https://openai.com/blog/openai-baselines-ppo/

OpenAI

Proximal Policy Optimization

We're releasing a new class of reinforcement learning algorithms, Proximal Policy Optimization (PPO), which perform comparably or better than state-of-the-art approaches while being much simpler to implement and tune. PPO has become the default reinforcement learning algorithm at OpenAI because of its ease of use and good performance.

JULY 20, 2017
3 MINUTE READ

VIEW ON GITHUBVIEW ON ARXIV PPO lets us train AI policies in challenging environments, like the Roboschool one shown above where an agent tries to reach a target (the pink sphere), learning to walk, run, turn, use its momentum to recover from minor hits, and how to stand up from the ground when it is knocked over.

Policy gradient methods are fundamental to recent breakthroughs in using deep neural networks for control, from video games, to 3D locomotion, to Go. But getting good results via policy gradient methods is challenging because they are sensitive to the choice of stepsize — too small, and progress is hopelessly slow; too large and the signal is overwhelmed by the noise, or one might see catastrophic drops in performance. They also often have very poor sample efficiency, taking millions (or billions) of timesteps to learn simple tasks.

Researchers have sought to eliminate these flaws with approaches like TRPO and ACER, by constraining or otherwise optimizing the size of a policy update. These methods have their own trade-offs — ACER is far more complicated than PPO, requiring the addition of code for off-policy corrections and a replay buffer, while only doing marginally better than PPO on the Atari benchmark; TRPO — though useful for continuous control tasks — isn't easily compatible with algorithms that share parameters between a policy and value function or auxiliary losses, like those used to solve problems in Atari and other domains where the visual input is significant.

PPO

With supervised learning, we can easily implement the cost function, run gradient descent on it, and be very confident that we'll get excellent results with relatively little hyperparameter tuning. The route to success in reinforcement learning isn't as obvious — the algorithms have many moving parts that are hard to debug, and they require substantial effort in tuning in order to get good results. PPO strikes a balance between ease of implementation, sample complexity, and ease of tuning, trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small.

We've previously detailed a variant of PPO that uses an adaptive KL penalty to control the change of the policy at each iteration. The new variant uses a novel objective function not typically found in other algorithms:

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

- \theta$\theta$ is the policy parameter
- \hat{E}_t$\hat{E}_t$ denotes the empirical expectation over timesteps
- r_{t}r_t is the ratio of the probability under the new and old policies, respectively
- \hat{A}_t$\hat{A}_t$is the estimated advantage at time tt
- \varepsilon$\varepsilon$ is a hyperparameter, usually 0.1 or 0.2

This objective implements a way to do a Trust Region update which is compatible with Stochastic Gradient Descent, and simplifies the algorithm by removing the KL penalty and need to make adaptive updates. In tests, this algorithm has displayed the best performance on continuous control tasks and almost matches ACER's performance on Atari, despite being far simpler to implement.

Controllable, Complicated Robots

Agents trained with PPO develop flexible movement policies that let them improvise turns and tilts as they head towards a target location.

We've created interactive agents based on policies trained by PPO — we can use the keyboard to set new target positions for a robot in an environment within Roboschool; though the input sequences are different from what the agent was trained on, it manages to generalize.

We've also used PPO to teach complicated, simulated robots to walk, like the 'Atlas' model from Boston Dynamics shown below; the model has 30 distinct joints, versus 17 for the bipedal robot. Other researchers have used PPO to train simulated robots to perform impressive feats of parkour while running over obstacles.

Baselines: PPO, PPO2, ACER, and TRPO

This release of baselines includes scalable, parallel implementations of PPO and TRPO which both use MPI for data passing. Both use Python3 and TensorFlow. We're also adding pre-trained versions of the policies used to train the above robots to the Roboschool agent zoo.

Update: We're also releasing a GPU-enabled implementation of PPO, called PPO2. This runs approximately 3X faster than the current PPO baseline on Atari. In addition, we're releasing an implementation of Actor Critic with Experience Replay (ACER), a sample-efficient policy gradient algorithm. ACER makes use of a replay buffer, enabling it to perform more than one gradient update using each piece of sampled experience, as well as a Q-Function approximate trained with the Retrace algorithm.

We’re looking for people to help build and optimize our reinforcement learning algorithm codebase. If you’re excited about RL, benchmarking, thorough experimentation, and open source, please apply, and mention that you read the baselines PPO post in your application.