Policy Gradients and Actor-Critic methods - A summary

April 12, 2024

1 Introduction

This short article will shed light on one of the most fundamental algorithms of Reinforcement Learning (RL), the Policy Gradient and its derived methods, the Actor-Critic. I will present them intuitively and analytically, how I understood them best!

2 Policy Gradient Foundation

Foundational RL algorithms focus on learning the state value function

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right]$$

and the action value function

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right]$$

measuring the discounted reward of following policy π in state s for $V^{\pi}(s)$ or the discounted reward of taking action a in state s and thereafter follow the policy π for $Q^{\pi}(s,a)$ in a tabular approach. We either use Monte-Carlo methods with Dynamic Programming or Temporal Difference algorithms to experience all state and action combinations and then update the current estimated value functions with the newly seen rewards. We result in a table where we can map each state or state-action pair to a specific value which then lets us choose the optimal policy by selecting a combination that yields maximum value.

However, as the environment gets intricate the possible states start to explode. For instance, take the game Breakout in Atari where the state s consists of the current frame of the game, i.e. an image of dimension (210,160,3). It should be straightforward that, let alone, all possible states of this image are almost

infinite. Taking the actions additionally into account, the state-action combinations even further increase.

The basic reasoning behind the Policy Gradient is the idea that instead of using a tabular approach we parametrize our policy using θ and adjust these parameters such that given a state s the policy suggests a distribution over actions which maximizes some performance measure $\mathbf{J}(\theta)$. Sutton and Barto (2020) opt to use the expected return of future rewards, which we denote with $v_{\pi(\theta)}(s_0)$, i.e. the true value function, as this measure. The policy Gradient Theorem shows that we can describe the gradient of $\mathbf{J}(\theta)$ as:

$$\nabla \mathbf{J}(\theta) = \nabla v_{\pi(\theta)} \propto \sum(s)\mu(s) \sum_a \pi(a|s) q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_a \pi(a|s) q^\pi(s,a) \right]$$

Hence, the Policy Gradient is proportional to the expected value of the action value function weighted by the probability of taking the action under the current policy π (Refer to pages 325ff of Sutton and Bartos book in chapter 13.2 for more details).
The cool part is, that we only need basic calculus rules for the proof. I will not go in detail, because the proof is presented very detailed by Sutton and Barto on page 325 in chapter 13.2.
If we now only consider the actual action a_t taken in this state (instead of summing over all possible actions) as suggested by the policy π and remember that $q^\pi(s,a) = G_t$, which is the discounted sum of rewards and simplify further we get:

$$\nabla \mathbf{J}(\theta) \propto \mathbb{E}_\pi \left[\frac{\nabla \pi(a_t|s_t,\theta)}{\pi(a_t|s_t,\theta)} G_t \right] = \mathbb{E}_\pi \left[\nabla \ln \pi(a_t|s_t,\theta) G_t \right]$$

Remember that we can plug in the ln since $\frac{\nabla x}{x} = \nabla \ln(x)$.

Using Monte Carlo sampling for the episodic case and applying stochastic gradient ascent we get the basic REINFORCE algorithm as described by Sutton and Barto.
In line 8 of algorithm 1 we can see the update rule. The parameters θ are adjusted in a way, such that the policy π increases the probability of actions a_t that yield high discounted returns and reduces the probability of actions that do yield low discounted returns.

This section presented the core idea of the Policy Gradient why it was introduced and how it works. The research area of RL used this foundation to build new, optimized algorithms to further increase the performance of Policy Gradient methods. We will explore these in the next chapters.

In summary, the inherent need to invent the Policy Gradient method was shown. Further, the derivation of the Policy Gradient was examined resulting in our fi-

Algorithm 1 REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π^*

Input: a differentiable policy parameterization $\pi(a|s, \theta)$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\theta \in \mathbb{R}^{d_0}$ (e.g., to 0)
loop (for each episode)
    Generate an episode $s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T$, following a$\pi(\cdot|\cdot, \theta)$
    for each step of the episode $t = 0, 1, ..., T-1$ **do**
        Calculate return $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
        Update $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \ln \pi(a_t|s_t, \theta)$
    end for
end loop

nal algorithm which successfully incorporated the Policy Gradient in the Monte-Carlo sampling method to construct a parametrized policy $\pi(\theta)$ that maximizes the discounted future rewards.

3 Baseline and Actor Critic methods

Policy gradient methods seek to optimize the policy directly by estimating the gradient of the expected return (cumulative rewards) with respect to the policy parameters. This estimation involves sampling trajectories according to the current policy and using the returns from these trajectories to update the policy parameters. However, the returns can vary significantly across different trajectories, leading to high variance in the gradient estimates. This variance can slow down learning or lead to instability in the training process, as the updates may swing wildly in different directions based on the sampled trajectories

To mitigate this fluctuation effect, the concept of a baseline was established. Basically, it is suggested to add a baseline b, which acts as a comparison to the generated return trajectory G_t. The baseline is added by subtracting b from G_t and can be any function as Sutton and Barto pointed out. This baseline effectively centers the returns around the baseline, which reduces the spread, i.e. the variance , of the adjusted returns.

$$G_t - b$$

The new update rule can then be described as:

$$\theta_{t+1} \leftarrow \theta_t + \alpha\gamma^t (G_t - b) \nabla_{\theta_t} \ln \pi(a_t|s_t, \theta)$$

We can apply the state value $v(s_t)$ as baseline. As this value is usually unknown, the estimate $\hat{v}(s_t)$ is a straightforward alternative. If we use a function

approximation approach by parametrizing the value estimate we arrive at a fundamental method called *Actor-Critic*.
Actor-Critic methods utilize two different function approximations, mostly neural-networks.

1. **The Critic** represents the learned state value function $\hat{v}(s_t, \phi)$. It should learn a mapping from states to values, disclosing information of how valuable each state is

2. **The Actor** takes over the decision part. It is simply the Policy Gradient of the previous section. However, the Critic's estimation is now added by using it as a baseline.

Using this, we can now introduce a new algorithm, the One-Step Actor-Critic algorithm. It is based on TD(0) errors and uses bootstrapping. It is very similar to algorithm 1, with a small additional twist. We now also update the Critic instead of solely the Actor.

Algorithm 2 One-step Actor-Critic (episodic), for estimating $\pi_\theta \approx \pi^*$ based on Sutton and Barto page 332.

1: **Input:** a differentiable policy parameterization $\pi(a|s, \theta)$
2: **Input:** a differentiable state-value function parameterization $\hat{v}(s, \phi)$
3: **Parameters:** step sizes $\alpha_\theta > 0$, $\alpha_w > 0$
4: Initialize policy parameter $\theta \in \mathbb{R}^{d_0}$ and state-value weights $\phi \in \mathbb{R}^d$ (e.g., to 0)
5: **loop** forever (for each episode):
6: Initialize S (first state of episode)
7: $I \leftarrow 1$
8: **loop** while S is not terminal (for each time step):
9: $A \sim \pi(\cdot|S, \theta)$
10: Take action A, observe S', R
11: $\delta \leftarrow R + \gamma \hat{v}(S', \phi) - \hat{v}(S, \phi)$ (if S' is terminal, then $\hat{v}(S', \phi) = 0$)
12: $\phi \leftarrow \phi + \alpha_\phi \delta \nabla_\phi \hat{v}(S, \phi)$
13: $\theta \leftarrow \theta + \alpha_\theta I \delta \nabla_\theta \log \pi(A|S, \theta)$
14: $I \leftarrow \gamma I$
15: $S \leftarrow S'$
16: **end loop**
17: **end loop**

Taking line 11 and 12 into account, one can see how we adjust the parameters of the state value function approximation. We start by calculating the classic TD(0) error, δ. Consequently, we perform the usual gradient descent step and update the parameters ϕ.

4 Acknowledgement

The external information used came from: [**SuttonBarto2018**]