# Week 2: Foundations of Policy Gradient Methods

BEH Chuen Yang

## Abstract

This document uses the foundations laid previously to explore Policy Gradient methods. We also study REINFORCE, a simple policy gradient algorithm formalized by Williams (1992), and discuss its use of *baselines* to reduce variance in the policy gradient estimates. Finally, we summarise the policy gradient method by devising a simple algorithm that uses the policy gradient to optimise the policy.

## 1 Recap

Consider a Markov Decision Process (MDP) $(S, A, \tau, r)$ where $S$ is the state space, $A$ is the action space, $\tau$ is the transition function and $r(s_t, a_t, s_{t+1})$ is the reward function. At any time step $t$, the agent has state $s_t$ and takes action $a_t$.

In Reinforcement Learning (RL), we are interested in learning a policy $a_t \sim \pi(\cdot|s_t)$ that maximises the *expected cumulative return*. Formally, where $T = (s_0, a_0, \ldots, a_{n-1}, s_n), s_0 \sim \rho_0$ describes a finite-horizon trajectory of length $n$,

$$\begin{aligned} \pi^* &= \arg\max_{\pi} \mathop{\mathbb{E}}_{T\sim\pi}\left[\sum_{t=0}^{n} r(s_t, a_t, s_{t+1})\right] \\ &= \arg\max_{\pi} \mathop{\mathbb{E}}_{T\sim\pi}\left[\sum_{t=0}^{n} \mathop{\mathbb{E}}_{s_{t+1}\sim\tau(s_t,a_t)}[r(s_t, a_t, s_{t+1})]\right] \\ &= \arg\max_{\pi} \mathop{\mathbb{E}}_{T\sim\pi}\left[\sum_{t=0}^{n} r(s_t, a_t)\right] \end{aligned} \tag{1}$$

## 2 Policy Gradient Methods

A straightforward technique to solve the above problem is to optimise for the expected cumulative return. This approach rests on the observation that the expected cumulative return is a differentiable function of the policy $\pi$, which allows us to approximate the optimal policy $\pi^*$, in a manner not too dissimilar to how we usually optimise differentiable functions in machine learning.

In deep learning especially, this can be done by using a *neural network* to represent the policy $\pi$, and then performing gradient ascent on the objective in Equation (1).

In order to further understand how this works, we will first derive the policy gradient.

Note that from here on, we assume the policy is a parameterised neural network $\theta$ whose architecture determines $\pi$. We will thus re-write the policy as $\pi_\theta$.

### 2.1 Deriving the Policy Gradient

(Note: The notation here has been synthesized from Achiam (2018), Levine et al. (2023) and Weng (2018)).

We first expand the objective in Equation (1) such that the inner terms are directly dependent on the policy $\pi_\theta$. Since $a_t \sim \pi_\theta(\cdot|s_t)$, $\pi_\theta(\cdot|s_t)$ contributes to the distribution of $s_{t+1}$. That is,

$$\begin{aligned}
\mathcal{P}(s_{t+1}|\pi_\theta) &= \sum_{a_t \in A} \mathcal{P}(\tau(s_t, a_t) = s_{t+1})\mathcal{P}(a_t|\pi_\theta, s_t) \\
&= \sum_{a_t \in A} \mathcal{P}(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)
\end{aligned} \tag{2}$$

We now extend this observation to consider whole trajectories $T$. Since the actions taken within each $T$ are *fixed*, we can write the probability of a *specific* trajectory $T$ under a given policy $\pi_\theta$ as:

$$\begin{aligned}
\mathcal{P}(T|\pi_\theta) &= \mathcal{P}(s_0|\rho_0) \prod_{t=0}^{n} \mathcal{P}(s_{t+1}|\pi_\theta) \\
&= \mathcal{P}(s_0|\rho_0) \prod_{t=0}^{n} \left( \sum_{a_t \in A} \mathcal{P}(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t) \right) \\
&= \mathcal{P}(s_0|\rho_0) \prod_{t=0}^{n} \left( \mathcal{P}(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t) \right)
\end{aligned} \tag{3}$$

As expected, the probability of a trajectory $T$ is dependent on the policy $\pi_\theta$, or rather, $\theta$. Now if we find $\nabla_\theta \mathcal{P}(T|\pi_\theta)$, we can use the sum rule to find the gradient of the expected cumulative return. This is because:

$$\begin{aligned}
\nabla_\theta \mathop{\mathbb{E}}_{T \sim \pi_\theta} \left[ \sum_{t=0}^{n} r(s_t, a_t) \right] &= \nabla_\theta \left[ \int_T \left[ \sum_{t=0}^{n} r(s_t, a_t) \right] \mathcal{P}(T|\pi_\theta) \right] \\
&= \int_T \left[ \sum_{t=0}^{n} r(s_t, a_t) \right] \nabla_\theta \mathcal{P}(T|\pi_\theta)
\end{aligned} \tag{4}$$

where, because we fixed the trajectory for each summand, $\sum_{t=0}^{n} r(s_t, a_t)$ are constants with respect to $\pi_\theta$.

With a product of probabilities this is difficult, but we can use a log transformation to change the expression to a sum of log probabilities, which is common in multivariable calculus [1]:

$$\nabla_\theta \mathcal{P}(T|\pi_\theta) = \mathcal{P}(T|\pi_\theta) \nabla_\theta \log(\mathcal{P}(T|\pi_\theta)) \tag{5}$$

Once again considering the entire trajectory $T$, we have:

$$\begin{aligned}
\nabla_\theta \log(\mathcal{P}(T|\pi_\theta)) &= \nabla_\theta \log(\mathcal{P}(s_0|\rho_0)) + \sum_{t=0}^{n} \nabla_\theta \log(\mathcal{P}(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t) \\
&= \cancelto{0}{\nabla_\theta \log(\mathcal{P}(s_0|\rho_0))} + \sum_{t=0}^{n} \left( \nabla_\theta \log(\mathcal{P}(s_{t+1}|s_t, a_t)) + \nabla_\theta \log(\pi_\theta(a_t|s_t)) \right) \\
&= \cancelto{0}{\nabla_\theta \log(\mathcal{P}(s_0|\rho_0))} + \sum_{t=0}^{n} \left( \cancelto{0}{\nabla_\theta \log(\mathcal{P}(s_{t+1}|s_t, a_t))} + \nabla_\theta \log(\pi_\theta(a_t|s_t)) \right) \\
&= \sum_{t=0}^{n} \nabla_\theta \log(\pi_\theta(a_t|s_t))
\end{aligned} \tag{6}$$

[1] Yes, it is also covered by Achiam (2018) and Williams (1992), but it just so happens to also have been covered in CS2040S. Thanks Eldric!

Now by using Equations (5) and (6), we can rewrite Equation (4) as:

$$\begin{aligned}
\nabla_\theta \mathop{\mathbb{E}}_{T\sim\pi_\theta}\left[\sum_{t=0}^{n} r(s_t, a_t)\right] &= \int_T \left[\sum_{t=0}^{n} r(s_t, a_t)\right] \nabla_\theta \mathcal{P}(T|\pi_\theta) \\
&= \int_T \left[\sum_{t=0}^{n} r(s_t, a_t)\right] \mathcal{P}(T|\pi_\theta) \nabla_\theta \log(\mathcal{P}(T|\pi_\theta) \\
&= \mathop{\mathbb{E}}_{T\sim\pi_\theta}\left[\sum_{t=0}^{n} r(s_t, a_t) \nabla_\theta \log(\mathcal{P}(T|\pi_\theta))\right] \\
&= \mathop{\mathbb{E}}_{T\sim\pi_\theta}\left[\left(\sum_{t=0}^{n} r(s_t, a_t)\right)\left(\sum_{t=0}^{n} \nabla_\theta \log(\pi_\theta(a_t|s_t))\right)\right]
\end{aligned} \tag{7}$$

For simplicity, we will henceforth define $J(\pi_\theta)$ as the policy objective and $\nabla_\theta J(\pi_\theta)$ as the policy gradient:

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{T\sim\pi_\theta}\left[\sum_{t=0}^{n} r(s_t, a_t)\right] \tag{8}$$

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{T\sim\pi_\theta}\left[\left(\sum_{t=0}^{n} r(s_t, a_t)\right)\left(\sum_{t=0}^{n} \nabla_\theta \log(\pi_\theta(a_t|s_t))\right)\right] \tag{9}$$

and write the gradient ascent update as:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta) \tag{10}$$

where $\alpha$ is a learning rate hyperparameter.

This completes the derivation of the *basic* policy gradient [2].

### 2.2 An Algorithm Using the Policy Gradient

Practically, however, we usually *estimate* $\nabla_\theta J(\pi_\theta)$ by sampling a batch of trajectories $T$ from the policy $\pi_\theta$ and the environment. This gives us the following estimation of the policy gradient, due to Sutton & Barto (2018) and Achiam (2018):

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=0}^{n} r(s_t, a_t)\right)\left(\sum_{t=0}^{n} \nabla_\theta \log(\pi_\theta(a_t|s_t))\right) \tag{11}$$

With that in mind, we can now summarise the above derivation into a simple algorithm that uses the policy gradient to optimise the policy $\pi_\theta$.

**Algorithm 1** Policy Gradient Algorithm

```
Input: Policy $\pi_\theta$, learning rate $\alpha$
Output: Updated policy $\pi_\theta$
Initialise policy parameters $\theta$
while not converged do
    Sample a batch of trajectories $T$ from the policy $\pi_\theta$ and the environment
    Estimate the policy gradient $\nabla_\theta J(\pi_\theta)$ using Equation (11)
    Update the policy parameters using (Equation (10))
end while
```

[2] However, with some effort, we can simplify the above expression further. For brevity, we will directly reference one such proof here (due to Achiam (2018)).

### 2.3 Improving the Policy Gradient

While the above derivation is theoretically sound, it can be improved in practice.

When we perform gradient ascent $J(\pi_\theta)$, we must do so by sampling a small number of trajectories $T$ from the policy $\pi_\theta$. This means our estimate of the policy gradient $\nabla_\theta J(\pi_\theta)$ will have variance of unknown magnitude, which can lead to instability and slow convergence in the training process.

One common approach is to consider only the *reward-to-go* $R_t$ at each time step $t$ (Achiam (2018), Levine et al. (2023)), which is the sum of the rewards from time step $t$ to the end of the trajectory:

$$R_t = \sum_{k=t}^{n} r(s_k, a_k) \tag{12}$$

This is intuitively explained by *causality*, since the rewards at time step $t$ depend only on the actions taken from time step $t$ onwards.

Another way to reduce the variance of the policy gradient estimate is to use a *state-dependent baseline* $b(s_t)$, as demonstrated in Achiam (2018) and Williams (1992). [3]

## 3 REINFORCE

REINFORCE is a simple policy gradient algorithm first formalized by Williams (1992) that uses the policy gradient derived above, but incorporates a baseline $b(s_t)$ to reduce the variance of the policy gradient estimate. [4]

### 3.1 Baseline

The baseline function $b(s_t)$ can reduce the variance of the policy gradient estimate due to the Expected Gradient Log Probability (EGLP) theorem (Achiam (2018)), which states that if $b(s_t)$ only depends on the state $s_t$, then

$$\mathop{\mathbb{E}}_{a_t \sim \pi_\theta(s_t)} [b(s_t) \nabla_\theta \log(\pi_\theta(a_t|s_t))] = 0 \tag{13}$$

This means that the baseline does not affect the expected value of the policy gradient estimate, but it can reduce the variance of the estimate by centering the rewards around the baseline.

This matches our intuition that a ”baseline” is a reference point on which we can anchor our perceptions of the rewards’ relative value.

### 3.2 REINFORCE Objective

Hence, the policy gradient with a baseline is given by:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{T \sim \pi_\theta} \left[ \sum_{t=0}^{n} [(r(s_t, a_t) - b(s_t)) \nabla_\theta \log(\pi_\theta(a_t|s_t))] \right] \tag{14}$$

where $b(s_t)$ is a state-dependent baseline function.

By Equation (13), we can see that the baseline does not affect the expected value of the policy gradient estimate, but it can reduce the variance of the estimate by centering the rewards around the baseline. This helps to stabilise the training process and improve convergence from a practical standpoint.

[3] Many sources, simply assert that the policy gradient has high variance, and *more crucially*, that the use of a baseline will lower variance in policy gradient estimates (see Weng (2018), Achiam (2018), Takeshi (2017), Sutton & Barto (2018)). However, these sources do not formally explain why. A proof is given in Wu et al. (2018).

[4] According to Williams (1992), any algorithm that uses a baseline is technically a REINFORCE algorithm.

### 3.3 Aside: Choices of Baseline

The choice of baseline function $b(s_t)$ can vary depending on the problem and the specific implementation of REINFORCE. However, a common choice is to use the value function $V_\theta(s_t)$ as the baseline (Achiam (2018), Sutton & Barto (2018), Weng (2018)), which is an estimate of the expected return from state $s_t$ under the current policy $\pi_\theta$.

This baseline is just the Bellman equation (Sutton & Barto (2018)), which states that the value function $V_\theta(s_t)$ is the expected return from state $s_t$ under the policy $\pi_\theta$:

$$V_\theta(s_t) = \mathop{\mathbb{E}}_{a_t \sim \pi_\theta(s_t)} \left[ r(s_t, a_t) + V_\theta(s_{t'})) \right] \tag{15}$$

This means that *an optimal* value function [5] $V_\theta(s_t)$ can be used as a baseline to reduce the variance of the policy gradient estimate.

Examples of $V_\theta(s_t)$'s use include Schulman et al. (2015) and Schulman et al., where they are used in two derivative algorithms, namely TRPO and PPO, respectively.

### 3.4 An Improved Policy Gradient Algorithm

We can now summarise the REINFORCE algorithm, which uses the policy gradient with a baseline to optimise the policy $\pi_\theta$.

For precision's sake (and to answer the homework question), we will rewrite the policy gradient estimate to incorporate a baseline, as well as the reward-to-go $R_t$:

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \left( (R_t - \sum_{t=0}^{n} b(s_t)) \right) \left( \sum_{t=0}^{n} \nabla_\theta \log(\pi_\theta(a_t|s_t)) \right) \tag{16}$$

**Algorithm 2** Improved Policy Gradient Algorithm

```
Input: Policy $\pi_\theta$, learning rate $\alpha$, baseline of choice $b(s_t)$
Output: Updated policy $\pi_\theta$
while not converged do
    Sample a batch of trajectories $T$ from the policy $\pi_\theta$
    for each trajectory $T$ in the batch do
        Compute the rewards-to-go $R_t$ for each time step $t$ in the trajectory
        Compute the baseline $b(s_t)$ for each time step $t$ in the trajectory
        Estimate the policy gradient $\nabla_\theta J(\pi_\theta)$ using Equation (16)
    end for
    Update the policy parameters using Equation (10)
end while
```

## 4 Conclusion

In this report, we have derived the policy gradient and discussed how it can be used to optimise policies in reinforcement learning. We have also shed light on a drawback of policy gradient methods, which is their high variance in practice. To tackle this, we studied REINFORCE by Williams (1992), which incorporates a baseline to reduce the variance of policy gradient estimates.

For now, this concludes theorywork on Reinforcement Learning. While we did not cover RL or MDPs in general (specifically, we only covered *finite-horizon*, *undiscounted reward* MDPs), we have covered the foundations of policy gradient methods in a way that makes the key intuitions behind them clear.

[5] Otherwise, $V_\theta(s_t)$ inevitably carries some bias. For more, see Schulman et al. (2016)

## References

Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

Sergey Levine, Kyle Stachowicz, Vivek Myers, Joey Hong, and Kevin Black, 2023. URL https://rail.eecs.berkeley.edu/deeprlcourse/.

John Schulman, Martin Wainwright, and Pieter Abbeel. Equivalence between policy gradient and q-learning. *ArXiv*, (1707.06347). URL https://arxiv.org/abs/1707.06347.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. *ArXiV*, 2015. URL https://arxiv.org/pdf/1502.05477.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *ArXiv*, 2016. URL https://arxiv.org/abs/1506.02438.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Daniel Takeshi. Going deeper into reinforcement learning: Fundamentals of policy gradients, 2017. URL https://danieltakeshi.github.io/2017/03/28/going-deeper-into-reinforcement-learning-fundamentals-of-policy-gradients/.

L. Weng. Policy gradient algorithms, 2018. URL https://lilianweng.github.io/posts/2018-04-08-policy-gradient/.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992. URL https://people.cs.umass.edu/~barto/courses/cs687/williams92simple.pdf.

Cathy Wu, Aravind Rajeswaran, Yan Duan, Vikash Kumar, Alexandre M. Bayen, Sham M. Kakade, Igor Mordatch, and P. Abbeel. Variance reduction for policy gradient with action-dependent factorized baselines. *ArXiv*, 2018. URL https://arxiv.org/abs/1803.07246.