Deep Reinforcement Learning (Sp25)

Instructor: Dr. Mohammad Hossein Rohban

Summary of Lecture 8: Advanced Methods

Summarized By: Amirhossein Asadi

Proximal Policy Optimization

- In **Policy Gradient** methods, it is crucial to ensure that the policy does not undergo drastic changes during updates. Significant changes in the policy can lead to large steps in the parameter space, which may result in a decrease in rewards. This decline in rewards can cause the generation of low-quality training data, creating a detrimental cycle that further degrades the model's performance.

$$D_{KL}\left(\pi_{\theta'}(\cdot|s) \parallel \pi_{\theta}(\cdot|s)\right) = \sum_{a} \pi_{\theta'}(a|s) \log \frac{\pi_{\theta'}(a|s)}{\pi_{\theta}(a|s)} \leq \epsilon.$$

- In general, the goal is to maximize the **advantage function** to ensure **policy improvement**. However, this cannot be done directly because the optimal policy, which we aim to optimize, is unknown. As a result, the data required to compute the expected value depends on the parameters θ, which are themselves the target of our maximization process and are not available. Instead, we have data generated from the previous policy. To address this, we use **importance sampling** to estimate the expected value under the new policy using samples from the old policy.

$$\max_{\pi_\theta} \mathbb{E}_{s,a\sim\pi_\theta}\left[A^{\pi_{\text{old}}}(s,a)\right] = \mathbb{E}_{a\sim\pi_{\text{old}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)} A^{\pi_{\text{old}}}(s,a)\right]$$

- Next, we use **clipping**. The benefit of clipping is that it restricts the ratio of π_θ to π_{old}, preventing large stepwise changes in the policy parameters. However, this restriction can slow down learning. If the situation is very poor (which is usually the case at the beginning), we use the following formula:

$$\frac{\pi_{\theta'}(a|s)}{\pi_{\theta}(a|s)} \cdot A^{\pi_\theta}(s,a),$$

  and then switch back to the normal mode :

$$\text{clip}\left(\frac{\pi_{\theta'}(a|s)}{\pi_{\theta}(a|s)}, 1-\epsilon, 1+\epsilon\right) A^{\pi_\theta}(s,a)$$

- In general, **PPO** performs significantly better than **TRPO** and other methods in unstable and more challenging environments. This is primarily due to its simplicity, stability, and ability to handle large policy updates without diverging.
- **TRPO** often performs worse than **PPO**, despite their similar underlying logic. The primary reason for this is that TRPO involves a complex optimization process. Specifically, TRPO explicitly aims to optimize the **KL divergence** between the old and new policies, which requires solving a constrained optimization problem. This process relies on several approximations and assumptions to simplify the optimization, which can lead to suboptimal performance and instability in practice.

  In contrast, PPO simplifies the optimization by using a clipped objective function, which avoids the need for explicit KL divergence constraints. This makes PPO more computationally efficient and easier to implement, while still maintaining stable and effective policy updates.
- Proximal Policy Optimization (PPO) is an **on-policy** algorithm designed to update the policy efficiently while ensuring stability.

Soft Actor Critic

- **PPO** suffers from **sample inefficiency**. This is because, after collecting a set of trajectories, it can only compute a single gradient and take a small policy update step using that gradient. Once the update is performed, the data is discarded, and new data must be collected. This repeated data collection process is equivalent to running the simulator multiple times, which incurs high computational costs and makes PPO sample inefficient.

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) + \alpha H(\pi(\cdot|s_t)) \right]$$

  We observe that the return is split into two parts: one part is the traditional return we had before, and the other part is the sum of entropies.

- To address the sample inefficiency of PPO, we need to move towards making it more **off-policy**. This is where **SAC** comes into play.

- By using **entropy**, SAC ensures that the entropy is maximized for each state. In this case, the **Bellman operator** is modified as follows:

$$\mathcal{T}^\pi Q(\mathbf{s}_t, \mathbf{a}_t) \triangleq r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \, \mathbb{E}_{\mathbf{s}_{t+1}\sim p} \left[V(\mathbf{s}_{t+1}) \right]$$

  where

$$V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t\sim\pi} \left[Q(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t|\mathbf{s}_t) \right]$$

- **SAC** improves rewards and encourages policy diversification through entropy regularization, enhancing robustness.

- It can be shown that in **SAC**, the optimal policy can be derived using the **softmax** function over the Q-values. Specifically, the optimal policy π^* is given by:

$$\pi^*(a|s) = \frac{\exp Q(s,a)}{\sum_{a'} \exp Q(s,a')}$$

- The **SAC** algorithm consists of the following key steps:

  1. **Q-function Update**:

$$Q(s,a) \leftarrow r(s,a) + \mathbb{E}_{s'\sim p,\, a'\sim\pi} \left[Q(s',a') - \log \pi(a'|s') \right].$$

  This update converges to Q^π, the Q-function under the current policy π.

  2. **Policy Update**:

$$\pi_{\text{new}} = \arg\min_{\pi'} D_{KL} \left(\pi'(\cdot|s) \middle\| \frac{1}{Z} \exp Q^{\pi_{\text{old}}}(s,\cdot) \right).$$

  In practice, only one gradient step is taken on this objective to ensure stability.

  3. **Interaction with the Environment**: Collect more data by interacting with the environment using the updated policy.