Reinforcement Learning

Zhuoyuan Chen*
Facebook AI Research
Menlo Park, CA 94025
chenzhuoyuan07@gmail.com

Abstract

RL: Policy Gradient, Q-Learning, actor-critic

1 Policy Gradient

Reward, trajectories: τ:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\sum_t r(s_t, a_t)] \approx \frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_{i,t}) \tag{1}$$

where

$$J(\theta) = \int \pi_\theta(\tau) r(\tau) d\tau$$

Then

$$\nabla_\theta J(\theta) = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau = \mathbb{E}_\tau[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

1. Causal in r;
2. Minus baseline to reduce variance, minimum variance achieved at

$$b = \frac{\mathbb{E}[g(\tau)^2 r(\tau)]}{\mathbb{E}[g(\tau)^2]}$$

3. On policy;

Importance sampling:

$$\mathbb{E}_{x \sim p(x)}[f(x)] = \mathbb{E}_{x \sim q(x)}[\frac{p(x)}{q(x)} f(x)]$$

Given a trajectory, we have

$$\frac{\pi_\theta(\tau)}{\bar{\pi}_\theta(\tau)} = \frac{\prod_{t=1}^T \pi_{\theta(a_t|s_t)}}{\prod_{t=1}^T \bar{\pi}_{\theta(a_t|s_t)}}$$

Then off-policy PG with IS:

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(a_t|s_t)(\prod_{t'=1}^t \frac{\pi_{\theta'}(a_{t'}|s_{t'}}{\pi_\theta(a_{t'}|s_{t'}})(\sum_{t'=t}^T r(s_{t'}, a_{t'}) \prod_{t''=t}^{t'} \frac{\pi_{\theta'}(a_{t''}|s_{t''})}{\pi_\theta(a_{t''}|s_{t''})})] \tag{2}$$

*

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

1.1 TRPO

$$\begin{aligned}
J(\theta') - J(\theta) &= J(\theta') - \mathbb{E}_{s_0 \sim p(s)}[V^{\pi_\theta}(s_0)] \\
&= J(\theta') - \mathbb{E}_{\tau \sim p_{\theta'}(\tau)}[V^{\pi_\theta}(s_0)] \\
&= \mathbb{E}_{\tau \sim p_{\theta'}(\tau)}[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t)] + [\sum_{t=1}^{\infty} \gamma^t (\gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t))] \\
&= \mathbb{E}_{\tau \sim p_{\theta'}(\tau)}[\sum_{t=1}^{\infty} \gamma^t A^{\pi_\theta}(s_t, a_t)]
\end{aligned}$$

1.2 PPO

Loss: let

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

Then,

$$L^{CLIP}(\theta) = \mathbb{E}[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{3}$$

PPO system design: same as A2C (check section 2.1 actor-critic), only difference:

```
step 4: agent.update()
  ratio = exp(action_log_prob - old_action_log_prob)
  surr1 = ratio * adv_target
  surr2 = clamp(ratio, 1.0-eps, 1.0+eps) * adv_target
  action_loss = -min(surr1, surr2)
```

1.3 ACKTR

1. Paper: Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation [NIPS 2017];
2. Use the **K-FAC** technique to approximate Fisher Information matrix in paper [J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature.]

Important properties of Kronecker operator $\otimes$:

1. $(P \otimes Q)^{-1} = P^{-1} \otimes Q^{-1}$;
2. $(P \otimes Q)vec(T) = PTQ^T$

Let $W \in \mathbb{R}^{C_{out} \times C_{in}}$ be the weight matrix of the l^{th} layer, with $s = Wa$, we have K-FAC:

$$F_l = \mathbb{E}[vec\{\nabla_W L\} vec\{\nabla_W L\}^T] = \mathbb{E}[aa^T \otimes \nabla_s L(\nabla_s L)^T] \tag{4}$$

$$\approx \mathbb{E}[aa^T] \otimes \mathbb{E}[\nabla_s L(\nabla_s L)^T] = A \otimes S := \hat{F}_l \tag{5}$$

Then,

$$vec(\triangle W) = \hat{F}_l^{-1} vec\{\nabla_W J\} = vec(A^{-1} \nabla_W J S^{-1}) \tag{6}$$

In actor-critic setting (two output heads– action a and critic v), we will try to apply $\mathbb{E}_{p(\tau}[\nabla \log p(a, v|s) \nabla \log p(a, v|s)^T]$.

Step-size, following TRPO, SGD-like $\theta \leftarrow \theta - \eta F^{-1} \nabla_\theta L$ will result in large updates in policy and make algorithm prematurely converge to near-deterministic. Therefore, we adopt step-size as:

$$\eta = \min(\eta_{\max}, \sqrt{\frac{2\delta}{\triangle\theta^T \hat{F} \triangle\theta}}) \tag{7}$$

System design:

1. Agent (ACKTR) initialization:

(a) Split modules (bias)
(b) Register forward pre-hook (save input)
(c) Register backward hook (save gradient output)
(d) Running stats:

```
self.m_aa = {} # save input.t() @ input
self.m_gg = {}
self.Q_a, self.Q_g = {}, {}
self.d_a, self.d_g = {}, {}
```

2. Agent (ACKTR) update:
   (a) Before forward: compute running stat of $a^T a$ and save in m-aa;
   (b) During backward: compute running stat of $g^T g$
3. Optimizer.step() for each module:
   (a) Eig SVD of inputs: $Cov(a, a) = Q_a D_a Q_a^T$;
   (b) Eig SVD of output gradient: $Cov(g, g) = Q_g D_g Q_g^T$;
   (c) Update with normal gradient G and update each module's gradient with $\triangle\theta$:
$$\triangle\theta = Q_g \frac{Q_g^T G Q_a}{d_g^T d_a + 0.01} Q_g^T \tag{8}$$
   (d) Normal SGD;

2 Actor-Critic

value function, advantage:

$$\begin{aligned} Q^\pi(s_t, a_t) &= \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}[r(s_{t'}|s_t, a_t)] \\ V^\pi(s_t) &= \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)}[Q^\pi(s_t, a_t)] \\ A^\pi(s_t, a_t) &= Q^\pi(s_t, a_t) - V^\pi(s_t) \end{aligned}$$

1. Take action $a \sim \pi_\theta(a|s)$, get (s, a, s', r);
2. Update $\hat{V}_\phi^\pi$ using $r + \gamma\hat{V}(s')$
3. Evaluate $A(s, a) = r(s, a) + \gamma\hat{V}(s') - \hat{V}(s)$
4. PG: $\nabla_\theta \approx \nabla_\theta \log \pi(a|s)\hat{A}^\pi(s, a)$
5. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

What value should we use to multiply with $\nabla \log \pi(a_t|s_t)$?

1. low-variance, biased: $r(s, a) + \gamma V(s_{t+1}) - V(s_t)$;
2. unbiased, high-variance: $\sum_{t'}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'}) - b$
3. Eligibility traces and n-step returns: a good balance
$$\hat{A}(s_t, a_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(s_{t'}, a_{t'}) + \gamma^n \hat{V}(s_{t+n}) - \hat{V}(s_t) \tag{9}$$
4. GAE:
$$\hat{A}_{GAE}^\pi(s_t, a_t) = \sum_{n=1}^{\infty} w_n \hat{A}_n^\pi(s_t, a_t) \tag{10}$$
   with $w_n \propto \lambda^{n-1}$
$$\hat{A}_{GAE}^\pi(s_t, a_t) = \sum_{n=1}^{\infty} (\gamma\lambda)^{t'-t} \delta_{t'} \tag{11}$$

2.1 System Design

1. Batch-size = num of games (=16 running envs in our scenarios);
2. Policy (actor-critic):
   (a) Shared CNNBase module: output a (?, n) dim feature;
   (b) (Optional) x, rnn-hxs = CNNBase.forward-gru(x, rnn-hxs, masks)
   (c) Actor head: feature -> $(?, n_a)$
   (d) Critic head: feature -> $(?, 1)$
   (e) Actor head wraps a distribution; for sampling action a_t and log-prob $\log \pi_\theta(a_t|s_t)$
3. Rollouts (data pool):
   (a) Observation: (T+1, ?, 4, 84, 84)
   (b) RNN hidden state: (T+1, ?, nh)
   (c) Rewards: (T, ?, 1)
   (d) Value-preds: (T+1, ?, 1)
   (e) Returns: (T+1, ?, 1)
   (f) Action-log-prob: (T, ?, 1)
   (g) Actions: (T, ?, 1)
   (h) Masks: (T+1, ?, 1); 1.0 for non-terminal, 0. for terminal;
4. Agent (A2C-ACKTR, gail, kfac, ppo)
   (a) actor-critic: policy;
   (b) Coefficients of different terms;
   (c) Optimizer (KFAC, RMSprop);
   (d) update();
5. Running Logic:

```
for n steps:
  step 1: for t = 1..5 (no-gradient mode)
    value, act, act-log-prob, rnn-h = actor_critic.act(obs[step], rnn-h, mask[step])
    obs, reward, done, infos = envs.step(action)
    update mask
    rollouts update: obs, rnn-h, action, action-log-prob,
       value, reward, masks, bad-masks
  step 2: (no-gradient mode)
    next_value = actor_critic.get_value(obs[-1]) # get value for T+1
  step 3: (n-step true target value)
    rollouts.compute_returns(next_value) # (T, ?, 1)
    can also be GAE;
  step 4: update parameter
     value_loss, action_loss, dist_entropy = agent.update()
       1. value, action-log-prob, entropy, rnn-hs = actor_critic.evaluate_actions()
         value, action-feat, rnn-hs = CNNBase(inputs, rnn, masks) # (T * ?, 4, 84, 84)
         dist (action distribution)
         action-log-probs
         dist-entropy
       2. Advantages: A(st, at) = rollouts.returns - values
       3.1 action-loss: -(adv * action-log-prob)
       3.2 value-loss: adv ^ 2
       3.3 entropy: -coeff * dist-entropy
  step 5: rollouts.after_update()
     reset T+1 step obs, rnn-hs, masks, bad-masks to [0]
```

3 Value Function

Q-Learning:

1. Collect (s, a, s', r) with some policy (**ϵ-greedy exploration**), could do with **replay buffer**
2. $y_i \leftarrow r(s_i, a_i) + \gamma \max_a Q(s', a)$
3. $\phi \leftarrow \arg\max_{\phi'} \sum_i ||Q_{\phi'}(s, a) - y||^2$

Boltzmann exploration:

$$\pi(a_t|s_t) \sim \exp(Q_\phi(s_t, a_t))$$

Advanced tricks:

1. Double Q-learning: another network to select a':
$$y = r + \gamma Q_{\phi'}(s', \arg\max_{a'} Q_\phi(s', a')) \tag{12}$$
2. Q-learning with n-steps: less-biased, typically faster learning especially early on;
3. Continuous actions: DDPG, another network for action a;
4. Prioritized experience replay;
5. Clip gradient or Huber loss;