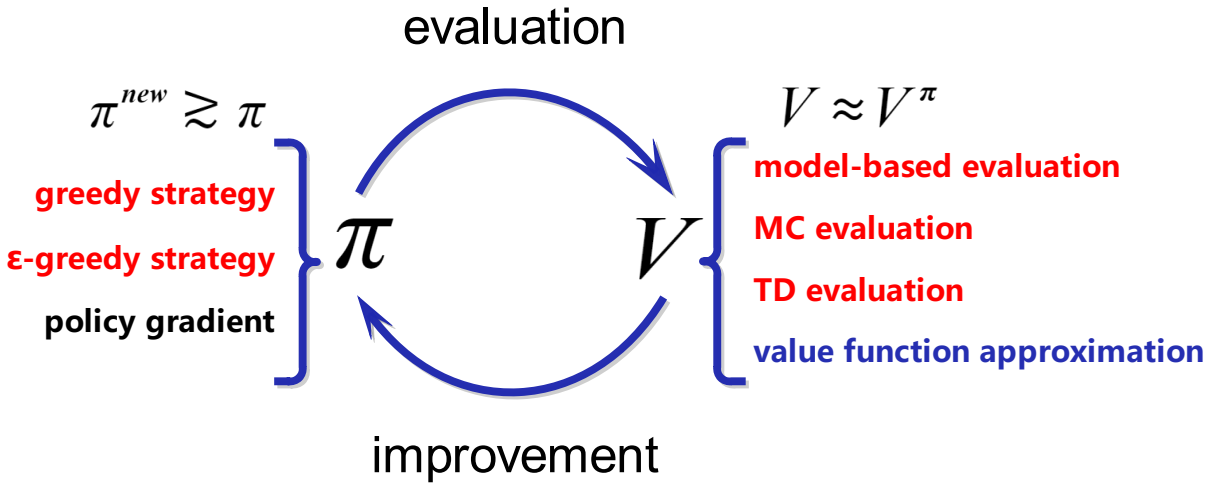
Algorithmic and Theoretical Foundations of RL

Value Function Approximation

Motivation

- We have assumed a tabular representation for value functions until now
  - every state s has a $v_\pi(s)$ or every state-action pair (s, a) has a $q_\pi(s, a)$
- Problem with large MDPs
  - Enormously many states/actions or continuous state/action space
  - Inefficient to learn every state/action value individually
- **Solution**: Value function approximation (reduction of problem dimension)

Value Function Approximation (VFA)



Approximately represent state/action values with functions

$$v_\pi(s) \approx v(s;\omega) \quad \text{or} \quad q_\pi(s,a) \approx q(s,a;\omega)$$

- Learn parameter ω instead of state/action value directly
- Generalize from seen states/actions to unseen states/actions

Which Functions?

- Many possible function approximation methods including
  - Linear function approximation
  - Neural networks
  - Decision trees
  - Nearest neighbors
  - Fourier/wavelet bases
- This lecture focuses on functions that are differentiable, in particular
  - Linear functions:

$$v(s;\omega) = \phi(s)^T \omega,$$

    where $\phi(s)$ is the feature vector of state s
  - Neural networks:

$$v(s;\omega) = \mathbf{NN}_\omega(s),$$

    where $\mathbf{NN}_\omega$ represents a neural network with weights ω.

Table of Contents

Overall Idea

Optimization with Oracle

Assume $v_\pi(s)$ is given by an oracle. Under VFA $v_\pi(s) \approx v(s;\omega)$, one way to find a good ω is by solving

$$\min_\omega J_E(\omega) = \mathbb{E}_{S\sim\mathcal{D}}\left[\left(v(S;\omega) - v_\pi(S)\right)^2\right],$$

where $\mathcal{D}$ is a distribution on $\mathcal{S}$ (which should be associated with π, will be discussed later). Here we consider $\|\cdot\|_2$-norm, but there are other options. The stochastic gradient descent (SGD) method for solving this problem is

$$\omega_{t+1} = \omega_t + \alpha_t \cdot (v_\pi(S) - v(S;\omega_t))\nabla_\omega v(S;\omega_t).$$

We restrict our attention on state values while the discussion for action values is similar.

Overall Idea

Optimization without Oracle

Since $v_\pi(s)$ is unknown, update in last page is not implementable. In this situation, we may replace $v_\pi(s)$ by an estimation.

- MC policy evaluation with function approximation: Letting $G(s)$ (unbiased estimator of $v_\pi(s)$) be the discounted return calculated starting from s following an episode sampled from π, then the update becomes

$$\omega_{t+1} = \omega_t + \alpha_t \cdot (G(s) - v(s;\omega_t))\nabla_\omega v(s;\omega_t).$$

- TD policy evaluation with function approximation: Let $(s, a, r, s') \sim \pi$, we can use $r + \gamma \cdot v(s';\omega_t)$ to estimate $v_\pi(s)$ (biased). Then the update becomes

$$\omega_{t+1} = \omega_t + \alpha_t \cdot (r + \gamma \cdot v(s';\omega_t) - v(s;\omega_t))\nabla_\omega v(s;\omega_t).$$

Caution that $(r + \gamma \cdot v(s';\omega_t) - v(s;\omega_t))\nabla_\omega v(s;\omega_t)$ is not the gradient of (sample version of Bellman error) $(r + \gamma \cdot v(s';\omega_t) - v(s;\omega_t))^2$ since gradient of $v(s';\omega_t)$ is not considered.

Linear Representation of State Values

$$v(s;\omega) = \phi(s)^T\omega, \quad \text{where } \omega \in \mathbb{R}^n \text{ and } \phi(s) = \begin{bmatrix} \phi_1(s) \\ \phi_2(s) \\ \vdots \\ \phi_n(s) \end{bmatrix} \in \mathbb{R}^n \implies \nabla_\omega v(s;\omega) = \phi(s)$$

- If $\phi(s) = e_s \in \mathbb{R}^{|\mathcal{S}|}$ for every $s \in \mathcal{S}$, it reduces to the tabular representation (i.e., represent each state value individually).

MC Policy Evaluation with Linear VFA

Algorithm 1: MC policy evaluation with linear VFA

Initialization: $\phi(s)$, $\omega = \omega_0$, policy π to be evaluated

for $k = 0, 1, 2, \ldots$ **do**

  Initialize s_0 and sample an episode following π:

$$(s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_{T-1}, a_{T-1}, r_{T-1}, s_T) \sim \pi$$

  $G \leftarrow 0$

  for $t = T-1, T-2, \ldots, 0$ **do**

    $G \leftarrow \gamma \cdot G + r_t$

    if s_t *does not appear in* $(s_0, \cdots, s_{t-1})$ **then**

$$\omega \leftarrow \omega + \alpha_{k,t} \left(G - \phi(s_t)^T \omega \right) \phi(s_t)$$

    end

  end

end

Convergence of MC Policy Evaluation with Linear VFA

Theorem 1. Let $\mathcal{D}$ be the stationary distribution over the states induced by the policy π. If α_t satisfies $\sum \alpha_{k,t} = \infty, \quad \sum \alpha_{k,t}^2 < \infty$, then MC policy evaluation with linear VFA converges to the parameter ω_{MC} that minimizes $J_E(\omega)$.

"An analysis of temporal-difference learning with function approximation" by Tsitsiklis and Roy, 1997.

TD Policy Evaluation with Linear VFA

Algorithm 2: TD(0) policy evaluation with linear aproxiamtion

Initialization: $\phi(s)$, ω_0, policy π to be evaluated and initial state s_0

for $k = 0, 1, 2, \ldots$ **do**

  Sample a tuple $(s_t, a_t, r_t, s_{t+1}) \sim \pi$ from s_t

  $\omega_{t+1} = \omega_t + \alpha_t \left(r_t + \gamma \cdot \phi(s_{t+1})^T \omega_t - \phi(s_t)^T \omega_t \right) \phi(s_t)$

end

Convergence of TD(0) Evaluation with Linear Approximation

Theorem 2. Let $\mathcal{D}$ be the stationary distribution over the states induced by the policy π. If α_t satisfies $\sum \alpha_t = \infty, \quad \sum \alpha_t^2 < \infty$, then TD policy evaluation with linear VFA converges to the parameter ω_{TD} such that

$$J_E(\omega_{TD}) \leq \frac{1}{1-\gamma} \min_{\omega} J_E(\omega).$$

- TD target is a biased estimation of $v_\pi(s)$. Thus, TD policy evaluation with Linear VFA is not a stochastic method that minimizes $J_E(\omega)$. Indeed, it is a stochastic method that minimizes the projected Bellman error with suitable norm.

"An analysis of temporal-difference learning with function approximation" by Tsitsiklis and Roy, 1997.

A Word about Stationary Distribution of States under a Policy

Recall from Lecture 1 that $P^\pi = (p^\pi_{ss'})$ is the transition matrix induced by a policy π. In particular, $p^\pi_{ss'}$ is the transition probability from s to s'. Thus, given an initial visiting probability vector $d_0 \in \mathbb{R}^{\mathcal{S}}$ on $\mathcal{S}$ at time 0, it is easy to show that

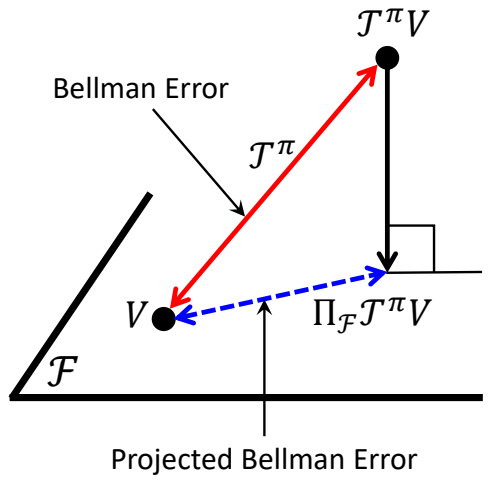
$$d_t = ((P^\pi)^T)^t d_0$$

is the visiting probability vector at time t. Then the probability vector d corresponding to the stationary distribution $\mathcal{D}$ is defined as

$$d = \lim_{t \to \infty} d_t,$$

which exists under mild conditions. When it exists, it is easy to see that d the eigenvector of $(P^\pi)^T$ corresponding to the eigenvalue 1.

Bellman Error and Projected Bellman Error



Given a policy π and $v \in \mathbb{R}^{|\mathcal{S}|}$,

$$\text{Bellman Error:} \quad \mathrm{BE}(v) = \|v - \mathcal{T}^\pi v\|,$$
$$\text{Projected Bellman Error:} \quad \mathrm{PBE}(v) = \|v - \Pi_\mathcal{F}\mathcal{T}^\pi v\|,$$

where $\mathcal{F}$ is a function approximation space.

- If $\mathrm{BE}(v) = 0$, there holds $v = v_\pi$. Note that both Bellman error and projected Bellman error does not involve v_π (unlike $J_E(\cdot)$), we can potentially design algorithms by minimizing them directly.

Remark about Convergence

In order to establish the convergence of a stochastic method, basically we need to show that it is a stochastic version of a GD for certain objective functions (provided GD converges).

- If the method does not follow the gradient of any objective function, the convergence cannot be guaranteed.
- If the method follows the gradient of other objective function, it is not expected to converge to the solution of the target objective function (e.g., TD policy evaluation for linear VFA since TD target is biased estimation of $v_\pi(s)$).

Table of Contents

Policy Evaluation of Actions Values with VFA

With an oracle for $q_\pi(s, a)$, we can form the following optimization problem

$$\min_\omega J_E(\omega) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\|q(s, a; \omega) - q_\pi(s, a)\|_2^2\right].$$

The SGD for this problem is given by

$$\omega_{t+1} = \omega_t + \alpha_t \cdot (q_\pi(s, a) - q(s, a; \omega_t))\nabla_\omega q(s, a; \omega_t).$$

Sample a tuple (s, a, r, s', a'). We can estimate $q_\pi(s, a)$ by $r + \gamma \cdot q(s', a'; \omega_t)$, yielding the update

$$\omega_{t+1} = \omega_t + \alpha_t \cdot (r + \gamma \cdot q(s', a'; \omega_t) - q(s, a; \omega_t))\nabla_\omega q(s, a; \omega_t).$$

We mainly consider TD learning in this section.

Linear VFA of Action Values

In linear VFA for action values, we have

$$q(s, a; \omega) = \phi(s, a)^T \omega, \quad \text{where } \omega \in \mathbb{R}^n \text{ and } \begin{bmatrix} \phi_1(s, a) \\ \phi_2(s, a) \\ \vdots \\ \phi_n(s, a) \end{bmatrix} \in \mathbb{R}^n.$$

It is clear that $\nabla_\omega q(s, a; \omega) = \phi(s, a)$.

SARSA with Linear VFA

Algorithm 3: SARSA with Linear VFA

Initialization: $\phi_{s,a}$, s_0, π_0, $a_0 \sim \pi_0(s_0)$

for $t = 0, 1, 2, \ldots$ **do**

Sample a tuple $(s_t, a_t, r_t, s_{t+1}, a_{t+1}) \sim \pi_t$ from (s_t, a_t)

$$\omega_{t+1} = \omega_t + \alpha_t \left(r_t + \gamma \cdot \phi(s_{t+1}, a_{t+1})^T \omega_t - \phi(s_t, a_t)^T \omega_t \right) \phi(s_t, a_t)$$

Update policy of visited state via ϵ_t-greedy:

$$\pi_{t+1}(a|s_t) = \begin{cases} 1 - \epsilon_t + \frac{\epsilon_t}{|\mathcal{A}|} & \text{if } a = \underset{a'}{\operatorname{argmax}}\, \phi(s_t, a')^T \omega_{t+1}, \\ \frac{\epsilon_t}{|\mathcal{A}|} & \text{otherwise.} \end{cases}$$

end

Q-Learning with Linear VFA

In Q-learning $q(s, a; \omega)$ is used to approximate $q^*(s, a)$. After observing a transition $(s_t, a_t, r_t, s_{t+1}) \sim b_t$, we can construct $r_t + \gamma \cdot \max_a q(s_{t+1}, a; \omega_t)$ as a better estimation of $q^*(s_t, a_t)$ than $q(s_t, a_t; \omega_t)$. Then we can update ω_t towards the gradient descent direction of
$\mathcal{L}(\omega) = \frac{1}{2}\left(r_t + \gamma \cdot \max_a q(s_{t+1}, a; \omega_t) - q(s_t, a_t; \omega)\right)^2$:

$$\omega_{t+1} = \omega_t + \alpha_t \left(r_t + \gamma \cdot \max_a q(s_{t+1}, a; \omega_t) - q(s_t, a_t; \omega_t)\right) \nabla_\omega q(s_t, a_t; \omega_t).$$

Algorithm 4: Q-Learning with linear VFA

Initialization: $\phi(s, a)$, s_0

for $t = 0, 1, 2, \ldots$ **do**

Sample a tuple $(s_t, a_t, r_t, s_{t+1}) \sim b_t$ from s_t where b_t is a behavior policy

Update parameter

$$\omega_{t+1} = \omega_t + \alpha_t \left(r_t + \gamma \cdot \max_a \phi(s_{t+1}, a)^T \omega_t - \phi(s_t, a_t)^T \omega_t\right) \phi(s_t, a_t)$$

end

Rule of Thumb for Convergence

Experiments show that divergence may arise whenever we combine three things (but any two without the third is ok):

- **Value Function approximation**: Generalize from seen values to unseen values,
- **Bootstrapping:** Learn value estimates from other value estimates,
- **Off-policy learning:** Learn a policy from data not due to that policy.

Table of Contents

Q-Learning with VFA as Incremental Approximate Q-Value Iteration

Recall that the Q-value iteration has the following form:

$$q_{t+1} = \mathcal{F} q_t, \quad \text{where } \mathcal{F} q\,(s,a) = \mathbb{E}_{s' \sim \mathbb{P}(\cdot|s,a)} \left(r\left(s,a,s'\right) + \gamma \cdot \max_{a' \in \mathcal{A}} q\left(s',a'\right) \right).$$

With q_t being replaced by its function approximation $q(:;\omega_t)$, there may not be a function $q(:;\omega_{t+1})$ such that $q(:;\omega_{t+1}) = \mathcal{F} q(:;\omega_t)$ holds exactly. Thus, we can instead solve for $q(:;\omega_{t+1})$ via (why the second equality holds?)

$$\begin{aligned}
\omega_{t+1} &= \operatorname*{argmin}_{\omega} \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\left(q(s,a;\omega) - \mathcal{F} q(s,a;\omega_t) \right)^2 \right] \\
&= \operatorname*{argmin}_{\omega} \mathbb{E}_{(s,a)\sim\mathcal{D}, s' \sim \mathbb{P}(\cdot|s,a)} \left[\left(q(s,a;\omega) - \left(r(s,a,s') + \gamma \cdot \max_{a' \in \mathcal{A}} q(s',a';\omega_t) \right) \right)^2 \right].
\end{aligned}$$

Solving this problem by one step SGD based on tuple (s_t, a_t, r_t, s_{t+1}) yields the Q-learning with VFA update.

This is an informal interpretation since the generation of (s_t, a_t, r_t, s_{t+1}) doesn't mean it follows the gradient of a population objective function for $\mathcal{D}$.

Batch Method

Let $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^n$ be a batch of experience data. At time t, we can form an sample version of $\mathbb{E}_{(s,a)\sim\mathcal{D}}\left[(q(s,a;\omega) - \mathcal{F}q(s,a;\omega_t))^2\right]$ and update parameter by finding a solution to the empirical risk minimization (or regression) problem

$$\omega_{t+1} = \operatorname*{argmin}_{\omega} \sum_{i=1}^{n} \left(q(s_i, a_i; \omega) - \left(r_i + \gamma \cdot \max_{a' \in \mathcal{A}} q(s_i', a'; \omega_t)\right)\right)^2.$$

Solving this problem by batch SGD yields an instance of Fitted Q-Iteration.

Fitted Q-Iteration (FQI): Offline Approximate Q-Value Iteration

Algorithm 5: FQI

Initialization: Dataset $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^{n}$, initial VFA parameter $\omega = \omega_0$

for $t = 0, 1, 2, \ldots$ *until some stopping criterion is met* **do**

  Copy parameter: $\tilde{\omega} \leftarrow \omega$

  for $k = 0, 1, 2, \ldots$ *until some stopping criterion is met* **do**

    Sample a mini-batch $\mathcal{B}$ of $\mathcal{D}$

    $\omega \leftarrow$

    $$\omega + \alpha \sum_{(s_i, a_i, r_i, s_i') \in \mathcal{B}} \left(r_i + \gamma \cdot \max_{a'} q(s_i', a'; \tilde{\omega}) - q\left(s_i, a_i; \omega\right) \right) \nabla_\omega q\left(s_i, a_i; \omega\right)$$

  end

end

Deep Q-Learning

Deep Q-learning is a variant of FQI which uses deep neural network for VFA and goes back to incremental learning by maintaining a buffer and experience replay.

Algorithm 6: DQN

Initialization: Replay buffer $\mathcal{D}$ to capacity N, Q network $q(s, a; \omega)$ with $\omega = \omega_0$, target Q network $q(s, a; \tilde{\omega})$ with $\tilde{\omega} = \omega$, SGD iteration number C, $k = 0$, and s_0

for $t = 0, 1, 2, \ldots$ *until some stopping criterion* **do**

- $k \leftarrow k + 1$
- Sample a tuple $(s_t, a_t, r_t, s_{t+1}) \sim b_t$ from s_t and add it to buffer $\mathcal{D}$
- sample a mini-batch $\mathcal{B}$ of $\mathcal{D}$
- $\omega \leftarrow$
- $\omega + \alpha \sum_{(s_i, a_i, r_i, s_i') \in \mathcal{B}} \left(r_i + \gamma \cdot \max_{a'} q(s_i', a'; \tilde{\omega}) - q\left(s_i, a_i; \omega\right) \right) \nabla_\omega q\left(s_i, a_i; \omega\right)$
- **if** $k == C$ **then**
  - $\tilde{\omega} \leftarrow \omega$
  - $k \leftarrow 0$
- **end**

end

Alternative Interpretation of Target Network

Recalling Bellman error for policy evaluation, Bellman error for VFA of optimal action values can be defined in one way as

$$\mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\left(q(s,a;\omega)-\mathcal{F}q(s,a;\omega)\right)^2\right]$$
$$\leq \mathbb{E}_{(s,a)\sim\mathcal{D},s'\sim\mathbb{P}(\cdot|s,a)}\left[\left(q(s,a;\omega)-\left(r(s,a,s')+\gamma\cdot\max_{a'\in\mathcal{A}}q(s',a';\omega)\right)\right)^2\right]$$

Given batch data $\mathcal{D}=\{(s_i,a_i,r_i,s_i')\}_{i=1}^n$, we can find optimal ω by solving

$$\min_{\omega}\sum_{i=1}^{n}\left(q(s_i,a_i;\omega)-\left(r_i+\gamma\cdot\max_{a'\in\mathcal{A}}q(s_i',a';\omega)\right)\right)^2.$$

When applying batch SGD to solve this problem, we not only need to consider the gradient with respect to ω in $q(s_i,a_i;\omega)$ but also in $\max_{a'\in\mathcal{A}}q(s_i',a';\omega)$. The target network can also be interpreted as fixing the parameter (or equivalently the TD target) in $\max_{a'\in\mathcal{A}}q(s_i',a';\omega)$ for a few iterations during the training process.

Practical Tricks

In addition to experience replay (for breaking dependence) and target network (for improving stability), there are also other tricks in deep Q-learning,

- Deep double Q-learning
- Prioritized replay
- Dueling networks
- Clip gradients or use Huber loss on Bellman error
- . . . . . .

Questions?