a_t and simulate the system up to stage $t+1$;
3. Using the sample (s_t, a_t, r_t, s_{t+1}), update the Q-function at (s_t, a_t) as

$$Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \alpha_t \Big[r_t + \gamma Q(s_{t+1}, \pi(s_{t+1})) - Q(s_t, a_t) \Big];$$

4. Set $t \leftarrow t+1$ and go to Step 2;

- Idea: Use Q-learning for approximate policy evaluation
- Challenge: Step 3?

Value Function Approximation via Stochastic (Incremental) Gradient Descent

Approximate policy evaluation:

$$\theta \in \arg\min_{\theta} J(\theta) := \frac{1}{2}\mathbb{E}[(Q_\theta^\pi(s_t, a_t) - \underbrace{y_t}_{\text{estimate of } Q^\pi(s_t,a_t)})^2]$$

Value Function Approximation via Stochastic (Incremental) Gradient Descent

Approximate policy evaluation:

$$\theta \in \arg\min_{\theta} J(\theta) := \frac{1}{2}\mathbb{E}[(Q_\theta^\pi(s_t, a_t) - \underbrace{y_t}_{\text{estimate of } Q^\pi(s_t, a_t)})^2]$$

- Stochastic gradient using a single sample:

$$g_t := (Q_\theta^\pi(s_t, a_t) - y_t)\nabla_\theta Q_\theta^\pi(s_t, a_t)$$

Note that

$$\mathbb{E}[g_t] = \nabla J(\theta).$$

Value Function Approximation via Stochastic (Incremental) Gradient Descent

Approximate policy evaluation:

$$\theta \in \arg\min_{\theta} J(\theta) := \frac{1}{2}\mathbb{E}[(Q^{\pi}_{\theta}(s_t, a_t) - \underbrace{y_t}_{\text{estimate of } Q^{\pi}(s_t,a_t)})^2]$$

- Stochastic gradient using a single sample:

$$g_t := (Q^{\pi}_{\theta}(s_t, a_t) - y_t)\nabla_{\theta} Q^{\pi}_{\theta}(s_t, a_t)$$

  Note that

$$\mathbb{E}[g_t] = \nabla J(\theta).$$

- Update:

$$\theta \leftarrow \theta + \alpha_t (Q^{\pi}_{\theta}(s_t, a_t) - y_t)\nabla_{\theta} Q^{\pi}_{\theta}(s_t, a_t)$$

  This gives an inexact policy evaluation.

Optimistic PI with Parametric Q-Function Approximation

Idea: Using Q-learning in approximate policy evaluation step

1. Given state s_t in stage t, choose an *arbitrary action* a_t and simulate the system up to stage $t+1$ to collect

$$(s_t, a_t, r_t, s_{t+1})$$

2. (Policy improvement) Generate the action a_{t+1} as

$$a_{t+1} \in \arg\max_{\boldsymbol{a} \in A} Q_\theta(s_{t+1}, \boldsymbol{a})$$

   To enhance exploration, one can use an ϵ-greedy selection.

3. (Inexact policy evaluation) Update the parameters as

$$\theta \leftarrow \theta + \alpha_t \nabla_\theta Q_\theta(s_t, a_t)(Q_\theta(s_t, a_t) - y_t),$$

   where target y_t is a sample-based estimate of $Q_\theta(s_t, a_t)$

SARSA

- With single-step approximation, the target can be chosen as

$$y_t := r_t + \gamma Q_\theta(s_{t+1}, a_{t+1}).$$

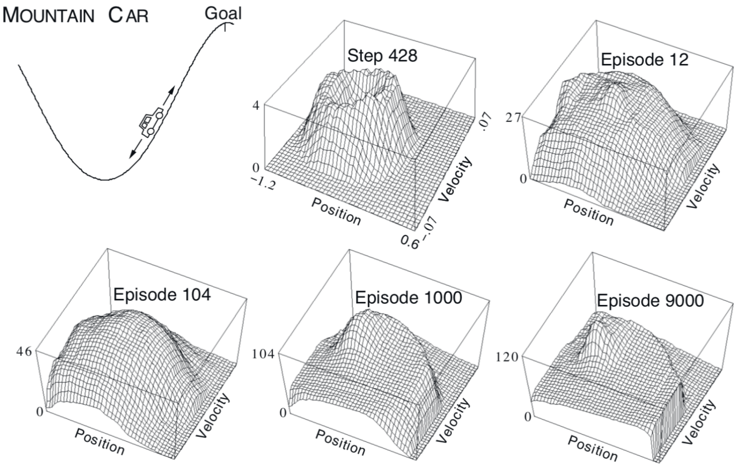
- This extreme (single-sample) optimistic PI algorithm is often called SARSA (State-Action-Reward-State-Action).
- The behavior of this algorithm is very complex: its theoretical convergence properties are unclear and there are no associated error bounds in the literature.
- The algorithm is vulnerable to bias (like TD(0))
- You should be very careful when using SARSA with function approximation although it is very convenient to implement!
- We will learn a batch-based idea using a buffer in DQN.

Example: Mountain Car

- Goal: to drive a car up a steep mountain road
- Difficulty: gravity is stronger than the car's engine – The only solution is to first move away from the goal and up the opposite slope on the left.
- Reward: -1 on all time steps until the car moves past its goal position at the top of the mountain, which ends the episode.
- Actions (acceleration): full throttle forward $(+1)$, full throttle reverse (-1), and zero throttle (0)
- Function approximation using a regular grid

Mountain Car: Cost-to-go learning results $(-\max_a Q_\theta(s,a))$

Each episode started from a random position in $[-0.6, -0.4)$ and zero velocity.



n-Step Sarsa: On-Policy Control

- With n-step approximation, the target can be chosen as

$$y_t := r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n Q_\theta(s_{t+n}, a_{t+n}).$$

- Here, the sample data are generated using the policy

$$\pi(\boldsymbol{s}) \in \arg\max_{\boldsymbol{a} \in A} Q_\theta(\boldsymbol{s}, \boldsymbol{a})$$

because y_t is a target for Q^π. So this is an on-policy method!

n-Step Sarsa: On-Policy Control

- With n-step approximation, the target can be chosen as

$$y_t := r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n Q_\theta(s_{t+n}, a_{t+n}).$$

- Here, the sample data are generated using the policy

$$\pi(\boldsymbol{s}) \in \arg\max_{\boldsymbol{a} \in A} Q_\theta(\boldsymbol{s}, \boldsymbol{a})$$

because y_t is a target for Q^π. So this is an on-policy method!

- Update the parameters as before:

$$\theta \leftarrow \theta + \alpha_t \nabla_\theta Q_\theta(s_t, a_t)(Q_\theta(s_t, a_t) - y_t)$$

- More robust than single-step off-policy Sarsa.

Off-Policy Method?

Q-Learning (tabular):

Off-Policy Method?

Q-Learning (tabular):

Initialize Q;

1. Take some action and observe (s, a, s', r);
2. Set $Q(s, a) \leftarrow (1 - \alpha) \underbrace{Q(s, a)}_{\text{old estimate}} + \alpha \underbrace{\left[r + \gamma \max_{a'} Q(s', a')\right]}_{\text{new estimate}}$;
3. Repeat until convergence;

Off-Policy Method?

Q-Learning (tabular):

Initialize Q;

1. Take some action and observe (s, a, s', r);
2. Set $Q(s,a) \leftarrow (1-\alpha) \underbrace{Q(s,a)}_{\text{old estimate}} + \alpha \underbrace{\left[r + \gamma \max_{a'} Q(s',a')\right]}_{\text{new estimate}}$;
3. Repeat until convergence;

Note:

- (s, a, s') gives information about transition $p(s'|s,a)$
- (s, a, r) gives information about reward $r(s,a)$

Approximate Q-Learning

For large-scale problems

Approximate Q-Learning

For large-scale problems

- **Parameterize Q-function:** $Q_\theta(s, a)$

Approximate Q-Learning

For large-scale problems

- **Parameterize Q-function:** $Q_\theta(s,a)$

Initialize θ;

1. Collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ using some policy;
2. For $i = 1 : N$
   - Set $\underbrace{y_i}_{\text{target}} \leftarrow \underbrace{r_i + \gamma \max_a Q_\phi(s'_i, a)}_{\text{new estimate}}$;
3. Set $\phi \leftarrow \arg\min_\phi \underbrace{\frac{1}{2}\sum_i \|Q_\phi(s_i, a_i) - y_i\|^2}_{\text{loss function}}$;
4. Repeat until sufficient improvement;

Approximate Q-Learning (stochastic gradient version)

Initialize θ;

1. Take some action and observe (s_i, a_i, s'_i, r_i);
2. Set $\underbrace{y_i}_{\text{target}} \leftarrow \underbrace{r_i + \gamma \max_a Q_\theta(s'_i, a)}_{\text{new estimate}}$;
3. Set $\theta \leftarrow \theta - \underbrace{\alpha}_{\text{stepsize}} \underbrace{\nabla_\theta Q_\theta (Q_\theta(s_i, a_i) - y_i)}_{\text{stochastic gradient}}$;

   완전히 converge 안해도 됨
4. Repeat until sufficient improvement;

- This is an off-policy algorithm
- Can use mini-batch and experience replay (DQN)

What's wrong with approximate Q-learning?

What's wrong with approximate Q-learning?

- Unclear how to parameterize Q-function

What's wrong with approximate Q-learning?

- Unclear how to parameterize Q-function
  (ex) linear parametrization (unclear feature selection):

$$Q_\phi(s,a) := \underbrace{\beta(s,a)^\top}_{\text{feature}} \phi$$

What's wrong with approximate Q-learning?

- Unclear how to parameterize Q-function
  (ex) linear parametrization (unclear feature selection):

$$Q_\phi(s,a) := \underbrace{\beta(s,a)^\top}_{\text{feature}} \phi$$

- Correlation between samples
  $((s_t, a_t, s_{t+1}, r_t)$ and $(s_{t+1}, a_{t+1}, s_{t+2}, r_{t+1}))$

What's wrong with approximate Q-learning?

- Unclear how to parameterize Q-function
  (ex) linear parametrization (unclear feature selection):

$$Q_\phi(s,a) := \underbrace{\beta(s,a)^\top}_{\text{feature}} \phi$$

- Correlation between samples
  $((s_t, a_t, s_{t+1}, r_t)$ and $(s_{t+1}, a_{t+1}, s_{t+2}, r_{t+1}))$

- Poor convergence property