CS294: Deep Reinforcement Learning [1]

Lecture Notes: Value Functions Introduction [2]

Fall 2017[3]

[1] Course Instructors: Sergey Levine

[2] Author: InnerPeace

[3] Un-official Lecture Notes

Keyphrases: Value Function. Advantage Function. Q-learning. Policy Iteration.
This set of notes introduces the idea of improving policy only by using a critic, without an actor. We then discuss how to extract policy from a value function. Lastly, we will introduce the *Q-learning* algorithm, how it works and practical considerations.

1 *Policy Iteration*

We learned that *policy gradient* is intuitive and closely related to the usual supervised learning algorithms we are familiar with, but it's noisy and has high variance. Can we find a way to get rid of it? In *actor-critic*, we estimated the advantage function $\hat{A}^\pi(\mathbf{s}_t, \mathbf{a}_t)$ which represents how much better is $\mathbf{a}_t$ than the average action according to π. It turns out that advantage function is quiet useful, cause we can find out the *best* action from $\mathbf{s}_t$ following π afterwards, which is $\arg\max_{\mathbf{a}_t} \hat{A}^\pi(\mathbf{s}_t, \mathbf{a}_t)$. And if we can find out the best action at every time-step, we can improve the policy with the mechanism, then problem's solved regardless of what the policy is. Besides, with this idea, we use only the advantage function without worrying about the high variance sort of problems of policy gradient.

The resulting mechanism is straightforward. The new policy assigns the probability 1 to the best action at that time-step and 0 to other actions. And the new policy is at least as good as π, then we can improve policy with this mechanism iteratively.

$$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} \hat{A}^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

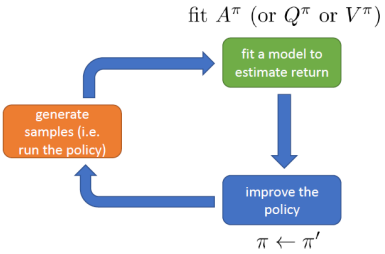

Figure 1: General scheme of improving policy with advantage.

Algorithm 1: Policy Iteration Algorithm

1: **while** *Not converged* **do**
2: Evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a})$.
3: Update $\pi \leftarrow \pi'$.
4: **end while**
5: **return** improved policy π

The way to evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a})$ is the same as *actor-critic* where

$$A^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')] - V^\pi(\mathbf{s}) \tag{2}$$

And we only need to evaluate V^π during iteration.

1.1 *Dynamic Programming*

To elaborate this idea, we start with a toy example assuming that we know the dynamic, and state and action are both discrete and small. Suppose that we are in a grid world which has the size of 4×4 (figure 2) with 16 states and 4 actions per state by walking to different directions. We can store full $V^\pi(\mathbf{s})$ in a table.

As for the value estimation, we discussed the *bootstrapped* mode in actor-critic, where the reward plus next value is the estimation of value function, and the expectation over actions of that is the value of current state.

Figure 2: Tabular case which has no approximation errors w.r.t the value function.

$$V^\pi(\mathbf{s}) \leftarrow E_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[r(\mathbf{s},\mathbf{a}) + \gamma E_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})}\left[V^\pi(\mathbf{s}')\right]\right] \tag{3}$$

From equation 1, only one action (best one) has non-zero probability which means that the policy π is deterministic $\pi(\mathbf{s}) = \mathbf{a}$. Then we can simplify equation 3 to

$$V^\pi(\mathbf{s}) \leftarrow r(\mathbf{s},\pi(\mathbf{s})) + \gamma E_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\pi(\mathbf{s}))}\left[V^\pi(\mathbf{s}')\right] \tag{4}$$

For the tabular case, there is no approximation errors for the estimation of $V^\pi(\mathbf{s})$.

2 *Value Iteration*

The Q-function measures the expected future rewards by taking action $\mathbf{a}_t$ and state $\mathbf{s}_t$, and the relationship between Q-function and value function is:

$$Q^\pi(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')] \tag{5}$$

$$V^\pi(\mathbf{s}) = E_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] \tag{6}$$

Interestingly, it turns out the best action w.r.t to advantage is the same as that of Q-function.

$$\arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t,\mathbf{a}_t) = \arg\max_{\mathbf{a}_t} Q^\pi(\mathbf{s}_t,\mathbf{a}_t) \tag{7}$$

Then we can replace advantage with Q-function to make it simpler for evaluating. For the tabular case, we can write out all the $Q(\mathbf{s},\mathbf{a})$ w.r.t every pair of state-action, and for every state, we just take the actions with biggest Q which is combined to form the policy. Besides, only $\arg\max_{\mathbf{a}_t} Q^\pi(\mathbf{s}_t,\mathbf{a}_t)$ has probability of 1 and together with equation 6 we obtain that:

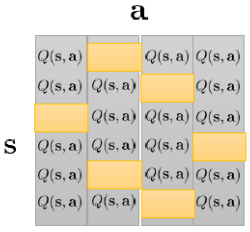

Figure 3: Q values for tabular case, the yellow block means the biggest Q value for the each state.

$$V^\pi(\mathbf{s}) = \max_{\mathbf{a}} Q(\mathbf{s},\mathbf{a}) \tag{8}$$

Then we skip the policy and compute values directly. Finally, if we know how to represent value function $V^\pi(\mathbf{s})$, then together with those ideas, it forms the *value iteration algorithm*.

Algorithm 2: Value Iteration Algorithm

1: **while** *Not converged* **do**
2: Evaluate $Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')]$.
3: Update $V^\pi(\mathbf{s}) = \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$.
4: **end while**

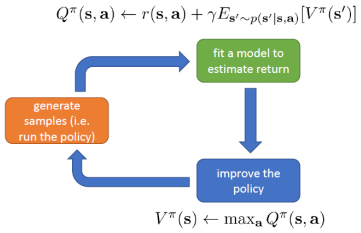

Figure 4: Value Iteration Algorithm

2.1 Q Iteration

If we compute $Q^\pi(\mathbf{s}, \mathbf{a})$ directly, it results in *Q iteration algorithm* which is basically the same as *value iteration algorithm* but with changing order of steps.

Algorithm 3: Q Iteration Algorithm

1: **while** *Not converged* **do**
2: Evaluate $V^\pi(\mathbf{s}) = \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$.
3: Update $Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')]$.
4: **end while**

3 Fitted Value Iteration

How do we represent $V(s)$ in value iteration algorithm? For the discrete state space, we can do that with a big table, one entry for each state. But when the dimension of state space goes large, it's impractical *e.g.* if the state is an image with size 200×200, the size of state space is $|\mathcal{S}| = (255^3)^{200 \times 200}$ which is more than atoms in the universe. It's also referred as the *curse of dimensionality*.

To deal with state space with large dimension, we can use approximator like neural network where the input is state $\mathbf{s}$, output is $V(\mathbf{s})$, and weights of the network is denoted as ϕ. As normal linear regression, we set the square distance as the loss function:

$$\mathcal{L}(\phi) = \frac{1}{2} \left|\left| V_\phi(\mathbf{s}) - \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a}) \right|\right|^2 \tag{9}$$

We usually refer it as *fitted value iteration algorithm*.

Algorithm 4: Fitted Value Iteration Algorithm

1: **while** *Not converged* **do**
2: Set $\mathbf{y}_i = \max_{\mathbf{a}_i}(r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}_i')])$.
3: Update $\phi = \arg\min_\phi 1/2 \sum_i ||V_\phi(\mathbf{s}_i) - \mathbf{y}_i||^2$.
4: **end while**

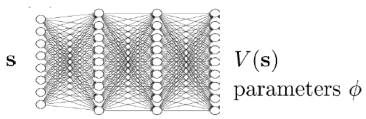

Figure 5: Value function approximator

When evaluating $\mathbf{y}_i$, it requires us to know the outcomes for different actions and we can do that with knowing dynamics or with sampling. But it also requires us go back to the exact same previous step after going to the next state which is usually impractical.

4 *Fitted Q iteration*

What about approximating $Q^\pi(\mathbf{s}, \mathbf{a})$ directly? From *Q iteration*, if we approximate $E[V^\pi(\mathbf{s}')]$ with one sample of the next state:

$$E[V^\pi(\mathbf{s}')] \approx \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}') \tag{10}$$

Then the resulting algorithms is referred as *fitted Q iteration algorithm*.

Algorithm 5: Fitted Q Iteration Algorithm

1: **while** *Not converged* **do**
2: Approximate $E[V(\mathbf{s}_i')] \approx \max_{\mathbf{a}_i'} Q_\phi(\mathbf{s}_i', \mathbf{a}_i')$.
3: Set $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V(\mathbf{s}_i')]$.
4: Update $\phi = \arg\min_\phi 1/2 \sum_i ||Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i||^2$.
5: **end while**

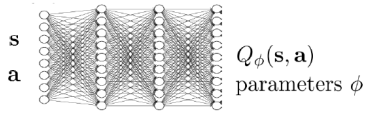

Figure 6: Q-function approximator

Where we can get rid of simulation of actions since the Q function tells the value for different actions. We still need the *max* operation, but it doesn't involve trying different actions in the real world, instead, we just need to try different actions in the function approximator.

It works even for off-policy [4] samples and has one network with input as state and action, no high-variance policy gradient involved. However, there's no convergence guarantees for non-linear function approximation [5].

The fully fitted Q iteration algorithm including the data collecting phrase has many other options in practice.

[4] If the algorithm estimates the value function of the policy generating the data, the method is called on-policy. Otherwise it is called off-policy.

[5] We alluded this when discussing trade-off of different RL algorithms in lecture 2, and we will talk more about it later.

Algorithm 6: Fully Fitted Q Iteration Algorithm

1: **while** *Not converged* **do**
2: Collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}_i', r_i)\}$ using some policy.
3: **while** *less than K steps* **do**
4: Set $\mathbf{y}_i \approx r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}_i'} Q_\phi(\mathbf{s}_i', \mathbf{a}_i')$.
5: **while** *less than S steps* **do**
6: Update $\phi = \arg\min_\phi 1/2 \sum_i ||Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i||^2$.
7: **end while**
8: **end while**
9: **end while**

Where K controls how further we choose to evaluate the Q-function and S controls how many iterations we take to improve

the function approximator.

Why is this algorithm off-policy? In algorithm 6, $\max_{\mathbf{a}_i'} Q_\phi(\mathbf{s}_i', \mathbf{a}_i')$ estimates the value of the current policy (optimal policy extracted from Q-function), and this estimation is valid regardless of where the state $\mathbf{s}'$ comes from, besides it updates with arg max in default even if the actual policy will not take the action that maximize Q function. It's like estimating the $Q(\mathbf{s}, \mathbf{a})$ in the big table of state-action space, what we are trying to do find out what the Q value is for each state-action pair regardless of where the state-action comes from.

It could be confusing what fitted Q iteration is trying to optimizing, cause the policy gradient is clear which heads to maximize the expected reward. Actually, the fitted Q iteration minimizes the *bellman error* $\mathcal{E}$.

$$\mathcal{E} = \frac{1}{2} E_{(\mathbf{s},\mathbf{a})\sim\beta}\Big[Q_\phi(\mathbf{s}, \mathbf{a}) - [r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}')]\Big] \tag{11}$$

Where the $(\mathbf{s}, \mathbf{a})$ is distributed from the behavior policy β. if $\mathcal{E} = 0$, then

$$Q_\phi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}') \tag{12}$$

If the distribution β has good coverage which assigns a probability to every possible state-action pair, then the Q function $Q_\phi(\mathbf{s}, \mathbf{a})$ satisfies the equation 12 is called the optimal policy, corresponding to optimal policy π' (following the mechanism in equation 13)

$$\pi(\mathbf{s}_t|\mathbf{a}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

which leads to the maximized reward. However, when we leave the tabular case, these guarantees about the optimality are lost *e.g.* using the neural network function approximator.

4.1 Online Q-learning

We can change fitted Q-iteration into online version by taking only one step, and update Q once with one step of gradient descent.

1: **while** *Not converged* **do**
2: Take action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}_i', r_i)$.
3: Set $\mathbf{y}_i \approx r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}_i'} Q_\phi(\mathbf{s}_i', \mathbf{a}_i')$.
4: Update $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$.
5: **end while**

Algorithm 7: Online Q-learning Algorithm

But which action do we take for data collecting? One way is to follow our final policy, also referred as *greedy policy* (equation 13),

but we might get stuck in some state which we think it's good and go to that state repeatedly without exploring other states. And this is a major problem in RL called *exploration problem* into which we will dive deeper. Besides, exploration will has a great impact on range of support of the behavior policy β. For now, we will introduce some methods with simple heuristics that people often choose to get around with the problem.

Epsilon greedy

The intuition is that the greedy policy is good but I will not fully commit to it, instead I will use a little factor ϵ to control the commitment, and do exploration with other actions with small probability.

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1-\epsilon & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ \epsilon/(|\mathcal{A}|-1) & \text{otherwise} \end{cases} \tag{14}$$

Where we think the other actions have equal probability and $\mathcal{A}$ is number of actions.

Boltzmann exploration

Another nuanced way is *Boltzmann exploration*, intuitively, the Q function gives clue about which action is better or worse then we just follow it and construct probability of actions based on Q-function. Besides, we need the probabilities to be positive and normalized, then the exponent of Q-function seems reasonably good.

$$\pi(\mathbf{a}|\mathbf{s}) \propto \exp(Q_\phi(\mathbf{s}_t, \mathbf{a}_t)) \tag{15}$$

Later, we will find out that the Boltzmann exploration will lead to interesting connect between RL with probabilistic inference.

5 *Leaning theory*

We alluded that value based learning algorithms have no guarantees that it will converge to optimal policy when using function approximator, why is that? And what about tabular case? Let's dive deeper to find out the mathematical explanation behind all of these.

Firstly, we define an operator $\mathcal{B}$:

$$\mathcal{B}V = \max_{\mathbf{a}} r_{\mathbf{a}} + \gamma \mathcal{T}_{\mathbf{a}} V \tag{16}$$

Where $r_{\mathbf{a}}$ is stacked vector of rewards at all states for action $\mathbf{a}$ *e.g.* it's the element-wise maximization along horizontal axis in figure 3, and $\mathcal{T}_{\mathbf{a}}$ is the matrix of transitions for action $\mathbf{a}$ such that $\mathcal{T}_{\mathbf{a},i,j} = p(\mathbf{s}' = i|\mathbf{s} = j, \mathbf{a})$.

We also define the *fixed point* of $\mathcal{B}$ as $V^\star$ which satisfies:

$$V^\star = \mathcal{B}V^\star = \max_{\mathbf{a}} r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\star(\mathbf{s}')] \tag{17}$$

In theory, the $V^\star$ always exists and is unique [6], besides it corresponds to the optimal policy. We can prove that value iteration in *tabular case* (algorithm 2 which is the same as equation 17 mathematically) reaches $V^\star$ because $\mathcal{B}$ is a *contraction* [7] which means for any V and $\bar{V}$, we have

$$||\mathcal{B}V - \mathcal{B}\bar{V}||_\infty \leq \gamma||V - \bar{V}||_\infty \tag{18}$$

After applying operator $\mathcal{B}$, the distance w.r.t to infinity norm gets small than that of before by discount factor γ. And if we choose $V^\star$ as $\bar{V}$, we will converge eventually.

For the non-tabular case *e.g.* fitted value iteration algorithm (algorithm 4) which includes the regression step finding the function V' which minimizes the squared error.

$$V' \leftarrow \arg\min_{V' \in \Omega} \frac{1}{2} \sum ||V'(\mathbf{s}) - (\mathcal{B}V)(\mathbf{s})||^2 \tag{19}$$

Where Ω is the class of functions for the function approximator (*e.g.* neural network). We can illustrate the update process with figure 8. The plane represents all the possible functions, and the blue line means the possible function represented by current function approximator (*e.g.* neural nets). We start from function V and applying operator $\mathcal{B}$ to jump off the manifold of Ω, trying to reach the fixed point of $\mathcal{B}$. Then with regression, we project $\mathcal{B}V$ back to Ω as V' ideally.

Mathematically, we define a new operator Π:

$$\Pi V = \arg\min_{V' \in \Omega} \frac{1}{2} \sum ||V'(\mathbf{s}) - (\mathcal{B}V)(\mathbf{s})||^2 \tag{20}$$

Where Π is a *projection* onto Ω in terms of l_2 norm. Then for fitted value iteration algorithm, we firstly apply *back up* $\mathcal{B}$, then project it with Π:

$$V \leftarrow \Pi\mathcal{B}V \tag{21}$$

However the problem is that, the $\mathcal{B}$ is a contraction *w.r.t.* infinity norm ("max" norm), and Π is a contraction *w.r.t.* l_2 norm (Euclidean distance), but $\Pi\mathcal{B}$ is not a contraction of any kind in general.

In a word, value iteration converges in tabular case and fitted value iteration does not converge in general. Besides, it also applies to Q-iteration algorithms.

[6] TODO: prove it.

[7] TODO: prove it

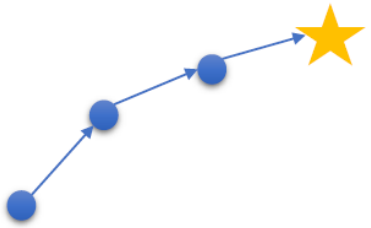

Figure 7: Illustration of value iteration convergence in tabular case

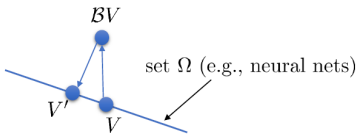

Figure 8: Illustration of updating process of fitted value iteration algorithm.

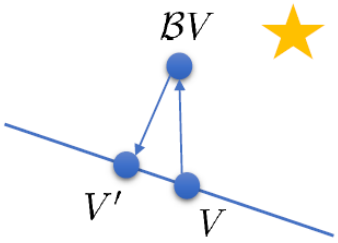

Figure 9: The star is the optimal function and learning procedure could drive away from the star in general.