Deep Reinforcement Learning for Sparse Rewards

Fucheng Li [1] Shelvin Pauly [1]

Abstract

Despite the major advances in reinforcement learning due to deep learning, oftentimes the performance of these algorithms falls short due to the sparsity of reward feedback. Many reward functions would only indicate success or partially success. These algorithms fail in this regard is largely due to the need of large number of exploration actions to construct a useful policy. Therefore, we have implemented a recent algorithm Learning Online with Guidance Offline (LOGO) algorithm and attempted to further improve its performance via Clipped Double Q-Learning and Proximal Policy Optimization Algorithm. Our algorithm is able to obtain a better performance in some of the commonly used benchmarks adjusted to sparse settings. At the end, we included a discussion on the further improvement of LOGO and similar algorithms.

1. Introduction

In many real-life applications, reinforcement learning algorithms have been introduced with a challenging problem of large state-action space. The exploration becomes especially important when a sparse intuitive reward function is implemented indicating success or partial success. One such example is a sparse recommendation system or a robotic way point task (Li, 2017). With no intermediate information available, it's difficult for the existing reinforcement learning algorithms to construct a viable policy within a reasonable amount of time.

In many of these problems, they would have access to data gathered using a sub-optimal behavioral policy to help the algorithm learning process. There are some existing policy gradient algorithms (Schulman et al., 2015; Lillicrap et al., 2015) that can perform well under dense reward settings. However, these algorithms were unable to construct a viable reward policy within reasonable time as shown in benchmarks (Rengarajan et al., 2022). Learning Online with Guidance Offline (LOGO) was introduced recently to utilize the Trust Region Policy Optimization (TRPO) algorithm with offline demonstration data for guidance in a sparse reward setting(Rengarajan et al., 2022). The algorithm can be described in two steps:

- generate a candidate policy using TRPO.
- search for a policy closest to the behavioral policy using the candidate policy.

This method outperforms all methods in all of its benchmarks as well as achieve great performance in a real-life robotic way point task.

With LOGO's performance in mind, we attempted to look into further improving its performance using two known reinforcement learning optimization techniques. We want to know can these common techniques still applicable for algorithms designed to work with sparse reward environments. At the end of this paper, we proposed a new direction to potentially reduce the run time of such algorithms.

1.1. Background

Reinforcement learning can be described as a Markov Decision Process (MDP) in which we denote as $< S, A, R, P, \gamma >$, where S is the state space, A is the action space, $R : S \times A \to \mathbb{R}$ is the reward function, $P : S \times A\times \to [0, 1]$ is the transition probability function with $P(s'|s, a)$ representing the transition probability from state s to s' if action a is taken, and $\gamma \in (0, 1)$ is the discount factor. A policy π is a mapping from S to probability distribution over A, so $\pi(s, a)$ is the probability distribution of taking a in state s. The objective of reinforcement learning is to find the optimal policy to maximize the following objective

$$J_R(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)] \tag{1}$$

, where τ is the trajectory generated by policy π such that $s_0 \sim \mu$, $a_t \sim \pi(s_t, \cdot)$, $s_{t+1} \sim P(\cdot|s_t, a_t)$, and μ is initial state distribution. citeoverview.

[1]Department of Computer Science, University of Maryland, College Park. Correspondence to: Fucheng Li <lifc@umd.edu>, Shelvin Pauly <spauly@umd.edu>.

A few functions is computed to measure the performance of the reinforcement algorithms. The value function of a policy π is defined as $V^{\pi}_{R}(s) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty}\gamma^{t}R(s_t,a_t)|s_0 = s]$, which is the average cumulative discounted reward by following the policy π starting from the state s. The action-value or $Q-$value function of a policy is π is $Q^{\pi}_{R}(s,a) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty}\gamma^{t}R(s_t,a_t|s_0 = 0s, a_0 = a]$. The advantage function is $A^{\pi}_{R}(s,a) = Q^{\pi}_{R}(s,a) - V^{\pi}_{R}(s)$. All of the expectation in computation would be replaced with empirical expectation. The discounted state visitation distribution for policy π is $d^{\pi}(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^{t}\mathbb{P}(s_t = s|\pi)$(Li, 2017).

Before introduction LOGO, it's important to discuss a few other algorithms building up to it. LOGO is built on top of TRPO, which is a policy gradient methods. Most of the policy gradient methods would define a loss function as

$$L(\theta) = \mathbb{E}[\log\pi_{\theta}(a_t|s_t)A_t] \tag{2}$$

, where θ is the parameters of the current policy and A_t is the estimated advantage function at timestep t. Then, the algorithm would obtain the gradient of this function

$$g_t = \mathbb{E}_t[\nabla_{\theta}\log\pi_{\theta}(a_t|s_t)A_t] \tag{3}$$

This gradient can then be plug into a gradient descent algorithm for optimization.

If we use a deep neural network to estimate the policy, then we would get the Deep Policy Gradient algorithm (Lillicrap et al., 2015). For a further improvement, we can utilized an idea from Double Deep Q-Learning (DDQN), using two separate neural networks to estimate the policy (van Hasselt et al., 2015). But DDQN has a tendency to overestimate in actor-critics or policy gradient setting, there's an improvement to introduce Clipped Double Q-Learning (CDQ) (Fujimoto et al., 2018), which can be formulated as

$$y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \pi_{\theta_1}(s')) \tag{4}$$

, y_1 is the target value estimate and θ' denotes the critics' parameters.

Trust Region Policy Optimization algorithm works by maximizing an objective function while subject to a constraint on the size of the policy update (Schulman et al., 2015). We are going to follow the notation found in the LOGO paper.

$$\arg\max_{\pi} \mathbb{E}_{s\sim d^{\pi_k}, a\sim\pi(s,\cdot)}[A^{\pi_k}_{R}(s,a)] \;\; \text{s.t.} \;\; D^{\pi_k}_{KL}(\pi,\pi_k) \le \delta \tag{5}$$

, where $D^{\pi_k}_{\mathrm{KL}}(\pi,\pi_k) = \mathbb{E}_{s\sim d^{\pi_k}}[\mathrm{KL}(\pi(s,\cdot),\pi_k(s,\cdot))]$ and KL is the Kullback-Leibler divergence between two distribution defined as $\mathrm{KL}(p,q) = \sum_x p(x)\log\frac{p(x)}{q(x)}$. This is an equivalent to the original TRPO formulation. Since we are sampling states from d^{π_k}, we can emit the original $\frac{\pi_{k+1}(a_t|s_t)}{\pi_k(a_t|s_t)}$ from thus formulation. An improvement on this algorithm is Proximal Policy Optimization (Schulman et al., 2017). PPO would optimize the following objective directly instead of subjecting to a constraint

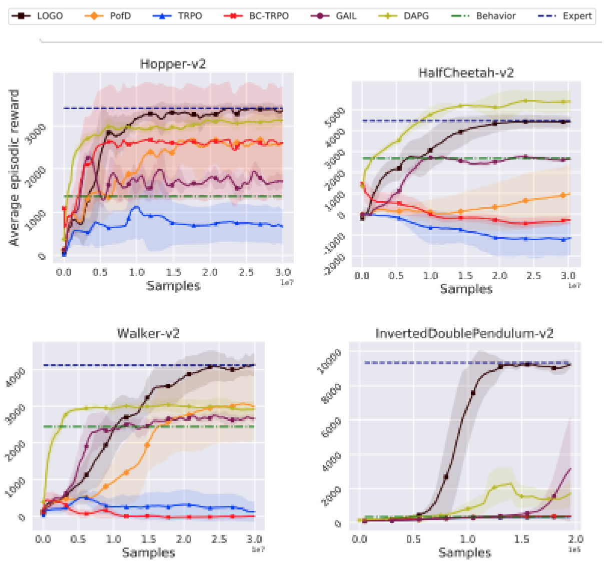

Figure 1. Evaluation of algorithms on four sparse reward MuJoCo environments with complete offline data. The expert line was created by running a TRPO algorithm in a dense reward environment(Rengarajan et al., 2022).

$$\arg\max_{\pi} \mathbb{E}_{s\sim d^{\pi_k}, a\sim\pi(s,\cdot)}[A^{\pi_k}_{R}(s,a) - \beta\mathrm{KL}(\pi,\pi_k)] \tag{6}$$

β is a penalty coefficient which can be modified adaptively. There were two methods proposed in the original TRPO paper, PPO-penalty and PPO-Clip. PPO-Clip was selected in our algorithm because it outperforms PPO-penalty in the benchmarks done by the original authors (Schulman et al., 2015). The clip function

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \ge 0 \\ (1-\epsilon)A & A < 0 \end{cases} \tag{7}$$

With all of these, LOGO can be broken down into a two step TRPO. The algorithm assumes there exists some demonstration data with the form $\mathcal{D} = {\tau^{i}}_{i=1}^{n}$, where $\tau^{i} = (s^{i}_{1}, a^{i}_{1}, ..., s^{i}_{T}, a^{i}_{T}), \tau_i \sim \pi_b$. One assumption is made for the behavior policy π_b, $\mathbb{E}_{a\sim\pi_b}[A^{\pi_k}_{R}(s,a) \ge \beta > 0], \forall s$. This assumption makes sense because the behavior policy is expected to outperform any untrained policy. The first step of LOGO is obtained by TRPO using (5) to obtain $\pi_{k+\frac{1}{2}}$. The second step can be computed according to (8).

$$\pi_{k+1} = \arg\min_{\pi} D^{\pi_k}_{\mathrm{KL}}(\pi,\pi_b) \;\; \text{s.t.} \;\; D_{\mathrm{KL}^{\max}}(\pi,\pi_{k+\frac{1}{2}}) \le \delta_k \tag{8}$$

Algorithm 1 LOGO Algorithm

```
1: Initialization: Initial policy $\pi_0$, demonstration data $\mathcal{D}$ or behavior policy $\pi_b$
2: for k = 0, 1, 2, ... do
3:     Collect $(s, a, r, s') \sim \pi_k$ and store in $\pi_{||}$
4:     if $\pi_b$ is known then
5:         $\mathcal{C}_{\pi_k}(s, a) = \log(\pi_k(s, a)/\pi_b(s, a))$
6:     else
7:         Train a discriminator $B(s, a)$ according to (11)
8:         $\mathcal{C}_{\pi_k}(s, a) = -\log B(s, a)$
9:     end if
10:    Estimate $g_k$ and $F_k$ using $\mathcal{D}_{\pi_k}$
11:    Perform policy improvement step
12:    Decay $\delta_k$ geometrically, $\delta_k \leftarrow \alpha * \delta_k$ if average return in the current iteration is too large.
13:    Estimate $h_k$ and $L_k$ using $C_{\pi_k}$ and $\mathcal{D}_{\pi_{k+\frac{1}{2}}}$
14:    Perform the policy guidance step
15: end for
```

Algorithm 2 LOGO-Modified Algorithm

```
1: Initialization: Initial policy $\pi_0$, demonstration data $\mathcal{D}$ or behavior policy $\pi_b$
2: for k = 0, 1, 2, ... do
3:     Collect $(s, a, r, s') \sim \pi_k$ and store in $\pi_{||}$
4:     if $\pi_b$ is known then
5:         $\mathcal{C}_{\pi_k}(s, a) = \log(\pi_k(s, a)/\pi_b(s, a))$
6:     else
7:         Train a discriminator $B(s, a)$ according to (11)
8:         $\mathcal{C}_{\pi_k}(s, a) = -\log B(s, a)$
9:     end if
10:    Perform policy improvement step with PPO using $\mathcal{D}_{\pi_k}$
11:    Decay $\delta_k$ geometrically, $\delta_k \leftarrow \alpha * \delta_k$ if average return in the current iteration is too large.
12:    Perform the policy guidance step with PPO using $C_{\pi_k}$ and $\mathcal{D}_{\pi_{k+\frac{1}{2}}}$
13: end for
```

Since it's infeasible to search through all possible policies, the authors of LOGO followed the linear search approach similar to the original TRPO and developed a practical algorithm. The parameters of the policy θ can be updated by

$$\theta_{k+\frac{1}{2}} = \theta_k + \sqrt{\frac{2\delta}{g_k^T F_k{-1} g_k}} F_k^{-1} g_k \tag{9}$$

$$\theta_{k+1} = \theta_{k+\frac{1}{2}} - \sqrt{\frac{2\delta_k}{h_k^T L_k^{-1} h_k}} L_k^{-1} h_k, \tag{10}$$

where $g_k = \nabla_\theta \mathbb{E}_{s\sim d^{\pi_k}, a\sim\pi(s,\cdot)}[A_R^{\pi_k}(s, a)]$, $F_k = \nabla_\theta^2 D_{\text{KL}}^{\pi_k}(\pi_\theta, \pi_k)$, $h_k = \nabla_\theta \mathbb{E}_{s\sim d^{\pi_{k+\frac{1}{2}}}, a\sim\pi_\theta(s,\cdot)}[A_{C_{\pi_{k+\frac{1}{2}}}}^{\pi_{k+\frac{1}{2}}}(s, a)]$, and $L_k = \nabla_\theta^2 D_{\text{KL}}^{\pi_{k+\frac{1}{2}}}(\pi_\theta, \pi_{k+\frac{1}{2}})$. $A_{C_\pi}^{\pi}$ or the estimated advantage is difficult to compute when the behavior policy π_b is unknown. The authors in LOGO used an idea from Generative Adversarial Networks (GANS) to train a discriminator using demonstration data $\mathcal{D}$ and the data $\mathcal{D}_{k+\frac{1}{2}}$ generated by the policy $\pi_{k+\frac{1}{2}}$ that can approximate the critic or reward function $C_{\pi_{k+\frac{1}{2}}}$. That can be expressed as training a discriminator function $B : \mathcal{S} \times \mathcal{A} \to [0, 1]$ as

$$\max_B \mathbb{E}_{(s,a)\sim\rho^{\pi_b}}[\log B(s, a)] + \mathbb{E}_{(s,a)\sim\rho^{\pi}}[\log((1 - B(s, a))], \tag{11}$$

where $\rho^\pi(s, a) = d^\pi(s)\pi(s, a)$.

2. Algorithm

Since LOGO is able to outperform many of the exisitng algorithms (see Figure 1), our goal for this project is to further improve upon the performance of LOGO using two well-known techniques in reinforcement learning. We have replaced both of the TRPO steps with a modified version of PPO-Clip (Algorithm 3). The new LOGO algorithm is described in Algorithm 2. The modified version of PPO-Clip would early stop if it's too far from the Trust Regions, which bounds this new region above by a factor of λ than LOGO's TRPO step. Clipped Double Q-Learning as the value function was used in the PPO step to reduce the overestimation.

3. Results and Discussion

Since PPO can be used directly as a replacement for TRPO, we had directed implemented our algorithms on top of the LOGO codebase. This allows us to replicate all of the testing cases in the LOGO benchmark environment. A sparse reward environment was created by reducing the amount of reward feedback in the simulation. In Hopper, HalfCheetah and Walker2d, the environment only providing reward of +1 each time after the agent has moved forward by 2, 20, and 2 units from its initial position, respectively. In InvertedDoublePendulum, the reward is only given at the end of the episode. We also experimented with the two modification individually, creating Clipped PPO with regular value function update and TRPO with CDQ value function update.

The modified algorithm is able to have some improvement over the original LOGO algorithm as shown in Figure. 2. However, it's under-performing in HalfCheetah environment. This is most likely due to CDQ doesn't perform as well in sparse environment. This can be seen across the four benchmarks. The sparse rewards should not allow overestimation to occur so choosing the minimum operation in

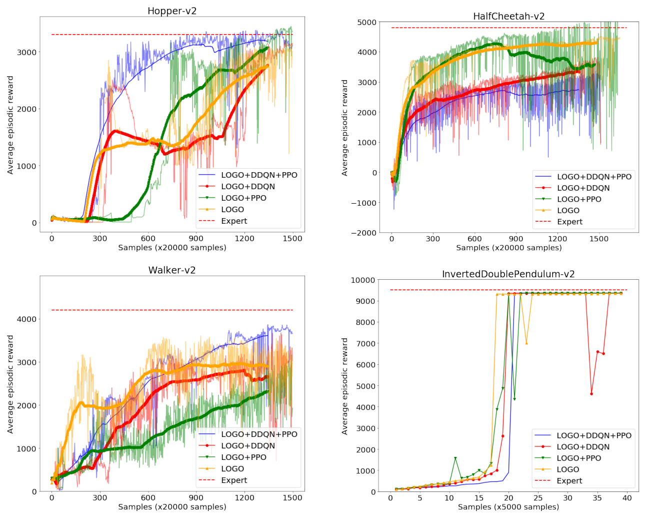

Figure 2. Evaluation of modified-LOGO on four sparse reward MuJoCo environments with complete offline data. The faded background representing the original data points and the solid lines are smoothed out trend-lines. LOGO+PPO+DDQN is the modified-LOGO algorithm described in Algorithm 2. LOGO+PPO is the modeified-LOGO using a regular value function update in the PPO step. LOGO+DDQN is using a CDQ as its value function.

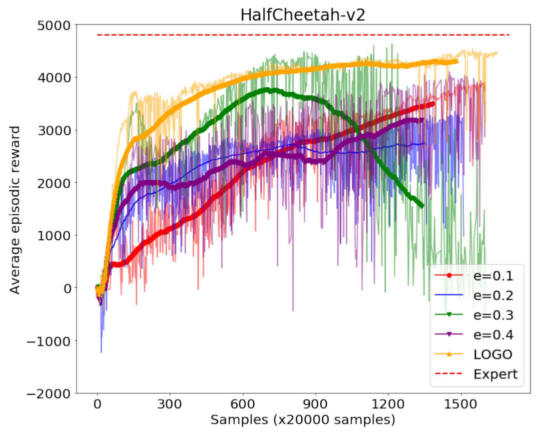

Figure 3. Evaluation of modified-LOGO with different clip epsilon on the HalfCheetah environment.

Algorithm 3 Early Stop PPO-Clip (Adapted from (Schulman et al., 2017))

```
1: Input: initial policy parameters $\theta_0$, initial value function parameters $\rho_0$
2: repeat
3:    Collect $(s, a, r, s') \sim \pi_k$ and store in $\mathcal{D}_{\pi_k}$
4:    Compute rewards-to-go $\hat{R}_t$
5:    Compute advantage estimates $\hat{A}_t$ based on the current value function $V_{\phi_k}$, this value function is defined by (4) using Clipped Double Q-Learning.
6:    Update the policy by maximizing the PPO-Clip objective:
      $$\pi_{k+1} = \arg\max_{\pi} \Bigg( \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in D_k} \mathbb{E}_{s \sim d^{\pi_k}, a \sim \pi(s,\cdot)} [\min(A_R^{\pi_k}(s,a), g(\epsilon, A_R^{\pi_k}(s,a)))] \Bigg)$$
7:    Fit the value function by regression on mean-square error (We have two value functions because of Clipped Double Q-Learning):
      $\phi^1_{k+1} = \arg\min_{\phi} \mathbb{E}_{s \sim d^{\pi_k}, a \sim \pi(s,\cdot)} [(V_\phi(s_t) - \hat{R}_t)^2]$
      $\phi^2_{k+1} = \arg\min_{\phi} \mathbb{E}_{s \sim d^{\pi_k}, a \sim \pi(s,\cdot)} [(V_\phi(s_t) - \hat{R}_t)^2]$
8: until $D_{\text{KL}}^{\pi_k}(\pi, \pi_k) \leq \lambda\delta$ and $k < iter$
```

CDQ would only hinder the performance of the algorithm.

We also attempted to optimize the performance by tuning the clipping epsilon in the PPO step. The performance of the model was increasing as we decreased epsilon, which is effectively making $\frac{\pi_{k+1}}{\pi_k} = 1$. That would mean the model is increasing if PPO is removed for the case of HalfCheetah. We do not have a good idea on why this is the case because PPO was able to help to a more optimal solution in the other three cases. There's one plausible direction with regarding to the decaying KL bound δ_k we are looking to explore. PPO-clip relies on a constant KL bound to perform optimization in original algorithm. For a changing KL bound, a more suitable algorithm would be PPO-Penalize (Algorithm 4). This would automatically adjust the loss function using a penalizing KL term.

4. Future and Conclusion

In this paper, we have empirically shown reinforcement optimization algorithms are effective in sparse reward settings. However, even with these improvements, the computational cost in training doesn't justify this process. Another direction to optimize such algorithm is reducing the computational cost while remaining a similar level of performance.

Algorithm 4 PPO-Penalize (Adapted from (Heess et al., 2017))

```
1: for  $i \in \{1,.. ,N\}$ do
2:     Run policy $\pi_\theta$ for $T$ timesteps, collecting $\{s_t, a_t, r_t\}$
3:     Estimate advantages $\widehat{A}_t = \sum_{t'>t} \gamma^{t'-t} r'_t - V_\varnothing(s_t)$
       $\pi_{old} \leftarrow \pi_\theta$
4:     for  $j \in \{1,.. , M\}$ do
5:         $J_{PPO}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \widehat{A}_t - [\pi_old|\pi_\theta]$
6:         Update $\theta$ by a gradient method w.r.t $J_{PPO}(\theta)$
7:     end for
8:     for $j \in \{1,.. , B\}$ do
9:         $L_{BL}(\varnothing) = -\sum_{t=1}^{T}(\sum_{t'>t} \gamma^{t'-t} r'_t - V_\varnothing(s_t))^2$
10:        Update $\varnothing$ by a gradient method w.r.t $L_{BL}(\varnothing)$
11:    end for
12:    if $KL[\pi_{old}|\pi_\theta] > \beta_{high} KL_{target}$ then
13:        $\lambda \leftarrow \alpha\lambda$
14:    else if $KL[\pi_{old}|\pi_\theta] > \beta_{low} KL_{target}$ then
15:        $\lambda \leftarrow \lambda/\alpha$
16:    end if
17: end for
```

Many of the computational cost comes from finding viable trajectories and updating the policy. We can borrow ideas from Bayesian Optimization (BO) to search better trajectories and therefore reducing the amount of feasible trajectories to update the policy (Müller et al., 2021). In recent papers, multiple techniques and regularization has shown to improve Bayesian Optimization under sparse reward environments (Liu et al., 2022). However, BO often suffers in high dimensional cases. This is often the case with reinforcement learning where the action space is massive. A solution would be Monte Carlo Tree Search, which has shown to work well in high dimensional Neural Architecture Search in combination with BO (Wang et al., 2020).

References

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods, 2018. URL https://arxiv.org/abs/1802.09477.

Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S. M. A., Riedmiller, M., and Silver, D. Emergence of locomotion behaviours in rich environments, 2017.

Li, Y. Deep reinforcement learning: An overview, 2017.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning, 2015.

Liu, S., Feng, Q., Eriksson, D., Letham, B., and Bakshy, E. Sparse bayesian optimization, 2022.

Müller, S., von Rohr, A., and Trimpe, S. Local policy search with bayesian optimization. In *Advances in Neural Information Processing Systems*, 2021.

Rengarajan, D., Vaidya, G., Sarvesh, A., Kalathil, D., and Shakkottai, S. Reinforcement learning with sparse rewards using guidance from offline demonstration, 2022.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. Trust region policy optimization, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning, 2015.

Wang, L., Fonseca, R., and Tian, Y. Learning search space partition for black-box optimization using monte carlo tree search, 2020.