Lecture 12: Policy Optimization II

Bolei Zhou

The Chinese University of Hong Kong

bzhou@ie.cuhk.edu.hk

March 4, 2020

Today's Plan

1. Review on policy optimization
2. Improving the policy gradient with a critic: Actor-Critic algorithm

Review on Policy Optimization

1. Softmax policy: weight actions using linear combination of features $\phi(s, a)^T\theta$

$$\pi_\theta(s, a) = \frac{\exp^{\phi(s,a)^T\theta}}{\sum_{a'} \exp^{\phi(s,a')^T\theta}}$$

2. Gaussian policy:
   1. Mean is a linear combination of state features $\mu(s) = \phi(s)^T\theta$
   2. Variance may be fixed σ^2 or can also be parameterized
   3. Policy is Gaussian, the continuous $a \sim \mathcal{N}(\mu(s), \sigma^2)$
3. Neural network policy: $\pi_\theta(a|s) = F(s)$ where $F(\cdot)$ is a neural network

Review on Policy Optimization

1. In policy optimization, for the policy function π_θ the objective is to maximize

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

   1. $R(\tau)$ could be any reasonable reward function on τ

2. So the gradient is

$$\begin{aligned} \nabla_\theta J(\theta) =& \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) \\ =& \mathbb{E}_\tau[R(\tau) \nabla_\theta \log P(\tau;\theta)] \end{aligned}$$

   1. Very nice log trick: $\nabla_\theta f_\theta(x) = f_\theta(x) \nabla_\theta \log f_\theta(x)$
   2. Trajectory distribution: $P(\tau;\theta) = \left[\mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)\right]$
   3. Very nice canceling $\nabla_\theta \log P(\tau;\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Review on Policy Gradient

1. After decomposing τ we derived that

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau\Big[\Big(\sum_{t=0}^{T-1} r_t\Big)\Big(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\Big)\Big]$$

2. After applying the causality, we have

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau\Big[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

   1. $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is the return for a trajectory at step t

REINFORCE: MC Policy Graident Algorithm

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Policy Gradient is On-Policy RL

1. $\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\nabla_\theta \log P_\theta(\tau) r(\tau)]$
2. In REINFORCE algorithm: there is the on-policy sampling (sample $\{\tau\}$ from π_θ)
3. On-policy learning can be extremely inefficient

Off-Policy Policy Gradient with Importance Sampling

1. $\theta^* = \arg\max_\theta J(\theta)$ where $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau)]$
2. Let's say we have a behavior policy $\hat{\pi}$ where we can generate samples, then

$$J(\theta) = \mathbb{E}_{\tau \sim \hat{\pi}}[\frac{\pi_\theta(\tau)}{\hat{\pi}(\tau)} r(\tau)]$$

3. Then we can derive the off-policy policy gradient:
   1. GPS (Guided Policy Search): Levine & Koltun (2013) ICML. deep RL with importance sampled policy gradient.

Reducing Variance by a Baseline

1. The original update

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

    1. $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is the return for a Monte-Carlo trajectory which might have high variance

2. We subtract a baseline $b(s)$ from the actual return of the policy gradient to reduce variance

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} \big(G_t - b(s_t)\big) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

3. A good baseline is the expected return

$$b(s_t) = \mathbb{E}[r_t + r_{t+1} + ... + r_{T-1}]$$

Reducing Variance by a Baseline

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} \big(G_t - b_{\mathbf{w}}(s_t) \big) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

1. Baseline $b(s)$ can reduce variance, without changing the expectation
2. $b_{\mathbf{w}}(s)$ also has a parameter $\mathbf{w}$ to learn so that we have two set of parameters $\mathbf{w}$ and θ

Vanilla Policy Gradient Algorithm with Baseline

procedure POLICY GRADIENT(α)
  Initialize policy parameters θ and baseline values $b(s)$ for all s, e.g. to 0
  for iteration $= 1, 2, \ldots$ **do**
    Collect a set of m trajectories by executing the current policy π_θ
    for each time step t of each trajectory $\tau^{(i)}$ **do**
      Compute the *return* $G_t^{(i)} = \sum_{t'=t}^{T-1} r_{t'}$
      Compute the *advantage estimate* $\hat{A}_t^{(i)} = G_t^{(i)} - b(s_t)$
    Re-fit the baseline to the empirical returns by updating $\mathbf{w}$ to minimize

$$\sum_{i=1}^{m} \sum_{t=0}^{T-1} \|b(s_t) - G_t^{(i)}\|^2$$

    Update policy parameters θ using the policy gradient estimate $\hat{g}$

$$\hat{g} = \sum_{i=1}^{m} \sum_{t=0}^{T-1} \hat{A}_t^{(i)} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

    with an optimizer like SGD ($\theta \leftarrow \theta + \alpha \cdot \hat{g}$) or Adam
  return θ and baseline values $b(s)$

Reducing Variance Using a Critic

1. The update is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

2. In practice, G_t is a sample from Monte Carlo policy gradient, which is the unbiased but noisy estimate of $Q^{\pi_\theta}(s_t, a_t)$
3. Instead we can use a critic to estimate the action-value function,

$$Q_{\mathbf{w}}(s, a) \approx Q^{\pi_\theta}(s, a)$$

4. Then the update becomes

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T-1} Q_{\mathbf{w}}(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

Actor-Critic Policy Gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T-1} Q_{\mathbf{w}}(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

1. It becomes Actor-Critic Policy Gradient
   1. Actor: the policy function used to generate the action
   2. Critic: the value function used to evaluate the reward of the actions
2. Actor-critic algorithms maintain two sets of parameters
   1. Actor: Updates policy parameters θ, in direction suggested by critic
   2. Critic: Updates action-value function parameters $\mathbf{w}$

Estimating the Action-Value Function

1. The critic is solving a familiar problem: policy evaluation
   1. How good is policy π_θ for current parameter θ
2. Policy evaluation was explored in previous lectures, e.g.
   1. Monte-Carlo policy evaluation
   2. Temporal-Difference learning
   3. Least-squares policy evaluation

Action-Value Actor-Critic Algorithm

1. Using a linear value function approximation: $Q_{\mathbf{w}}(s,a) = \psi(s,a)^T \mathbf{w}$
   1. Critic: update $\mathbf{w}$ by a linear TD(0)
   2. Actor: update θ by policy gradient

Algorithm 1 Simple QAC

1: **for** each step **do**
2: generate sample s, a, r, s', a' following π_θ
3: $\delta = r + \gamma Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a)$ #TD error
4: $\mathbf{w} \leftarrow \mathbf{w} + \beta \delta \psi(s, a)$
5: $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_{\mathbf{w}}(s, a)$
6: **end for**

Reducing the Variance of Actor-Critic by a Baseline

1. Recall Q-function / state-action-value function:

$$Q^{\pi,\gamma}(s,a) = \mathbb{E}_\pi[r_1 + \gamma r_2 + ...|s_1 = s, a_1 = a]$$

2. State value function can serve as a great baseline

$$\begin{aligned} V^{\pi,\gamma}(s) =& \mathbb{E}_\pi[r_1 + \gamma r_2 + ...|s_1 = s] \\ =& \mathbb{E}_{a\sim\pi}[Q^{\pi,\gamma}(s,a)] \end{aligned}$$

3. Advantage function: combining Q with baseline V

$$A^{\pi,\gamma}(s,a) = Q^{\pi,\gamma}(s,a) - V^{\pi,\gamma}(s)$$

4. Then the policy gradient becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi,\gamma}(s,a)]$$

N-step estimators

1. We used the Monte-Carlo estimates of the reward
2. We can also us TD methods for the policy gradient update, or any intermediate blend between TD and MC methods:
3. Consider the following n-step returns for $n = 1, 2, \infty$

$$
\begin{aligned}
n = 1(TD) \quad G_t^{(1)} =& r_{t+1} + \gamma v(s_{t+1}) \\
n = 2 \qquad\quad G_t^{(2)} =& r_{t+1} + \gamma r_{t+2} + \gamma^2 v(s_{t+2}) \\
n = \infty(MC) \quad G_t^{(\infty)} =& r_{t+1} + \gamma r_{t+2} + ... + \gamma^{T-t-1} r_T
\end{aligned}
$$

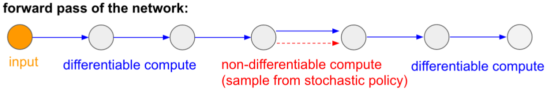
4. Then the advantage estimators become

$$
\begin{aligned}
\hat{A}_t^{(1)} =& r_{t+1} + \gamma v(s_{t+1}) - v(s_t) \\
\hat{A}_t^{(2)} =& r_{t+1} + \gamma r_{t+2} + \gamma^2 v(s_{t+2}) - v(s_t) \\
\hat{A}_t^{(\infty)} =& r_{t+1} + \gamma r_{t+2} + ... + \gamma^{T-t-1} r_T - v(s_t)
\end{aligned}
$$

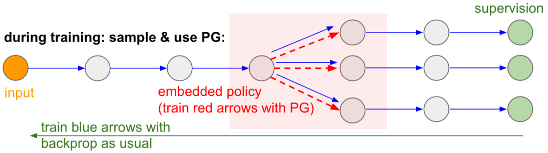
$\hat{A}^{(1)}$ has low variance and high bias. $\hat{A}^{(\infty)}$ has high variance but low bias

Overcoming Non-differentiable Computation

1. Another interesting advantage of Policy Gradient is that it allows us to overcome the non-differentiable computation



2. During training we will produce several samples (indicated by the branches below), and then we'll encourage samples that eventually led to good outcomes (in this case for example measured by the loss at the end)



Extension of Policy Gradient

1. State-of-the-art RL methods are almost all policy-based
   1. **A2C and A3C**: Asynchronous Methods for Deep Reinforcement Learning, ICML'16. Representative high-performance actor-critic algorithm: `https://openai.com/blog/baselines-acktr-a2c/`
   2. **TRPO**: Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization: deep RL with natural policy gradient and adaptive step size
   3. **PPO**: Schulman, Wolski, Dhariwal, Radford, Klimov (2017). Proximal policy optimization algorithms: deep RL with importance sampled policy gradient

Different Schools of Reinforcement Learning

1. Value-based RL: solve RL through dynamic programming
   1. Classic RL and control theory
   2. Representative algorithms: Deep Q-learning and its variant
   3. Representative researchers: Richard Sutton (no more than 20 pages on PG out of the 500-page textbook), David Silver, from DeepMind
2. Policy-based RL: solve RL mainly through learning
   1. Machine learning and deep learning
   2. Representative algorithms: PG, and its variants TRPO, PPO, and others
   3. Representative researchers: Pieter Abbeel, Sergey Levine, John Schulman, from OpenAI, Berkelely
3. Some random essay I wrote on Zhihu:`https://www.zhihu.com/question/316626294/answer/627373838`

请问DeepMind和OpenAI身后的两大RL流派有什么具体的区别?

周博磊

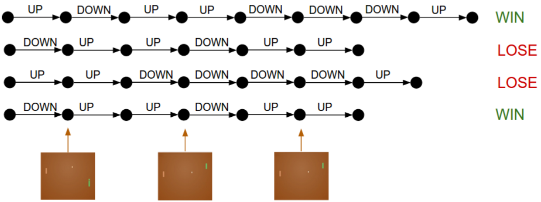
机器学习、深度学习（Deep Learning）、人工智能 话题的优秀回答者

1,681 人赞同了该回答

Policy gradient code example

1. A very nice summary of policy graident algorithms: `https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html`
2. Code example of policy gradient: `https://github.com/cuhkrlcourse/RLexample/blob/master/pg-pong.py`
3. Educational blog by Karpathy: `http://karpathy.github.io/2016/05/31/rl/`

4. Episodes/trajectories generated by the game, so PG encourages the good actions and penalizes the bad actions



Concluding Remarks

1. **Derive the policy gradient by yourself to get a deeper understanding!**
2. Next Tuesday: Mid-term review for the course project
3. 5-min presentation per team through ZOOM, team order will be randomly generated and sent out shortly
4. Presentation order is posted at Piazza `https://piazza.com/class/k6rmne6pqcl57s`