Addison Kalanther (addikala@berkeley.edu)

Lecture 7: Value Function Methods

In the last lecture, we went over actor-critic algorithms, which used value functions to define our advantage function for the policy gradient step. In this lecture, we explore how we can omit the policy gradient completely to recover pure value function based methods.

Omitting the policy gradient

The advantage $A^\pi(\mathbf{s}_t, \mathbf{a}_t)$ tells us how much better $\mathbf{a}_t$ is than the average action according to π.

What if we implicitly define the policy using $\arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t)$. This would be the best action from $\mathbf{s}_t$ if we follow π. This guarantees that our action is at *least* as good as any action $\mathbf{a}_t \sim \pi(\mathbf{a}_t \mid \mathbf{s}_t)$, regardless of π.

More rigorously, we can define our policy as the following.

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

This is guaranteed to be as good as π and will likely be better.

Policy Iteration

The idea behind policy iteration is to repeat the above policy redefinition until convergence. This happens when the policy iteration update (step 2) no longer changes the policy.

Policy Iteration Algorithm

1. evaluate $A^\pi(\mathbf{s}_t, \mathbf{a}_t)$
2. set $\pi \leftarrow \pi'$
3. Repeat

As before, $A^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma E[V^\pi(\mathbf{s}_{t+1})] - V^\pi(\mathbf{s}_t)$

Evaluating $V^\pi(\mathbf{s})$

Dynamic Programming Let's assume we know the transition dynamics $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$ and that $\mathbf{s}$ and $\mathbf{a}$ are both discrete (and small).

We can store all $V^\pi(\mathbf{s})$ in a table, updating $V^\pi(\mathbf{s})$ per step through the transition tensor $\mathcal{T}$ using the following bootstrapped update.

$$V^\pi(\mathbf{s}) \leftarrow E_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}[r(\mathbf{s}, \mathbf{a}) + \gamma E_{\mathbf{s}'\sim o(\mathbf{s}'|\mathbf{s},\mathbf{a})}[V^\pi(\mathbf{s}')]]$$

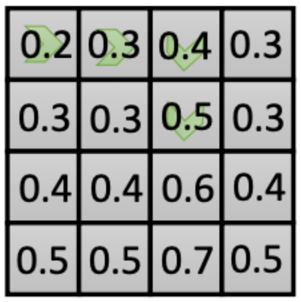
This can be simplified to

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

Repeating this until convergence, this gives us our **policy evaluation** algorithm.

Policy evaluation algorithm

1. $V^\pi(\mathbf{s}) \leftarrow r(\mathbf{s}, \pi(\mathbf{s})) + \gamma E_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[V^\pi(\mathbf{s}')]$
2. Repeat



16 states, 4 actions per state

can store full $V^\pi(\mathbf{s})$ in a table!

$\mathcal{T}$ is $16 \times 16 \times 4$ tensor

Figure 1: Fully defined value function table

Value Iteration

An even simpler method can be computed if we don't explicitly compute our policy. Instead, if we implicitly define our policy through finding the $\arg\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$ once Q converges. This is because

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

This means we can evaluate

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

which is a bit simpler.

Value iteration algorithm

1. set $Q^\pi(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E[V(\mathbf{s}')]$
2. set $V(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$
3. repeat

We recover our policy by $\arg\max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a}) \rightarrow$ policy.

Fitted Value Iteration

For most problems, we can't have a full table for each $Q^\pi(\mathbf{s}, \mathbf{a})$ and $V^\pi(\mathbf{s})$ entry (due to curse of dimensionality). When this is the case, we need a way to model our Q- and value functions.

Rather than have a tabular representation of the value functions $V(\mathbf{s})$, what if we used a neural net function $V : \mathcal{S} \rightarrow \mathbb{R}$ with parameters ϕ?

We would get **fitted value iteration**, which fits a neural net to approximate the value function and then performs value iteration using the previous steps fitted value function.

Fitted value iteration algorithm

1. set $\mathbf{y}_i \leftarrow \max_{\mathbf{a}_i}(r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}_i')])$
2. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i ||V_\phi(\mathbf{s}_i) - \mathbf{y}_i||^2$
3. Repeat

Step 1 is the evaluation step, and step 2 is the update step. We go until convergence or desired performance has been reached, since convergence is only guaranteed in the tabular case.

The problem with fitted value iteration is that we still need to know the transition dynamics to evaluate step 1 (to find $\max_{a_i}$ and $E[V_\phi(\mathbf{s}_i')]$). We can forgo the transition dynamics if we use Q-values instead.

Fitted Q-iteration

With fitted q-iteration. We don't need to know the transition dynamics. It can instead be inferred from the collected samples.

Basic fitted Q-iteration algorithm

1. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}_i')]$
    - approximate $E[V(\mathbf{s}_i')] \approx \max_{\mathbf{a}'} Q_\phi(\mathbf{s}_i', \mathbf{a}_i')$
2. set $\phi \leftarrow \arg\min_\phi \frac{1}{2}\sum_i ||Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - y_i||^2$
3. Repeat

A few advantages of q-iteration is that it works for off-policy samples, unlike actor-critic, and only requires one network without a high-variance policy gradient. The issue is that there are no convergence guarantees for non-linear function approximations like this one.

Full fitted Q-iteration algorithm

The fully fitted q-iteration algorithm collects a dataset using some policy and then runs q-iteration K times before collecting more samples from the environment. This allows for the Q-value approximator $Q_\phi(\mathbf{s}, \mathbf{a})$ to sufficiently update it's parameters before using new samples.

Full fitted Q-iteration algorithm (step-by-step)

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}_i', r_i)\}$ using some policy
2. Run the following procedure K times
    1. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}_i'} Q_\phi(\mathbf{s}_i', \mathbf{a}_i')$
    2. set $\phi \leftarrow \arg\min_\phi \frac{1}{2}\sum_i ||Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - y_i||^2$
3. Repeat

From Q-iteration to Q-Learning

What makes Q-iteration off-policy?

The dataset in step 1 of the full fitted Q-iteration algorithm doesn't specify we need to use the current policy to collect samples. This means we can keep a dataset of transitions in a replay buffer R. Following this, in step 1 of the s sub-procedure, the $\mathbf{s}_i, \mathbf{a}_i$ transition is independent of π, and the $\max_{\mathbf{a}_i'}$ can be computed using the current policy without having to sample from the environment.

What is fitted Q-iteration optimizing?

Q-iteration is optimizing the *bellman error*. It is defined as the following

$$\epsilon = \frac{1}{2} E_{(\mathbf{s},\mathbf{a})\sim\beta}\left[\left(Q_\phi(\mathbf{s}, \mathbf{a}) - [r(\mathbf{s}, \mathbf{a}) + \gamma \max_{a'} Q_\phi(\mathbf{s}', \mathbf{a}')]\right)^2\right]$$

If $\epsilon = 0$, then $Q_\phi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \max_{a'} Q_\phi(\mathbf{s}', \mathbf{a}')$, the neural net has converged to the optimal Q-function, corresponding to the optimal policy π'.

The optimal policy is sometimes written using Q^* and π^*.

Online Q-iteration

Online q-iteration is like our full fitted q-iteration algorithm, but the dataset is of size $N = 1$ and continuously sampled from our environment, and our subprocess does not repeat ($K = 1$).

Online Q-iteration algorithm

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}_i} Q_\phi(\mathbf{s}', \mathbf{a}')$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(Q_\phi(\mathbf{s}', \mathbf{a}') - \mathbf{y}_i)$

The final policy would follow from taking the $\arg\max_{\mathbf{a_t}} Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$. However, this wouldn't be the optimal policy for taking some action in step 1. This is because the policy would not explore actions other than what is immediately giving the best outcome (it would never realize rewards from lucrative long-horizon trajectories). Because of this, we have to use a policy that is prone to *exploration*, which describes how much it deviates from the current best policy to explore other less immediately-optimal actions.

Two common policies that incorporate exploration are **epsilon-greedy**:

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

($|\mathcal{A}|$ is the cardinality of the action space)

and **Boltzman exploration**:

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

Some might recognize this as being a form of *softmax*, which is why this is also called *softmax exploration.*

Q-iteration vs Q-learning

The difference between standard fitted q-iteration and q-learning is that q-iteration takes from any $\mathbf{s}_t, \mathbf{a}_t$ transition pair while Q-learning is online and samples its transitions from some form of its current policy. Online fitted q-iteration is a form of q-learning.

It is important to note that q-learning specifically makes use of a neural network to learn it's q-values. Standard q-iteration does not have this restriction (for example, could be tabular as discussed before).

Value function learning theory

Does it converge?

A question with these methods is whether or not it converges. We discussed previously that tabular q-iteration converges, but why?

Let's define an operator $\mathcal{B} : \mathcal{B}V = \max_{\mathbf{a}} r_{\mathbf{a}} + \gamma \mathcal{T}_{\mathbf{a}} V$, $\mathcal{T}$ is a matrix of transitions for action $\mathbf{a}$ such that $\mathcal{T}_{\mathbf{a},i,j} = p(\mathbf{s}' = i \mid \mathbf{s} = j, \mathbf{a})$.

V^* is a *fixed point* of $\mathcal{B}$. We know that $V^*(\mathbf{s}) = \max_{\mathbf{a}} r(\mathbf{s}, \mathbf{a}) + \gamma E[V^*(\mathbf{s}')]$, so $V^* = \mathcal{B}V^*$, since value iteration converges to a policy that does not change given max action or expectation. This results in the equality above as this is only true if the chosen action gives the maximum expected reward for every state.

This point always exists and is always unique, but it is not always guaranteed we will reach it.

Value iteration (tabular)

We can prove that value iteration reaches V^* because $\mathcal{B}$ is a *contraction.*

Contraction A function f is a contraction if it satisfies the following,

$$d(f(x), f(y)) \leq kd(x, y)$$

where d is some distance measure.

In this case, for any V and $\bar{V}$, we have $||\mathcal{B}V - \mathcal{B}\bar{V}||_\infty \leq \gamma||V - \bar{V}||_\infty$.

From this, we see that for every value iteration update, the gap always gets smaller by γ. We can choose $\hat{V}$ to be V^*, which leaves us with $||\mathcal{B}V - V^*||_\infty \leq \gamma||V - V^*||_\infty$.

This means for every value iteration step, we get closer to the optimal value function, since the value iteration algorithm can be reformulated as applying the operator $\mathcal{B}$ until conversion.

Value iteration algorithm (using $\mathcal{B}$)

1. $V \leftarrow \mathcal{B}V$
2. Repeat

Fitted value iteration

Let's define a new operator Π as $\Pi V = \arg\min_{V \in \Omega} \frac{1}{2} \sum ||V'(\mathbf{s}) - V(\mathbf{s})||^2$. We see that Π is a *projection* onto Ω (in terms of l_2 norm).

Using Π and $\mathcal{B}$, we can rewrite the fitted value iteration update step as

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

We can now rewrite our algorithm using these operators.

Fitted value iteration algorithm (using Π and $\mathcal{B}$)

1. $V \leftarrow \Pi\mathcal{B}V$

We can visualize the contraction by $\mathcal{B}$ then the projection onto Ω. The contraction will try to take the point closer to the optimum, but the projection will take the point and put it back on the line corresponding to the set Ω (e.g. representing neural nets).

This causes the point to continuously 'rubber-band' between moving closer to the point and being projected back to the Ω set. Over iterations, this can actually cause the point moving away from the optimum, *leaving us with no convergence guarantee.*

This is despite the fact that $\mathcal{B}$ is a contraction w.r.t. ∞-norm ("max" norm) as $||\mathcal{B}V - \mathcal{B}\bar{V}||_\infty \leq \gamma||V - \bar{V}||_\infty$ and Π is a contraction w.r.t. l_2-norm (Euclidean distance) as $||\Pi V - \Pi\bar{V}||^2 \leq \gamma||V - \bar{V}||^2$. The composite operator $\Pi\mathcal{B}$ is not a contraction of any kind.

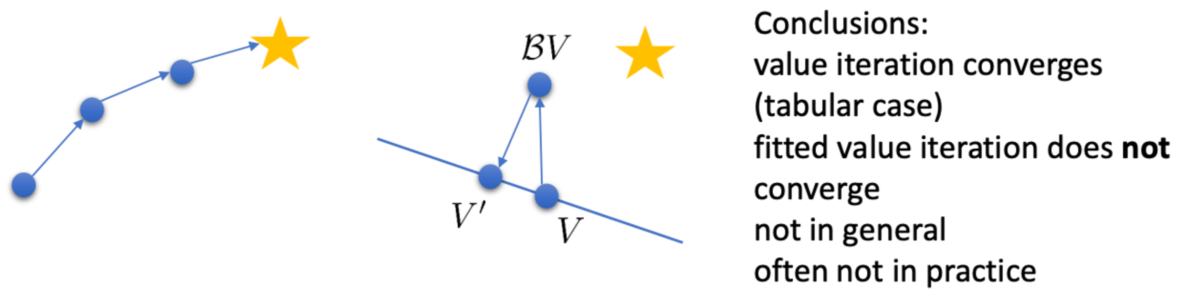
Fitted Q-iteration

Fitted Q-iteration has the same problem as fitted value iteration. The update step can be rewritten as $Q \leftarrow \Pi\mathcal{B}Q$, which means that it is prone to the same contraction/projection problem.

Online Q-iteration

This also applies to online q-iteration, despite the fact that the update step is different ($\phi \leftarrow \phi - \alpha\frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$).

One might think that the update step is just gradient descent, so it has to converge. However, it is not gradient descent, as there is no gradient through the target value $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}_i{}', \mathbf{a}_i{}')$.



Conclusions:
value iteration converges
(tabular case)
fitted value iteration does **not**
converge
not in general
often not in practice

Figure 2: Fitted value iteration update

A sad corollary There is a similar update step in the batch-actor critic algorithm to update the value function to use as the critic. However, there is the same issue of there being an l_∞ contraction when evaluating $\mathbf{y}_i$ for datapoints in a fitted bootstrapped policy iteration and the l_2 contraction when using the mean squared error to fit the parameters of your fitted value function $V_\phi(\mathbf{s})$.