Reinforcement Learning for Robotic Locomotions

Bo Liu
Stanford University
121 Campus Drive
Stanford, CA 94305, USA
bliuxix@stanford.edu

Huanzhong Xu
Stanford University
121 Campus Drive
Stanford, CA 94305, USA
xuhuanvc@stanford.edu

Songze Li
Stanford University
121 Campus Drive
Stanford, CA 94305, USA
songzeli@stanford.edu

Abstract

In Reinforcement Learning, it is usually more convenient to optimize in the policy $\pi(a|s)$ space than forming a policy indirectly by accurately evaluating state-action function $Q(s, a)$ or value function $V(s)$. Because the value function doesn't prescribe actions and the state-action function might be very hard to estimate in continuous or large action space. Therefore, in an environment that suffers from a high cost of evaluation, such as robotic locomotion systems, policy optimization method is a better option due to the significantly smaller policy space. In this project, we investigate a particular policy optimization method, the Trust Region Policy Optimization algorithm, explore possible variants of it, and propose a new method based on our understanding of this algorithm.

1 Introduction

The original motivation of this project comes from the desire to understand human motor control unit in the field of bioengineering. Controlling simulated human body to achieve complex motion using an artificial brain will provide not only better understanding of how human motor system works but also theoretical foundations for surgeries for people having physical disabilities. Therefore, we focus on finding efficient Reinforcement Learning algorithms in environment with high cost evaluation. We eventually decided on exploring policy gradient method because searching in policy space can be much more efficient. In particular, we investigate into a recent work called Trust Region Policy Optimization (Schulman et al., 2015a), in which the author formulates the reinforcement learning objective as an optimization problem subject to a "trust region" constraint. This algorithm works well in practice and has theoretical guarantee to improve in each episode if the objective and constraint are evaluated exactly. In the original paper, the trust region corresponds to the region that is close to the old policy in the policy space. The author defines closeness in terms of the KullbackLeibler (KL) divergence between the old and new policy distributions, e.g. $D_{KL}(\pi_{old}(\cdot|s) \parallel \pi_{new}(\cdot|s))$. In fact, during optimization step, it uses a second order approximation of this KL constraint which involves the Hessian of the KL divergence. Although using conjugate gradient method, as suggested by the author, allows us to avoid computing the Hessian exactly, in general this is still computationally expensive. Therefore, we question to what extent can KL constraint outperforms other easy-to-compute constraints, or even no constraint at all, to compensate for its cost. Based on our understanding of this model, we also view this problem from another perspective and propose a new method that estimates the advantage value using neural net and updates policy directly.

2 Related Works

There are mainly two approaches for the proposed problem in reinforcement learning: policy gradient methods and Q-function methods.

A policy gradient method directly optimizes a parametrized control policy by gradient descent. It belongs to the class of policy search techniques

that maximizes the expected return of a policy in a fixed policy class, while traditional value function approximation approaches derive policies from a value function. Policy gradient methods allow the straightforward incorporation of domain knowledge in the policy parametrization and require significantly fewer parameters to represent the optimal policy than the corresponding value function. They are guaranteed to converge to a policy locally optimal at least. Furthermore, they can handle continuous states and actions, even including imperfect state information. Except for the vanilla policy gradient method, there exist variants like natural policy gradient (Kakade, 2002) and algorithms that use trust regions (Schulman et al., 2015a).

Instead of parameterizing the policy, the Q-function focuses on the state-action function $Q_\pi(s_t, a_t)$ and updates it with Bellman Equation. The optimal Q is obtained via value iteration, in which we repeatedly apply a Bellman operator until it converges. It has been shown that under some mild assumptions, for any initial Q, this value iteration algorithm is guaranteed to converge to Q_{π^*}, where π^* is the optimal policy (Baird and others, 1995). Variants of this basic value iteration algorithm include the neural fitted Q-iteration parameterizes Q-function with a neural network and replaces the Bellman operator with minimizing MSE between two Q-functions. Q-function methods do not work as generally as policy gradient methods although they are more sample efficient when they do work.

Algorithm	Simple & Scalable	Data Efficient
Vanilla Policy Gradient	Good	Bad
Natural Policy Gradient	Bad	OK
Q-learning	Good	OK

Table 1: comparison of different algorithms

3 Notation

To make further derivation and description clear, we provide our notation for classic Markov Decision Process (MDP) and review the conventional reinforcement learning objective.

MDP is a 6-tuple of $(S, A, T, r, \rho, \gamma)$, where S is the set of possible states, A is the set of possible actions, $T : S \times A \times S \to R$ is the transition probability, $r : S \to R$ is the reward function, $\rho : S \to R$ is the initial distribution of state s_0 and γ is the discount factor.

Let π denote a stochastic policy $\pi : S \times A \to [0, 1]$. The objective of classic RL is to maximize the expected future rewards:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots}\Big[\sum_{t=0}^{\infty} \gamma^t r(s_t)\Big]$$

where $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi(a_t|s_t)$, $s_{t+1} \sim T(s_{t+1}|s_t, a_t)$. Let $Q_\pi(s, a)$ and $V_\pi(s)$ denote the standard state-action function and value function, where

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\Big[\sum_{l=t}^{\infty} \gamma^t r(s_l)\Big]$$

$$V_\pi(s_t) = \mathbb{E}_{a_t}\, Q_\pi(s_t, a_t)$$

Define the advantage as:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

In addition, we define $\rho_\pi : S \to R$ as the discounted visitation frequencies function:

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \ldots$$

4 Method

4.1 Policy Gradient Method

Policy gradient method, as the name suggests, tries to estimate the derivative of objective with respect to policy parameter directly. The most commonly used gradient estimator has the form

$$\hat{g} = \mathbb{E}_t[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\hat{A}_t]$$

The TRPO algorithm is a specific derivative under this class of algorithms.

4.2 Trust Region Policy Optimization

Trust region policy optimization (TRPO) method combines the idea of Minorize-Maximization and policy gradient method. It defines a surrogate function which is easier to optimize and provides a strict lower bound for the original objective function. But

in order to use this surrogate, the optimization has to be done in space near the previous policy.

Let π_0 denote the baseline policy and π denote any policy. The following identity expresses the expected return of π (Kakade and Langford, 2002):

$$\begin{aligned}\eta(\pi) &= \eta(\pi_0) + \mathbb{E}_{s_0,a_0,\cdots\sim\pi}[\sum_{t=0}^{\infty} A_{\pi_0}(s_t,a_t)] \\ &= \eta(\pi_0) + \sum_{t=0}^{\infty}\sum_{s} P(s_t=s|\pi)\sum_{a}\pi(a|s)\gamma^t A_{\pi_0}(s,a) \\ &= \eta(\pi_0) + \sum_{s}\sum_{t=0}^{\infty}\gamma^t P(s_t=s|\pi)\sum_{a}\pi(a|s)A_{\pi_0}(s,a) \\ &= \eta(\pi_0) + \sum_{s}\rho_{\pi}(s)\sum_{a}\pi(a|s)A_{\pi_0}(s,a)\end{aligned}$$

By substituting ρ_π with ρ_{π_0}, we get an approximation of the discounted future reward under the new policy π. The new objective is:

$$L(\pi) = \eta(\pi_0) + \sum_{s}\rho_{\pi_0}(s)\sum_{a}\pi(a|s)A_{\pi_0}(s,a) \quad (1)$$

It has been pointed out that this approximation matches the original objective $\eta(\pi)$ to first order (Schulman et al., 2015a). Therefore, a small enough step $\pi_{\theta_{old}} \rightarrow \pi$ that improves $L(\pi_{\theta_{old}})$ also improves $\eta(\pi)$. But the problem is what "small" means here. The author suggests using the KL divergence measure and provides a rigorous theoretical proof. For simplicity, we will not include full derivation here but only the eventual surrogate objective:

$$\max_{\theta} L_{\theta_{old}}(\theta) - CD_{KL}(\theta_{old},\theta)$$

where C is some properly chosen constant.

However, in practice, if we used the penalty coefficient C recommended by the theory, the step size would be too small. To find faster and more robust optimization, a trust region is introduced and the objective becomes maximizing L subject to D_{KL} inside the trust region:

$$\max_{\theta} L_{\theta_{old}}(\theta)$$

$$\text{subject to } \; D_{KL}(\theta_{old},\theta) \leq \delta$$

We can further estimate the objective using importance sampling based on the π_{old} distribution. The constraint can also be estimated in the same way using Monte-Carlo sampling and second order approximation.

4.3 Model for TRPO

For TRPO and its variants, we use neural networks for both policy and value model. For both networks, the input is the state observation $\vec{s}$. The output of policy model is a multi-variate normal distribution for action parameterized by its mean and standard deviation. The output of value model is a single number that is the estimation of the value at this particular state observation.

4.4 Substitution for KL Divergence

The basic idea of TRPO is to optimize the surrogate loss function under the the restriction that the new policy is close enough to the old one. However, one may wonder why we have to use KL divergence here to measure the closeness between the two policies. A natural substitution for the KL divergence is the mean-square error (MSE) $\|\theta_{old} - \theta\|_2^2$. Replacing KL constraint with the MSE will significantly speed up the training procedure since MSE is easier to estimate. Moreover, we recognize that using MSE corresponds to the original policy gradient method because when the MSE is sufficiently small, the direction that maximizes the objective corresponds to its gradient. We experiment TRPO with both KL and MSE constraint. For a baseline, we also implement the method that purely optimizes the objective without any constraint. We will further analyze the comparison in following sections with figures.

4.5 Neural Network Advantage Estimation

Back to equation (1)

$$L(\pi) = \eta(\pi_0) + \sum_{s}\rho_{\pi_0}(s)\sum_{a}\pi(a|s)A_{\pi_0}(s,a)$$

Since $\eta(\pi_0)$ is a constant, maximizing $L(\pi)$ is equivalent to making the $L(\pi) - \eta(\pi_0)$ as positive as possible, for example:

$$\sum_{s}\rho_{\pi_0}(s)\sum_{a}\pi(a|s)A_{\pi_0}(s,a) > 0$$

The intuition here is that the advantage value $A_{\pi_0}(s,a)$ is an indication of how good a certain

action is and we would like to increase the probabilities for better actions that have large positive advantage values. In other words, if advantage values can be accurately estimated, we can optimize the problem by directly maximizing the probabilities of good sampled actions.

In practice, our actions are sampled from a multi-variate Gaussian in a continuous space. We estimate advantage of sampled actions directly from the Monte-Carlo sampling trajectory using the generalized advantage estimation (GAE) (Schulman et al., 2015b), but we also want an estimation of the advantage value as if mean action were taken at each time step. Then by calculating the difference, whenever mean action advantage is smaller (in practice, we require the difference to be larger than a threshold to remove some noise), we mask out the probability of the sampled action and maximize them. Hence, the only problem is how to estimate advantage value for mean actions, the actions that we didn't take during roll-out. We solve this problem by using a 3-layer feed-forward neural network. The input of the network is the concatenation of state observation s and action a; the output of the network is the estimated advantage value.

During implementation, we use TRPO for the first few episodes until the MSE between calculated and estimated advantages is smaller than a threshold. The above method starts after this.

5 Experiments

We test the different methods in the MuJoCo simulator, a physics engine from openAI gym. The 3 environments we use have increasing complexity:

- Swimmer: 10-dimensional state space, linear reward
- Hopper: 12-dimensional state space, same reward as Swimmer, with a positive bonus for being in non-terminal state
- Walker: 18-dimensional state space, same reward as Hopper with an added penalty for strong impacts of the feet against the ground.

6 Results and Analysis

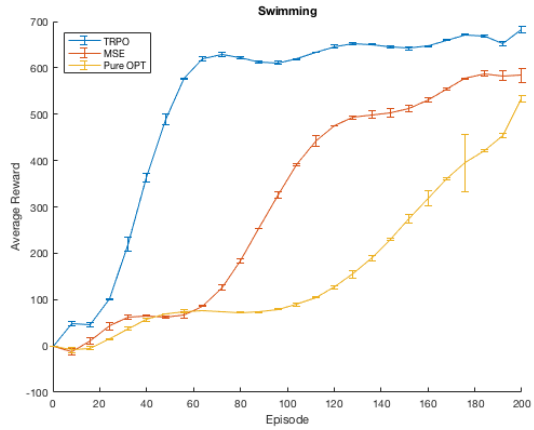

Figure 1: Swimmer: Average reward v.s. Episode

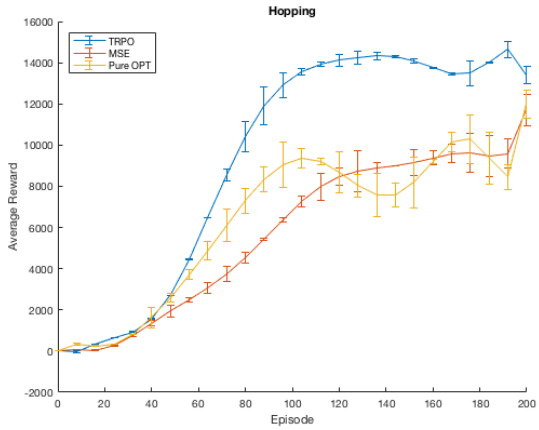

Figure 2: Hopper: Average reward v.s. Episode

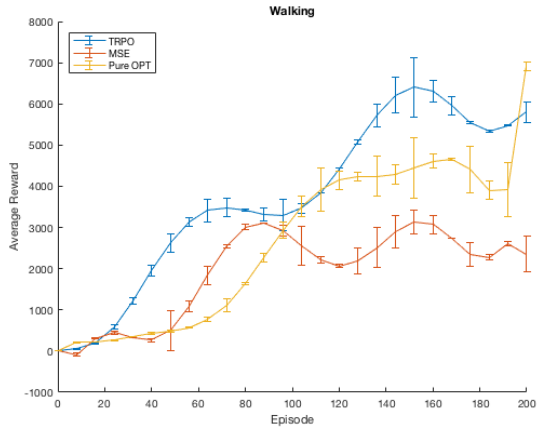

Figure 3: Walker: Average reward v.s. Episode

As shown above, KL divergence does outperform its variants. In spite of the high cost, KL constraint is better when measuring distance between policies. Because in MSE case, though intuitively closer parameters correspond to similar policies, in larger policy space even small update in parameter will result in much different policies. In contrast, the KL constraint is directly estimated upon different $\pi(\theta)$ and roll-outs from different policies are guaranteed to be similar. However, MSE can significantly shorten the training time. So our suggestion is a combination: apply TRPO with MSE constraint to let the model quickly climb to a certain point and then use original TRPO later for faster convergence.

Our advantage estimation method did not work at first. We spent a lot of time debugging. One thing we noticed is that the algorithm tended to have large MSE loss for advantage estimation after some consecutive epochs of improvement. In other words, this provides evidence that our model works when estimation is accurate. In addition, we conjecture that the unexpected drop might result from the over-fitting of our advantage neural net. So whenever it sees a new observation and action that differ from previous experience, it gives wrong estimation of advantage values and the policy will update along the wrong gradient direction.

Therefore, by carefully tuning L2-regularization constant and dropout for the neural network, the model indeed improves, as shown in the figure below.

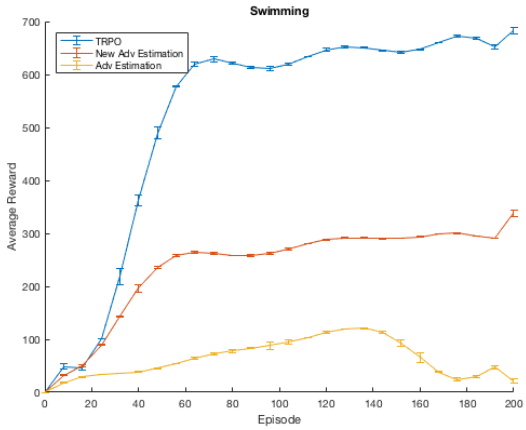

Figure 4: TRPO and advantage estimation

In addition to the analysis above, we also notice an interesting fact about TRPO: despite the fact that the average reward increases in each episode, the algorithm is highly sensitive to initialization. As even using different random seeds, the model's performance varies largely.

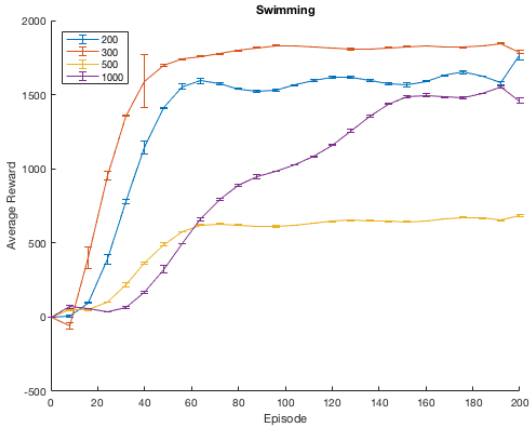

Figure 5: TRPO with different random seeds

7 Conclusion and Future work

In this project, we explore the cutting-edge TRPO algorithm in the class of policy gradient method and try out some possible variants of it. We also experiment our own modification using advantage estimation neural network. We may continue our investigation into other possible substitutions for KL divergence. In addition, we would also like to do more error analysis on why even the TRPO model is not robust enough and find methods to lower the variance.

8 Contributions

- Bo Liu: responsible for coding of TRPO, its variants and advantage estimation method; participated in writeup of project proposal, milestone, poster and final report; presented poster.

- Huanzhong Xu: responsible for analyzing cost of Conjugate Gradient in KL, and estimating feasibility of its variants; helped debugging; collected data and ploted all figures; participated in milestone, poster and final report writeup.

- Songze Li: responsible for project idea and running experiments; participated in milestone and final report writeup.

References

Leemon Baird et al. 1995. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the twelfth international conference on machine learning*, pages 30–37.

Sham Kakade and John Langford. 2002. Approximately optimal approximate reinforcement learning.

Sham M Kakade. 2002. A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015a. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015b. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.