Reinforcement Learning based on Policy Gradient

Ângelo Gregório Lovatto

Supervised by Leliane Nunes de Barros

Departamento de Ciência da Computação - IME-USP

Introduction

In this work we analyze and compare *policy gradient methods*, which offer novel solutions to the reinforcement learning problem. In this setting, an agent interacts with an environment at each stage by performing an action that can change the current state of the environment, producing a reward signal and taking the environment to a next state. Environments are formally modeled as Markov Decision Processes (MDPs).

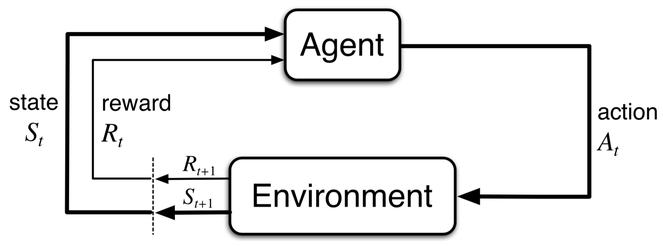

Figure 1: The agent-environment interface in an MDP. Source: Sutton & Barto [4]

We consider episodic control tasks were the agent's objective is to learn a stochastic policy $\pi(a \mid s)$ that maximizes the expected accumulated reward signal throughout successive interactions with the environment:

$$J(\pi) = \mathbb{E}_{\substack{s_{0:\infty} \\ a_{0:\infty}}} \left[\sum_{t=0}^{\infty} r(s_t, a_t) \right], \text{ where } a_t \sim \pi(\cdot \mid s_t),\ s_{t+1} \sim P(\cdot \mid s_t, a_t).$$

The Policy Gradient Approach

Policy gradient methods allows one to reduce the reinforcement learning problem to that of stochastic optimization by considering smoothly parameterized stochastic policies: $\{\pi_{\boldsymbol{\theta}};\ \boldsymbol{\theta} \in \mathbb{R}^d\}$. By directly parameterizing the policy, these methods can handle large, continuous state and action spaces in combination with deep learning models.

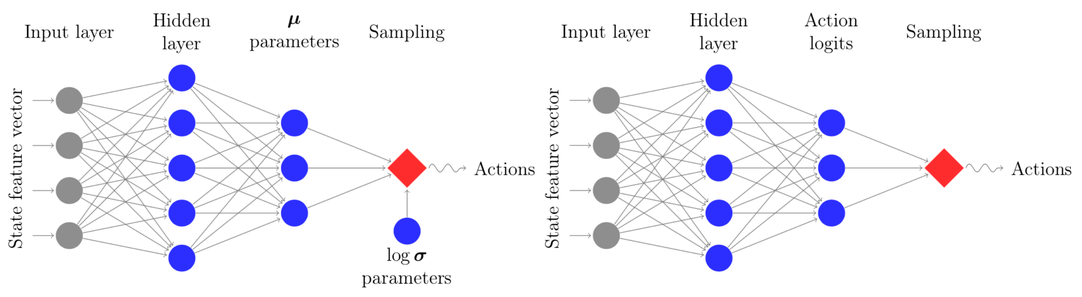

Figure 2: Policy parameterization using multilayer perceptrons for continuous and discrete action spaces, from left to right, respectively; Sampling nodes consist of parameterized multivariate normal and categorical distributions.

The *policy gradient theorem* provides the basis for these methods: an expression for the gradient of performance that can be sampled from experiences:

$$\boldsymbol{g} = \mathbb{E}_{\substack{s_{0:\infty} \\ a_{0:\infty}}} \left[\sum_{t=0}^{\infty} A^{\pi}(s_t, a_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t \mid s_t) \right],$$

where the advantage function is the difference between the action and state values induced by the policy and MDP: $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$. This allows them to be combined with advanced stochastic optimization methods such as the Adam algorithm to find approximately optimal policies. This simple use of the gradient $\boldsymbol{g}$ is usually called the *Vanilla Policy Gradient* method.

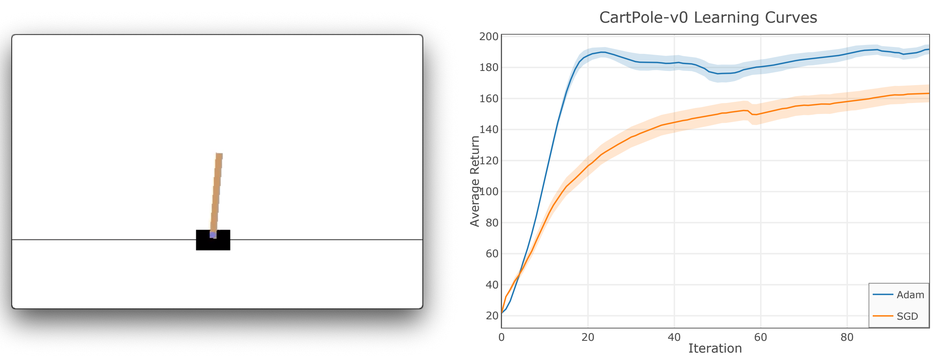

Figure 3: (left) A pole is attached by an un-actuated joint to a cart, which can only move left or right. Reward is given as long as the pole stays balanced. (right) The learning curves obtained using stochastic gradient descent and the momentum-based Adam algorithm.

Reinforcement Learning and Information Geometry

A policy $\pi_{\boldsymbol{\theta}}$ and an MDP together induce a probability distribution over *trajectories*, i.e. given a trajectory $\tau = (s_0, a_0, s_1, \ldots, s_{n-1}, a_{n-1}, s_n)$, the probability distribution over τ given $\pi_{\boldsymbol{\theta}}$ is:

$$Pr(\tau \mid \pi_{\boldsymbol{\theta}}) = P(s_0) \prod_{t=0}^{n-1} P(s_{t+1} \mid s_t, a_t) \pi_{\boldsymbol{\theta}}(a_t \mid s_t).$$

The set of all such distributions defines a statistical manifold, where each choice of policy parameters $\boldsymbol{\theta}$ correspond to a probability distribution over trajectories. In this setting, the Fisher information matrix gives a local approximation of the Kullback-Leiber (KL) divergence between trajectory distributions:

$$D_{\mathrm{KL}}\left(Pr(\cdot \mid \pi_{\boldsymbol{\theta}_{\mathrm{old}}}) \parallel Pr(\cdot \mid \pi_{\boldsymbol{\theta}})\right) \approx (\boldsymbol{\theta} - \boldsymbol{\theta}_{old})^{\mathsf{T}} F(\boldsymbol{\theta}_{\mathrm{old}})(\boldsymbol{\theta} - \boldsymbol{\theta}_{\mathrm{old}}),$$
$$F(\boldsymbol{\theta}_{\mathrm{old}}) = \mathbb{E}_{s \sim \pi_{\boldsymbol{\theta}_{\mathrm{old}}}} \left[\nabla^2_{\boldsymbol{\theta}} D_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}_{\mathrm{old}}}(\cdot \mid s) \parallel \pi_{\boldsymbol{\theta}}(\cdot \mid s)\right) \right].$$

The *natural policy gradient* is the steepest ascent direction of the performance measure J when distances are measured by KL divergence on the trajectory distribution manifold:

$$\tilde{\boldsymbol{g}} = F(\boldsymbol{\theta})^{-1} \boldsymbol{g}.$$

This gives the same direction in the space of policies, regardless of the parameterization used to define them! Most notably, using this direction instead of the vanilla gradient can substantially increase performance in tasks where positive rewards are few and far between.

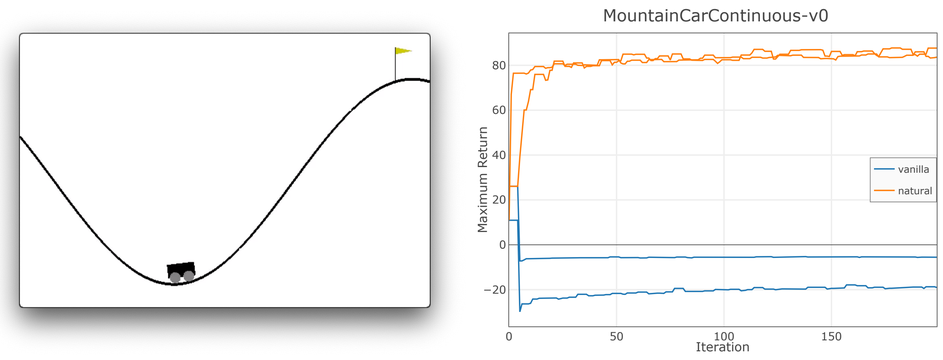

Figure 4: (left) A car starts in a valley and reward is given if it reaches the flag on the right hill. The agent must learn to build momentum, given that its engine is too weak. (right) A comparison between the Vanilla and Natural Policy Gradient methods; both methods start with a policy which manages to reach the top occasionally, however only the Natural Policy Gradient method manages to learn and replicate the behaviour.

Reducing Learning to Numerical Optimization

Trust Region Policy Optimization (TRPO) [3] treats the learning problem at every iteration as the following constrained optimization problem:

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & L_{\pi_{\boldsymbol{\theta}_{\mathrm{old}}}}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{s,a \sim \pi_{\boldsymbol{\theta}_{\mathrm{old}}}} \left[\frac{\pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_{\boldsymbol{\theta}_{\mathrm{old}}}(a \mid s)} A^{\pi_{\boldsymbol{\theta}_{\mathrm{old}}}}(s,a) \right] \\ \text{s.t.} \quad & \mathbb{E}_{s \sim \pi_{\boldsymbol{\theta}_{\mathrm{old}}}} \left[D_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}_{\mathrm{old}}}(\cdot \mid s) \parallel \pi_{\boldsymbol{\theta}}(\cdot \mid s)\right) \right] \leq \delta \end{aligned}$$

The intuition behind this formulation is that, if policies are sufficiently close to one another, we can pessimistically predict the performance of a new policy $\pi_{\boldsymbol{\theta}}$ with data sampled from the current one, $\pi_{\boldsymbol{\theta}_{\mathrm{old}}}$, using a surrogate objective function. TRPO solves the problem above using a linear approximation to the objective and a quadratic approximation to the constraint:

$$L_{\pi_{\boldsymbol{\theta}_{\mathrm{old}}}}(\pi_{\boldsymbol{\theta}}) \approx \boldsymbol{g}^{\mathsf{T}}(\boldsymbol{\theta} - \boldsymbol{\theta}_{\mathrm{old}}),$$
$$\mathbb{E}_{s \sim \pi_{\boldsymbol{\theta}_{\mathrm{old}}}} \left[D_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}_{\mathrm{old}}}(\cdot \mid s) \parallel \pi_{\boldsymbol{\theta}}(\cdot \mid s)\right) \right] \approx (\boldsymbol{\theta} - \boldsymbol{\theta}_{\mathrm{old}})^{\mathsf{T}} F(\boldsymbol{\theta}_{\mathrm{old}})(\boldsymbol{\theta} - \boldsymbol{\theta}_{\mathrm{old}}).$$

The solution is a natural policy gradient step, scaled to take into account the trust region constraint, keeping successive policies in a reasonable neighborhood. This approach is much more efficient than the previous methods, allowing one to tackle high-dimensional control problems such as navigation in robotics.

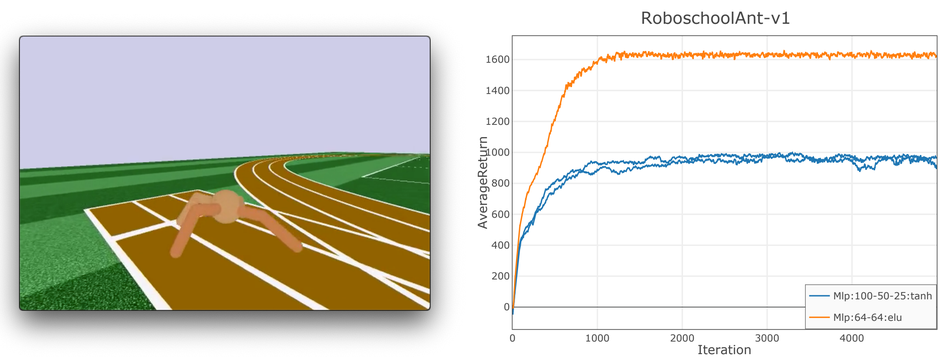

Figure 5: A 3D-quadruped robot, with 29 state dimensions and 9 actuated degrees of freedom. TRPO with deep learning models enables learning of stable walking animations. The policies took about 5 hours to train on a dual core laptop.

Materials and Methods

Policies and algorithms were implemented using the PyTorch deep learning framework [2]. The environments were supplied by the OpenAI Gym toolkit for reinforcement learning research [1].

Funding provided by the CNPq-PIBIC undergraduate research program.

Conclusions

Policy gradient methods offer solutions for reward-related learning problems which enable integration of reinforcement learning theory with deep learning methods and models. Thus, these can be applied to more realistic tasks, when compared with other reinforcement learning solutions. However, there's still a lot of room for improvement in regards to sample efficiency and hyperparameter tuning.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[2] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[3] John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.