PPO(Proximal Policy Optimization)

Paper Link: https://arxiv.org/pdf/1707.06347.pdf

Key Features

- PPO is an SGD(Stochastic Gradient Descent)-based optimization algorithm that learns with the goal of surrogate objective
- PPO is an algorithm that alternates between policy-based data sampling and sample-based optimization
- Compared to TRPO, PPO has higher stability and reliability for TR(trust region)
- PPO has a simple structure, so it is easy to implement, but its performance is strong.

Background

The problem with Policy Gradient algorithm

Policy gradient algorithm has the property of performing multiple optimizations for the same sample. This property is likely to destabilize learning by updating the agent's policy empirically.

The problem with TRPO(Trust Region Policy Optimization)

TRPO(Trust Region Policy Optimization) is a policy gradient series algorithm based on a stochastic policy. Trust region refers to a section that guarantees an update in the direction in which performance rises. TRPO uses trust region to update policy to optimum.

TRPO collects state-action pairs using the single path method or vine method and approximates Q-functions in the Monte Carlo method. Single path method is a method of updating the policy by Q-function after collecting a series of states and actions for the old policy. Vine method is a method of learning more diverse distributions by collecting all of the states available for visit due to action in a parallelized trajectories method. After taking average for the collected samples, true objective(η) is obtained through the following equation.

$$\eta(\theta) \geq L_{\theta_{old}}(\theta) - CD_{KL}^{max}(\theta_{old}, \theta)$$

Improvement of true objective η is guaranteed through surrogate function maximization. However, a practical problem appears here. In this process, the step size for the lower boundary becomes very small as the penalty coefficient C actually enters a very large value. To solve this problem, trust region constraint was implemented using KL divergence as a construct as follows, not by giving penalty using KL divergence.

$$maximize\ L_{\theta_{old}}(\theta) \tag{1}$$
$$subject\ to\ D_{KL}^{max}(\theta_{old}, \theta) \leq \delta \tag{2}$$

Since it is practically impossible to obtain the maximum value for KL divergence in this process, it is replaced by obtaining the average value of sampling-based KL divergence as follows.

–

$$\bar{D}_{KL}^{\rho}(\theta_1, \theta_2) := \mathbb{E}_{s\sim p}[D_{KL}(\pi_{\theta_1}(\cdot|s) \parallel \pi_{\theta_2}(a|s))], \tag{3}$$

$$\underset{\theta}{maximize}\ L_{\theta_{old}}(\theta), \tag{4}$$

$$subject\ to\ \bar{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta \tag{5}$$

If you develop the local approximation term in the above formula, it can be expressed as follows.

$$maximize \sum_{s}\rho_{\theta_{old}}(s)\sum_{a}\pi_{\theta}(a|s)A_{\theta_{old}}(s,a) \tag{6}$$

$$subject\ to\ \ \bar{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta \tag{7}$$

For this equation, the sum for state is replaced to an expectation for visitation frequency, the advantage function is replaced to Q-function, and the sum for action is replaced by an importance sampling estimator. Through this transformation, general and practical formulas were derived as follows, and these are the optimization formulas used in TRPO.

$$\underset{\theta}{maximize}\ \mathbb{E}_{s\sim\rho_{\theta},a\sim}[\frac{\pi_{\theta}(a|s)}{q(a|s)}Q_{\theta_{old}}(s,a)] \tag{8}$$

$$subject\ to\ E_{s\sim\rho_{\theta_{old}}}\left[D_{KL}(\pi_{\theta_{old}}(\cdot|s) \parallel \pi_{\theta}(\cdot|s))\right] \leq \delta \tag{9}$$

However, the TRPO optimization method discussed above has a disadvantage in that the content is difficult and thus the implementation becomes complicated.

In addition, the use of these surrogate objectives is likely to increase the policy update step and thus can’t guarantee TRPO’s monotonous improvement. So, If this is

maximized without constraint, the policy(π) learning becomes unstable due to the increase in the update width of the policy parameters(θ).

Method

Clipped surrogate objective

The problem of instability in learning according to the policy update width of TRPO is solved by placing constraint on surrogate objective as follows.

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

The new constraint restrict surrogate objective in a way that excludes probability ratio($r_t(\theta)$) whose computation value exists outside of clipping range. Based on this equation, it is possible to learn stably because the constraint restrict amount of change, which is difference between old policy(θ_{old}) and new policy(θ), in clipping range. In addition, another advantage is that it is much easier to implement than the TRPO method.

Algorithm

In PPO, L^{CLIP} should be used instead of existing L^{PG} for automatic differential calculation and mini-batch SGA(Stochastic Gradient Ascent) based on this surrogate objective should be carried out.

Most of the techniques for calculating the variance-reduced advantage-function estimators use a learned state-value estimation(V). However, if policy network(actor) and value estimator(critic) share parameters, loss function should be combined with surrogate objective of policy network and error term of value estimator. In addition, if entropy bonus is added to the surrogate objective, It makes algorithm’s performance

stronger because entropy bonus ensure sufficient exploration. The loss function is shown as follows.

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$$

where $c1,\ c2$ are coefficient, and S denotes an entropy bonus, and L_t^{VF} is a squared-error loss of the value $(V_\theta(s_t) - V_t^{targ})^2$. One popular way to implement this is to sample data, which is not episode(full-batch) but multi-step(mini-batch), for policy during time step(T). This style requires an advantage $\hat{A}$ estimator that does not look beyond time step T. The estimator is as follows. Prior to calculating the objective function based on sampling, the first thing to be pointed out is the GAE(Generalized Advantage Estimation), an essential concept for calculating $\hat{A}$.

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T)$$

The GAE obtains a weighted sum by taking an exponential moving average for TD errors as follows. In addition, $(1-\lambda)$ is needed to make weighted sum which converge to 1.

$$TD(\lambda) \to \sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} A_t^{(n)}, \tag{10}$$

$$lim_{n\to\infty} TD(\lambda) = 1 \tag{11}$$

GAE uses an equation which is as follows expressed as the sum of δ to calculate the above formula.

$$A_t^{(1)} = R_t + \gamma V(s_t + 1) \triangleq \delta_t \tag{12}$$
$$A_t^{(2)} = R_t + \gamma R_{t+1} + \gamma^2 V(s_t + 1) \tag{13}$$
$$= (R_t + \gamma V(s_{t+1}) - V(s_t)) + \gamma(R_{t+1} + \gamma V(s_{t+2}) - V(s_{t+1})) \tag{14}$$
$$= (\delta_t + \gamma\delta_{t+1}) + \gamma(\delta_{t+1}), \tag{15}$$
$$A_t^n = \sum_{k=t}^{t+n-1} \gamma^{r-t}\delta_k \tag{16}$$

Equation (10) is expressed according to the above equation as follows. Equation (21) is called GAE.

$$TD(\lambda) \to \sum_{n=1}^{\infty}(1-\lambda)\lambda^{n-1}A_t^{(n)} \tag{17}$$
$$= (1-\lambda)(\delta_t + \lambda(\delta_t + \gamma\delta_{t+1}) + \lambda^2(\delta_t + \delta_{t+1} + \gamma(\delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2})) \tag{18}$$
$$= (1-\lambda)(\delta_t(1 + \lambda + \lambda^2 + \cdots) + \gamma\delta_{t+1}(\lambda + \lambda^2 + \cdots) + \cdots) \tag{19}$$
$$= (1-\lambda)(\delta_t \times \frac{1}{1-\lambda} + \gamma\delta_{t+1} \times \frac{1}{1-\lambda} + \gamma^2\delta_{t+2} \times \frac{1}{1-\lambda} + \cdots) \tag{20}$$
$$= \sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta_k \triangleq A_t^{GAE} \tag{21}$$

PPO uses advantage $\hat{A}$ calculated in this way as follows.

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Implementation on JORLDY

- PPO JORLDY Implementation

```
### learn function ###
# 1. calculate probability of old policy and advantage
# 2. calculate probability of policy
# 3. calculate ratio(importance), actor loss by surrogate objective,
#    critic loss, entropy bonus
# 4. calculate L(CLIP+VF+S)

def learn(self):
...

    # set prob_a_old and advantage
    with torch.no_grad():
        if self.action_type == "continuous":
            mu, std, value = self.network(state)
            m = Normal(mu, std)
            z = torch.atanh(torch.clamp(action, -1 + 1e-7, 1 - 1e-7))
            log_prob = m.log_prob(z)
        else:
            pi, value = self.network(state)
            log_prob = pi.gather(1, action.long()).log()
        log_prob_old = log_prob

        next_value = self.network(next_state)[-1]
        delta = reward + (1 - done) * self.gamma * next_value - value
        adv = delta.clone()
        adv, done = adv.view(-1, self.n_step), done.view(-1, self.n_step)
        for t in reversed(range(self.n_step - 1)):
            adv[:, t] += (
                (1 - done[:, t]) * self.gamma * self._lambda * adv[:, t + 1]
            )
```

```
        ret = adv.view(-1, 1) + value

        if self.use_standardization:
            adv = (adv - adv.mean(dim=1, keepdim=True)) / (
                adv.std(dim=1, keepdim=True) + 1e-7
            )

        adv = adv.view(-1, 1)

    mean_ret = ret.mean().item()

    # start train iteration
    actor_losses, critic_losses, entropy_losses, ratios, probs = [], [], [], [], []
    idxs = np.arange(len(reward))
    for _ in range(self.n_epoch):
        np.random.shuffle(idxs)
        for offset in range(0, len(reward), self.batch_size):
            idx = idxs[offset : offset + self.batch_size]

            _state, _action, _value, _ret, _adv, _log_prob_old = map(
                lambda x: [_x[idx] for _x in x] if isinstance(x, list) else x[idx],
                [state, action, value, ret, adv, log_prob_old],
            )

            if self.action_type == "continuous":
                mu, std, value_pred = self.network(_state)
                m = Normal(mu, std)
                z = torch.atanh(torch.clamp(_action, -1 + 1e-7, 1 - 1e-7))
                log_prob = m.log_prob(z)
            else:
                pi, value_pred = self.network(_state)
                m = Categorical(pi)
                log_prob = m.log_prob(_action.squeeze(-1)).unsqueeze(-1)

            ratio = (log_prob - _log_prob_old).sum(1, keepdim=True).exp()
            surr1 = ratio * _adv
            surr2 = (
                torch.clamp(
                    ratio, min=1 - self.epsilon_clip, max=1 + self.epsilon_clip
                )
                * _adv
            )
            actor_loss = -torch.min(surr1, surr2).mean()

            value_pred_clipped = _value + torch.clamp(
                value_pred - _value, -self.epsilon_clip, self.epsilon_clip
            )

            critic_loss1 = F.mse_loss(value_pred, _ret)
            critic_loss2 = F.mse_loss(value_pred_clipped, _ret)

            critic_loss = torch.max(critic_loss1, critic_loss2).mean()

            entropy_loss = -m.entropy().mean()
```

```
            loss = (
                actor_loss
                + self.vf_coef * critic_loss
                + self.ent_coef * entropy_loss
            )

...
```

References

Relevant papers

- High-Dimensional Continuous Control Using Generalized Advantage Estimation
  (Schulman et al, 2016)
- Emergence of Locomotion Behaviours in Rich Environments
  (Heess et al, 2017)

Public implementations

- baselines
  (OpenAI)
- modularRL
  (Caution: this implements PPO-penalty instead of PPO-clip)
- rllab
  (Caution: this implements PPO-penalty instead of PPO-clip)
- rllib
  (Ray)