Demystifying Policy Gradients

Arun Kumar

Oct 2019

1 Introduction

Reinforcement Learning algorithms are used extensively in learning controls because of the fact that you really don't need to know the complex dynamics of the *agent* and the *environment*. The agent learns to act optimally in the environment as the time progresses. This, it does by exploring the world around and learning from the *rewards*. The positive rewards encourage the actions and negative rewards discourage the actions. Eventually, the agent has to learn a policy π which is basically a mapping between the *states* and *actions*. There are *value-based* methods that learn the state-values and then there are *policy-based* methods that directly learn the policy. In the case of value-based methods like Q-learning, the policy is derived from the value function (e.g, ϵ - greedy).

2 Policy Gradients

Policy based methods can very easily learn the direct mapping of states with action probabilities. The way it is done is by parameterizing the policy e.g, by a neural network. The input to this network is the states vector and the output is the probability distribution over possible actions. Example, in the game of pong, there can be two outputs - up and down. The states vector could be MxN pixels snapshot of the game. In this case, the two outputs of the network will give the probability of going up or down respectively. Most of the time it will happen that these decisions made by the network are not correct. During training process, we want to tune the network parameters so that it takes decisions(hence actions) that get maximum expected sum of rewards over time.

Consider a control policy parameterized by parameter vector θ.We have to maximize expected sum of rewards $\mathrm{E}[\sum_{\mathrm{t}=0}{}^{\mathrm{H}} R(s_{\mathrm{t}})|\pi_\theta]$.

In the value-based methods, θ used to be a parameter vector of the network that predicted the q-values, now θ parameterizes the policy. Policy optimization is better than it's counterpart Value based (Q or V) because of three reasons: 1. Often π can be easier than finding Q and V. 2. V doesn't directly prescribe actions. It would need dynamics model and one Bellman backup. 3. Q

need to be able to efficiently solve argmax_{u} Q_θ(s, a) - challenge for high dimensional/continuous action spaces.In case of DQN you can have one output per action but then, these are discrete actions. What if there are continuous actions. Even if you discretize the action space, there can be millions of such continuous actions. The simplest way to train a policy is to apply Likelihood Ratio Policy Gradient.

2.1 Likelihood Ratio Policy Gradient

We let τ denote a state-action sequence s0, u0, ... , sH, uH. We overload the notation: $\text{R}(\tau) = \sum_{\text{t}=0}{}^{\text{H}} R(st, ut)$.

$$U(\theta) = E[\sum_{t=0}^{H} R(st, ut); \pi_\theta] = \sum_{\tau} P(\tau; \theta) R(\tau) \tag{1}$$

In our new notation, our goal is to find θ:

$$max_\theta U(\theta) = max_\theta \sum_{\tau} P(\tau; \theta) R(\tau) \tag{2}$$

To find the $\max_\theta \text{U}(\theta)$, we have to find the gradient of $\text{U}(\theta)$.

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau; \theta) R(\tau) \\
&= \sum_{\tau} \nabla_\theta P(\tau; \theta) R(\tau) \\
&= \sum_{\tau} \frac{P(\tau; \theta)}{P(\tau; \theta)} \nabla_\theta P(\tau; \theta) R(\tau) \\
&= \sum_{\tau} P(\tau; \theta) \frac{\nabla_\theta P(\tau; \theta)}{P(\tau; \theta)} R(\tau) \\
&= \sum_{\tau} P(\tau; \theta) \nabla_\theta \log P(\tau; \theta) R(\tau)
\end{aligned}$$

The above expectation can be approximated with the empirical estimate for m sample paths under the policy π_θ:

$$\nabla_\theta U(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \text{log} P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

From the above expression it can be seen that $\nabla_\theta U(\theta)$ is the average of the grad. log probability of the trajectories multiplied with the total rewards in those trajectories. Intuitively, if the total reward of a trajectory is negative, the log probability of that trajectory will be decreased and if the total reward of a trajectory is positive, the log probability of that trajectory will be increased. In this way, the gradient always tunes the parameter vector in the direction that favours maximum rewards.

2.2 Inside the Paths

In the above section, we saw the term grad. log probability of the trajectory given a policy network parameterized by θ. In this section, we'll dive a little deeper into it and observe what it is composed of. We can write $\nabla_\theta \log P(\tau^{(i)};\theta)$ as follows:

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \nabla_\theta \log [\prod_{t=0}^{H} P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)}).\pi_\theta(u_t^{(i)}|s_t^{(i)})]$$

Note that individual trajectory τ^{i} is composed of a state-action sequence s0, u0, ... , sH, uH. From the multiplication rule in probability it can be said that the probability of a trajectory is the product of the probabilities of each transition that occurred from initial state s0 to the terminal state sH. Now how do we calculate probability of each transition? The probability of a transition is the product of the probability of ending up in the next state $\mathrm{s_{t+1}}$ from current state $\mathrm{s_t}$(dynamics model) and the probability of taking an action $\mathrm{u_t}$ given current state $\mathrm{s_t}$ under the policy π_θ(policy). We can rewrite the above expression as follows:

$$= \nabla_\theta [\sum_{t=0}^{H} \log P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)})]$$

As the first term doesn't depend on θ, the above expression boils down to:

$$= \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)})$$

So the final expression doesn't depend on the dynamics model. Hence applying policy gradient is so convenient.

2.3 Baselines

We want to increase the probability of the paths that are better than average and decrease the probability of the paths that are worse than average. So instead of multiplying the log prob. of path with the total rewards accumulated over the path, we multiply it with the total rewards minus the average sum of the rewards over that path.

$$\nabla_\theta U(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)(R(\tau^{(i)}) - b)$$

The term b is the baseline which is a measure of the average performance of the policy in terms of average rewards.

2.4 Intuition

What is the intuition behind these equations? The log term is the maximum log likelihood. In deep learning, it measures the likelihood of the observed data. In our context, it measures how likely the trajectory is under the current policy. By multiplying it with the rewards, we want to increase the likelihood of a policy if the trajectory results in a high positive reward. On the contrary, we want to decrease the likelihood of a policy if it results in a high negative reward. In short, keep what is working and throw out what is not.

One very good thing about policy gradients is that it avoids the problem of vanishing gradients. Usually, in deep-learning, the gradient of highly correlated sequences causes the vanishing gradient problem.In the case of policy gradient, such problem does not exist. The probability of the trajectory (a long sequence) is defined as the product of the probability of each transition in that trajectory as shown in section 2.2. And log of the product results in sum which is much easier to work with as far as derivatives are concerned. This breaks the curse of multiplying a long sequence.

3 Revisiting Maximum Likelihood Estimation