Lecture 16 - Value Function Approximation

Guiliang Liu

The Chinese University of Hong Kong, Shenzhen

DDA4230: Reinforcement Learning
Course Page: [Click]

Value Function Approximation

Motivations:

- So far we represented the value function by a lookup table where each state has an entry, $V(s)$, or each state-action pair has an entry, $Q(s,a)$.
- However, this approach might not generalize well to problems with large state and action spaces. A popular solution is via value function approximation (VFA)

$$V^{\pi}(s) \approx \hat{V}(s,\mathrm{w}) \quad \text{or} \quad Q^{\pi}(s,a) \approx \hat{Q}(s,a,\mathrm{w}).$$

Value Function Approximation

$$V^{\pi}(s) \approx \hat{V}(s, \mathrm{w}) \quad \text{or} \quad Q^{\pi}(s, a) \approx \hat{Q}(s, a, \mathrm{w}).$$

In the approximation, w is usually referred to as the parameter or weights of our function approximator. Some choices for function approximators are listed below.

- Linear combinations of features
- Neural networks
- Decision trees
- Nearest neighbors
- Fourier and wavelet basis

Linear feature representations

In linear function representations, we use a feature vector to represent a state

$$x(s) = (x_1(s), x_2(s), \ldots, x_d(s))^T,$$

where d is the dimensionality of the feature space. We then approximate our value functions using a linear combination of features as

$$\hat{V}(s, \mathrm{w}) = x(s)^T \mathrm{w} = \sum_{j=1}^{d} x_j(s) \mathrm{w}_j .$$

Linear feature representations

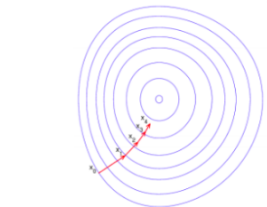
The error of the approximation is defined on the measure space of the occupancy measure, which denotes the cumulative probability that a state is visited under π

$$\rho^{\pi}(s) = \lim_{T \to \infty} \frac{\sum_{t=0}^{T} \gamma^t \mathbb{P}(s_t = s \mid \pi)}{\sum_{t=0}^{T} \gamma^t}.$$

The quadratic objective function (also known as the loss function) of the approximation error is then defined as

$$J(\mathrm{w}) = \mathbb{E}_{s \sim \rho^{\pi}(s)} \left[\left(V^{\pi}(s) - \hat{V}(s, \mathrm{w}) \right)^2 \right].$$

Gradient Descent



A common technique to minimize the above objective function is gradient descent.

- Start at some particular spot x_0, corresponding to some initial value of our parameter w.
- Evaluate the gradient at x_0, which is the direction of the steepest increase of objective.
- Take a step along the negative direction of the gradient vector and arrive at x_1.
- This process is repeated until convergence.

Gradient Descent

Mathematically, this can be summarized as

$$\nabla_{\mathrm{w}} J(\mathrm{w}) = \left(\frac{\partial J(\mathrm{w})}{\partial \mathrm{w}_1}, \frac{\partial J(\mathrm{w})}{\partial \mathrm{w}_2}, \ldots, \frac{\partial J(\mathrm{w})}{\partial \mathrm{w}_n} \right), \quad \text{compute the gradient}$$

$$\Delta \mathrm{w} = -\frac{1}{2} \alpha \nabla_{\mathrm{w}} J(\mathrm{w}), \quad \text{compute an update step using gradient descent}$$

$$\mathrm{w} \leftarrow \mathrm{w} + \Delta \mathrm{w}, \quad \text{take a step towards the local minimum}$$

where α is the learning rate.

Gradient Descent

Stochastic Gradient Descent

- In practice, gradient descent is not a sample-efficient optimizer. so we use stochastic gradient descent (SGD).
- In minibatch SGD, we sample a minibatch of past experiences, compute our objective function on that minibatch, and update our parameters using gradient descent on the minibatch.

Monte-Carlo policy evaluation with linear VFA

Algorithm 1: Monte-Carlo policy evaluation with linear VFA

Initialize $\mathbf{w} = 0$, $R(s) = 0\ \forall s$, $k = 1$

while *true* **do**

  Sample k-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,H_k})$ given π

  for $t = 1, \ldots, H_k$ **do**

    if *first visit to s in episode k* **then**

      Append $\sum_{j=t}^{H_k} r_{k,j}$ to $R(s_t)$

      $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\text{avg}(R(s_t)) - \hat{V}(s_t, \mathbf{w}))x(s_t)$

  $k = k + 1$

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Monte-Carlo policy evaluation with linear VFA

This algorithm is a modification of first-visit Monte-Carlo policy evaluation, while we replace our value function with our linear VFA. Recall that the mean squared error of a linear VFA for a particular policy π relative to the true value is:

$$J(\mathrm{w}) = \sum_s \rho^\pi(s)(V^\pi(s) - \hat{V}^\pi(s,\mathrm{w}))^2 .$$

Lemma

Monte-Carlo policy evaluation with linear VFA converges to the weights w_{MC} *with minimum mean squared error.*

$$J(\mathrm{w}_{MC}) = \min_{\mathrm{w}} \sum_s \rho^\pi(s)(V^\pi(s) - \hat{V}^\pi(s,\mathrm{w}))^2 .$$

Temporal-difference methods with linear VFA

Recall that in the tabular setting, we approximate V^π via bootstrapping and sampling and update $V^\pi(s)$ by

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha(r + \gamma V^\pi(s') - V^\pi(s)),$$

where $r + \gamma V^\pi(s')$ represents our TD target. Using linear VFA, we replace V^π with $\hat{V}^\pi$ and our update equation becomes

$$\begin{aligned} \mathrm{w} &\leftarrow \mathrm{w} + \alpha(r + \gamma \hat{V}^\pi(s', \mathrm{w}) - \hat{V}^\pi(s, \mathrm{w}))\nabla_{\mathrm{w}} \hat{V}^\pi(s, \mathrm{w}) \\ &= \mathrm{w} + \alpha(r + \gamma \hat{V}^\pi(s', \mathrm{w}) - \hat{V}^\pi(s, \mathrm{w}))x(s). \end{aligned}$$

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Temporal-difference methods with linear VFA

In value function approximation, although our target is a biased and approximated estimate of the true value $V^{\pi}(s)$, linear TD(0) will still converge to some global approximate optimum.

Lemma

TD(0) policy evaluation with VFA converges to the weights w_{TD} *which is optimum up to* $1/(1-\gamma)$ *of the minimum mean squared error.*

$$J(\mathrm{w}_{TD}) \leq \frac{1}{1-\gamma} \min_{\mathrm{w}} \sum_{s} \rho^{\pi}(s)(V^{\pi}(s) - \hat{V}^{\pi}(s,\mathrm{w}))^2 .$$

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Control using VFA

Use function approximators for action-value functions by $\hat{Q}(s, a, \mathrm{w}) \approx Q^{\pi}(s, a)$. We may then interleave policy evaluation, by approximating using $\hat{Q}(s, a, \mathrm{w})$, and policy improvement, by ε-greedy policy improvement. To be more concrete, we define our objective function $J(\mathrm{w})$ as

$$J(\mathrm{w}) = \mathbb{E}_{\pi}\left[\left(Q^{\pi}(s, a) - \hat{Q}^{\pi}(s, a, \mathrm{w})\right)^2\right].$$

Similar to what we did earlier in policy evaluation, we may then use either gradient descent or stochastic gradient descent to minimize the objective function.

Control using VFA

For example, for a linear action value function approximator, this can be summarized as

$$x(s,a) = (x_1(s,a), x_2(s,a), \ldots, x_n(s,a))^T, \quad \text{action value features}$$

$$\hat{Q}(s,a,\mathrm{w}) = x(s,a)^T \mathrm{w}, \quad \text{action value linear in features}$$

$$J(\mathrm{w}) = \mathbb{E}_\pi \left[(Q^\pi(s,a) - \hat{Q}^\pi(s,a,\mathrm{w}))^2 \right], \quad \text{objective function}$$

$$-\frac{1}{2}\nabla_{\mathrm{w}} J(\mathrm{w}) = \mathbb{E}_\pi \left[(Q^\pi(s,a) - \hat{Q}^\pi(s,a,\mathrm{w}))\nabla_{\mathrm{w}} \hat{Q}^\pi(s,a,\mathrm{w}) \right]$$

$$= \mathbb{E}_\pi \left[(Q^\pi(s,a) - \hat{Q}^\pi(s,a,\mathrm{w}))x(s,a) \right], \quad \text{compute the gradient}$$

$$\Delta \mathrm{w} = -\frac{1}{2}\alpha \nabla_{\mathrm{w}} J(\mathrm{w})$$

$$= \alpha(Q^\pi(s,a) - \hat{Q}^\pi(s,a,\mathrm{w}))x(s,a), \quad \text{compute the update}$$

$$\mathrm{w} \leftarrow \mathrm{w} + \Delta \mathrm{w}. \quad \text{take a step of gradient descent}$$

Control using VFA

For Monte Carlo methods, we substitute our target $Q^\pi(s,a)$ with a return G_t.

$$\Delta \mathrm{w} = \alpha(G_t - \hat{Q}(s,a,\mathrm{w}))\nabla_{\mathrm{w}}\hat{Q}(s,a,\mathrm{w}).$$

For SARSA, we substitute our target with a TD target

$$\Delta \mathrm{w} = \alpha(r + \gamma\hat{Q}(s',a',\mathrm{w}) - \hat{Q}(s,a,\mathrm{w}))\nabla_{\mathrm{w}}\hat{Q}(s,a,\mathrm{w}).$$

For Q-learning, we substitute our target with a maximum TD target

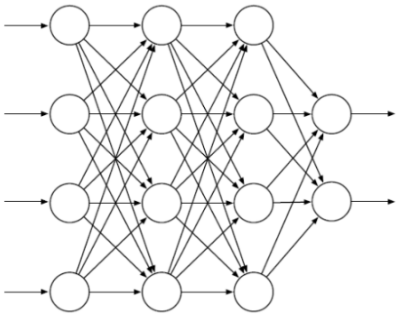
$$\Delta \mathrm{w} = \alpha(r + \gamma\max_{a'}\hat{Q}(s',a',\mathrm{w}) - \hat{Q}(s,a,\mathrm{w}))\nabla_{\mathrm{w}}\hat{Q}(s,a,\mathrm{w}).$$

Neural networks

Motivations:

- Although linear VFAs often work well given the right set of features, it can also be difficult to hand-craft such feature set.
- Neural networks provide a much richer function approximation class that is able to directly go from states without requiring an explicit specification of features.

Neural networks



- Neural networks with a single hidden layer can have the "universal approximation" property, which has been demonstrated both empirically and theoretically.
- Complicated functions can be approximated with a hierarchical composition of multiple hidden layers.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Question and Answering (Q&A)

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen