Lecture 9: Linear and Nonlinear Function Approximation

Course: Reinforcement Learning Theory
Instructor: Lei Ying
Department of EECS
University of Michigan, Ann Arbor

Bellman Equation

The Bellman Equation:

$$J_k^*(x_k) = \max_u E\left[r_k(x_k, u) + J_{k+1}^*(x_{k+1})\right]$$
$$= \max_u \sum_r r P(r_k = r | x_k, u) + \sum_x J_{k+1}^*(x) P(x_{k+1} = x | x_k, u)$$

Difficulties in solving the Bellman equation:

1. Model-based: need to know $P(X_{k+1} | X_k, U_k)$.
   $\rightarrow$ model free, data-driven methods. Q-Learning, SARSA, etc
2. Curse-of-dimensionality: large state and action spaces (the 9×9 Go game has 10^{35} states).
   $\rightarrow$ function approximation.Function Approximation
3. Find the maximization.
   $\rightarrow$ actor-critic, policy gradient

Linear approximation

- Linear function approximation (feature-based)

$$Q(i,u) = \sum_{k=1}^{K} \theta_k \phi_k(i,u) = \theta^T \phi(i,u)$$

  - $\phi(i,u)$: feature vector for (i,u) and is given
  - θ: weight vector (for all state-action pairs)

Learning $\underset{|S|\times|A|}{\underset{\Uparrow}{Q(i,u)}}$ is equivalent to learning $\underset{K}{\underset{\Uparrow}{\theta}}$.

- If $K << |S| \times |A|$, the complexity reduces significantly.
- Learning θ also allows us to generalize when facing new state-action pair.

Linear approximation

TD error:
Q-learning without function approximation:

$$r(x_k, u_k) + \alpha \max_v Q_k(x_{k+1}, v) - Q_k(x_k, u_k)$$

Q-learning with linear function approximation:

$$\underbrace{r(x_k, u_k) + \alpha \max_v \theta_k^T \phi(x_{k+1}, v)}_{\text{new estimate according to Bellman's equation}} - \underbrace{\theta_k^T \phi(x_k, u_k)}_{\Uparrow \text{ old estimate}}$$

Linear approximation

TD error:

Q-learning without function approximation:

$$r(x_k, u_k) + \alpha \max_v Q_k(x_{k+1}, v) - Q_k(x_k, u_k)$$

Q-learning with linear function approximation:

$$\underbrace{r(x_k, u_k) + \alpha \max_v \theta_k^T \phi(x_{k+1}, v)}_{\text{new estimate according to Bellman's equation}} - \underset{\substack{\Uparrow \\ \text{old estimate}}}{\theta_k^T \phi(x_k, u_k)}$$

Temporal-difference algorithm

Update θ according to

$$\theta_{k+1} = \theta_k + \beta_t(r(x_k, u_k) + \alpha \max_v \theta_k^T \phi(x_{k+1}, v) - \theta_k^T \phi(x_k, u_k))\phi(x_k, u_k).$$

Linear approximation

TD(0) with linear function approximation given policy μ:

$$d_k = r_k + \alpha\theta_k^T\phi(x_{k+1}, u_{k+1}) - \theta_k^T\phi(x_k, u_k)$$

$$\theta_{k+1} = \theta_k + \beta_k d_k \phi(x_k, u_k)\phi(x_k, u_k).$$

TD(λ) with linear function approximation given policy μ:

$$\theta_{k+1} = \theta_k + \beta_k \left(\sum_{m=k}^{\infty} \lambda^{m-k} d_m\right) \phi(x_k, u_k).$$

Linear approximation

Backward (online) TD(λ)

$$\theta_{k+1} = \theta_k + \beta_k e_k d_k,$$

where the eligibility trace e_k is defined to be

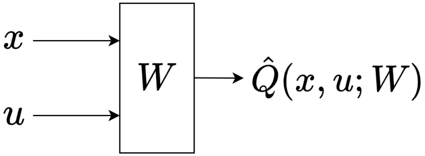
$$e_k = \sum_{m=0}^{k} (\alpha\lambda)^{k-m} \phi(x_m, u_m).$$

. Then,

$$\implies \begin{cases} \theta_{k+1} = \theta_k + \beta_k d_k e_k \\ e_{k+1} = \alpha\lambda e_k + \phi(x_t, u_k) \\ d_k = r_k + \alpha\theta_t^T \phi(x_{k+1}, u_{k+1}) - \theta_k^T \phi(x_t, u_k) \end{cases}$$

Nonlinear function approximation using neural networks

- Approximate $Q(x, u)$ using a neural network.



We can use neural networks for approximating value functions or Q-functions (or in policy-gradient).

- With linear function approximation, we are interested in

$$\min_{\theta} \|\Phi^T \theta - J_\mu\|_\Pi.$$

(Will cover later)

Nonlinear function approximation using neural networks

- Q-learning with NN: We are interested in

$$\min_\theta E\left[\frac{1}{2}(\hat{Q}(x_k, u_k; \theta) - r_k - \alpha \max_v \hat{Q}(x_{k+1}, v; \theta))^2\right]$$

- SGD algirithm:

$$\theta_{k+1} = \theta_k + \beta_k d_k \nabla_\theta \hat{Q}(x_k, u_k; \theta_k)$$

$$\underset{\text{(TD error)}}{d_k} = r_k + \alpha \max_v \hat{Q}(x_{k+1}, v; \theta_k) - \hat{Q}(x_k, u_k; \theta_k)$$

Nonlinear function approximation using neural networks

- Define

$$G(\theta) = \frac{1}{2}(\hat{Q}(x_k, u_k; \theta) - r_k - \alpha \max_v \hat{Q}(x_{k+1}, v; \theta))^2$$

Then

$$\frac{\partial G(\theta)}{\partial \theta} \approx (\hat{Q}(x_k, u_k, \theta) - r_k - \alpha \max_v \hat{Q}(x_{k+1}, v; \theta))\nabla \hat{Q}(x_k, u_k, \theta)$$

- r_k: from samples
- $\max_v \hat{Q}(x_{k+1}, v; \theta_k), \hat{Q}(x_k, u_k; \theta_k)$: from neural networks
- Question: how to compute $\nabla_\theta \hat{Q}(x, u; \theta)$?

Reference

- This lecture is based on R. Srikant's lecture notes on *Q-Learning* available at `https://sites.google.com/illinois.edu/mdps-and-rl/lectures?authuser=1`

Acknowledgements: I would like to thank Alex Zhao for helping prepare the slides, and Honghao Wei and Zixian Yang for correcting typos/mistakes.