DDA4230 Reinforcement learning	Policy optimization
Lecture 18	
Lecturer: Guiliang Liu	*Scribe: Jing Dong*

1 Goal of this lecture

In this lecture we discuss how to implement the stochastic gradient once we have an estimator.

Suggested reading: *Trust region policy optimization*, by Schulman, Levine, Moritz, Jordan, and Abbeel; *Proximal policy optimization algorithms*, by Schulman, Wolski, Dhariwal, Radford, and Klimov;

2 Recap: Policy gradient

In the past lectures, we have covered many value-based methods, which require us to learn the Q or V functions. The policy optimization methods learn the policy π directly. There are a few reasons why this may be preferred over value based methods. We list a few here.

- The value function V does not prescribe us actions. It would also need the dynamic model to compute the actions.
- For the Q function, we may not be able to efficiently solve $\arg\max_a Q(s,a)$. This is especially challenging for continuous and high-dimensional action spaces.

Before moving on to policy optimization methods, we first review the policy gradient method. Let τ denote a state-action sequence $s_0, a_0, \ldots, s_T, a_T$, and we overload the notation to let $r(\tau) = \sum_{t=0}^{T} r(s_t, a_t)$. Then for a policy π parameterized by θ and the corresponding occupancy measure $P^{\pi_\theta}(\tau)$, we desire to find

$$\max_{\theta} \mathbb{E}\left[P^{\pi_\theta}(\tau) r(\tau)\right] .$$

Taking gradient (denoted as g) with respect to θ gives us

$$g = \mathbb{E}\left[r(\tau)\nabla_\theta \log\left(\sum_{t=1}^{T} P\left(s_{t+1} \mid s_t, a_t\right) \cdot \pi_\theta\left(a_t \mid s_t\right)\right)\right] = \mathbb{E}\left[r(\tau)\sum_{t=1}^{T} \nabla_\theta \log(\pi_\theta\left(a_t \mid s_t\right))\right] .$$

This gradient is unbiased and we do not need access to the dynamic model to compute this. There are several different related expressions for the policy gradient, which have the form

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right)\right] ,$$

where Ψ_t might be one of the following:

1. $\sum_{t=0}^{\infty} r_t$: the total reward of the trajectory;
2. $Q^{\pi}(s_t, a_t)$: the action value function;
3. $\sum_{t'=t}^{\infty} r_{t'}$: the reward following action a_t;
4. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: the reward following action a_t with a baseline;
5. $A^{\pi}(s_t, a_t)$: the advantage function;
6. $r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$: the TD residual.

The latter formulas use the definitions $A^{\pi}(s_t, a_t) := Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$, which is the advantage function.

In policy optimization methods, the data samples we have collected may not be corresponding to the policy we wish to optimize. In this case, some special handling is needed so that the gradient estimates can be unbiased. Let π_1 be the policy we are currently following and π_2 be the policy we want to optimize. Let them be parameterized by θ_1, θ_2, respectively. Then we can use importance sampling to re-weight our objective as

$$\max_{\theta_1} \mathbb{E}\left[\frac{P^{\pi_{\theta_2}}(\tau)}{P^{\pi_{\theta_1}}(\tau)} r(\tau)\right].$$

3 Trust region policy optimization (TRPO)

Using the gradients to optimize our policy requires us to provide a step size (a.k.a learning rate). In the classic supervised learning setting or in the optimization literature, having a bad step size may not be terrible. This is because the next update can partially correct the error in the previous steps. In reinforcement learning and particularly policy optimization, this is going to be a greater problem. When the step size steps are too far, we obtain a terrible policy. This indicates that the next batch of data will be collected under this terrible policy. Then it becomes not clear how to recover short of going back and shrinking the step size.

One method of choosing the step size is by line search. The procedure is summarized as follows

1. Calculate the initial loss and initialize the step size to be a large value;
2. Update the parameter with the gradients under the current step size can calculate the new loss;
3. Decrease the value of step size until we have found a new loss that is less than the initial loss.

This simple method helps us to make gradient descent at a relatively good step size. However, it may be expensive to compute so many gradients (evaluation along the line). Moreover, this methods ignores the quality of our gradients (our gradients may be of poor quality as it is stochastic). In fact, if the noise of the gradient is large, choose the best step sizes

along the line could be analogous to choosing the optimum among the random signals (the multi-hypothesis test).

An alternative method is the trust region method. Intuitively, it finds us a point where it is a good approximation of the actual descent gradient within a "trust region".

Let us first denotes $P(\tau \mid \theta) = P(s_0) \cdot \Pi_{t=1}^{T} P(s_{t+1} \mid s_t, a_t)\pi_\theta(a_t \mid s_t)$. Then the trust region method finds us the next parameter $\theta+\delta\theta$ through solving the following maximization problem.

$$\begin{aligned} &\max_{\delta\theta} \ \ g^\top \delta\theta \\ &\text{subject to } d_{\mathrm{KL}}\left(P(\tau|\theta)||P(\tau|\theta+\delta\theta)\right) \le \epsilon\,, \end{aligned}$$

where d_{KL} denotes the KL divergence, g is our gradient estimate, and ϵ is a parameter we can set. Here, the change of the objective function is estimated by assuming that the objective function in this neighbouring area is linear.

Using the expression of the KL divergence, we have

$$\begin{aligned} d_{\mathrm{KL}}(P(\tau;\theta)\|P(\tau;\theta+\delta\theta)) &= \sum_\tau P(\tau;\theta)\log\frac{P(\tau;\theta)}{P(\tau;\theta+\delta\theta)} \\ &= \sum_\tau P(\tau;\theta)\log\frac{P\left(s_0\right)\prod_{t=0}^{T-1}\pi_\theta\left(a_t \mid s_t\right)P\left(s_{t+1} \mid s_t, a_t\right)}{P\left(s_0\right)\prod_{t=0}^{T-1}\pi_{\theta+\delta\theta}\left(a_t \mid s_t\right)P\left(s_{t+1} \mid s_t, a_t\right)} \\ &= \sum_\tau P(\tau;\theta)\log\frac{\prod_{t=0}^{T-1}\pi_\theta\left(a_t \mid s_t\right)}{\prod_{t=0}^{T-1}\pi_{\theta+\delta\theta}\left(a_t \mid s_t\right)}\,. \end{aligned}$$

With M samples, this term can be approximated by the sample average and we may rewrite the maximization problem to be

$$\begin{aligned} &\max_{\delta\theta} \ \ g^\top \delta\theta \\ &\text{subject to } \frac{1}{M}\sum_{(s,a)}\log\frac{\pi_\theta\left(a \mid s\right)}{\pi_{\theta+\delta\theta}\left(a \mid s\right)} \le \epsilon\,. \end{aligned}$$

This maximization problem with the constraint can be hard to enforce given complicated policies like neural networks. Therefore we would need to approximate the KL divergence further for a feasible objective. This is done through second order approximation with fisher matrix F_θ.

$$\begin{aligned} d_{\mathrm{KL}}\left(\pi_\theta(a \mid s)\|\pi_{\theta+\delta\theta}(a \mid s)\right) &\approx \delta\theta^\top\left(\sum_{(s,a)\sim\theta}\nabla_\theta\log\pi_\theta(a \mid s)\nabla_\theta\log\pi_\theta(a \mid s)^\top\right)\delta\theta \\ &= \delta\theta^\top F_\theta\delta\theta\,. \end{aligned}$$

And now our maximization problem is simplified to

$$\begin{aligned} &\max_{\delta\theta} \ \ g^\top \delta\theta \\ &\text{subject to } \delta\theta^\top F_\theta\delta\theta \le \epsilon\,, \end{aligned}$$

which is a linear objective quadratic constrained optimization problem and could be solved analytically using the Karush–Kuhn–Tucker conditions (though the algorithm will not rely on the analytical solution).

Thus the final TRPO objective is given as

$$\begin{aligned} \text{Surrogate loss: } & \max_{\pi} L(\pi) = \mathbb{E}_{\pi_{\text{old}}}\left[\frac{\pi(a \mid s)}{\pi_{\text{old}}(a \mid s)} A^{\pi_{\text{old}}}(s,a)\right] \\ \text{Constraint: } & \mathbb{E}_{\pi_{\text{old}}}\left[d_{\text{KL}}\left(\pi \| \pi_{\text{old}}\right)\right] \leq \epsilon\,, \end{aligned}$$

where A denotes the advantage function. This corresponds to a general policy gradient form we have mentioned earlier.

The last remaining issue is that θ can be high-dimensional. In this case building and inverting fisher matrix F_θ can be impractical. Thus the TRPO method approximately solve the constraint to avoid the expense caused by exact computation. Though the approximation alleviate the computation overhead, it still requires second order information such as Hessian matrix. This can still be expensive both in computation and space.

4 Proximal policy optimization (PPO)

Even though the TRPO method has invested a great effort into making the constraint set feasible, they may still be hard to enforce in complicated policy architectures. For example, one with stochasticity like dropout or with parameter sharing. Moreover, the computation overhead can still be high.

The PPO method enforces a "soft" constraint instead, which means that during the training process, the constraint could be violated. It adds a proximal value to the objective function. The objective is the following

$$L(\pi) = \mathbb{E}_{\pi_{\text{old}}}\left[\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} A^{\pi_{\text{old}}}(s,a) - \beta d_{\text{KL}}\left(\pi_{\theta_{\text{old}}}, \pi_\theta\right)\right].$$

The β can be fixed or adaptively chosen (or simply set to 0). One reason why one may wish to adaptively choose β is because it can be hard to find one β that performs well across different problems.

Another surrogate objective proposed by PPO stems from the observation that the policy's performance can fluctuate greatly when $\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ changes too quickly. Thus it limits ρ to a range of $[1-\epsilon, 1+\epsilon]$ such that no abrupt updates to the policy will be made. The surrogate objective is then written as

$$L^{CLIP}(\pi) = \mathbb{E}\left[\min\{\rho_t(\theta)A(s,a), \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)A(s,a)\}\right].$$

We take the minimum of the constrained and unconstrained objectives such that our final objective is a lower bound of the unclipped objective. With this scheme, we only ignore the change in probability ratio when it would make the objective improve, and we include it when it makes the objective worse.

The two objective functions $L(\pi)$ and $L^{CLIP}(\pi)$ mentioned above can be used separately or together.

algorithm	avg. normalized score
No clipping or penalty	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
Adaptive KL $d_{\text{targ}} = 0.003$	0.68
Adaptive KL $d_{\text{targ}} = 0.01$	0.74
Adaptive KL $d_{\text{targ}} = 0.03$	0.71
Fixed KL, $\beta = 0.3$	0.62
Fixed KL, $\beta = 1.$	0.71
Fixed KL, $\beta = 3.$	0.72
Fixed KL, $\beta = 10.$	0.69

Table 1: Results from continuous control benchmark. Average normalized scores (over 21 runs of the algorithm, on 7 environments) for each algorithm / hyperparameter setting . β was initialized at 1.

5 Implementations of policy optimization

Although we have summarized the main algorithmic contributions of PPO and TRPO, there exist many different practical implementations of them. The implementation details indeed influence the algorithm performance by a lot. For example, one may choose to clip the learnt value function just like how we clip ρ in $[1-\epsilon, 1+\epsilon]$ and to scale the rewards obtained from the environment. The findings are summarized in a paper focusing on implementation details [EIS+20].

In a later paper [HMDH20], two common design choices in PPO are revisited, precisely I) clipped policy probability ratio for regularization and II) to parameterize policy action space by continuous Gaussian or discrete softmax distribution. They first identify three failure cases in PPO and proposed replacements for these two designs.

The failure modes are:

- On continuous action spaces, standard PPO is unstable when rewards vanish outside bounded support.
- On discrete action spaces with sparse high rewards, standard PPO often gets stuck at suboptimal actions. The policy is sensitive to initialization when there are locally optimal actions close to initialization.

We encourage the students to obtain the high-level ideas of TRPO and PPO and try them on different tasks to obtain a better understanding of the algorithm. These algorithms are also good choices of first-to-try algorithms when one is presented with a task for policy-based reinforcement learning.

Acknowledgement

This lecture notes takes reference from Pieter Abbeel's lecture notes on reinforcement learning.

References

[EIS+20] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on PPO and TRPO. In *International Conference on Learning Representations*, 2020.

[HMDH20] Chloe Ching-Yun Hsu, Celestine Mendler-Dünner, and Moritz Hardt. Revisiting design choices in proximal policy optimization. 2020.