DDA4230 Reinforcement learning	Value function approximation
Lecture 15	
Lecturer: Guiliang Liu	*Scribe: Baoxiang Wang*

1 Goal of this lecture

In this lecture we investigate methods where the value function is approximated by linear in some known feature vector or by a neural network.
Suggested reading: Chapter 9, 10 and 11 of *Reinforcement learning: An introduction*;

2 Value function approximation

So far we have represented value function by a lookup table where each state has a corresponding entry, $V(s)$, or each state-action pair has an entry, $Q(s,a)$. However, this approach might not generalize well to problems with very large state and action spaces, or in other cases we might prefer quickly learning approximations over converged values of each state. A popular approach to address this problem is via value function approximation (VFA)

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

In the approximation, $\mathbf{w}$ 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

In this lecture, we will explore two popular classes of differentiable function approximators: Linear feature representations and neural networks. The reason for demanding a differentiable function is by the feasibility of the derivation of the algorithms.

3 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

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

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

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

3.1 Gradient descent

A common technique to minimize the above objective function is gradient descent. Figure 1 provides a visual illustration. We start at some particular spot x_0, corresponding to some initial value of our parameter $\mathbf{w}$. We then evaluate the gradient at x_0, which tells us the direction of the steepest increase in the objective function. To minimize our objective function, we take a step along the negative direction of the gradient vector and arrive at x_1. This process is repeated until we reach some convergence criteria.

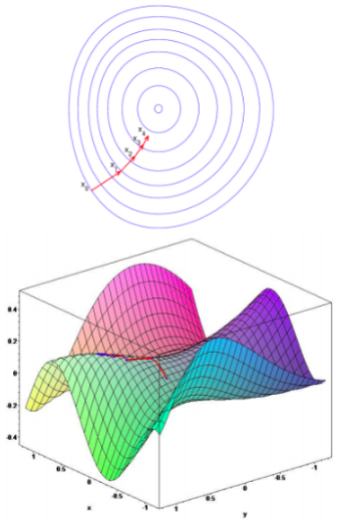

Figure 1: Visualization of gradient descent. We wish to reach the center point where our objective function is minimized. We do so by following the red arrows, which points in the opposite direction of our evaluated gradient.

Mathematically, this can be summarized as

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

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

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

where α is the learning rate.

3.2 Stochastic gradient descent

In practice, gradient descent is not considered a sample efficient optimizer and stochastic gradient descent (SGD) is used more often. Although the original SGD algorithm referred to updating the weights using a single sample, due to the convenience of vectorization, people often refer to gradient descent on a minibatch of samples as SGD as well. 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. Let us now go back to several algorithms we covered in previous lectures and see how value function approximations can be incorporated.

3.3 Monte-Carlo policy evaluation with linear VFA

Algorithm 1: Monte-Carlo policy evaluation with linear VFA

```
Initialize w = 0, R(s) = 0 ∀s, k = 1
while true do
    Sample k-th episode (s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, ..., s_{k,H_k}) given π
    for t = 1, ..., H_k do
        if first visit to s in episode k then
            Append Σ_{j=t}^{H_k} r_{k,j} to R(s_t)
            w ← w + α(avg(R(s_t)) − V̂(s_t, w))x(s_t)
    k = k + 1
```

Algorithm 1 is a modification of first-visit Monte-Carlo policy evaluation, while we replace our value function with our linear VFA. We also make a note that, although our return, $\text{avg}(R(s_t))$, is an unbiased estimate, it is with a high variance, which makes the algorithm hard to converge in practice. The algorithm can be modified to its every-visit variant by removing the if condition.

Recall that the mean squared error of a linear VFA for a particular policy π relative to the true value is

$$J(\mathbf{w}) = \sum_{s \in \mathcal{S}} \rho^{\pi}(s) (V^{\pi}(s) - \hat{V}^{\pi}(s, \mathbf{w}))^2 .$$

Lemma 1 *Monte-Carlo policy evaluation with linear VFA converges to the weights* $\mathbf{w}_{MC}$ *with minimum mean squared error.*

$$J(\mathbf{w}_{MC}) = \min_{\mathbf{w}} \sum_{s \in \mathcal{S}} \rho^{\pi}(s)(V^{\pi}(s) - \hat{V}^{\pi}(s, \mathbf{w}))^2 \,.$$

3.4 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

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

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

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

We omit the proofs here and encourage interested readers to look at *An analysis of temporal-difference learning with function approximation* by Tsitsiklis and Van Roy (1997) or its follow-up studies for some in-depth discussion.

3.5 Control using VFA

Similar to VFAs for the state value function, we can also use function approximators for action value functions. That is, we let $\hat{Q}(s, a, \mathbf{w}) \approx Q^{\pi}(s, a)$. We may then interleave policy evaluation, by approximating using $\hat{Q}(s, a, \mathbf{w})$, and policy improvement, by ϵ-greedy policy improvement. To be more concrete, we write out this mathematically.

First, we define our objective function $J(\mathbf{w})$ as

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

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. For example, for a linear

action value function approximator, this can be summarized as

$$
\begin{aligned}
x(s,a) &= (x_1(s,a), x_2(s,a), \ldots, x_n(s,a))^T\,, && \text{action value features}\\
\hat{Q}(s,a,\mathbf{w}) &= x(s,a)^T\mathbf{w}\,, && \text{action value linear in features}\\
J(\mathbf{w}) &= \mathbb{E}_\pi\left[(Q^\pi(s,a) - \hat{Q}^\pi(s,a,\mathbf{w}))^2\right]\,, && \text{objective function}\\
-\frac{1}{2}\nabla_{\mathbf{w}} J(\mathbf{w}) &= \mathbb{E}_\pi\left[(Q^\pi(s,a) - \hat{Q}^\pi(s,a,\mathbf{w}))\nabla_{\mathbf{w}}\hat{Q}^\pi(s,a,\mathbf{w})\right]\\
&= \mathbb{E}_\pi\left[(Q^\pi(s,a) - \hat{Q}^\pi(s,a,\mathbf{w}))x(s,a)\right]\,, && \text{compute the gradient}\\
\Delta\mathbf{w} &= -\frac{1}{2}\alpha\nabla_{\mathbf{w}} J(\mathbf{w})\\
&= \alpha(Q^\pi(s,a) - \hat{Q}^\pi(s,a,\mathbf{w}))x(s,a)\,, && \text{compute the update}\\
\mathbf{w} &\leftarrow \mathbf{w} + \Delta\mathbf{w}\,. && \text{take a step of gradient descent}
\end{aligned}
$$

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

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

For SARSA, we substitute our target with a TD target

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

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

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

We note that because our use of value function approximations to carry out the Bellman backup operator, convergence is not guaranteed. We refer users to look for Baird's counterexample for a more concrete illustration. The rest of the algorithms have contraction guarantees, which is summarized in Table 1.

Algorithm	Tabular	Linear VFA	Nonlinear VFA
Monte-Carlo control	Yes	(Yes)	No
SARSA	Yes	(Yes)	No
Q-learning	Yes	No	No

Table 1: Summary of convergence of Control Methods with VFA. A "(Yes)" means the result chatters around some near-optimal value function.

4 Neural networks

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.

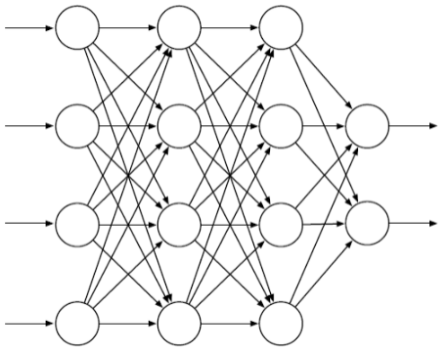

Figure 2: A generic feedforward neural network with 4 input units, 2 output units, and 2 hidden layers.

Figure 2 illustrates a generic feedforward neural network. The network in the figure has an output layer consisting of two output units, an input layer with four input units, and two hidden layers, which are layers that are neither input nor output layers. A real-valued weight is associated with each link. The units are typically semi-linear, meaning that they compute a weighted sum of their input signals and then apply a nonlinear function to the result. This is usually referred to as activation functions. 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.

Some theory and recent results of neural networks have been applied to solve bandit and reinforcement learning problems. We refer readers of interest to conduct Google search on those recent advances.

Acknowledgement

This lecture notes partially use material from *Reinforcement learning: An introduction* and *CS234: Reinforcement learning* from Stanford.