Generalized Advantage Estimation

广义优势估计

东川路第一可爱猫猫虫

主要内容

- 优势函数
- N步优势函数估计器
- 方差与偏差的权衡
- 指数加权平均
- γ与λ
- 代码实现

优势函数

- $A(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t)$
- $Q_\pi(s_t, a_t)$

  a_t的动作价值
- $V_\pi(s_t)$

  s_t的状态价值

对期望$Q_\pi(s_t, a_t) - V_\pi(s_t)$进行估计

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}\sim P}[r_t + \gamma V^\pi(s_{t+1})]$$
$$V^\pi(s_{t+1}) = \mathbb{E}_{a_{t+1}, s_{t+2}}[r_{t+1} + \gamma V^\pi(s_{t+2})]$$
$$\begin{aligned} Q^\pi(s_t, a_t) &= \mathbb{E}_{s_{t+1}}\left[r_t + \gamma \mathbb{E}_{a_{t+1}, s_{t+2}}[r_{t+1} + \gamma V^\pi(s_{t+2})]\right] \\ &= \mathbb{E}_{s_{t+1}, a_{t+1}, s_{t+2}}[r_t + \gamma r_{t+1} + \gamma^2 V^\pi(s_{t+2})] \end{aligned}$$

- Advantage estimator

  $r_t + \gamma V(s_{t+1}) - V(s_t)$

  $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$ 时序差分误差

  1-step advantage estimator
- $r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t)$

  2-step advantage estimator
- $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots - V(s_t)$

  无穷步优势估计器

  蒙特卡洛（Monte Carlo）估计

$$r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots - V(s_t)$$

- 回顾蒙特卡洛估计

  无穷步优势估计器从时刻t开始收集直到回合结束的所有奖励

  采样的是每条轨迹的真实回报

  方差大

  无偏性的必然结果

- $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots$

  从时间t开始的完整折扣回报

n-step advantage estimator里的方差和偏差

- $\hat{A}_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t)$

  高偏差 低方差
- $\hat{A}_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t)$
- ……
- $\hat{A}_t^{(inf)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots - V(s_t)$

  高方差 低偏差
- k步优势估计器

$$\hat{A}_t^{(k)} = \Sigma_{l=0}^{k-1} \gamma^l \, r_{t+l} + \gamma^k V(s_{t+k}) - V(s_t)$$

web.stanford.edu/class/cs234/slides/lecture7pre.pdf

用$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$表示n-step advantage estimator

- $\hat{A}_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t)$
  $= \delta_t^V$
- $\hat{A}_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t)$
  $= \delta_t^V + \gamma \delta_{t+1}^V$
  望远镜求和 telescoping sum
  $\delta_{t+1}^V = r_{t+1} + \gamma V(s_{t+2}) - V(s_{t+1})$
  $\gamma \delta_{t+1}^V = \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - \gamma V(s_{t+1})$
- $\hat{A}_t^{(k)} = \Sigma_{l=0}^{k-1} \gamma^l r_{t+l} + \gamma^k V(s_{t+k}) - V(s_t)$
  $= \Sigma_{l=0}^{k-1} \gamma^l \delta_{t+l}^V$

web.stanford.edu/class/cs234/slides/lecture7pre.pdf

GAE用指数进行指数加权平均

$$\hat{A}_t^{(1)} = \delta_t^V$$
$$\hat{A}_t^{(2)} = \delta_t^V + \gamma\delta_{t+1}^V$$
$$\hat{A}_t^{(k)} = \Sigma_{l=0}^{k-1}\gamma^l\delta_{t+l}^V$$

- 几何分布

在独立重复试验中，试验次数预先不能确定

设每次试验成功的概率为λ，将试验进行到成功一次为止，以X表示所需的试验次数，则X的分布律为

$$P\{X = k\} = (1-\lambda)^{k-1}\lambda, k = 1,2,\ldots$$

称随机变量X服从参数为λ的几何分布

$$\hat{A}_t^{GAE(\gamma,\lambda)} = (1-\lambda)(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \ldots)$$

- 几何分布 首项为λ，公比为$1-\lambda$
- GAE权重 首项为$1-\lambda$，公比为λ

指数加权平均

web.stanford.edu/class/cs234/slides/lecture7pre.pdf

GAE就是k步优势估计器的指数加权平均

$$
\begin{aligned}
\hat{\boldsymbol{A}}_t^{\mathrm{GAE}(\gamma,\lambda)} &= (1-\lambda)\left(\hat{\boldsymbol{A}}_t^{(1)} + \lambda \hat{\boldsymbol{A}}_t^{(2)} + \lambda^2 \hat{\boldsymbol{A}}_t^{(3)} + \ldots\right) \\
&= (1-\lambda)\left(\delta_t^V + \lambda\left(\delta_t^V + \gamma\delta_{t+1}^V\right) + \lambda^2\left(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V\right) + \ldots\right) \\
&= (1-\lambda)\left(\delta_t^V\left(1 + \lambda + \lambda^2 + \ldots\right) + \gamma\delta_{t+1}^V\left(\lambda + \lambda^2 + \ldots\right)\right. \\
&\quad \left. + \gamma^2\delta_{t+2}^V\left(\lambda^2 + \lambda^3 + \ldots\right) + \ldots\right) \\
&= (1-\lambda)\left(\delta_t^V \frac{1}{1-\lambda} + \gamma\lambda\delta_{t+1}^V \frac{1}{1-\lambda} + \gamma^2\lambda^2\delta_{t+2}^V \frac{1}{1-\lambda} + \ldots\right) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned}
$$

- exponentially-weighted average
- GAE(γ, λ) λ为GAE的权重参数

GAE$(\gamma, 0)$与GAE$(\gamma, 1)$

- GAE$(\gamma, 0)$即为TD
  - GAE退化成了经典Actor-Critic算法
  - 基于TD(0)残差的优势函数估计
  - 依赖单步reward
  - 偏差高，方差低
- GAE$(\gamma, 1)$即为MC
  - GAE退化成了reinforce算法
  - 基于蒙特卡洛方法的优势函数估计
  - 偏差低，方差高

GAE(γ,λ)里的γ

- $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots - V(s_t)$

  γ是折扣因子?

We consider an undiscounted, episodic formulation of the policy optimization problem. The initial state s_1 is sampled from ρ_1. A trajectory $(s_1, a_1, s_2, a_2, \ldots, s_T, a_T)$ is generated by sampling actions according to the policy $a_t \sim \pi(a_t \mid s_t)$ and sampling the states according to the dynamics $s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)$. A cost $c_t = c(s_t, a_t, s_{t+1})$ is incurred at each step. The goal is to minimize the expected total cost $\sum_{t=1}^{T} c_t$. Although we use episodes of length

- 以无折扣成本总和为优化目标

  没有折扣因子γ

We will introduce a parameter γ that allows us to reduce variance by downweighting costs corresponding to delayed effects, at the cost of introducing bias. This parameter is related to the discount factor used in discounted formulations of MDPs, but we treat it as a variance reduction parameter, following prior work [16, 7], and analyze different settings in our experiments. The discounted value functions are given by:

$$V^{\pi,\gamma}(s_t) := \mathbb{E}_{\substack{s_{t+1:T},\\ a_{t:T}}}\left[\sum_{l=0}^{\infty} \gamma^l c_{t+l}\right] \qquad Q^{\pi,\gamma}(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:T},\\ a_{t+1:T}}}\left[\sum_{l=0}^{\infty} \gamma^l c_{t+l}\right] \tag{4}$$

$$A^{\pi,\gamma}(s_t, a_t) := Q^{\pi,\gamma}(s_t, a_t) - V^{\pi,\gamma}(s_t). \tag{5}$$

γ

- 论文里的γ是一个为了工程实践而引入的算法参数

  而非问题定义的一部分

- 引入γ以后

  优化目标从无折扣变成了有折扣

  这就引入了偏差 变成了有偏估计

  用一个有偏差但低方差的梯度估计器，去优化一个无偏差的目标函数

GAE的代码实现

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$$

$$\begin{aligned}\hat{\boldsymbol{A}}_t^{\mathrm{GAE}(\gamma,\lambda)} &= (1-\lambda)\left(\hat{\boldsymbol{A}}_t^{(1)} + \lambda\hat{\boldsymbol{A}}_t^{(2)} + \lambda^2\hat{\boldsymbol{A}}_t^{(3)} + \ldots\right) \\ &= \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^V\end{aligned}$$

$$\hat{A}_t = \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots$$

$$\hat{A}_{t+1} = \delta_{t+1} + \gamma\lambda\delta_{t+2} + (\gamma\lambda)^2\delta_{t+3} + \cdots$$

$$\delta_t + \gamma\lambda\hat{A}_{t+1} = \delta_t + \gamma\lambda(\delta_{t+1} + \gamma\lambda\delta_{t+2} + \cdots) = \hat{A}_t$$

https://github.com/labmlai/annotated_deep_learning_paper_implementations/tree/master/labml_nn/rl/ppo/gae.py

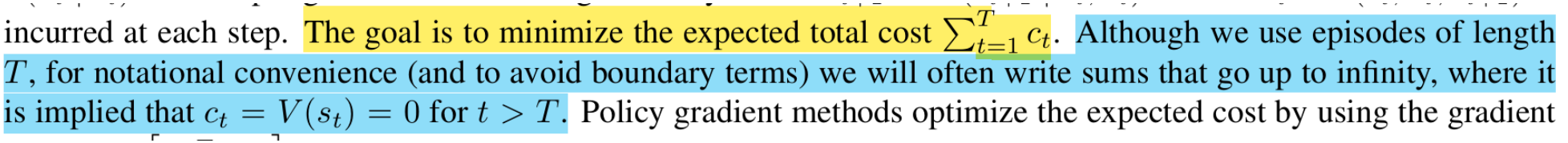
$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$$

$$\delta_t + \gamma\lambda\hat{A}_{t+1} = \delta_t + \gamma\lambda(\delta_{t+1} + \gamma\lambda\delta_{t+2} + \cdots) = \hat{A}_t$$

```python
import numpy as np
class GAE:
    def __init__(self, n_workers: int, worker_steps: int, gamma: float, lambda_:
float):
        self.lambda_ = lambda_
        self.gamma = gamma
        self.worker_steps = worker_steps
        self.n_workers = n_workers
    def __call__(self, done: np.ndarray, rewards: np.ndarray, values: np.ndarray) ->
np.ndarray:
        advantages = np.zeros((self.n_workers, self.worker_steps), dtype=np.float32)
        last_advantage = 0
        last_value = values[:, -1]
        for t in reversed(range(self.worker_steps)):
            mask = 1.0 - done[:, t]
            last_value = last_value * mask
            last_advantage = last_advantage * mask
            delta = rewards[:, t] + self.gamma * last_value - values[:, t]
            last_advantage = delta + self.gamma * self.lambda_ * last_advantage
            advantages[:, t] = last_advantage
            last_value = values[:, t]
        return advantages
```

参数: n_workers=2, worker_steps=3, gamma=0.9, lambda=0.8

输入数据:


incurred at each step. The goal is to minimize the expected total cost $\sum_{t=1}^{T} c_t$. Although we use episodes of length T, for notational convenience (and to avoid boundary terms) we will often write sums that go up to infinity, where it is implied that $c_t = V(s_t) = 0$ for $t > T$. Policy gradient methods optimize the expected cost by using the gradient

done:
[[0. 0. 1.]
 [0. 0. 0.]]
rewards:
[[1. 2. 3.]
 [0.5 1.5 2.5]]
values:
[[0.1 0.2 0.3 0.]
 [0.1 0.2 0.3 0.]]
初始化: last_value = [0. 0.], last_advantage = 0

--- 时间步 t=2 ---
done[:, 2] = [1. 0.]
mask = 1 - done[:, 2] = [0. 1.]
应用mask前: last_value = [0. 0.], last_advantage = 0
应用mask后: last_value = [0. 0.], last_advantage = [0. 0.]
rewards[:, 2] = [3. 2.5]
values[:, 2] = [0.3 0.3]
delta = rewards + gamma*last_value - values = [2.7 2.2]
new last_advantage = delta + gamma*lambda*last_advantage = [2.7 2.2]
advantages[:, 2] = [2.7 2.2]
更新last_value = values[:, 2] = [0.3 0.3]

--- 时间步 t=1 ---
done[:, 1] = [0. 0.]
mask = 1 - done[:, 1] = [1. 1.]
应用mask前: last_value = [0.3 0.3], last_advantage = [2.7 2.2]
应用mask后: last_value = [0.3 0.3], last_advantage = [2.7 2.2]
rewards[:, 1] = [2. 1.5]
values[:, 1] = [0.2 0.2]
delta = rewards + gamma*last_value - values = [2.07   1.5699999]
new last_advantage = delta + gamma*lambda*last_advantage = [4.014 3.154]
advantages[:, 1] = [4.014 3.154]
更新last_value = values[:, 1] = [0.2 0.2]

--- 时间步 t=0 ---
done[:, 0] = [0. 0.]
mask = 1 - done[:, 0] = [1. 1.]
应用mask前: last_value = [0.2 0.2], last_advantage = [4.014 3.154]
应用mask后: last_value = [0.2 0.2], last_advantage = [4.014 3.154]
rewards[:, 0] = [1. 0.5]
values[:, 0] = [0.1 0.1]
delta = rewards + gamma*last_value - values = [1.0799999 0.58   ]
new last_advantage = delta + gamma*lambda*last_advantage = [3.97008
2.8508801]
advantages[:, 0] = [3.97008   2.8508801]
更新last_value = values[:, 0] = [0.1 0.1]

最终advantages矩阵:
[[3.97008   4.014   2.7   ]
 [2.8508801 3.154   2.2   ]]