COS 435/ECE 433 Week 6 Precept Notes

March 21, 2024

1 Advantage Actor-Critic (A2C)

In the lecture, we studied the vanilla actor-critic method, which maintains an additional Q-function Q_ϕ by a critic network, and incorporates this into the following estimation of the gradient:

$$\hat{g}_t = \sum_{h\geq 0} \nabla_\theta \log \pi_{\theta^t}(a_h^t|s_h^t) Q_{\phi_t}(s_h^t, a_h^t).$$

This is motivated by the fact that $\mathbb{E}[\sum_{h\geq 0} \nabla_\theta \log \pi_t(a_h|s_h) Q_{\pi_t}(s_h, a_h)]$ is an unbiased estimator for $\nabla J(\theta) := \mathbb{E}[\sum_{h\geq 0} r_h(s_h, a_h)]$. In this section, we will introduce another variant of the classical actor-critic method. First, let our recall the following lemma we proved in Precept 3:

Lemma 1.1 (Causality)**.** $\mathbb{E}_{\pi_\theta}[b_h(s_0, a_0, \dots, a_{h-1}s_h)\nabla_\theta \log \pi_\theta(a_h|s_h)] = 0$ *for all* $h \in [H]$.

Applying the causality lemma, we have

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{h=0}^{H} \nabla_\theta \log \pi_\theta(a_h|s_h)\left\{Q^{\pi_\theta}(s_h, a_h) - V^{\pi_\theta}(s_h)\right\}\right].$$

We define the *advantage function* $A^\pi(s_h, a_h) := Q^{\pi_\theta}(s_h, a_h) - V^{\pi_\theta}(s_h)$. A natural unbiased estimator for the gradient would then be

$$g_t^A = \sum_{h\geq 0} \nabla_\theta \log \pi_{\theta^t}(a_h^t|s_h^t) A_{\pi_t}(s_h^t, a_h^t).$$

Intuitively, when we have access to an accurate enough approximation of A^{π_θ}, such an estimation allows us to further reduce the variance of the gradient, as we "subtract" a mean-zero noise that is a function of state from the estimate. Motivated by this, we obtain the Advantage Actor-Critic (A2C) method. Usually, A2C maintains two neural networks during the update: the first one acts as the "critic", i.e. it outputs the current value function $V_\phi(s_t)$ (different from vanilla actor-critic, which maintains $Q_\phi(s, a)$), and the second one is the "actor", which represents our policy $\pi_\theta(\cdot|s_t) : \mathcal{S} \to \Delta^{\mathcal{A}}$. In every episode, we update the actor with the following gradient:

$$\hat{g}_t^A = \eta \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \left(\hat{Q}(s_t, a_t) - V_\phi(s_t)\right).$$

Here $\hat{Q}(s_t, a_t)$ is a surrogate of Q-value function constructed using the maintained V_ϕ. Usually, we have two methods to estimate Q:

- $\sum_{h\geq t} \gamma^{h-t} r(s_h, a_h)$,

- $r(s_t, a_t) + \gamma \cdot V_\phi(s_{t+1})$,

note that the first estimate incurs a larger variance but a zero bias, while the second one might suffer from a larger bias (estimation of V_ϕ), but usually a smaller variance (no randomness from the transition after $t+1$). The critic V_ϕ is updated by

$$\min_\phi \sum_t \left(r(s_t, a_t) + \gamma V_\phi^{old}(s_{t+1}) - V_\phi(s_t) \right)^2,$$

or by

$$\min_\phi \sum_t \left(V_\phi(s_t) - \sum_{h \geq t} \gamma^{h-t} r(s_h, a_h) \right)^2.$$

2 Deep Deterministic Policy Gradient (DDPG)

Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:
            $\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$
        Update the target networks:
            $\theta^{Q'} \leftarrow \tau \theta^Q + (1-\tau)\theta^{Q'}$
            $\theta^{\mu'} \leftarrow \tau \theta^\mu + (1-\tau)\theta^{\mu'}$
    end for
end for
```

Figure 1: DDPG algorithm.

Regular actor-critic algorithms work well when we have only finite actions, as it is essentially learning a mapping from $\mathcal{S}$ to $\mathbb{R}^{\mathcal{A}}$, which can be well-approximated by a neural network with an $\mathcal{A}$-dimensional output. However, when it comes to the case that we have infinite candidate actions, e.g. in the scenario of robotic control, how can we learn the optimal action given the current state?

DDPG offers a viable solution. DDPG maintains 4 neural networks: an estimation Q_ϕ as the critic, a policy network $\mu_\theta : \mathcal{S} \to \mathcal{A}$ as the actor, and two corresponding target networks, $Q_{\phi'}$ and $\mu_{\theta'}$. By deterministic, we mean μ does not include randomness, as it is hard to approximate a probability with a continuous support with a simple neural network.

There are 3 major differences between DDPG and A2C:

- **Actor update.** To adapt to continuous action space and deterministic policy, in DDPG, the actor is updated by the continuous version of policy gradient:

$$\Delta\theta = \eta \cdot \mathrm{mean}_t\big(\nabla_a Q_\phi(s_t, \mu_\theta(s_t)) \cdot \nabla_\theta \mu_\theta(s_t)\big).$$

  Also, refer to the lecture notes for details.

- **Replay buffer & Exploration strategy.** To mitigate the catastrophic forgetting issue in actor-critic (as you might have already seen in previous examples), DDPG "borrows" the concept of replay buffer from DQN, and correspondingly an exploratory strategy when choosing action. In every step, DDPG samples a (s_t, a_t, s_{t+1}, r_t) from the replay buffer, and update the policy. Similar to the ε-greedy exploration in DQN, DDPG adds noise to the chosen action given by the actor to improve exploration:

$$a_t = \mu_\theta(s_t) + \varepsilon_t,$$

  where ε_t is a Gaussian noise. Then the agent receives the new s_{t+1} with reward and stores it in the replay buffer.

- **Target network & Critic update.** To mitigate inconsistency during temporal difference backups, DDPG borrows the concept of target network from double DQN. Specifically, in every episode, DDPG updates the critic Q_ϕ by performing gradient descent with the following improved Bellman loss:

$$L(\phi) = \sum_{t\geq 0} \Big(r(s_t, a_t) + \gamma \cdot Q_{\phi'}(s_{t+1}, \mu_{\theta'}(s_{t+1})) - Q_\phi(s_t, a_t)\Big)^2,$$

  Note that the TD target is calculated by the target networks $Q_{\phi'}$, and $Q_{\phi'}$ should not be trained here! This would decrease inconsistency during training. However, different from DQN, which updates the target network with a slower frequency, DDPG chooses to update $Q_{\phi'}$ and $\mu_{\theta'}$ in every episode but with a slower rate after updating ϕ and θ:

$$\phi' = \tau\phi + (1-\tau)\phi', \qquad \theta' = \tau\theta + (1-\tau)\theta'.$$

  Usually τ is set to be some small constant, e.g. 0.005, to prevent rapid updates to the target networks.

3 Twin Delayed DDPG (TD3)

Similar to DDPG, TD3 also maintains a continuous actor update with a deterministic policy gradient and a replay buffer with an exploratory strategy. The difference between TD3 and DDPG lies in the critic update. Unlike Double DQN, due to the slow update of the policy (the θ in previous examples), the current and target values still remain similar even when using a double Q-update technique. While the implementation of an independent target network allows for less biased value estimation, even an unbiased estimate with high variance can still lead to future overestimations in local regions of state space, which in turn can negatively affect the global policy.

To address this issue, TD3 updates the actor/critic network in the following way:

```
Algorithm 1 Twin Delayed DDPG
 1: Input: initial policy parameters $\theta$, Q-function parameters $\phi_1$, $\phi_2$, empty replay buffer $\mathcal{D}$
 2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
 3: repeat
 4:   Observe state $s$ and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
 5:   Execute $a$ in the environment
 6:   Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
 7:   Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
 8:   If $s'$ is terminal, reset environment state.
 9:   if it's time to update then
10:     for $j$ in range(however many updates) do
11:       Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:       Compute target actions
              $a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \quad \epsilon \sim \mathcal{N}(0, \sigma)$
13:       Compute targets
              $y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$
14:       Update Q-functions by one step of gradient descent using
              $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$
15:       if $j \mod$ policy_delay $= 0$ then
16:         Update policy by one step of gradient ascent using
              $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$
17:         Update target networks with
              $\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$
              $\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$
18:       end if
19:     end for
20:   end if
21: until convergence
```

Figure 2: TD3 algorithm.

- TD3 maintains **6 neural networks**: Q_{ϕ_1}, Q_{ϕ_2} as two independent update for the Q function, $Q_{\phi'_1}, Q_{\phi'_2}$ as their corresponding target networks, and $\mu_\theta, \mu_{\theta'}$ as the actor and its target network. Q_{ϕ_1} and Q_{ϕ_2} are independently initialized, and all target networks are initialized to be the same as their corresponding counterparts. When updating Q-functions, in every episode, DDPG updates the critic

Q_ϕ by performing gradient descent with the following improved Bellamn loss:

$$L(\phi_i) = \sum_t \left(r(s_t, a_t) + \gamma \cdot \min_{i=1,2} Q_{\phi_i'}(s_{t+1}, \tilde{a}_{t+1}) - Q_{\phi_i}(s_t, a_t) \right)^2,$$

where $\tilde{a}_t = \mu_{\theta'}(s_{t+1}) + \text{clip}_{[-c,c]}(\mathcal{N}(0, \tilde{\sigma}))$, which is different from the TD target in DDPG, as it take the minimum of the two future Q-values and a "disturbed" future action. Here we clip $\tilde{a}_t$ to prevent it from going too far. Intuitively, TD3 evaluates the Q-value of s_{t+1} in a more "conservative" way.

- When updating μ_θ, TD3 only utilizes Q_{ϕ_1}:

$$\Delta\theta = \eta \cdot \sum_t \big(\nabla_a Q_{\phi_1}(s_t, \mu_\theta(s_t)) \cdot \nabla_\theta \mu_\theta(s_t) \big),$$

where η is the stepsize. At the end of every episode, similar to DDPG, all parameters are updated by

$$\phi_i' = \tau\phi_i + (1-\tau)\phi_i', \qquad \theta' = \tau\theta + (1-\tau)\theta'.$$

- The actor update is delayed: it only updates once every several times the critic updates.