Addison Kalanther (addikala@berkeley.edu)

Lecture 6: Actor-Critic Algorithms

This lecture first introduces us to actor-critic algorithms.

Improving the policy gradient

The policy gradient involved us estimating the gradients of a weighted log-likelihood, using reward-to-go as the weights. Rather than use the reward-to-go of a single trajectory, what if we used the expected value from the given state? Can we get a better estimate?

Q-function

We calculate the expected reward when taking action $\mathbf{a}_t$ from state $\mathbf{s}_t$ using the following equation. This is known as the Q-function $Q(\mathbf{s}_t, \mathbf{a}_t)$.

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

Value function

We can calculate the expected reward from state $\mathbf{s}_t$ by taking the expectation over the action distribution from that state. This is known as the value function $V(\mathbf{s}_t)$.

$$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$$

Advantage function

Advantage functions give us the expected advantage from taking action $\mathbf{a}_t$ from state $\mathbf{s}_t$.

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

The better the advantage function estimate, the lower the variance in the policy gradient.

$$\nabla * \theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla * \theta \log \pi_\theta(\mathbf{a}_t^i \mid \mathbf{s}_t^i) \underline{A^\pi(\mathbf{s}_t^i, \mathbf{a}_t^i)}$$

Value function fitting

We can rewrite the q-function $Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$ and advantage $A^\pi(\mathbf{s}_t, \mathbf{a}_t)$ using only the value function $V^\pi(\mathbf{s}_t)$.

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + V^\pi(\mathbf{s}_t)$$

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

This shows us that we only need to fit a value function. We can fit a value function $\hat{V}^\pi(\mathbf{s})$ using a deep neural net.

Policy evaluation

We can get an estimate of the value function $V^\pi(\mathbf{s}_t)$ for a given policy π by taking a monte carlo estimate $V^\pi(\mathbf{s}_t) \approx \sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$, which admittedly is not as good as an average over N samples, but it is still decent.

Fit these to a neural net using supervised learning, taking the current state $\mathbf{s}_t$ as input and the reward-to-go as the output.

Bootstrapped estimate

Rather than have our target y be the reward-to-go, we can use the reward of the current state-action pair $r(\mathbf{s}_t, \mathbf{a}_t)$ and the previously fitted value function to get a lower variance approximation of the expected value function. Our ideal target becomes

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

This is sometimes referred to as the “bootstrapped” estimate.

Examples

For a game of backgammon or chess, the fitted value function $\hat{V}_\phi^\pi(\mathbf{s}_t)$ could be the expected outcome given board state $\mathbf{s}_t$.

From Evaluation to Actor-Critic

Aside: discount factors for policy gradients

We have been mostly talking about processes with a finite horizon T, but what if we have an infinite-step process?

A simple trick is to force the robot to get rewards sooner than later by using a *discount factor* $\gamma \in [0, 1]$. $\gamma = 0.99$ often works well.

This turns our old target estimated value $y_{i,t}$ to

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

This changes the MDP and can be interpreted as the robot having a $1 - \gamma$ chance to end its run from any given state $\mathbf{s}_t$.

Batch Actor-Critic Algorithm

The batch actor-critic algorithm is the most basic of the algorithms. It uses monte carlo evaluation to fit a value function $\hat{V} * \phi^\pi(\mathbf{s}_t)$ using a neural net. Then, using the fitted function to evaluate the advantage $\hat{A}_\phi^\pi(\mathbf{s}_t, \mathbf{a}_t)$. We then use the advantage to perform our policy gradient step.

The algorithm

1. sample $\{\mathbf{s}_t, \mathbf{a}_t\}$ from $\pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$ (run the agent)
2. fit $\hat{V} * \phi^\pi(\mathbf{s}_t)$ to sample reward sums
3. evaluate $\hat{A}_\phi^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{t+1}) - \hat{V} * \phi^\pi(\mathbf{s}_t)$ for all $\mathbf{s}_t, \mathbf{a}_t$ pairs
4. $\nabla * \theta J(\theta) \approx \sum_t \nabla * \theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \hat{A}_\phi^\pi(\mathbf{s}_t, \mathbf{a}_t)$
5. $\theta \leftarrow \theta + \alpha \nabla * \theta J(\theta)$
6. Repeat

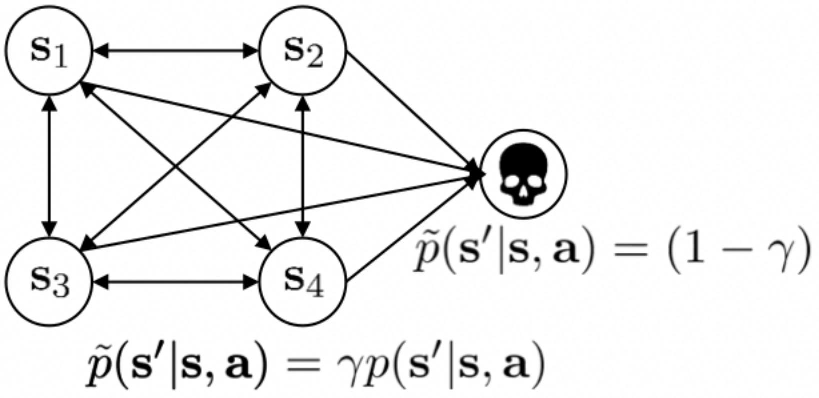
$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
$\mathbf{s}_4$
$\tilde{p}(\mathbf{s}'|\mathbf{s},\mathbf{a}) = (1-\gamma)$
$\tilde{p}(\mathbf{s}'|\mathbf{s},\mathbf{a}) = \gamma p(\mathbf{s}'|\mathbf{s},\mathbf{a})$

Figure 1: Discount as death

Online Actor-Critic Algorithm

Online actor-critic differs from batch actor-critic in that the policy is updated for every sampled action.

The algorithm

1. take action $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$, get $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, r)$
2. update $\hat{V} * \phi^\pi(\mathbf{s}_t)$ using target $r + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{t+1})$
3. evaluate $\hat{A}_\phi^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{t+1}) - \hat{V} * \phi^\pi(\mathbf{s}_t)$
4. $\nabla * \theta J(\theta) \approx \sum_t \nabla * \theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \hat{A}_\phi^\pi(\mathbf{s}_t, \mathbf{a}_t)$
5. $\theta \leftarrow \theta + \alpha \nabla * \theta J(\theta)$
6. Repeat

Actor-critic design decisions

Architecture

In an actor-critic algorithm, we have two networks: one for the value function $\hat{V} * \phi^\pi(\mathbf{s}_t)$ and one for the policy $\pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$. We can either use two separate networks or a shared network with two different heads.

With two separate networks, it would be easier to train and be more stable (due to no opposing gradients), but there would be no shared features between actor & critic (no sharing of information).

Online actor-critic in practice

Online actor-critic often does not do a policy gradient step per samples. Instead, it takes advantage of batching using parallel workers.

Synchronized parallel workers have n threads going one step at a time and then updates θ once all threads complete their run.

Asynchronous parallel workers don't wait for each other and update once it gets an sample from a thread. While this does mean that the gradients for some threads won't be for the current policy due to race conditions, if the sampling is fast enough, it should not introduce too much bias and still works.

Off-policy assumption

Basic actor-critic algorithms are on-policy. However, what if we can use old, previously seen transitions not from the current policy. If we can, we recover an off-policy algorithm.

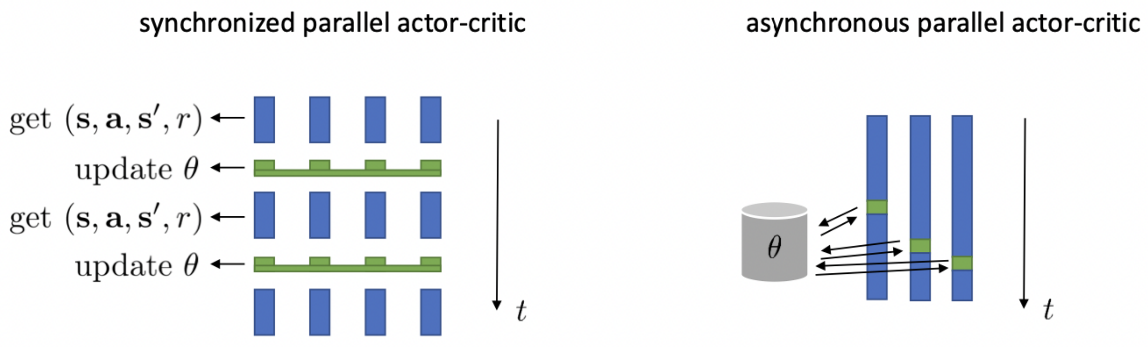
synchronized parallel actor-critic
asynchronous parallel actor-critic
get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
update θ
get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
update θ
t
θ
t

Figure 2: Parallel actor critic

We can store transitions $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, r)$ from all past policies in a *replay buffer* $\mathcal{R}$.

Since the value function $\hat{V} * \phi^{\pi}(\mathbf{s}_t)$ is no longer specific to a policy, we want to fit a Q-function $\hat{Q}^{\pi}_{\phi}(\mathbf{s}_t, \mathbf{a}_t)$ instead.

Online off-policy actor-critic algorithm

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a} \mid \mathbf{s})$, get $(\mathbf{s}.\mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, r_t\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}^{\pi}_{\phi}$ using targets $y_i = r_i + \gamma \hat{Q}^{\pi}_{\phi}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$
   - Note that $\mathbf{a}_{t+1} \sim \pi_\theta(\mathbf{a}_{t+1} \mid \mathbf{s}_{t+1})$ is sampled from the current policy, **not** from the replay buffer $\mathcal{R}$
4. $\nabla * \theta J(\theta) \approx \frac{1}{N} \sum_t \nabla * \theta \log \pi_\theta(\mathbf{a}^{\pi}_t \mid \mathbf{s}_t) \hat{Q}^{\pi}_{\phi}(\mathbf{s}_t, \mathbf{a}^{\pi}_t)$ where $\mathbf{a}^{\pi}_t \sim \pi_\theta(\mathbf{a} \mid \mathbf{s}_t)$
5. $\theta \leftarrow \theta + \alpha \nabla * \theta J(\theta)$
6. Repeat

A problem with this algorithm is that $\mathbf{s}_t$ from buffer $\mathcal{R}$ does not come from the π_θ distribution. There is nothing we can do about this, but this is okay, as we want the optimal policy on $p_\theta(\mathbf{s})$, but we instead get an optimal policy on a *broader* distribution.

Aside: Reparameterization trick We can also use the *reparameterization trick* in step 4 to better estimate the integral of our log probabilities.

A fundamental issue with our Monte Carlo integration is that our $\nabla * \theta J(\theta)$ estimate is often *very high variance*. Even when we take the average over a large number of samples, we might miss some rare states that greatly affect our expectation.

By rewriting the expectation as a function of the random variable p, we can chose it's distribution and control the estimate such that the chances of drawing formerly rare states are more common in the p distribution.

Using this, we can rewrite the expectation of some arbitrary G_θ as the following

$$G_\theta = \nabla * \theta E_{\epsilon \sim p}[J(\theta, \epsilon)] = E_{\epsilon \sim p}[\nabla * \theta J(\theta, \epsilon)]$$

The reparameterization trick can alleviate the high variance issue, and even if $\nabla * \theta J(\theta, \epsilon)$ is not a good estimator of G_θ, if large contributions from ϵ are very rare, we don't see it often enough in optimization for it to matter all too much (it would essentially become noise).

We often see the reparameterization trick in the context of VAE's, where we need to sample from a large space, and we control that sampling and the probabilities of states by choosing our p distribution.

In reinforcement learning, we can control our Monte Carlo integration so we get more or less rare / desired state-action samples by choosing our p distribution, rather than having it depend on θ.

Critics as baselines

Actor-critics use a learned critic as a baseline ($r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}_{t+1}) - \hat{V} * \phi^{\pi}(\mathbf{s}_t)$) during the policy gradient update. This results in a lower variance (due to the critic), but it would not be unbiased if the critic is not perfect.

- This is because the next state used in $\hat{V}^{\pi}_{\phi}(\mathbf{s}_{t+1})$ is dependant on the current policy's action, which is dependent on θ.

Base policy gradients which use some constant as a baseline ($\left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_t, \mathbf{a}_t)\right) - b$) has no bias, but has a higher variance due to its single-sample estimate.

We can use the critic without the bias by using the critic as the baseline $\left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_t, \mathbf{a}_t)\right) - \hat{V} * \phi^{\pi}(\mathbf{s}_t)$. This has the advantage of being unbiased while having a lower variance (baseline is closer to rewards).

Control variates: action-dependent baselines

Action-dependent baselines can be used to lower variance at the cost of introducing some bias.

Take $\hat{A}^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{Q}^{\pi}_{\phi}(\mathbf{s}_t, \mathbf{a}_t)$, this goes to zero in expectation if critic is correct. However, critic is never 100% correct, so it is never correct.

Let's go back to our critic advantage function $\hat{A}^{\pi}_{\mathrm{C}}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}_{t+1}) - \hat{V} * \phi^{\pi}(\mathbf{s}_t)$, this has the advantage of being lower variance, but higher bias if wrong (which it always is).

Our single estimate Monte Carlo advantage $\hat{A}^{\pi}_{\mathrm{MC}}(\mathbf{s}_t, \mathbf{a}_t) = \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_t, \mathbf{a}_t)\right) - \hat{V} * \phi^{\pi}(\mathbf{s}_t)$ has no bias, but has a higher variance.

Can we combine these two to control the bias/variance tradeoff?

Generalized advantage estimation (GAE)

Let's first look at an n-step critic advantage function

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

By controlling n, we control how much of the single sample Monte Carlo estimate we want to use to estimate the advantage function. With higher n, we introduce less bias and more variance. With lower n, we introduce more variance and less bias.

However, what if we don't have to choose just one n? Instead, let's choose to cutoff our paths everywhere all at once!

This is what the **Generalized Advantage Estimation (GAE)** entails. It is a weighted combination of n-step returns of the form

$$\hat{A}^{\pi}_{\mathrm{GAE}}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{n=1}^{\infty} w_n \hat{A}^{\pi}_{n}(\mathbf{s}_t, \mathbf{a}_t)$$

We prefer cutting earlier for less variance, so we would like our weights to have an exponential falloff $w_n \propto \lambda^{n-1}$.

Using this, we can rewrite our GAE as

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

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

The $\gamma\lambda$ term looks like our discount factor γ in typical reward-to-go. From this, we can conclude that discount is a form of variance reduction as well.

Actor-critic suggested readings

Classic papers

- Sutton, McAllester, Singh, Mansour (1999). Policy gradient methods for reinforcement learning with function approximation: actor-critic algorithms with value function approximation

Deep reinforcement learning actor-critic papers

- Mnih, Badia, Mirza, Graves, Lillicrap, Harley, Silver, Kavukcuoglu (2016). Asynchronous methods for deep reinforcement learning: A3C – parallel online actor-critic
- Schulman, Moritz, L., Jordan, Abbeel (2016). High-dimensional continuous control using generalized advantage estimation: batch-mode actor-critic with blended Monte Carlo and function approximator returns
- Gu, Lillicrap, Ghahramani, Turner, L. (2017). Q-Prop: sample-efficient policy- gradient with an off-policy critic: policy gradient with Q-function control variate