Evaluation of Proximal Policy Optimization Algorithms

Shansong (Sam) Huang, Qinyun (Peter) Yu

April 1st, 2022

DS8010: Interactive Learning in Decision Process

Introduction

In machine learning there are three main subcategories of learning: supervised, unsupervised and reinforcement learning. Out of the three categories, reinforcement learning remains as the most applicable to real life situations as its framework can help with modeling reward-based learning. This reward-based learning occurs under a trial-and-error process that directly interacts with its environment, and what makes the framework particularly appealing is the lack of required interaction from its users [Sutton and Barto, (2020)].

To apply reinforcement learning to a problem, a general path is taken. Formulating a problem, creating an environment, reward shaping, creating and training an agent and finally deploying a policy. This utilizes the Markov Decision Process framework while following the Generalized Policy Iteration approach in order to solve the Bellman equation, so that an optimal policy can be computed for [Sutton and Barto, (2020)]. In the 20th century, there have been a number of different approaches, namely Dynamic Programming (DP), Monte Carlo (MC), and Temporal Difference (TD) learning methods, which all have their respective advantages and disadvantages, but ultimately all fall under the category of value function approximation algorithms and are considered tabular methods. The major shortcoming of the previously mentioned methods are simply that they are considered tabular, only work under small datasets and cannot be applied to a real life scenarios [Sutton and Barto, (2020)].

In 2012, reinforcement learning was revitalized when Krizhevsky et al., published a paper on the "*ImageNet Classification with Deep Convolutional Neural Networks*", or better known for its architecture - AlexNet. This paper gave reinforcement learning a new light as suddenly researchers could utilize consumer-grade graphical processing units and use them in tandem with large amounts of data and neural networks coupled with backpropagation. This revitalization was shortly seen with the 2015 paper by DeepMind, titled as "*Human-level control through deep reinforcement learning*", which revolutionized how reinforcement learning was approached, as suddenly neural networks could allow Q-Learning, a traditional approach, to learn to play Atari games through Deep Q-Learning Networks (DQN) [Mnih et al, (2015)] .

Deep Q-Learning Networks were by no means perfect though. While it was very sample efficient, as per Sutton and Barto, DQNs sacrificed a lot to attain its performance and fell under what the book classified as the "*deadly triad*" whereby DQNs would suffer convergence and instability issues with its learning. As a result, much research was poured into studying alternatives to DQNs [Sutton and Barto, (2020)].

One known alternative to DQNs are the policy gradient methods. According to Peters (2010), by parameterizing a policy, one could instead directly learn about a policy rather than approximating through V- or Q-functions and the policy could be updated through gradient ascent. This is exactly what happened in 2015, when a paper titled "Trust Region Policy Optimization" (TRPO) was published where the TRPO method addressed DQN instability, while also not sacrificing out too much on the data efficiency. Essentially, TRPO worked very well having addressed some of the many shortcomings of policy gradient methods and allowed researchers to apply the algorithm to real life scenarios [Schulman et al., (2015)]. Unfortunately, however, the algorithm itself was simply very complicated to understand and very difficult to implement. This led to proximal policy optimization, which was released in the subsequent years. This new algorithm was inherently very similar to the previously published TRPO paper and had the same lead author, however given slight modifications, PPO was simpler, faster, and much more data efficient [Schulman et al., (2017)].

Reinforcement learning is still an ongoing research topic and much research is still being done in the area. Despite proximal policy optimization being one of the newest algorithms published, it is not an algorithm that works with every given scenario, and a number of different state-of-the-art algorithms are still being developed to this day. To study the shortcomings of PPO, we plan on implementing the PPO algorithm and evaluating the performance when benchmarked to a number of different Atari environments, and comparing its results to the paper results.

Literature Review

Reinforcement learning starts with solving a Bellman equation which is denoted as the following:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], for\ all\ s \in S \tag{1}$$

[Sutton and Barto, (2020)]

This occurs while following the Generalized Policy Iteration approach where:

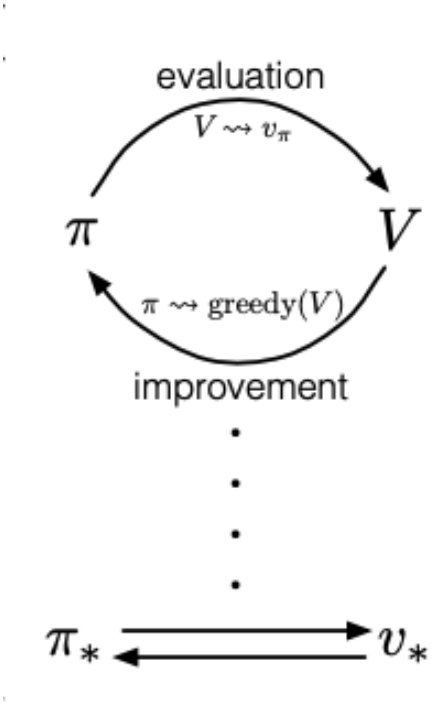

Figure 1: Generalized Policy Iteration[Sutton and Barto, (2020)]

and relies on the Markov Decision Process framework:

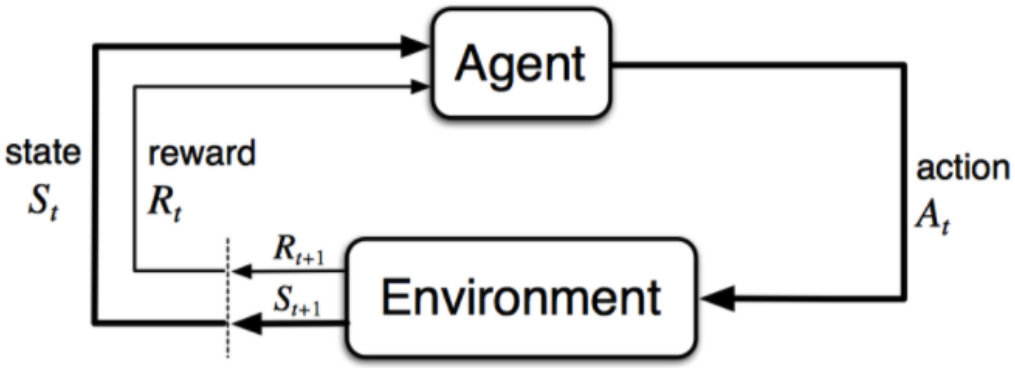

Figure 2: MDP framework[Sutton and Barto, (2020)]

By alternating between policy evaluation and policy improvement framed as a Markov Decision Process, an optimal policy can be obtained through solving the Bellman equation. These three elements are very critical in many reinforcement learning algorithms, and many algorithms approach this problem in different ways. Some of the most common early implementations of reinforcement learning include dynamic programming (DP), Monte Carlo (MC) and Temporal Difference (TD) methods, which all ultimately suffer from the curse of dimensionality, and rely on discrete states in order to allow value functions to be approximated [Sutton and Barto, (2020)].

With the release of AlexNet came new innovations in the field of reinforcement learning. By enhancing Q-Learning, a variant of Temporal Difference Learning, and pairing its optimal-policy learning with deep neural networks, the Bellman equation could update the network's weights. This was very successful as suddenly the tabular limitation of Q-Learning was addressed as now a neural network acted as a function approximator, and states and actions did not have to be stored in a Q-table, which suffered from scalability issues. Human-level control was then attained over a number of different Atari games through these Deep Q-Learning Networks [Mnih et al, (2015)].

According to Sutton and Barto (2020), an algorithm falls within the deadly triad of divergence if the algorithm relies on the following:

- Function approximation
- Bootstrapping
- Off-policy learning

For DQNs, this was fatal as the neural network acted as a function approximator, bootstrapping occurred as Q-Learning is a form of TD-Learning which has updates relying on future Q-values, and because it learned off-policy. This resulted in the method being unstable for certain problem spaces. As a result, researchers then studied this space and found alternatives. One such alternative was through policy gradient approaches.

Policy gradient approaches (explained on page 7) were not new around this time however they simplified reinforcement learning and instead of relying on V- or Q- function approximations, they directly parameterized the policy and used randomly instantiated weights to try to maximize value under such parameterized policy, given a state, whereby gradient ascent

updates were then performed [Peters, (2010)]. While simpler said, vanilla policy gradient approaches still had a number of drawbacks.

When implementing vanilla policy gradient approach, the natural gradient involves a second-order derivative matrix which makes it not suitable for large scale problems. In addition, the computational complexity is rather high. The vanilla policy gradient methods also suffer from poor data efficiency and lack robustness. The standard method to reduce the variance in gradient estimates was to use an advantage function, which estimates how good an action is compared to average action as a baseline when given a specific state. The idea was to reduce the high variance in the gradient estimates between the old policy and the new policy, which thereby would increase the stability of the vanilla policy gradient algorithm.

In 2015, the Trust Region Policy Optimization (TRPO) algorithm was published, which addressed many of the shortcomings of vanilla policy gradient methods. The TRPO algorithm added KL divergence constraints for enabling the trust-region for the optimization process. In the trust region, the search for the local minimum or local maximum is bounded in that region, and the local minimum or local maximum determines the direction of the optimal point. Where as in TRPO, it added KL divergence constraints so that the newly updated policy would not be too far away from the old policy, hereby lowering the policy update deviation [Karunakaran, (2020)]. However, the TRPO algorithm was relatively complicated to implement and understand, and was not compatible with architectures that included noise or parameter sharing. For many, implementing the TRPO algorithm could prove very challenging. Therefore, the Proximal Policy Optimization algorithm was then introduced that maintained the data efficiency and reliable performance of TRPO, while using only first-order optimization[Karunakaran, (2020)].

Notations

- $\hat{E}_t$ denotes the empirical expectation over a finite batch of samples
- π_θ is a stochastic policy parameterized by θ
- $\hat{A}_t$ = Q(s,a) - V(s), $\hat{A}_t$ is an estimator of the advantage function at time step t
- V(s) is the State-value function measures the expected return of state s
- $Q(s, a)$ is the Action-value function. It assesses the expected return of a pair of state and action (s,a)

Policy Optimization

Policy Gradient Methods

Policy gradient methods work by computing an estimator of the policy gradient and plugging it into a stochastic gradient ascent algorithm. The most commonly used gradient estimator with advantage function has the form:

$$\hat{g} = \hat{E}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{2}$$

Trust Region Policy Optimization (TRPO)

In order to avoid parameter updates that change the policy too much at one step to improve training stability, the TRPO introduce a KL divergence constraint on the size of policy update at each iteration.

$$\underset{\theta}{maximize} \quad \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}\hat{A}_t\right] \tag{3}$$

$$subject\ to \quad \hat{E}_t\left[KL[\pi_{\theta old}(\cdot|s_t), \pi_\theta(\cdot|s_t)]\right] \leq \delta \tag{4}$$

KL(Kullback-Leibler) divergence measures the distance between old and new probability distribution. θ_{old} is the vector of policy parameters before the update.

$$D_{KL}(\pi_{old}||\pi) = \int_x \pi_{old}(x) \log \frac{\pi_{old}(x)}{\pi(x)} dx \tag{5}$$

TRPO maximizes the objective function and subjects to the trust region constraint which enforces the distance between old and new policies measured by KL-divergence to be small enough, within a parameter of δ. However, this makes TRPO a complicated algorithm to implement. The KL constraint adds additional overhead in the form of hard constraints to the optimization process.

Proximal Policy optimization (PPO)

Given that TRPO is relatively complicated, here comes the PPO which has a much simpler approach. PPO simplifies TRPO by using a clipped objective function while retaining similar

performance. First, similar to TRPO objective function, it defines the probability ratio between the new policy and old policy as $r(\theta)$ [Schulman et al, (2017)].

$$r(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \tag{6}$$

Then, the objective function of TRPO becomes:

$$L^{CPI}(\theta) = \hat{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \hat{A}_t \right] = \hat{E}_t[r_t(\theta)\hat{A}_t] \tag{7}$$

CPI refers to conservative policy iteration. Without a constraint, maximization of L^{CPI} would lead to an excessively large policy update and big policy ratios. Next, PPO imposes the constraint by forcing $r(\theta)$ to be within a small interval around 1, [1 - ϵ, 1 + ϵ], where ϵ is a hyperparameter.

The main objective PPO proposed:

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t), clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{8}$$

The first term inside the min function is L^{CPI}. The second term clip$(r(\theta), 1-\epsilon, 1+\epsilon)$ clips the ratio $r(\theta)$ to be within $[1-\epsilon, 1+\epsilon]$. By taking the minimum of the clipped and unclipped objective we can have the final objective represent a lower bound on the unclipped objective [Schulman et al, (2017)]. A positive $\hat{A}_t$ indicates the action taken is good, otherwise if $\hat{A}_t$ is negative, the action taken by the agent is bad[Karunakaran, (2020)]. For both cases, with $L^{CLIP}(\theta)$ objective function, the PPO ensures there is no large policy change.

Methods

Using the Python OpenAI Gym environment, we plan on implementing the Proximal Policy Optimization algorithm as follows below in Keras and testing and benchmarking our implementation of PPO on the *CartPole-v1* environment well as *SpaceInvaders-v0* environment. The mean score of the last 100 episodes will then be taken and compared to the scores obtained in the paper.

PPO-Clip Algorithm

Firstly, the PPO algorithm collects a set of trajectories for each epoch by sampling from the latest version of the stochastic policy. Then, the reward and the estimated advantage

function are calculated for updating the policy and fit the value function. The policy is updated through a stochastic gradient ascent optimizer, and the value function is fitted by some gradient descent algorithm [Chrysovergis, (2021)].

Algorithm 1 PPO-Clip Algorithm

Input: initial policy parameter θ_0, initial value function parameters ϕ_0

for `<k = 0,1,2,... >` **do**

Collect set of trajectories $D_k = \tau_i$ by running policy $\pi_k = \pi(\theta_k)$ in the environment

Compute rewards-to-go $\hat{R}_t$

Compute advantage estimates, $\hat{A}_t$ based on the current value function $V_{\phi k}$

Update the policy by maximizing the PPO-Clip objective:

$\theta_{k+1} = \text{argmax}_\theta \frac{1}{|D_k|T} \sum_{r \in D_k} \sum_{t=0}^{T} min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta k} a_t|s_t} A^{\pi\theta_k}(s_t, a_t), g(\epsilon, A^{\pi\theta_k}(s_t, a_t)))$

Fit value function by regression on mean-squared error:

$\phi_{k+1} = \text{argmin}_\phi \frac{1}{|D_k|T} \sum_{r \in D_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2$

end for

References

Chrysovergis, I. (2021, June 24). Keras Documentation: Proximal policy optimization. Keras. Retrieved April 1, 2022, from https://keras.io/examples/rl/ppo$_{c}artpole/$

Karunakaran, D. (2020, December 4). Proximal policy optimization(ppo)- a policy-based reinforcement learning algorithm. Medium. Retrieved April 1, 2022, from https://medium.com/intro-to-artificial-intelligence/proximal-policy-optimization-ppo-a-policy-based-reinforcement-learning-algorithm-3cf126a7562d

Kirzhevsky, A., Sutskever, I., amp; Hinton, G. E. (n.d.). ImageNet classification with deep convolutional ... - neurips. Retrieved April 2, 2022, from https://proceedings.neurips.cc/paper/2012/file/c3
Paper.pdf

Peters, J. (n.d.). Policy gradient methods. Scholarpedia. Retrieved April 1, 2022, from http://www.scholarpedia.org/article/Policy$_{g}radient_{m}ethods$

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., amp; Abbeel, P. (2017, April 20). Trust region policy optimization. arXiv.org. Retrieved April 1, 2022, from https://arxiv.org/abs/1502.05477

Sutton, R. S., Barto, A. G. (2020). Reinforcement learning: An introduction. pg 73-78.The MIT Press.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017, August 28). Proximal policy optimization algorithms. arXiv.org. https://arxiv.org/abs/1707.06347