Algorithmic and Theoretical Foundations of RL

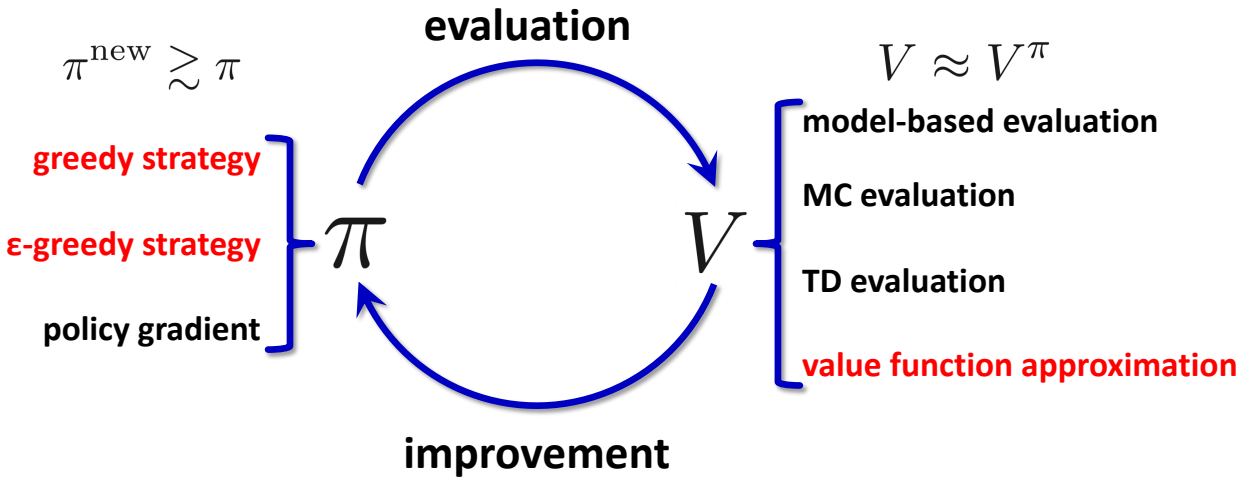
Value Function Approximation

Ke Wei
School of Data Science
Fudan University

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:

$$V(s;\omega) = \phi(s)^T\omega \quad \text{or} \quad Q(s,a;\omega) = \phi^T(s,a)\omega,$$

    where $\phi(s)$ and $\phi(s,a)$ are feature vectors at s and (s,a), respectively.
  - Neural networks:

$$V(s;\omega) = \mathbf{NN}_\omega(s) \quad \text{or} \quad Q(s,a;\omega) = \mathbf{NN}_\omega(s,a),$$

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

Table of Contents

Overall Idea

Optimization with Oracle

Given π, assume $V^\pi(s)$ is given. Under VFA $V^\pi(s) \approx V(s;\omega)$, one way to find a good ω is by solving

$$\min_\omega J(\omega) = \mathbb{E}_{s\sim\mathcal{D}}\left[\left(V(s;\omega) - V^\pi(s)\right)^2\right],$$

where $\mathcal{D}$ is a distribution on $\mathcal{S}$. Here we consider $\|\cdot\|_2$-norm, but there are other options. 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 with Oracle

When $V^\pi(s)$ is unknown, we may replace it by a statistical estimation.

- MC policy evaluation with VFA: Letting $G(s)$ (unbiased estimator of $V^\pi(s)$) be 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 VFA: Given $(s, a, r, s') \sim \pi$, use $r + \gamma \cdot V(s';\omega_t)$ to estimate $V^\pi(s)$ (biased). Then the update becomes

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

Iteratively refining target is key feature of RL methods, differing from supervised learning methods.

Other Perspectives for TD Evaluation with VFA

Approximate Bellman Iteration

Recall that Bellman iteration for state values has the form

$$V^{t+1} = \mathcal{T}^\pi V^t, \quad \text{where } [\mathcal{T}^\pi V](s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\mathbb{E}_{s'\sim P(\cdot|s,a)}\left[r(s,a,s') + \gamma V(s')\right].$$

Given $V(:;\omega_t)$, since $\mathcal{T}^\pi V(:;\omega_t)$ may not equal to some $V(:;\omega_{t+1})$, it is natural to seek ω_{t+1} by solving

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

Solving it via one-step SGD yields TD Evaluation with VFA.

Other Perspectives for TD Evaluation with VFA

Semi-Gradient for Optimizing Bellman Error

Recall the goal is to find an ω such that $V(:;\omega) \approx V^\pi$. Since V^π satisfies Bellman equation $\mathcal{T}^\pi V^\pi = V^\pi$, it is natural to find the desirable ω by minimizing

$$\mathbb{E}_{s\sim\mathcal{D}} \|V(s;\omega) - [\mathcal{T}^\pi V](s;\omega)\|_2^2,$$

which is known as Bellman error (BE) and can be optimized without knowing V^π in contrast to $J(\omega)$.

- BE can be defined under different norms. Under infinity norm, one has

$$\|V - V^\pi\|_\infty \leq \frac{\|V - \mathcal{T}^\pi V\|_\infty}{1-\gamma}.$$

- BE can be similarly defined for Bellman optimality operator $\mathcal{T}$ as well as for action values in the same fashion, which is useful in interpreting Q-learning.

Other Perspectives for TD Evaluation with VFA

Semi-Gradient for Optimizing Bellman Error

SGD for minimizing BE is (neglecting evaluation of $\mathcal{T}^\pi$ for conceptual clarity)

$$\omega_{t+1} = \omega_t + \alpha_t([\mathcal{T}^\pi V](s;\omega_t) - V(s;\omega_t))\nabla_w\big(V(s;\omega_t) - [\mathcal{T}^\pi V](s;\omega_t)\big).$$

It is not easy to compute $\nabla_w \mathcal{T}^\pi V(s;\omega_t)$. TD$(0)$ with VFA replaces $\mathcal{T}^\pi V(s;\omega)$ by $\mathcal{T}^\pi V(s;\omega_t)$ in objective function, and thus the term $\nabla_w[\mathcal{T}^\pi V](s;\omega_t)$ vanishes. This known as semi-gradient and resembles iterative reweighted scheme for solving nonlinear least squares. Note that due to the replacement, the iterate may not exact minimizing Bellman error, but some projected Bellman error.

Linear Representation of State Values

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

- If $\phi(s) = e_s \in \mathbb{R}^{|\mathcal{S}|}$ for every $s \in \mathcal{S}$, it reduces to the tabular representation; that is, 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**

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

end

end

end

TD Policy Evaluation with Linear VFA

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

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

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

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

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

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

Similarly, there are also different perspectives for the update.

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, $a_0 \sim \pi_0(\cdot|s_0)$

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 $Q^*(s, a)$. Having 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 $Q(s_t, a_t; \omega_t)$ since one-step lookahead reward r_t is accurate (or approximate error is discounted by γ), and update ω_t via

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

to reduce $\mathcal{L}(\omega) = \frac{1}{2} \left(r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \omega_t) - Q(s_t, a_t; \omega) \right)^2$.

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

Remark

- Both SARSA and Q-learning follow the TD target of the form $[\mathcal{F}^{\pi}Q](s_t, a_t; \omega_t)$. For SARSA, this quantity is estimated using random samples, while for Q-learning, π is greedy and $[\mathcal{F}^{\pi}Q](s_t, a_t; \omega_t) = [\mathcal{F}Q](s_t, a_t; \omega_t)$.
- Recall that policy iteration can be viewed as updating value and policy alternatively via policy evaluation and policy improvement. SARSA and Q-learning can also be viewed as updating value parameter and policy alternately. Conceptually, it is anticipated to fix policy parameter and update value parameter extensively to fit the policy, and then update the policy through the new parameter. However, SARSA and Q-learning with linear VFA expands this process by policy update right after value parameter update. This will not be a problem for simple VFA, but will cause stability issue for complex VFA, such as the case in deep Q-learning, where target network is introduced to fix policy parameter for a few number of iterations.

Table of Contents

Q-Learning with VFA as Approximate Q-Value Iteration

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

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

With Q^t being replaced by $Q(:;\omega_t)$, there may not be a function $Q(:;\omega_{t+1})$ such that $Q(:;\omega_{t+1}) = \mathcal{F}Q(:;\omega_t)$ holds exactly. We can solve for $Q(:;\omega_{t+1})$ via

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

Solving it via one step SGD yields Q-learning with VFA.

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 ω 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 ω

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 adopts 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 ω, 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

Semi-Gradient Perspective of Deep Q-Learning

Bellman error for optimal action values with VFA can be defined in one way as

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

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 gradient with respect to ω in $Q(s_i,a_i;\omega)$ but also in $\max_{a'\in\mathcal{A}}Q(s_i',a';\omega)$. Thus, deep Q-learning applied semi-gradient method to solve this problem, where 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?