UNDERSTANDING PHASIC POLICY GRADIENT

A PREPRINT

Ken Ming Lee
km23lee@uwaterloo.ca
University of Waterloo

November 25, 2020

ABSTRACT

Recent advancements in the field of reinforcement learning (RL) sparked the creation of various algorithms that achieved remarkable success in domains like complex video games and robotic control. However, to a novice RL researcher, concepts discussed in state-of-the-art papers may be unfamiliar. This goal of this paper is to provide sufficient background content in a succinct manner for readers to understand the intuition behind the research progression leading up to Phasic Policy Gradient (PPG) [1], and PPG's purpose and contributions to the RL field.

1 Introduction

This paper assumes that the reader is comfortable with fundamental concepts of RL, such as the Markov Decision Process (MDP), and fundamental model-based and model-free algorithms - namely Policy/Value Iteration, Sarsa and Q-Learning. All equations shown below assume a discrete-action space.

2 Vanilla Policy Gradient (REINFORCE)

The basis of policy gradient methods is to maximize the following objective function:

$$J(\theta) = E_{\pi_\theta}\left[log\,\pi_\theta(s,a)\,Q^{\pi_\theta}(s,a)\right]$$

The REINFORCE algorithm [2] optimizes the policy directly, using the discounted cumulative return, $R_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l}$ as the best estimate of $Q^{\pi_\theta}(s,a)$. Relying on R_t as our estimate of $Q^{\pi}(s,a)$ gives us a low bias but high variance estimate.

3 Deep Q-Learning (DQN)

DQN [3] implements Q-learning with a value function approximator and was the first successful algorithm that allowed an agent to learn from visual inputs. DQN estimates $Q^{\pi}(s,a)$ and uses bootstrapping to minimize the loss function, thus eliminating much of the issue with high variance as seen in REINFORCE. More specifically, DQN's objective function is to minimize the following Mean-Squared TD Error:

$$\mathcal{L}(w) = E_{s,a,r,s'}[(r + \gamma\, max_{a'} Q(s',a';w^-) - Q(s,a;w))^2]$$

Furthermore, DQN's contributions to the RL field include:

- Experience replay (ER) buffer - this allows us to decorrelate data and be sample efficient.
- Fixed Q-targets - stability during training is achieved by maintaining two different versions of the network (new and old), where we update our new network (which is also our current network) using the TD target from our old network.

4 Asynchronous Advantage Actor Critic (A3C)

The issues with DQN and other algorithms that use an ER buffer are:

- Higher memory usage and more computation per interaction.
- Algorithms are limited to performing only offline updates.

A3C [4] was one of the first algorithms that successfully implemented actor-critic using deep neural networks. And this was possible due to the combination of the following techniques:

- Shared architecture between actor and critic increases learning efficiency within network's starting layers.
- Update on fixed-length segments of experience (forward-view n-step TD) strikes balance between bias and variance.
- Parallel worker threads, each with separate instances of the environment that update a global network asynchronously. The variety of states seen by different workers help decorrelate data.

Although not necessarily introduced in A3C, it should be noted that instead of using cumulative discounted reward to update the policy gradient (e.g. REINFORCE), A3C uses the advantage function. The advantage function is formulated by subtracting a baseline function from R_t, giving us $\sum_{l=0}^{n-1} \gamma^l r_{t+l} + \gamma^n V(s_{t+l}) - V(s_t)$.

4.1 Synchronous A3C - A2C

The key difference between A3C and A2C [5] is that A2C waits for all workers to complete their segment before updating the main network all at once. The main advantages of doing so are:

- More efficient use of GPU since experiences from individual workers are aggregated into larger batches.
- Ensures that all workers are using the latest parameters of the main network.

5 Generalized Advantage Estimation (GAE)

One of the main issues that arise from using forward view n-step returns (as used in A3C) is that the optimal number of steps per update varies heavily between different environments. GAE [6] introduces parameter λ that allows for much better generalization across different environments compared to having fixed-length segments of experience.

The key contribution of GAE is the parameterized advantage function, $A_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$, where δ_t represents one-step TD error at timestep t. It is important to note that:

- Setting $\lambda = 0$ gives us one-step return - GAE(γ, 0): $A_t = \delta_t$
- Setting $\lambda = 1$ gives us Monte Carlo - GAE(γ, 1): $A_t = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l} = \sum_{l=0}^{\infty} \gamma^l r_{t+l} - V(s_t)$

6 Trust-Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO)

TRPO [7] and PPO [8] are motivated by the same issue, which is how can we take the largest optimization step without changing the policy too much, which may result in a performance collapse.

To provide more context, prior online algorithms that we have seen using the loss function $\mathcal{L}^{pg} = E_{\pi_\theta}\left[log\, \pi_\theta(s,a)\, A_t\right]$ performs gradient ascent optimization once per training sample, which is not sample efficient. As our advantage A_t is a noisy estimate, performing optimization over the same sample multiple times will rapidly drive $\pi_\theta(s,a)$ to 0 or 1 (for discrete action-space), which may result in a worse policy.

For TRPO and PPO, our objective function is instead $J(\theta) = E_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t] = E_t[r_t(\theta) A_t]$, which is differentiably equivalent to $\mathcal{L}^{pg}$. The main difference between TRPO and PPO is that while TRPO uses Kullback-Leibler divergence as a constraint to prevent our new policy from diverging too much from our previous policy, PPO proposes optimizing the following clipped surrogate objective:

$$J(\theta) = E_t\left[min\left(r_t(\theta)\, A_t,\ clip\left(r_t(\theta),\, 1-\epsilon,\, 1+\epsilon\right) A_t\right)\right]$$

The clipping nature of PPO enables it to perform gradient ascent multiple times per training sample, without taking steps that are too large. This significantly increases sampling efficiency.

7 Phasic Policy Gradient (PPG)

Sharing a network between the policy and value function have the clear advantage of allowing features (e.g. high-level representations) learnt by both objectives to complement each other, allowing the network to generalize better. However, competing objectives meant that relative weight must be assigned to updates from the policy and value function. PPG was motivated by the problem that regardless of the hyperparameter, there is still a risk that optimization of one objective will interfere with the other. There was also always an assumption that the optimal amount of sample reuse for both the policy and value function are the same.

PPG proposes a network architecture with separated actor and critic networks, where the policy network (actor) has an additional auxiliary head. The key ideas behind this design are:

- Separate policy and value function networks meant that we can now optimize them separately with different levels of sample reuse without worrying about competing objectives affecting performance. That value function network has also empirically shown that it can tolerate more sample reuse compared to the policy.
- Auxiliary head of the policy allows features learnt from value function to be distilled into the policy, while minimizing distortions to the policy.

8 Conclusion and Future Directions

Summarizing RL algorithms above, we have made major advancements in the following areas:

- Offline RL algorithms can be implemented using deep neural-networks through utilization of techniques such as experience buffers.
- Online RL algorithms can also be implemented using deep neural-networks by running multiple parallel instances simultaneously to decorrelate data. Sampling efficiency is also greatly increased with online algorithms that allow for sample reuse (e.g. PPO and PPG).
- Successful implementation of n-step TD algorithms allow us to strike a better balance between bias and variance. Parameterization of n from algorithms like GAE allows us to generalize better across different environments.

While this progress is significant, below are some of the barriers still faced by state-of-the-art algorithms that hinders it from being applicable to many real-world systems:

- Sampling efficiency is still a large issue, an agent needs too many interaction with environment to properly learn from it. This is often not feasible for real-world environments.
- Environments with sparse reward signals still pose a large challenge to RL agents.
- Generalization to never-seen-before states is still mediocre at best.
- Uncontrolled exploration makes it dangerous to employ online RL algorithms into the real world.

References

[1] Karl Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient, 2020.

[2] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12:1057–1063, 1999.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[4] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[5] Yuhuai Wu. Openai baselines: Acktr & a2c, Jun 2020.

[6] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[7] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.