

Lecture 9: Value Function Methods

Zhi Wang

Department of Control Science and Intelligent Engineering
Nanjing University

2025/04/14

Table of Contents

- 1 Omit policy gradient from actor-critic
- 2 Fitted value iteration
- 3 Fitted Q-iteration
- 4 Theories of value function methods

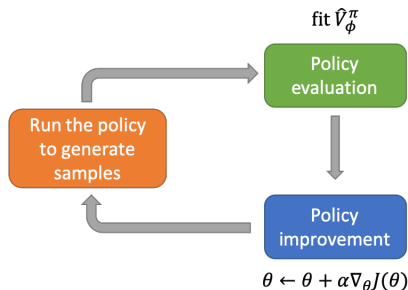
Contents and Goals

- What if we just use a critic, without an actor?
- Extracting a policy from a value function
- The fitted value iteration, fitted Q-iteration algorithms
- Goals
 - Understand how value functions give rise to policies
 - Understand the Q-learning with function approximation algorithm
 - Understand practical considerations for Q-learning

Review: the actor-critic algorithm

- Loop:

1. sample $\{s_i, a_i, r_i, s'_i\}$ from $\pi_\theta(a|s)$ (run it on the robot)
2. **policy evaluation**: fit $\hat{V}_\phi^\pi(s)$ to sampled reward sums
3. evaluate $\hat{A}^\pi(s_i, a_i) = r_i + \gamma \hat{V}_\phi^\pi(s'_i) - \hat{V}_\phi^\pi(s_i)$
4. **policy improvement**: $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i) \hat{A}^\pi(s_i, a_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

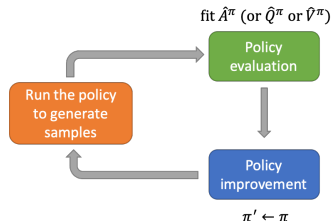


Can we omit policy gradient completely?

- The advantage function $A^\pi(s_t, a_t)$:
 - how much better is a_t than the average action according to π
- $\arg \max_{a_t} A^\pi(s_t, a_t)$: best action from s_t , if we then follow π
 - forget about approximating policies directly
 - just **derive policies from value functions**

- Is π' better than π , i.e., $\pi' \geq \pi$?
 - **The policy improvement theorem!**

$$\pi'(a_t|s_t) = \begin{cases} 1 & \text{if } a_t = \arg \max_{a_t} A^\pi(s_t, a_t) \\ 0 & \text{otherwise} \end{cases}$$



Review: Policy Improvement Theorem

- Let π and π' be any pair of deterministic policies such that,

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s), \quad \forall s \in \mathcal{S}.$$

Then the policy π' must be as good as, or better than, π .

Review: Policy improvement theorem

$$\begin{aligned} V^\pi(s) &\leq Q^\pi(s, \pi'(s)) \\ &= \mathbb{E}[R_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma Q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma V^\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] | S_t = s] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 V^\pi(S_{t+2}) | S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V^\pi(S_{t+3}) | S_t = s] \\ &\leq \dots \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots | S_t = s] \\ &= V^{\pi'}(s) \end{aligned}$$

Review: Policy improvement theorem

- Consider the new **greedy** policy, π' , selecting at each state the action that appears best according to $Q^\pi(s, a)$

$$\begin{aligned}\pi'(s) &= \arg \max_a Q^\pi(s, a) \\ &= \arg \max_a \mathbb{E}[R_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \arg \max_a \sum_{s', a} p(s', r | s, a) [r + \gamma V^\pi(s')]\end{aligned}$$

- The process of making a new policy that improves on an original policy, by making greedy w.r.t. the value function of the original policy, is called **policy improvement**
 - The greedy policy meets the conditions of the policy improvement theorem

The Generalized Policy Iteration framework

- High-level idea: **policy iteration** algorithm. Loop:

- Policy evaluation**: evaluate $A^\pi(s, a)$
- Policy improvement**: set $\pi \leftarrow \pi'$

$$\pi'(a_t|s_t) = \begin{cases} 1 & \text{if } a_t = \arg \max_{a_t} A^\pi(s_t, a_t) \\ 0 & \text{otherwise} \end{cases}$$

- Question**: How to evaluate A^π ?

$$A^\pi(s, a) = r(s, a) + \gamma V^\pi(s') - V^\pi(s)$$

- As we did in actor-critic algorithms, just evaluate V^π !

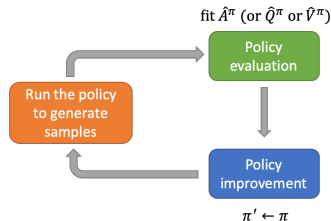
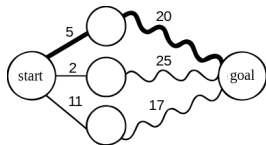


Table of Contents

- 1 Omit policy gradient from actor-critic
- 2 Fitted value iteration
 - Review: Policy iteration, Value iteration
 - Fitted value iteration with function approximation
- 3 Fitted Q-iteration
- 4 Theories of value function methods

Review: Dynamic Programming (DP)

- It refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a **recursive** manner.



- Finding the shortest path in a graph using optimal substructure
- A straight line: a single edge, a wavy line: a shortest path
- The bold line: the overall shortest path from start to goal

Review: Dynamic Programming (DP)

- A collection of algorithms that can be used to compute optimal policies given a perfect model of the environment (MDP)
 - Of limited utility in RL both because of their assumption of a perfect model and because of their great computational expense
 - Important theoretically, provide an essential foundation for the understanding of RL methods
 - **RL methods can be viewed as attempts to achieve much the same effect as DP**, only with less computation and without assuming a perfect model of the environment

Review: Policy evaluation in DP

- Compute the state-value function V^π for an arbitrary policy π

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) [r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) [r + \gamma V^\pi(s')] \end{aligned}$$

0.5	0.8	0.3	0.4
0.4	0.3	0.8	0.5
0.7	0.6	0.6	0.7
0.9	0.5	0.1	0.2

- 16 states, 4 actions per state
- can store full $V^\pi(s)$ in a table
- iterative sweeping over the state space

Review: Policy improvement in DP

- Consider the new **greedy** policy, π' , selecting at each state the action that appears best according to $Q^\pi(s, a)$

$$\begin{aligned}\pi'(s) &= \arg \max_a Q^\pi(s, a) \\ &= \arg \max_a \mathbb{E}[R_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \arg \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V^\pi(s')]\end{aligned}$$

- The process of making a new policy that improves on an original policy, by making greedy w.r.t. the value function of the original policy, is called **policy improvement**
 - The greedy policy meets the conditions of the policy improvement theorem

Review: Policy iteration

- Using policy improvement theorem, we can obtain a sequence of monotonically improving policies and value functions

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

- This process is guaranteed to converge to an optimal policy and optimal value function in a finite number of iterations
 - Each policy is guaranteed to be a strictly improvement over the previous one unless it is already optimal
 - A finite MDP has only a finite number of policies

Review: Policy iteration algorithm

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:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s', r | s, \pi(s)) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

old-action $\leftarrow \pi(s)$

$\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$

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

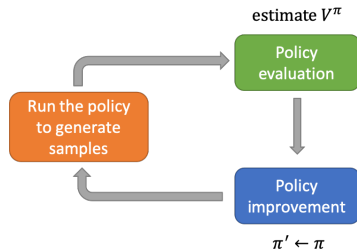
Dynamic programming with policy iteration

- Policy iteration. Loop:

1. **Policy evaluation:** evaluate $V^\pi(s)$
2. **Policy improvement:** set $\pi \leftarrow \pi'$

$$\pi'(a|s) = \begin{cases} 1 & \text{if } a = \arg \max_a Q^\pi(s, a) \\ 0 & \text{otherwise} \end{cases}$$

$$Q^\pi(s, a) = \sum_{s', r} p(s', r|s, a)[r + \gamma V^\pi(s')]$$



0.5	0.8	0.3	0.4
0.4	0.3	0.8	0.5
0.7	0.6	0.6	0.7
0.9	0.5	0.1	0.2

- 16 states, 4 actions per state
- can store full $V^\pi(s)$ in a table
- iterative sweeping over the state space

- In policy iteration, stop policy evaluation after just one sweep

$$V_{k+1}(s) = \sum_{s',r} p(s', r|s, \pi_k(s)) [r + \gamma V_k(s')]$$

$$\pi_{k+1}(s) = \arg \max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V_{k+1}(s')]$$

- Combine into one operation, called **value iteration** algorithm

$$V_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V_k(s')]$$

- For arbitrary V_0 , the sequence $\{V_k\}$ converges to V^* under the same conditions that guarantee the existence of V^*

Review: Value iteration

- Bellman optimality equation

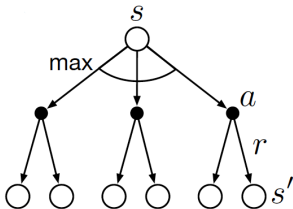
$$V^*(s) = \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V^*(s')]$$

- Value iteration

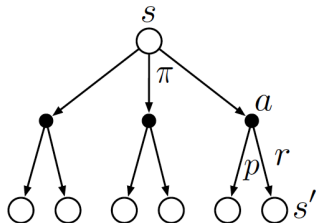
$$V_{k+1}(s) = \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V_k(s')]$$

- Turn Bellman optimality equation into an update rule
- Directly approximate the optimal state-value function, V^*

Review: Value iteration vs. Policy evaluation



Backup diagram for
value iteration



Backup diagram for
policy evaluation

Review: Value iteration algorithm

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 $V(\text{terminal}) = 0$

Loop:

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

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

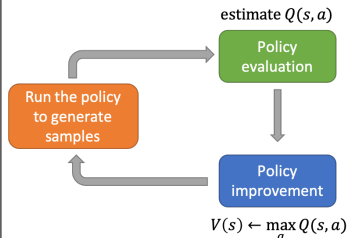
- One sweep = one sweep of policy evaluation + one sweep of policy improvement

Dynamic programming with value iteration

- Value iteration. Loop:

1. Policy evaluation: evaluate $Q(s, a)$
2. **Implicit** policy improvement:
set $V(s) \leftarrow \max_a Q(s, a)$

$$Q(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma V(s')]$$



- Skip the policy and compute values directly!

$$V(s) \leftarrow \max_a Q(s, a) \quad \text{equivalent} \quad \Rightarrow \quad \pi'(a|s) = \begin{cases} 1 & \text{if } a = \arg \max_a Q(s, a) \\ 0 & \text{otherwise} \end{cases}$$

Review: For large/continuous state/action spaces

- **Curse of dimensionality:** Computational requirements grow exponentially with the number of state variables
 - Theoretically, all state-action pairs need to be visited infinite times to guarantee an optimal policy
 - In many practical tasks, almost every state encountered will never have been seen before
- **Generalization:** How can experience with a limited subset of the state space be usefully generalized to produce a good **approximation** over a much larger subset?

Review: Curse of dimensionality

0.5	0.8	0.3	0.4
0.4	0.3	0.8	0.5
0.7	0.6	0.6	0.7
0.9	0.5	0.1	0.2

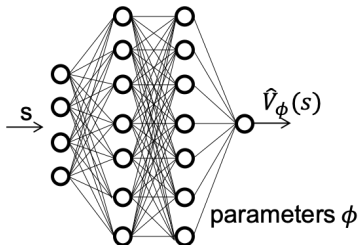
- In discrete case, represent $V(s)$ as a table
 - 16 states, 4 actions per state
 - can store full $V(s)$ in a table
 - iterative sweeping over the state space



- An image
 - $|\mathcal{S}| = (255^3)^{200 \times 200}$
 - more than atoms in the universe
 - can we store such a large table?

Review: Function approximation

- It takes examples from a desired function (e.g., a value function) and attempts to generalize from them to construct an approximation to the entire function
 - Linear function approximation: $V(s) = \sum_i \phi_i(s)w_i$
 - Neural network approximation: $V(s) = V_\phi(s)$

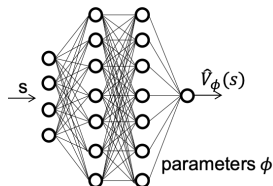


Review: Function approximation

- Function approximation is an instance of **supervised learning**, the primary topic studied in machine learning, artificial neural networks, pattern recognition, and statistical curve fitting
 - In theory, any of the methods studied in these fields can be used in the role of function approximator within RL algorithms
 - RL with function approximation involves a number of **new issues** that do not normally arise in conventional supervised learning, e.g., non-stationarity, bootstrapping, and delayed targets

Fitted value iteration with function approximation

- How do we represent $V(s)$?
 - Discrete: big table, one entry for each s
 - Continuous: neural network function
 $V_\phi : \mathcal{S} \rightarrow \mathbb{R}$



- Fitted value iteration. Loop:
 1. set target value: $y \leftarrow \max_a Q(s, a)$
 2. update neural net
 $\phi \leftarrow \arg \min_\phi \|V_\phi(s) - y\|^2$

$$Q(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma V_\phi(s')]$$

- Loss function: Mean squared error (MSE)

$$\mathcal{L}(\phi) = \frac{1}{2} \|V_\phi(s) - \max_a Q(s, a)\|^2$$

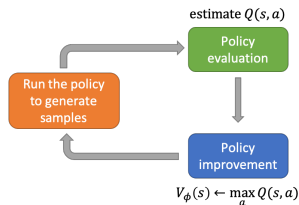


Table of Contents

- 1 Omit policy gradient from actor-critic
- 2 Fitted value iteration
- 3 Fitted Q-iteration**
- 4 Theories of value function methods

Review: Determine policies from value functions

- For $V^*(s)$: a one-step search
 - Actions that appear best after one-step search will be optimal actions
 - $a^* = \arg \max_a Q^*(s, a) = \arg \max_a \sum_{s', r} p(s', r | s, a)(r + \gamma V^*(s'))$
- For $Q^*(s, a)$: no need to do a one-step-ahead search
 - $a^* = \arg \max_a Q^*(s, a)$
 - The optimal action-value function allows optimal actions to be selected without having to know anything about possible successor states and their values, i.e., **without having to know anything about the environment's dynamics**

Review: Value iteration and Q-learning

- From prediction problems to control problems
- From model-based to model-free
- From state-value functions $V(s)$ to action-value functions $Q(s, a)$

$$V_{k+1}(s) \leftarrow \max_a \sum_{s', r} p(s', r|s, a) [r + \gamma V_k(s')]$$
$$Q_{k+1}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} Q_k(s', a')$$

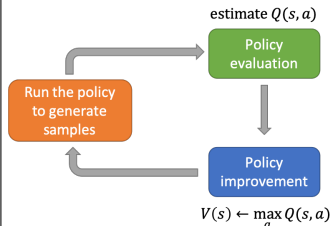
- Directly approximate the optimal state-value function, V^*
- **Need to know outcomes for different actions**
- Directly approximate the optimal action-value function, Q^*
- Fit the value function using only samples (s, a, r, s')

Value iteration \rightarrow Fitted value iteration

- Value iteration. Loop:

1. Policy evaluation: evaluate $Q(s, a)$
2. **Implicit** policy improvement:
set $V(s) \leftarrow \max_a Q(s, a)$

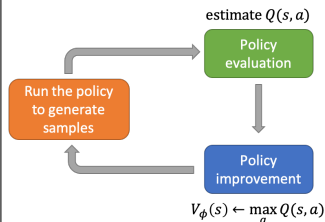
$$Q(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma V(s')]$$



- Fitted value iteration. Loop:

1. set target value: $y \leftarrow \max_a Q(s, a)$
2. update neural net
 $\phi \leftarrow \arg \min_{\phi} ||V_{\phi}(s) - y||^2$

$$Q(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma V_{\phi}(s')]$$



What if we don't know the transition dynamics?

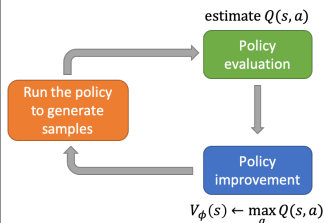
- Fitted value iteration. Loop:

1. set target value: $y \leftarrow \max_a Q(s, a)$

2. update neural net

$$\phi \leftarrow \arg \min_{\phi} ||V_{\phi}(s) - y||^2$$

$$Q(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma V_{\phi}(s')]$$



- Need to know outcomes for different actions!
- Question:** Can we extend the idea of fitted value iteration to a model-free learner?
 - The most popular model-free algorithm: Q-learning
 - Fitted value iteration \rightarrow fitted Q-learning?

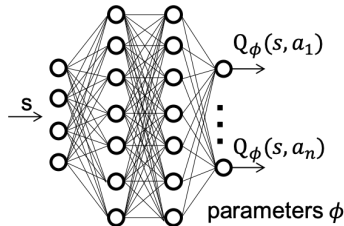
Q-learning \rightarrow Fitted Q-iteration (FQI)

- Q-learning. Loop:

1. Policy evaluation: evaluate $Q_k(s, a)$
2. Implicit policy improvement, set:
$$Q_{k+1}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} Q_k(s', a')$$

- Fitted Q-iteration. Loop:

1. set target value:
$$y \leftarrow r(s, a) + \gamma \max_{a'} Q_\phi(s', a')$$
2. update neural net
$$\phi \leftarrow \arg \min_{\phi} ||Q_\phi(s, a) - y||^2$$



Q-learning \rightarrow Fitted Q-iteration (FQI)

- Q-learning. Loop:

1. Policy evaluation: evaluate $Q_k(s, a)$

2. Implicit policy improvement, set:

$$Q_{k+1}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} Q_k(s', a')$$

- Temporal difference error (Bellman residual) for Q-learning:

$$\delta = r(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)$$

- Fitted Q-iteration. Loop:

1. set target value: $y \leftarrow r(s, a) + \gamma \max_{a'} Q_\phi(s', a')$

2. update neural net $\phi \leftarrow \arg \min_{\phi} \|Q_\phi(s, a) - y\|^2$

- Loss function (Bellman residual) for fitted Q-iteration:

$$\mathcal{L}(\phi) = \|r(s, a) + \gamma \max_{a'} Q_\phi(s', a') - Q_\phi(s, a)\|^2$$

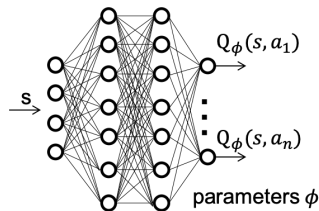
Full fitted Q-iteration algorithm

- Loop:

1. collect dataset $\{(s_i, a_i, r_i, s'_i)\}$ using behavior policy π
loop for K iterations:
 2. set $y_i \leftarrow r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$
 3. set $\phi \leftarrow \arg \min_\phi \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$

- Hyper-parameters:

- dataset size N
- behavior policy π for data collection
- iterations K
- gradient steps S for minimizing $\mathcal{L}(\phi)$



Review: On-policy vs. Off-policy

Target policy $\pi(a s)$	Behavior policy $b(a s)$
To be evaluated or improved Make decisions finally	To explore to generate data Make decisions in training phase

- **On-policy** methods: $\pi(a|s) = b(a|s)$
 - Evaluate or improve the policy that is used to make decisions during training
 - e.g., SARSA
- **Off-policy** methods: $\pi(a|s) \neq b(a|s)$
 - Evaluate or improve a policy different from that used to generate the data
 - Separate exploration from control
 - e.g., Q-learning

Review: Q-learning vs. SARSA

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

- Q-learning approximates the optimal action-value function for an optimal policy, $Q \approx Q_* = Q_{\pi_*}$
 - The target policy is greedy w.r.t Q , $\pi(a|s) = \arg \max_a Q(s, a)$
 - The behavior policy can be others, e.g., $b(a|s) = \varepsilon$ -greedy
-

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

- SARSA approximates the action-value function for the behavior policy, $Q \approx Q_\pi = Q_b$
 - The target and the behavior policy are the same, e.g., $\pi(a|s) = b(a|s) = \varepsilon$ -greedy

Why FQI is off-policy?

- Loop:

1. collect dataset $\{(s_i, a_i, r_i, s'_i)\}$ using behavior policy π

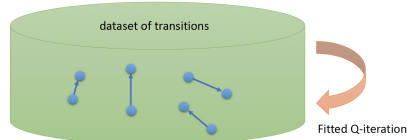
loop for K iterations:

2. set $y_i \leftarrow r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$

3. set $\phi \leftarrow \arg \min_\phi \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$

- The target greedy policy:

$$\pi(a|s) = \begin{cases} 1 & \text{if } a = \arg \max_a Q_\phi(s, a) \\ 0 & \text{otherwise} \end{cases}$$



What is FQI optimizing?

- Loop:

1. collect dataset $\{(s_i, a_i, r_i, s'_i)\}$ using behavior policy π

loop for K iterations:

2. set $y_i \leftarrow r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$

3. set $\phi \leftarrow \arg \min_\phi \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$

- Loss function (Bellman residual) for fitted Q-iteration:

$$\mathcal{L}(\phi) = \|r(s, a) + \gamma \max_{a'} Q_\phi(s', a') - Q_\phi(s, a)\|^2$$

- If $\mathcal{L}(\phi) = 0$, then we have **Bellman optimality equation**:

$$Q_{\phi^*}(s, a) = r(s, a) + \gamma \max_{a'} Q_{\phi^*}(s', a')$$

Exploration strategies - Behavior policy

- Loop:

1. collect dataset $\{(s_i, a_i, r_i, s'_i)\}$ using some behavior policy $b(a|s)$
loop for K iterations:
 2. set $y_i \leftarrow r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$
 3. set $\phi \leftarrow \arg \min_\phi \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$

- The target policy is the greedy policy w.r.t. $Q_\phi(s, a)$:

$$\pi(a|s) = \begin{cases} 1 & \text{if } a = \arg \max_a Q_\phi(s, a) \\ 0 & \text{otherwise} \end{cases}$$

- **Question:** Is it a good idea to use the target greedy policy as the behavior policy to collect samples?

Exploitation-exploration dilemma

- ϵ -greedy strategy:

$$b(a_t|s_t) = \begin{cases} 1 - \epsilon & \text{if } a_t = \arg \max_{a_t} Q_\phi(s_t, a_t) \\ \epsilon / (|\mathcal{A}| - 1) & \text{otherwise} \end{cases}$$

- $1 - \epsilon$: the exploitation part
 - $\epsilon / (|\mathcal{A}| - 1)$: the exploration part
 - **balance/trade-off** between exploitation and exploration: decrease ϵ as the learning proceeds
- Boltzmann strategy, softmax strategy:

$$b(a_t|s_t) = \frac{e^{Q(s_t, a_t)/\tau}}{\sum_a e^{Q(s_t, a)/\tau}}$$

- τ : temperature parameter
- **balance/trade-off** between exploitation and exploration: decrease τ as the learning proceeds

Batch-mode (offline) vs. online

- Batch-model (offline) algorithms
 - Collect a batch of samples using some policy
 - Fit the state- or action-value function iteratively
- Online algorithms
 - Take some action to collect one sample
 - Fit the value function
 - Iteratively alternate the above two steps

Online Q-learning algorithms

- Batch-mode fitted Q-iteration algorithm. Loop:
 1. collect dataset $\{(s_i, a_i, r_i, s'_i)\}$ using behavior policy π
loop for K iterations:
 2. set $y_i \leftarrow r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$
 3. set $\phi \leftarrow \arg \min_\phi \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$

- Online fitted Q-iteration algorithm. Loop:
 1. observe one sample (s_i, a_i, r_i, s'_i) using behavior policy π
 2. set $y_i \leftarrow r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$
 3. set $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(s_i, a_i)}{d\phi} (Q_\phi(s_i, a_i) - y_i)$

Looking back from DRL

- Tabular RL algorithms
 - R. S. Sutton and A. G. Barto, **Reinforcement Learning: An Introduction**, 2nd Edition.
- RL with linear function approximation
 - M. G. Lagoudakis and R. Parr, **Least-Squares Policy Iteration**, *Journal of Machine Learning Research*, 2003.
- RL with nonlinear function approximation, e.g., neural network
 - J. A. Boyan, et al., **Generalization in reinforcement learning: Safely approximating the value function**, NIPS 1995.
 - R. S. Sutton, et al., **Policy gradient methods for reinforcement learning with function approximation**, NIPS 2000.
 - Martin Riedmiller, **Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method**, ECML 2005.
- RL with deep neural networks
 - **Human-level performance, the milestone of artificial intelligence**

- Value-based methods
 - Don't learn a policy explicitly
 - Just learn state- or action-value function
- If we have value function, we have a policy
- Fitted value iteration
- Fitted Q-iteration
 - Batch mode, off-policy method

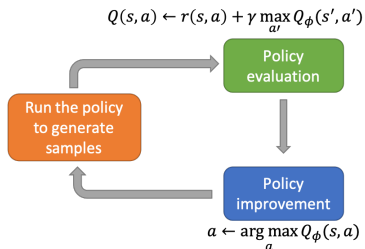


Table of Contents

- 1 Omit policy gradient from actor-critic
- 2 Fitted value iteration
- 3 Fitted Q-iteration
- 4 Theories of value function methods

Value iteration theory in tabular cases

- Bellman optimality equation

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

- Value iteration

$$V_{k+1}(s) = \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V_k(s')]$$

- Turn Bellman optimality equation into an update rule
- Directly approximate the optimal state-value function, V^*
- For arbitrary V_0 , the sequence $\{V_k\}$ converges to V^* under the same conditions that guarantee the existence of V^*

- Value iteration algorithm. Loop:

1. Policy evaluation: evaluate $Q(s, a)$
2. Implicit policy improvement: set $V(s) \leftarrow \max_a Q(s, a)$

$$Q(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma V(s')]$$

- Define a Bellman operator $\mathcal{B} : \mathcal{B}V = \max_a (r + \gamma \mathcal{T}_a V)$
 - \mathcal{T} : state transition function, $\mathcal{T}_{a,i,j} = p(s' = i | s = j, a)$
- V^* is a **fixed point** of \mathcal{B} , $V^* = \mathcal{B}V^*$, **Bellman optimality equation**:

$$V^*(s) = \sum_{s', r} p(s', r | s, a) [r + \gamma V^*(s')]$$

- always exists, is always unique, always corresponds to the optimal policy
- ...but will we reach it?

Bellman operator is a contraction

- V^* is a fixed point of \mathcal{B} , i.e., $V^* = \mathcal{B}V^*$, Bellman optimality equation:

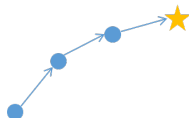
$$V^*(s) = \sum_{s', r} p(s', r | s, a) [r + \gamma V^*(s')]$$

- We can prove that value iteration reaches V^* since \mathcal{B} is a **contraction**

- For any V and \bar{V} , we have $\|\mathcal{B}V - \mathcal{B}\bar{V}\|_\infty \leq \gamma \|V - \bar{V}\|_\infty$
- The gap always gets smaller by $\gamma \in (0, 1)$

- Choose V^* as \bar{V} , note that $\mathcal{B}V^* = V^*$, we have:

$$\|\mathcal{B}V - V^*\|_\infty \leq \gamma \|V - V^*\|_\infty$$



Norm (mathematics)

- A norm is a function from a real or complex vector space to the nonnegative real numbers
 - behave in certain ways like the distance from the origin: it commutes with scaling, obeys a form of the triangle inequality, and is zero only at the origin
 - the length or size of a vector in a certain space
- p -norm: $\|x\|_p = (\sum_{i=1}^n |x_i|^p)^{(1/p)}$
 - l_1 norm: $\|x\|_1 = \sum_i |x_i|$
 - l_2 norm, the Euclidean norm: $\|x\|_2 = \sqrt{x_1^2 + \dots + x_n^2}$
 - l_∞ norm, the infinity norm: $\|x\|_\infty = \max_i |x_i|$

Non-tabular value function learning

- Define a new operator Π :

$$\Pi V = \arg \min_{V' \in \Omega} \|V'(s) - V(s)\|^2$$

- Π is a **projection** onto Ω in terms of l_2 norm

- Fitted value iteration. Loop:

1. \mathcal{B} operator: set $y \leftarrow \max_a \sum_{s', r} p(s', r | s, a)(r + \gamma V_\phi(s'))$
2. Π operator: set $\phi \leftarrow \arg \min_\phi \|V_\phi(s) - y\|^2$

$$V' \leftarrow \arg \min_{V' \in \Omega} \|V'(s) - (\mathcal{B}V)(s)\|^2$$

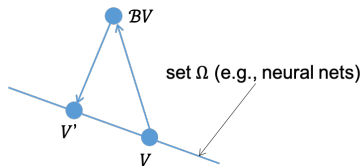
Non-tabular value function learning

- Fitted value iteration. Loop:

- \mathcal{B} operator: set $y \leftarrow \max_a \sum_{s',r} p(s',r|s,a)(r + \gamma V_\phi(s'))$
- Π operator: set $\phi \leftarrow \arg \min_{\phi} \|V_\phi(s) - y\|^2$

$$V' \leftarrow \arg \min_{V' \in \Omega} \|V'(s) - (\mathcal{B}V)(s)\|^2$$

- $(\mathcal{B}V)(s)$: updated value function
- Ω : all value functions represented by, e.g., neural nets
- Π is a **projection** onto Ω in terms of l_2 norm



Non-tabular value function learning

- fitted value iteration using \mathcal{B} and Π . Loop:

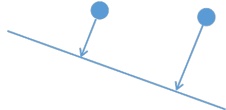
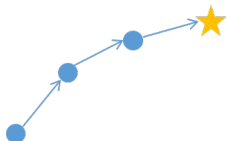
1. $V \leftarrow \Pi \mathcal{B} V$

- \mathcal{B} is a contraction w.r.t. l_∞ norm
(“max” norm)

$$\|\mathcal{B}V - \mathcal{B}\bar{V}\|_\infty \leq \gamma \|V - \bar{V}\|_\infty$$

- Π is a contraction w.r.t. l_2 norm
(Euclidean distance)

$$\|\Pi V - \Pi \hat{V}\|^2 \leq \|V - \bar{V}\|^2$$



Non-tabular value function learning

- \mathcal{B} is a contraction w.r.t. l_∞ -norm (“max” norm)

$$\|\mathcal{B}V - \mathcal{B}\bar{V}\|_\infty \leq \gamma \|V - \bar{V}\|_\infty$$

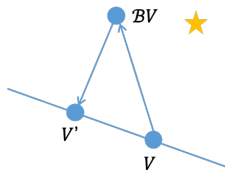
- Π is a contraction w.r.t. l_2 norm (Euclidean distance)

$$\|\Pi V - \Pi \hat{V}\|^2 \leq \|V - \bar{V}\|^2$$

- But... $\Pi\mathcal{B}$ is not contraction of any kind!

- Conclusions:

- value iteration converges (tabular case)
- fitted value iteration does **not** converge
- not in general
- often not in practice



What about fitted Q-iteration?

- Define a Bellman operator $\mathcal{B} : \mathcal{B}Q = r + \gamma \mathcal{T} \max_a Q$
- Define an operator $\Pi : \Pi Q = \arg \min_{Q' \in \Omega} \|Q'(s, a) - Q(s, a)\|^2$

- Fitted Q-iteration. Loop:

1. \mathcal{B} operator: set

$$y \leftarrow r(s, a) + \gamma \max_{a'} Q_\phi(s', a')$$

2. Π operator: set

$$\phi \leftarrow \arg \min_{\phi} \|Q_\phi(s, a) - y\|^2$$

- Using \mathcal{B} and Π . Loop:

1. $Q \leftarrow \Pi \mathcal{B} Q$

- Conclusions:

- \mathcal{B} is a contraction w.r.t. l_∞ norm (“max” norm)
- Π is a contraction w.r.t. l_2 norm (Euclidean distance)
- But... $\Pi \mathcal{B}$ is not contraction of any kind!

Fitted Q-iteration – Regression, but not gradient descent

- Online fitted Q-iteration algorithm. Loop:
 1. observe one sample (s_i, a_i, r_i, s'_i) using behavior policy π
 2. set $y_i \leftarrow r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$
 3. set $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(s_i, a_i)}{d\phi} (Q_\phi(s_i, a_i) - y_i)$

- isn't this just gradient descent? that converges, right?
- Fitted Q-iteration is not gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi(s_i, a_i)}{d\phi} \left(Q_\phi(s_i, a_i) - \underbrace{\left(r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i) \right)}_{\text{no gradient through target value!}} \right)$$

Value function learning for actor-critic

- Batch actor-critic algorithm. Loop:
 1. sample $\{s_i, a_i, r(s_i, a_i), s'_i\}$ from $\pi_\theta(a|s)$ (run it on the robot)
 2. **policy evaluation**: fit $\hat{V}_\phi^\pi(s)$ to sampled reward sums
 3. evaluate $\hat{A}^\pi(s_i, a_i) = r(s_i, a_i) + \gamma \hat{V}_\phi^\pi(s'_i) - \hat{V}_\phi^\pi(s_i)$
 4. policy improvement: $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i) \hat{A}^\pi(s_i, a_i)$
 5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
- An aside regarding terminology
 - V^π : value function for policy π
this is what the critic (policy evaluation) does
 - V^* : value function for optimal policy π^*
this is what value iteration does

Value function learning for actor-critic

- Batch actor-critic algorithm. Loop:

1. sample $\{s_i, a_i, r(s_i, a_i), s'_i\}$ from $\pi_\theta(a|s)$ (run it on the robot)
2. **policy evaluation**: fit $\hat{V}_\phi^\pi(s)$ to sampled reward sums
3. evaluate $\hat{A}^\pi(s_i, a_i) = r(s_i, a_i) + \gamma \hat{V}_\phi^\pi(s'_i) - \hat{V}_\phi^\pi(s_i)$
4. policy improvement: $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i) \hat{A}^\pi(s_i, a_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

- Define a Bellman operator $\mathcal{B} : \mathcal{B}V = r + \gamma \mathcal{T}V$

- l_∞ contraction, without the max operator; $y_i = r(s_i, a_i) + \gamma V_\phi^\pi(s'_i)$

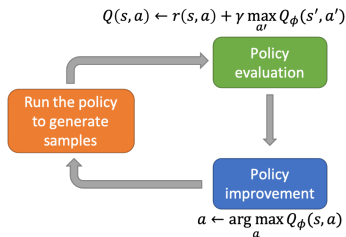
- Define an operator $\Pi : \Pi V' = \arg \min_{V' \in \Omega} \|V'(s) - V(s)\|^2$

- l_2 contraction; $\mathcal{L}(\phi) = \sum_i \|V_\phi^\pi(s_i) - y_i\|^2$

- Value function learning for the critic:

- $V \leftarrow \Pi \mathcal{B}V$; **fitted bootstrapped policy evaluation doesn't converge!**

- Value iteration theory
 - Linear operator for backup
 - Linear operator for projection
 - Backup is contraction
 - Value iteration converges
- Convergence with function approximation
 - Projection is also a contraction
 - Projection + backup is not a contraction
 - Fitted value iteration does not in general converge
- Implications for Q-learning
 - Q-learning, fitted Q-iteration, etc. does not converge with function approximation
- But we can make it work in practice!



Learning objectives of this lecture

- You should be able to...
 - Extend discrete value iteration to fitted value iteration with function approximation
 - Extend discrete Q-learning to fitted Q-iteration with function approximation
 - Be aware of some theories of value function methods

- Lecture 7 of CS285 at UC Berkeley, **Deep Reinforcement Learning, Decision Making, and Control**
 - <http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-7.pdf>
- Classical papers
 - Lagoudakis. 2003. **Least-squares policy iteration**: linear function approximation
 - Riedmiller. 2005. **Neural fitted Q-iteration**: batch mode Q-learning with neural networks

THE END