CS294: Deep Reinforcement Learning [1]

Lecture Notes: Actor-critic Algorithm [2]

Fall 2017[3]

[1] Course Instructors: Sergey Levine

[2] Author: InnerPeace

[3] Un-official Lecture Notes

Keyphrases: Policy Gradient. Actor-critic. Discount factor.
This set of notes introduces the idea of improving policy gradient with a critic. We then discuss how to evaluate policy and introduce discount factors to deal with infinite horizon tasks. Lastly, we will introduce the *actor-critic* algorithm and how it works.

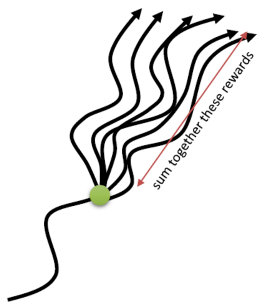
1 *Policy Evaluation*

From REINFORCE algorithm, we learned that the policy gradient is:

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\Big(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\Big)\right] \\ &= \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\hat{Q}_{i,t}\right] \qquad (1)\end{aligned}$$

Where the $\hat{Q}_{i,t}$ is the estimation of expected reward if we take action $\mathbf{a}_{i,t}$ in state $\mathbf{s}_{i,t}$ which is also referred as "reward to go". However it estimates how good the state-action pair is with a single sample, and the true "reward to go" from a state-action pair should be the expectation of many possible futures due to randomly chosen actions or stochastic dynamics of the environment (figure 1).

Figure 1: Illustration of actual "reward to go" from a state-action pair.



$$Q(\mathbf{s}_t,\mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t] \qquad (2)$$

If we know the true "reward to go" $Q(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$ and replace $\hat{Q}_{i,t}$ with it, we will obtain a better estimation of policy gradient with lower variance which is essential to better convergence.

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})Q(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right] \qquad (3)$$

We can also add baseline to the reward estimation:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})(Q(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) - b_t) \qquad (4)$$

We usually choose the average reward as the baseline, so b_t should average over $Q(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$:

$$b_t = \frac{1}{N}\sum_i Q(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) \qquad (5)$$

Besides, the expectation of Q-function over policy is the value function:

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

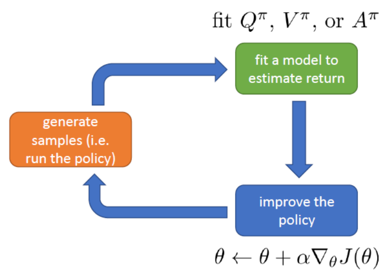
We denote that:

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

Where $A^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ is referred as *Advantage* which represent how much better $\mathbf{a}_t$ is than the average actions. Since b_t average over samples drawn from policy distribution π_θ, we can substitute b_t with $V(\mathbf{s}_t)$ and the policy gradient should be:

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})(Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) - V(\mathbf{s}_{i,t})) \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A^{\pi}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})\end{aligned} \tag{8}$$

Figure 2: Diagram of actor-critic. Comparing to REINFORCE algorithm, we fit Q^π or V^π in model fitting phrase instead of summing over the future rewards.


fit Q^π, V^π, or A^π
fit a model to estimate return
generate samples (i.e. run the policy)
improve the policy
$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Then the problem is how we achieve a better estimation of the *Advantage*.

Intuitively, if we need to compute something complicated, we can use neural network to estimate it. But should we directly estimate the *Advantage*? From equation 7, we learned that *Advantage* depends on state and action, and with more inputs, we need more parameters which results in higher variance. Can we estimate *Advantage* with just state or action? To do that, we can take advantage of the relationship between Q-function and value-function. Firstly, we rewrite $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ as sum of the reward at time-step t and expectation of future rewards.

$$\begin{aligned}Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) &= r(\mathbf{s}_t, \mathbf{a}_t) + \sum_{t'=t+1}^{T} E_{\pi_\theta}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t] \\ &= r(\mathbf{s}_t, \mathbf{a}_t) + E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}[V^{\pi}(\mathbf{s}_{t+1})]\end{aligned} \tag{9}$$

Besides, the expectation of future rewards is also the expectation of value function at the next time-step $t+1$ which is distributed according to the dynamics of the environment. Then we approximate the expectation with the value function at a specific state $\mathbf{s}_{t+1}$. Obviously, the approximation is not perfect, but from the perspective illustrated in figure 1, we take the first step and estimate the next step with value function which is good enough.

$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) \approx r(\mathbf{s}_t, \mathbf{a}_t) + V^{\pi}(\mathbf{s}_{t+1}) \tag{10}$$

Moreover, we can estimate *Advantage* with only state.

$$A^{\pi}(\mathbf{s}_t, \mathbf{a}_t) \approx r(\mathbf{s}_t, \mathbf{a}_t) + V^{\pi}(\mathbf{s}_{t+1}) - V^{\pi}(\mathbf{s}_t) \tag{11}$$

Then we just fit value function $\hat{V}^{\pi}_{\phi}(\mathbf{s})$ with neural network (parameters denoted as ϕ) in model-fitting phrase. And the objective is just the expectation of value function of the first state $\mathbf{s}_1$.

$$J(\theta) = E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)}[V^{\pi}(\mathbf{s}_1)] \tag{12}$$

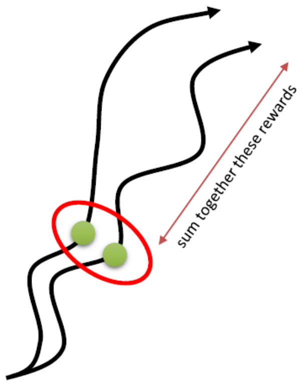
1.1 Evaluate with MC

If we need to fit $\hat{V}^{\pi}_{\phi}(\mathbf{s})$, what the training example should be? We know the input is $\mathbf{s}_t$, but what the target should be? *i.e.*, what's the ground truth of value function? Usually, we use Monte Carlo policy evaluation:

$$V^{\pi}(\mathbf{s}_t) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \tag{13}$$

Where we sample trajectories from the state $\mathbf{s}_t$ and average the "reward to go". However, it requires us to reset the simulator to the state $\mathbf{s}_t$ which is impractical. Then what about just the "reward to go" of current trajectory? Since we use neural network as the approximator, it will average the results of value functions with states quiet similar to each other (figure 3).

Figure 3: The same function should fit multiple samples, and since the two states are close to each other, the estimated value functions are also close.



We denote the target of trajectory i at time-step t as $y_{i,t}$:

$$y_{i,t} = \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \tag{14}$$

Then the training data is $\left\{\left(\mathbf{s}_{i,t}, \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})\right)\right\}$. We construct the learning process as supervised regression where the lost function is the squared distance:

$$\mathcal{L}(\phi) = \frac{1}{2}\sum_{i}\left|\left|\hat{V}^{\pi}_{\phi}(\mathbf{s}_i) - y_i\right|\right|^2 \tag{15}$$

1.2 Bootstrap mode

From the perspective of ideal target $y_{i,t}$ (*i.e.*, the true value function), can we fit a more accurate $\hat{V}_{\phi}$?

$$\begin{aligned} y_{i,t} &= \sum_{t'=t}^{T} E_{\pi_\theta}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_{i,t}] \\ &\approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \sum_{t'=t}^{T} E_{\pi_\theta}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_{i,t+1}] \\ &\approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + V^{\pi}(\mathbf{s}_{i,t+1}) \end{aligned} \tag{16}$$

Where we applied the same trick as equation 9, 10. The value function could be approximated by the reward of the current time-step plus the value function of the next time-step under the assumption that we took a deterministic action $\mathbf{a}_t$. Then we apply the "boot-strap" trick where we plug in the neural network estimator $\hat{V}_\phi$.

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}^\pi_\phi(\mathbf{s}_{i,t+1}) \tag{17}$$

The intuition is that if the $\hat{V}_\phi$ is better than cumulative rewards, then use it the estimate the target will "bootstrap" the performance. After that, the training date should be $\left\{\left(\mathbf{s}_{i,t}, r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}^\pi_\phi(\mathbf{s}_{i,t+1})\right)\right\}$ and we use the same loss function as equation 15.

In practice, we always initialize the neural network such that the output is small for the sake of stability, *e.g.* making the weights of the last linear layer to be small. Then for the first step, the output of $\hat{V}_\phi$ should look like $r(\mathbf{s}_0, \mathbf{a}_0)$. And the target for the second iteration should be $y_1 = r(\mathbf{s}_0, \mathbf{a}_0) + \hat{V}_\phi(\mathbf{s}_1)$ and $\hat{V}_\phi(\mathbf{s}_1)$ is something look like a reward. Iteratively, it will learn the value function over trajectories after many steps of iterations.

2 *Actor-critic Algorithm*

2.1 *Discount Factors*

However, there is a problem that if T (episode length) is infinite, $\hat{V}^\pi_\phi$ can get infinitely large in the case where we want infinite or approximate infinite episodes *e.g.* training Humanoid robot to run . Hence, we introduce the *discount factor* to address the issue. Suppose that someone grant you a fortune but he will give it to 10 years after, then you may consider to wait cause 10 years is not far away. But if he change it to 100 years after, you may care less cause your descendents can benefit from it. What about 1000 years? The number of years strongly affects the value of the fortune to you. Usually, we prefer the reward which is more reachable *i.e.*, better to get rewards sooner than later. By doing that, we introduce a number γ in front of value:

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma\hat{V}^\pi_\phi(\mathbf{s}_{i,t+1}) \tag{18}$$

Where $\gamma \in [0,1]$, and usually 0.99 works pretty well. From the perspective of MDP, γ changes the transition probability:

$$\hat{p}(\mathbf{s}'|\mathbf{s}, \mathbf{a}) = \gamma p(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \tag{19}$$

Is γ a parameter of the *critic*? Actually, γ is just the property of the MDP and highly depends on the problem setting.

How can we plug γ in policy gradient (equation 1)? There are two options:

Option 1:

After we apply the property of causality as equation 1, we can plug γ directory in front of the reward.

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\Big[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\Big(\sum_{t'=t}^{T}\gamma^{t'-t} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\Big)\Big] \tag{20}$$

Option 2:

Another option is to plug in γ before temporal decomposed policy gradient.

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \frac{1}{N}\sum_{i=1}^{N}\Big[\Big(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\Big)\Big(\sum_{t=1}^{T}\gamma^{t-1} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\Big)\Big] \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\Big(\sum_{t'=1}^{T}\gamma^{t'-1} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\Big) \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\gamma^{t-1}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\Big(\sum_{t'=t}^{T}\gamma^{t'-t} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\Big) \end{aligned} \tag{21}$$

Obviously, the two options are not the same. Option 2 discounts the gradient from step 1 *i.e.*, later steps matter less, which is quiet reasonable. However, we usually use option 1 in practice which is approximating the average reward as in the infinite horizon tasks, and Philip Thomas et al. discussed it in the paper "Bias in natural actor-critic algorithms" (link).

2.2 *Batch Actor-critic algorithm*

With policy evaluation with estimated value function, we derived the batch actor-critic algorithm by plugging value function estimation to REINFORCE algorithm:

Algorithm 1: Batch Actor-critic Algorithm

```
1: while Not converged do
2:     sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ by running the policy.
3:     Fit $\hat{V}^\pi_\phi(\mathbf{s})$ to sampled reward sums.
4:     Evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) \approx r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
5:     $\nabla_\theta J(\theta) \approx 1/N \sum_{i=1}^N \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
6:     $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
7: end while
8: return trained $\pi_\theta$
```

We often refer $\hat{V}^\pi_\phi(\mathbf{s})$ as the *critic*. During training, we generate batch of trajectories. And for the $\hat{V}^\pi_\phi(\mathbf{s})$ fitting step, one can choose to update $\hat{V}_\phi^\pi(\mathbf{s})$ for many iterations, or just train one-time for speeding up. For the fitting method, bootstrap is more popular than Monte Carlo evaluation.

2.3 *Online Actor-critic Algorithm*

We can also derive the on-line version of actor-critic algorithm, where after each time we take a step, we estimate the value function and update the policy.

Algorithm 2: Online Actor-critic Algorithm

```
1: while Not converged do
2:     Take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$.
3:     Update $\hat{V}_\phi^\pi$ with $r + \gamma \hat{V}_\phi^\pi(\mathbf{s}')$.
4:     Evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) \approx r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}') - \hat{V}_\phi^\pi(\mathbf{s})$
5:     $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
6:     $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
7: end while
8: return trained $\pi_\theta$
```

2.4 *Architecture Design*

We need to estimate *critic* $\hat{V}_\phi^\pi(\mathbf{s})$ and the policy $\pi_\theta(\mathbf{a}|\mathbf{s})$. Naturally, we can achieve the estimation with two explicit networks (figure 4), it's simple and stable for training without heavy tuning, but there is no shared features between actor and critic. Another option is taking advantage of the mutual features as in common CV tasks where the basic features like line, circle is mutual for different task. With the shared network design (figure 5), we use two heads as the prediction of actor and critic which is more efficient [4]. The shortage is that the shared network is tougher to train cause we push different kinds of gradients to the backbone.

Figure 4: Two network design

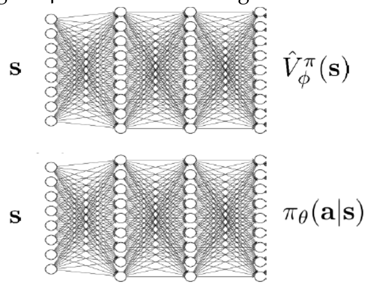
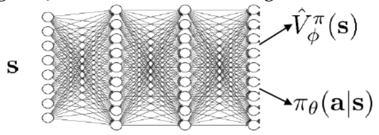

$\mathbf{s}$
$\hat{V}_\phi^\pi(\mathbf{s})$
$\mathbf{s}$
$\pi_\theta(\mathbf{a}|\mathbf{s})$

Figure 5: Shared network design


$\mathbf{s}$
$\hat{V}_\phi^\pi(\mathbf{s})$
$\pi_\theta(\mathbf{a}|\mathbf{s})$

[4] AlphaGo Zero used the shared network to train the actor and critic

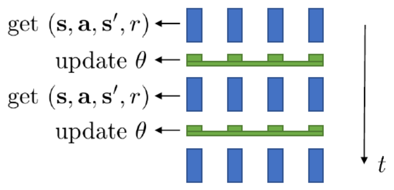
2.5 *Parallelism*

We learned the the two main problems of policy gradient are high variance and slow convergence, and we are trying to address the high variance issue with causality and baseline trick. However, for the one-line actor-critic algorithm where we estimate $\hat{V}_\phi^\pi(\mathbf{s})$ and policy $\pi_\theta(\mathbf{a}|\mathbf{s})$ with data of a single step, it contributes to higher the variance. One way to deal with that is working with a batch *i.e.*, increasing the samples to reduce variance. A practical way to do it is using parallel workers.

Synchronized parallel actor-critic:

Suppose that we have four workers (figure 6) and each worker works independently. During training, each worker takes a step and collects data $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$. When all the workers finishing collecting data, the main worker collects the four samples and updates the policy. In this way, we have four times examples. Obviously, the more worker we have, the lower variance and faster convergence it will be. However parallelism is not trivial in some cases, and we need to make the trade-off.

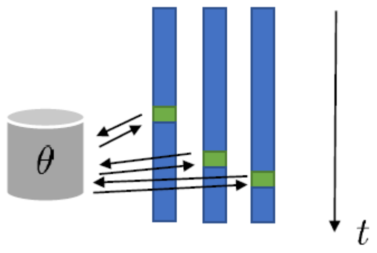
Figure 6: Synchronized parallel actor-critic


get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
update θ
get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
update θ
t

Asynchronous parallel actor-critic:

The synchronized version could be comparably slow, we have to wait until all workers has finished data collecting to update the policy. Another more advanced scheme is *asynchronous parallelism* (figure 7) where we have multiple workers and a parameter server which is responsible for storing and updating parameter θ. Once any worker has finished collecting data, it sends the estimated gradient to parameter server and updating policy immediately. At the same time, the other worker may still works with old parameters.

Figure 7: Asynchronous parallel actor-critic



2.6 State-dependent Baselines

The REINFORCE algorithm (equation 20) is unbiased estimation w.r.t policy gradient, since adding the constant baselines has no impact on gradients. But it has high variance due to single-sample estimation of "reward to go". The actor-critic algorithm (1) has lower variance due to critic. However, it's not biased as the neural network approximation is not perfect. One clever way to combining the advantage of the algorithms is using $\hat{V}^{\pi}_{\phi}(\mathbf{s})$ as the baseline and keeping the naive "reward to go" estimation.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) - \hat{V}^{\pi}_{\phi}(\mathbf{s}_{i,t}) \right) \right] \tag{22}$$

Where the baselines is $\hat{V}^{\pi}_{\phi}(\mathbf{s})$ which is only state-dependent, and during differentiating w.r.t θ, it could be seen as a *constant*, just like the average baseline b. And the result enjoys the benefits of unbiased estimation and lower variance compared to REINFORCE algorithm.

2.7 Action-dependent Baselines

Adding only state-dependent baselines, we will have an unbiased and higher variance estimation of *advantage*. But if we also incorporate action into the baseline, the difference between "reward to go" and baselines gets lower, even goes to zero in expectation if critic is correct. [5]

[5] If the policy is deterministic, it will probably go to zero.

$$\hat{A}^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{Q}^{\pi}_{\phi}(\mathbf{s}_t, \mathbf{a}_t) \tag{23}$$

However, the resulting policy gradient is biased as the baselines $\hat{Q}^{\pi}_{\phi}(\mathbf{s}_t, \mathbf{a}_t)$ depends on action. But we can manage to make it unbiased with adding a extra term[6].

[6] More about variance reduction later

$$\begin{aligned}\nabla_\theta J(\theta) \approx & \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\Big(\sum_{t'=t}^{T}\gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) - \hat{Q}^{\pi}_{\phi}(\mathbf{s}_{i,t})\Big)\right] \\ & + \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta E_{\mathbf{a}\sim\pi_\theta(\mathbf{a}_t|s_{i,t})}\left[\hat{Q}^{\pi}_{\phi}(\mathbf{s}_{i,t}, \mathbf{a}_t)\right]\end{aligned} \tag{24}$$

Following the proof of constant baselines is unbiased (lecture 3), we could obtain that:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\hat{Q}^{\pi}_{\phi}(\mathbf{s}_{i,t}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta E_{\mathbf{a}\sim\pi_\theta(\mathbf{a}_t|s_{i,t})}\left[\hat{Q}^{\pi}_{\phi}(\mathbf{s}_{i,t}, \mathbf{a}_t)\right] \tag{25}$$

Then with equation 24, we use the action-dependent baselines provided the second term can be evaluated *e.g.* the case where $\hat{Q}^{\pi}_{\phi}$ is quadratic and the policy π_θ is distributed according to Gaussian distribution. It's referred as *control variates*.

2.8 *Generalized Advantage Estimation*

The actor-critic (algorithm 1) has lower variance but higher bias due to the value function estimation is not accurate, and the REINFORCE with state-dependent baselines (equation 22) has no bias but higher variance due to single-sampled estimation of "reward to go".

$$\hat{A}^{\pi}_{C}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}_{t+1} - \hat{V}^{\pi}_{\phi}(\mathbf{s}_t)) \tag{26}$$

$$\hat{A}^{\pi}_{MC}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}^{\pi}_{\phi}(\mathbf{s}_t)) \tag{27}$$

So can we combine these ideas to control the bias-variance tradeoff?

The intuition is that starting from a state, the further to the future, the higher the variance will be. So at the closer region to the starting point, the estimation has low variance. Then we can try to draw a "line" after several steps to stopping estimating with single sample and plug in the value function to estimate for the future.

Another way to look at it is that, the difference between equation 26 and 27 is, actor-critic is the reward of *one-step* plus discounted future value as the "reward to go", and REINFORCE with value critic is cumulative rewards of *all time-steps* as the "reward to go". One

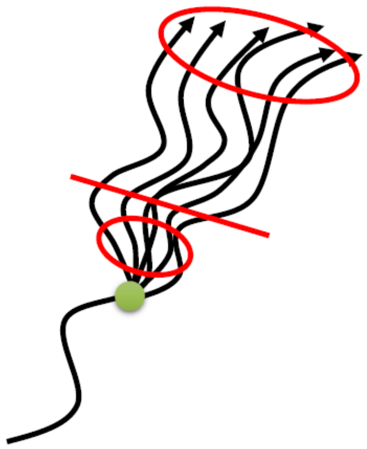


Figure 8: Illustration of bias-variance trade-off with a cutting line not far from the current state

straightforward way to control the trade-off is changing the number of steps (from 1 to T) to sum and plugging in discounted value for the future. Sometimes, it's referred as *n-step returns*.

$$\hat{A}_n^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) + \gamma^n \hat{V}_{\phi}^{\pi}(\mathbf{s}_{t+n} - \hat{V}_{\phi}^{\pi}(\mathbf{s}_t)) \tag{28}$$

In practice, choosing $n > 1$ often works better. [7]

[7] During early stages of training, choose $n = 4$ or 5 contributes faster learning.

Going further from the idea, what about choosing different n and finding a way to blend all? It leads the *generalized advantage estimation* where we take n from current time-step to the end T and blend all with wighted combination of n-step returns.

$$\hat{A}_{GAE}^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{n=1}^{T} w_n \hat{A}_n^{\pi}(\mathbf{s}_t, \mathbf{a}_t) \tag{29}$$

With the intuition of the closer to current time step, lower variance it is, we choose the weights proportional to exponent of parameter λ:

$$w_n = \lambda^{n-1}(1 - \lambda) \tag{30}$$

With some derivation [8], the GAE is:

[8] Refer paper of Schulman et al. for details of derivation.

$$\hat{A}_{GAE}^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{n=1}^{T} (\gamma\lambda)^{t'-t} \delta_{t'}$$

where

$$\delta_{t'} = r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) + \gamma \hat{V}_{\phi}^{\pi}(\mathbf{s}_{t'+1} - \hat{V}_{\phi}^{\pi}(\mathbf{s}_{t'})$$

When $\lambda = 0$, it reduces to equation 26 and $\lambda = 1$, it is the same as equation 27. Then we can choose $\lambda \in [0, 1]$ to control the bias-variance trade-off.

3 *Suggested Readings*

Classic Papers

- Sutton et al. Policy gradient methods for reinforcement learning with function approximation: actor-critic algorithms with value function approximation. link.

Recent Deep RL

- Mnih et al. Asynchronous methods for deep reinforcement learning: A3C with parallel online actor-critic. link.
- Schulman et al. High-dimensional continuous control using generalized advantage estimation: batch-mode actor-critic with blended Monte Carlo and function approximator returns. link.

- Gu et al. Q-prob: sample efficient policy-gradient with an off-policy critic: policy gradient with Q-function control variate. link.