# Continual Policy Gradient for Real-world Learning

Akshay Raman
ar8692@nyu.edu

## 1 Batch Policy Gradient

Policy gradient methods are a family of algorithms in reinforcement learning that optimize and model the policy directly. It aims to maximise the expected total reward by repeatedly estimating the gradient $g = \mathbb{E}[\sum_{t=0}^{\infty} r_t]$. There are several different related expressions for the policy gradient, which have the form:

$$g = \mathbb{E}[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)]$$

where $\Psi_t$ may be one of the following:

1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t=t'}^{\infty} r_t$: reward following action $a_t$.
3. $\sum_{t=t'}^{\infty} r_t - b(s_t)$: baselined total reward.
4. $Q^\pi(s_t, a_t)$: state-action value function.
5. $Q^\pi(s_t, a_t) - V^\pi(s_t)$: advantage function.
6. $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$: TD residual.

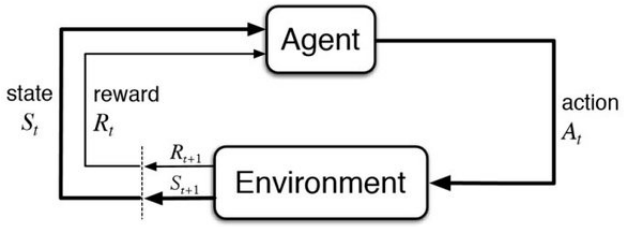
To estimate this gradient, we typically collect batches of trajectories and perform stochastic gradient descent to update the agent. Moreover, for the first three types of advantages, we need to rollout the entire episode and collect rewards to go for each state in the episode. The choice of advantage largely depends on the task in hand since there are benefits and drawbacks of each type.

Using the total reward is often referred as the Monte-carlo approach. The algorithm gives equal advantage to all actions in a rollout. This is most often not desirable since not all actions of the agent carry equal importance. Additionally, the monte-carlo method is has extremely high variance. All the other methods use some form of a critic network to estimate the value of a state or state-action pair. These methods lower the variance considerably, but at the cost of high bias.

## 2 Continual Policy Gradient

Batch policy gradient essentially uses stochastic gradient descent to estimate the gradient. Therefore, larger batch sizes aka. larger batch rollouts would give better and more accurate results. This however is not scalable since large rollouts implies large data buffers to store encountered states, actions, and rewards.

We need a method that can estimate the gradient above without the need of a large data buffer. This is crucial for long-horizon tasks. While the agent may update the gradient in batches, the algorithm can also use the current observation and reward to estimate the gradient. This is known as **Continual Policy Gradient**.




Continual policy gradient method do not update the agent using stored batches. Instead, it can only use information for the current and next step to update the agent. Thus, the following data is available to the agent at each timestep: $s_t, a_t, r_t, s_{t+1}$. The algorithm must only use this information to update agent. From the equations above, we can see that types 1-3 cannot be implemented directly since they require an episode rollout to calculate advantages. Types 4-6, on the other hand, can be modified to implement continual policy gradient. These advantages, however, suffer from high bias due to its over reliance on the critic network. A bad critic initialisation may prevent the policy from ever converging.

## 3 Generalised Advantage Estimation and TD($\lambda$)

A popular advantage estimate to combat the drawbacks of the other methods is the generalised advantage estimate which unifies one-step TD and Monte-Carlo methods. This advantage estimate is inspired from the $TD(\lambda)$ method, first introduced by Richard Sutton. The GAE is defined as an exponentially weighted average of $n$-step TD errors:

$$\Psi_t = A_t^{\text{GAE}} = (1-\lambda)(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} + \cdots)$$

where $A_t^n = \sum_{k=1}^{n} \gamma^{k-1} r_{t+k-1} + \gamma^n V(s_{t+n}) - V(s_t)$

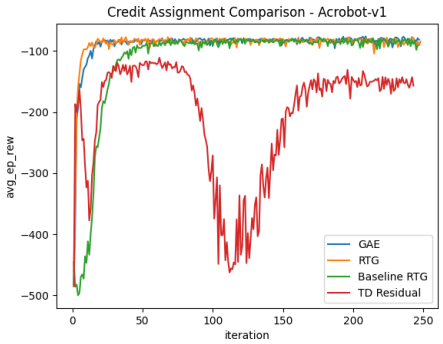
More importantly, the sum above can be written as a discounted sum of TD residuals.

$$A_t^{\text{GAE}} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V$$

where $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$

GAE offers many advantages which are discussed in later sections. One benefit is it allows us to choose the level of bias we want for our task. Choosing $\lambda = 0$ is equivalent to a one-step TD residual advantages. This has high bias but low variance. On the other extreme, $\lambda = 1$ results in the Monte-Carlo estimate (type 2).

GAE tries to solve a big issue in reinforcement learning known as the *credit assignment problem*. The difficulty lies in the long time delay between actions and their positive or negative effect on rewards. GAE simply assigned credit in an exponentially weighted way such that actions just preceding rewards are assigned more credit than later ones. However, instead of one-step or n-step TD, all actions may be updated in a trajectory.




The figure above shows the relative performance of various advantage estimators discussed. The graph show the best performance observed over 4 trials on the Acrobot-v1 tasks from the classic control suite in gymnasium. This task requires cumulative effort of actions to generate reward. Therefore, assignment credit to distant actions or even all actions is a good choice. The curves show that GAE ($\lambda = 0.9$) perform very well. Moreover, it also showed the least variance among all the advantage estimators.

# 4 Continual GAE with Eligibility Traces

The previous section showed that GAE is a good advantage estimator to choose to balance variance and bias. Another benefit of GAE is that it can be reformulated to work with continual policy gradients. This might seem improbable given that GAE consists of an average of n-step returns which require episodic rollouts. However, from the previous section notice that the estimate can also be written as a discounted sum of TD errors:

$$
\begin{aligned}
A_t^{\mathrm{GAE}} &= (1-\lambda)(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} + \cdots) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned}
$$

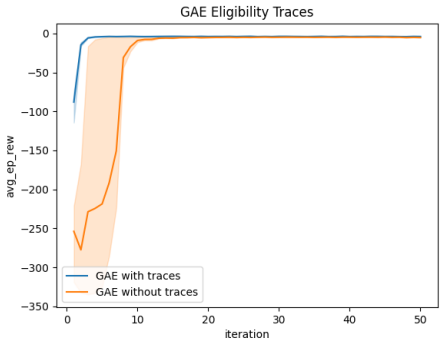
To make this estimate continual, we employ a mathematical trick known as eligibility traces which store a temporary record marking the occurrence of an event. For deep neural networks, we associate a trace for each weight in the network. At each step, we update the trace be decaying the existing trace and adding the gradient associated with the action $\nabla \log \pi_\theta$.

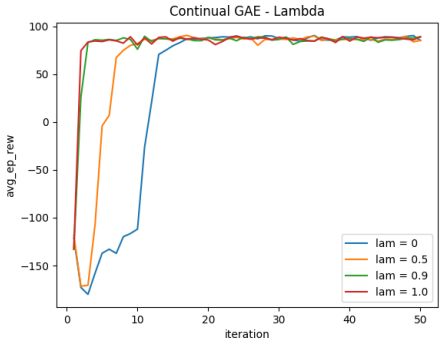
$$
\begin{aligned}
\epsilon_0 &:= 0 \\
\epsilon_t &:= (\gamma\lambda)\epsilon_{t-1} + \nabla \log \pi_\theta(a_t|s_t)
\end{aligned}
$$

Search for research labs

$$
g = \sum_{t=0}^{\infty} \delta_t^V \epsilon_t = \sum_{t=0}^{\infty} A_t^{\mathrm{GAE}} \nabla_\theta \log \pi_\theta(a_t|s_t)
$$

Eligibility traces can be implemented in Pytorch by designing a custom optimizer for the actor and critic networks. We associated a trace for each weight in the network. Furthermore, special functions must be implement to set/reset the trace and also broadcasting the TD error across the networks. The link to the code can be found here.






The first figure compares the performance of batch and continual GAE. We can see that both approaches converge to the same solution.

# References

[1] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

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

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

[4] Khadija Shaheen, Muhammad Abdullah Hanif, Osman Hasan, and Muhammad Shafique. Continual learning for real-world autonomous systems: Algorithms, challenges and frameworks. *Journal of Intelligent & Robotic Systems*, 105(1):9, 2022.

[5] Hado van Hasselt, Sephora Madjiheurem, Matteo Hessel, David Silver, Andr Barreto, and Diana Borsa. Expected eligibility traces, 2021.