Linear Function Approximation [1]

[1] Sections 4.1: Yang&Ying

Reinforcement Learning with Function Approximations

Challenges in solving the Bellman equation:

- Model-based algorithms: need to know $\bar{r}(i, u)$ and $P_{ij}(u)$
  $\rightarrow$ model free, data-driven methods. Q-Learning, SARSA, etc

Reinforcement Learning with Function Approximations

Challenges in solving the Bellman equation:

- Model-based algorithms: need to know $\bar{r}(i, u)$ and $P_{ij}(u)$
  $\rightarrow$ model free, data-driven methods. Q-Learning, SARSA, etc
- Scalability or Curse-of-dimensionality: large state and action spaces (the 9×9 Go game has 10^{35} states).
  $\rightarrow$ function approximation. Linear Function Approximation or Deep Neural Networks

Linear Function Approximation (Feature-Based)

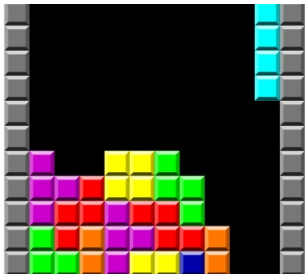
$$Q(i,u) \approx \sum_{m=1}^{M} w_m \phi_m(i,u) = w^T \phi(i,u)$$

- $\phi(i,u)$: M-dimensional feature vector for (i,u) and is assumed to be given
- w: M-dimensional weight vector; common for all (state, action) pairs.

Linear Function Approximation (Feature-Based)

$$Q(i,u) \approx \sum_{m=1}^{M} w_m \phi_m(i,u) = w^T \phi(i,u)$$

- $\phi(i,u)$: M-dimensional feature vector for (i,u) and is assumed to be given
- w: M-dimensional weight vector; common for all (state, action) pairs.



Features: the height of the highest column, differences in height between neighboring columns, holes, and wells

Picture is from `https://en.wikipedia.org/wiki/Tetris`

Linear Function Approximation (Feature-Based)

$$Q(i,u) \approx \sum_{m=1}^{M} w_m \phi_m(i,u) = w^T \phi(i,u)$$

- $\phi(i,u)$: M-dimensional feature vector for (i,u) and is assumed to be given
- w: M-dimensional weight vector; common for all (state, action) pairs.

Learning $Q(i,u)$ ($|\mathcal{S}| \times |\mathcal{A}|$) pairs) is equivalent to learning w.

- If $M << |\mathcal{S}| \times |\mathcal{A}|$, the complexity reduces significantly.
- Learning w also allows us to generalize when facing new state-action pair.

Q-Learning with Linear Function Approximation

Temporal Difference

$$\delta_k = r(x_k, u_k) + \alpha \max_v \tilde{Q}(x_{k+1}, v) - \tilde{Q}(x_k, u_k)$$

Q-Learning with Linear Function Approximation

Temporal Difference

$$\delta_k = r(x_k, u_k) + \alpha \max_v \tilde{Q}(x_{k+1}, v) - \tilde{Q}(x_k, u_k)$$

Temporal Difference with Linear Function Approximation

$$\delta_k(\tilde{w}) = r(x_k, u_k) + \alpha \max_v \tilde{w}^T \phi(x_{k+1}, v) - \tilde{w}^T \phi(x_k, u_k)$$

Q-Learning with Linear Function Approximation

Temporal Difference

$$\delta_k = r(x_k, u_k) + \alpha \max_v \tilde{Q}(x_{k+1}, v) - \tilde{Q}(x_k, u_k)$$

Q-Learning

$$\tilde{Q}(x_k, u_k) \leftarrow \tilde{Q}(x_k, u_k) + \beta_k \left(r(x_k, u_k) + \alpha \max_v \tilde{Q}(x_{k+1}, v) - \tilde{Q}(x_k, u_k) \right)$$

$$\tilde{Q}(x_k, u_k) \leftarrow \tilde{Q}(x_k, u_k) + \beta_k \delta_k \approx \tilde{Q}(x_k, u_k) - \frac{1}{2} \beta_k \frac{\partial \delta_k^2}{\partial \tilde{Q}(x_k, u_k)}$$

Q-Learning with Linear Function Approximation

Temporal Difference with Linear Function Approximation

$$\delta_k(\tilde{w}) = r(x_k, u_k) + \alpha \max_v \tilde{w}^T \phi(x_{k+1}, v) - \tilde{w}^T \phi(x_k, u_k)$$

Q-Learning with Linear Function Approximation

Temporal Difference with Linear Function Approximation

$$\delta_k(\tilde{w}) = r(x_k, u_k) + \alpha \max_v \tilde{w}^T \phi(x_{k+1}, v) - \tilde{w}^T \phi(x_k, u_k)$$

Q-Learning with Linear Function Approximation

$$\tilde{w} \leftarrow \tilde{w} - \frac{1}{2}\beta_k \frac{\partial \delta_k^2(\tilde{w})}{\partial \tilde{w}}$$

Q-Learning with Linear Function Approximation

Temporal Difference with Linear Function Approximation

$$\delta_k(\tilde{w}) = r(x_k, u_k) + \alpha \max_v \tilde{w}^T \phi(x_{k+1}, v) - \tilde{w}^T \phi(x_k, u_k)$$

Q-Learning with Linear Function Approximation

$$\tilde{w} \leftarrow \tilde{w} - \frac{1}{2}\beta_k \frac{\partial \delta_k^2(\tilde{w})}{\partial \tilde{w}}$$

Q-Learning with Linear Function Approximation

$$\tilde{w} \leftarrow \tilde{w} + \beta_k \delta_k(\tilde{w}) \phi(x_k, u_k)$$

Convergence of Q-Learning with Linear Function Approximation

For a Fixed Policy μ

The experiences $\{(x_k, u_k, r(x_k, u_k), x_{k+1}, u_{k+1})\}$ are generated under policy μ.

$$\tilde{w} \leftarrow \tilde{w} + \beta_k \delta_k(\tilde{w}) \phi(x_k, u_k),$$

where

$$\delta_k(\tilde{w}) = r(x_k, u_k) + \alpha \tilde{w}^T \phi(x_{k+1}, u_{k+1}) - \tilde{w}^T \phi(x_k, u_k).$$

Convergence

It has been proved that $\tilde{w} \to w^*$.

Convergence of Q-Learning with Linear Function Approximation

For a Fixed Policy μ

The experiences $\{(x_k, u_k, r(x_k, u_k), x_{k+1}, u_{k+1})\}$ are generated under policy μ.

$$\tilde{w} \leftarrow \tilde{w} + \beta_k \delta_k(\tilde{w}) \phi(x_k, u_k),$$

where

$$\delta_k(\tilde{w}) = r(x_k, u_k) + \alpha \tilde{w}^T \phi(x_{k+1}, u_{k+1}) - \tilde{w}^T \phi(x_k, u_k).$$

Convergence

It has been proved that $\tilde{w} \to w^*$. What is w^*?

High-Level Intuition

We expect w^* to satisfy

$$0 = E\left[\delta(w^*)\phi(x,u)\right].$$

We further assume μ is a deterministic policy, so the action is a deterministic function of the state and we can ignore action in our notation.

$$0 = \sum_i \pi_i \phi(i) \left(r(i) + \alpha \sum_j P_{ij} {w^*}^T \phi(j) - {w^*}^T \phi(i) \right)$$

Recall that where

$$T_\mu(V) = r(i) + \alpha \sum_j P_{ij} V(j).$$

$$\Phi \Pi_\mu (T_\mu(\Phi^T w^*) - \Phi^T w^*) = 0$$

High-Level Intuition

Consider a system with three states and two features.

$$\begin{pmatrix} \phi_1(1), \phi_1(2), \phi_1(3) \\ \phi_2(1), \phi_2(2), \phi_2(3) \end{pmatrix} \begin{pmatrix} \pi(1), 0, 0 \\ 0, \pi(2), 0 \\ 0, 0, \pi(3) \end{pmatrix} \left(T_\mu(\Phi^T w^*) - \begin{pmatrix} \phi_1(1), \phi_2(1) \\ \phi_1(2), \phi_2(2) \\ \phi_1(3), \phi_2(3) \end{pmatrix} \begin{pmatrix} w_1^* \\ w_2^* \end{pmatrix} \right) = 0$$

Without function approximation,

$$T_\mu(V_\mu) - V_\mu = 0.$$

With function approximation

$$T_\mu(\Phi^T w^*) - \Phi^T w^* \neq 0 \quad \text{but} \quad \Phi\Pi_\mu(T_\mu(\Phi^T w^*) - \Phi^T w^* = 0.$$

High-Level Intuition

With function approximation

$$T_\mu(\Phi^T w^*) - \Phi^T w^* \neq 0 \quad \text{but} \quad \Phi\Pi_\mu(T_\mu(\Phi^T w^*) - \Phi^T w^*) = 0.$$

- Bellman Error: $T_\mu(\Phi^T w^*) - \Phi^T w^*$
- Feature space: Φ

$\Phi\Pi_\mu(T_\mu(\Phi^T w^*) - \Phi^T w^*) = 0$ implies that the Bellman error is orthogonal to features (or the projected Bellman error is zero).

Divergence

$\Phi\Pi_\mu(T_\mu(\Phi^T w^*) - \Phi^T w^*) = 0$ assume it is an on-policy algorithm.

Divergence Example

- Two states $\mathcal{S} = \{1, 2\}$
- Transition probability matrix under the given policy: $P = \begin{pmatrix} \frac{1}{6} & \frac{5}{6} \\ \frac{1}{6} & \frac{5}{6} \end{pmatrix}$
- Reward $r(x, u) = 0$ for all (x, u) and $\alpha = \frac{10}{11}$.
- Scalar features: $\phi(1) = 1$ and $\phi(2) = 2$.

It is clear that $w^* = 0$ since $V_\mu = 0$; and the steady state of the Markov chain is $\pi(1) = \frac{1}{6}$ and $\pi(2) = \frac{5}{6}$.

Divergence

Divergence Example

- Two states $\mathcal{S} = \{1, 2\}$
- Transition probability matrix under the given policy: $P = \begin{pmatrix} \frac{1}{6} & \frac{5}{6} \\ \frac{1}{6} & \frac{5}{6} \end{pmatrix}$
- Reward $r(x, u) = 0$ for all (x, u) and $\alpha = \frac{10}{11}$.
- Scalar features: $\phi(1) = 1$ and $\phi(2) = 2$.

Suppose that state 1 and state 2 are with probability $2/3$ and $1/3$, respectively.

Divergence

Divergence Example

- Two states $\mathcal{S} = \{1, 2\}$
- Transition probability matrix under the given policy: $P = \begin{pmatrix} \frac{1}{6} & \frac{5}{6} \\ \frac{1}{6} & \frac{5}{6} \end{pmatrix}$
- Reward $r(x, u) = 0$ for all (x, u) and $\alpha = \frac{10}{11}$.
- Scalar features: $\phi(1) = 1$ and $\phi(2) = 2$.

Suppose that state 1 and state 2 are with probability $2/3$ and $1/3$, respectively.

$$
\begin{aligned}
w \leftarrow & w + \beta \begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} \frac{2}{3} & 0 \\ 0 & \frac{1}{3} \end{pmatrix} \left(\frac{10}{11} \begin{pmatrix} \frac{1}{6} & \frac{5}{6} \\ \frac{1}{6} & \frac{5}{6} \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} w - \begin{pmatrix} 1 \\ 2 \end{pmatrix} w \right) \\
= & w + \frac{2\beta}{9} w = \left(1 + \frac{2\beta}{9} \right) w \to \infty.
\end{aligned}
$$

Reference

- **Convergence proof and the divergence example:** John Tsitsiklis and Benjamin Van Roy. "An Analysis of Temporal-Difference Learning with Function Approximation." IEEE Transactions on Automatic Control (1997).