Notes on RL

Tatiana Ushakova

March 2025

1 Policy based RL

If we proceed the game following the policy - or the rules on how to act based on what we see happening in the game - we have trajectories of game evolution determined by chain of states and actions, with probability determined by game structure and the policy:

$$P_{traj} = \rho(s_0) \prod \left[P(s_{j+1}|s_j, a_j)\pi(a_j|s_j) \right]$$

which mean that we start with some initial state that samples from initial state and we sample both the actions and states - actions are sampled from policy and states from how out game or env work internally. And we see the main assumption here is the markov process - action depend only on current state and next state only on prev one. Of course philisophically we can determine the new state as $S_j = \{s_j, s_{j-1}..s_1\}$ so every game that doesnt satisfy this condition can be rewritten to one that satisfies.

From general considerations, we want to compute the total estimated reward that we are getting and differentiate it by the parameters in our policy so we can achieve the better policy:

$$J = \langle R(\tau) \rangle = \sum_{\tau - traj} P_\tau R(\tau)$$

We are dealing with the potentially infinite games - we have no upper boundary on steps in many steps in many games, so we need out definitions to be correctly defined, so instead of naively defined reward as a sum of distinct rewards we introduce the discounted one (and another different way to deal with that is to introduce the finite cutoff of the amount of steps backward to account in our reward).

$$R(\tau) = \sum \gamma^t r_t$$

Then we can treat our expected reward as the function to optimize for with respect to the internal parameters θ of our policy to maximize the expected reward:

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta J$$

where α is called the learning rate.

The reward r_t depends only on the state, not on policy (though in general also can depend on state probabilistically not deterministically), so the only part in J that depend on optimized parameter are policy action sampling terms $\pi(a_j|s_j)$.

We can compute the gradients of the trajectories

$$\nabla_\theta P(\tau) = P(\tau) \sum \nabla_\theta \log \pi(a_j|s_j)$$

and so

$$\nabla_\theta J = \sum_\tau R(\tau) P(\tau) \sum_{a_j, s_j \in \tau} \log \pi(a_j|s_j) = \langle R(\tau) \nabla_\theta \sum_{a_j, s_j \in \tau} \log \pi(a_j|s_j) \rangle$$

Somehow with the handwaivings we get from that that we actually need to minimize the function that is

$$L = -R(\tau) \log \pi(a_j|s_j)$$

which is obviously a kind of stretch of our derivation, as we definitely know that

$$\nabla_\theta J = -\langle \nabla_\theta L \rangle \neq -\nabla_\theta \langle L \rangle$$

in general and specifically in our case as we saw that probability specifically depends on the parameters of our policy.

In practice, in our ML learning implementations, we use the gradient of the expected reward as

$$\nabla_\theta J = \frac{1}{B} \sum_B \sum_\tau R(\tau) \nabla_\theta \log \pi_\theta(a_j|s_j)$$

Where we run the game several time (B - batch size number of games), and then use the equation to update the policy parameters.

And while instead in the implementations we use the formula

$$L = \frac{1}{B} \sum_B \sum_\tau R(\tau) \log \pi_\theta(a_j|s_j)$$

instead and then take the derivative, its actually turns out to be the correct derivative, as the probability of the trajectory multiplier appear in out formula not explicitly but by sampling through a lot of trajectories, wich means that we include fixed probability $P_{\theta_k}(\tau)$ in the point θ_k instead of dependent on θ, so in practice we actually having formula

$$\nabla_\theta L = \nabla_\theta \frac{1}{B} \sum_B \sum_\tau R(\tau) \log \pi_\theta(a_j|s_j) = \nabla_\theta \sum_\tau P_{\theta_k}(\tau) R(\tau) \log \pi(a_j|s_j) = \langle R(\tau) \nabla_\theta \sum_\tau \log \pi(a_j|s_j) \rangle$$

as probability calcultes in the point and doesnt depend on θ, and here above I ommited sum of logs for better readability.

1.1 Rewards-to-Go and Advantages Estimate

So in the algorithm description of Vanila PO - the symplest version of Policy learning method, we see the following:

Algorithm 1 Vanilla Policy Gradient Algorithm

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, ...$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

7: Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

or via another gradient ascent algorithm like Adam.
8: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
9: **end for**

2 PPO

Despite the name 'Policy Optimization', PPO is actually a mixed Policy-Value method that involves training of both netwoorks for policy and value.

3 Qiestions so far

- Why do we want to maximize the future reward and not the total reward (for the rewards to go)? We will use the policy in the future games and there we would want to maximize the total reward. (I guess if we assume that the game process is time invariant or time is included in the state, then its fine)

- What is the best way to not lose the data collected in the prev trainign with older policies? feels kind of ineficient usage of data to just throw it away, but I might be wrong, didnt think much about it