IFT-7014: Directed reading

Deep Reinforcement Learning - Basic Policy gradient

v:1.1

Luc Coupal
Université Laval
Montréal, QC, Canada
Luc.Coupal.1@uLaval.ca

Under the supervision of:

Professor **Brahim Chaib-draa**
Directeur du programme de baccalauréat en génie logiciel de l'Université Laval
Québec, QC, Canada
Brahim.Chaib-draa@ift.ulaval.ca

October 1, 2019

CONTENTS

Basic policy gradient

Policy gradient is a on-policy method which seek to directly optimize the policy π_θ by using sampled trajectories τ as weight. Those weights will then be used to indicate how good the policy performed. Based on that knowledge, the algorithm updates the parameters θ of his policy to make action leading to similar good trajectories more likely and similar bad trajectories less likely. In the case of Deep Reinforcement Learning, the policy parameter θ is a neural net. For this essay, I've studied and implemented the basic version of policy gradient also known as REINFORCE [1]. I've also complemented my reading with the following ressources:

- CS 294–112 *Deep Reinforcement Learning*: lecture 4 5 and 9 by Sergey Levine from University Berkeley;
- *OpenAI: Spinning Up: Intro to Policy Optimization*, by Josh Achiam;
- and also *Lil' Log blog:Policy Gradient Algorithms* by Lilian Weng, research intern at *OpenAI*;

You can find my implementaion at https://github.com/RedLeader962/LectureDirigeDRLimplementation

Comment on notation:

Since their is a lot of different notations across paper, I've decided to follow (for the most part) the convention established by Sutton & Barto in their book Reinforcement Learning: An Introduction [2]

$(\mathcal{S}, \mathcal{A}, T, \mathcal{R})$	Markov decision process with $\mathcal{S}$ the state space, $\mathcal{A}$ the action space, the transition fct T and a reward space $\mathcal{R}$
$t \in [1, \mathsf{T}]$	time step
T	Time horizon time step $t \in [1, \mathsf{T}]$; Case 1: T is finite, case 2: T is infinite ($\mathsf{T} = \infty$)
τ	a trajectory $\mathbf{s}_1, \mathbf{a}_1, r_2, \mathbf{s}_2, \mathbf{a}_2, \ldots, r_\mathsf{T}, \mathbf{s}_\mathsf{T}, \mathbf{a}_\mathsf{T}$ aka: episode, trial, rollout
θ	Parameters vector a vector of m parameter $\theta = (\theta_1, \ldots, \theta_m)$
$\pi_\theta(\tau)$	Policy over trajectory τ
$\pi_\theta(\mathbf{a}_t \vert \mathbf{s}_t)$	Policy of parameter θ
$s, \mathbf{s}, s_t,$ or $\mathbf{s}_t$	State or state vector $s, \mathbf{s} \in \mathcal{S}$. $\mathbf{s}$ is a state compose of multiple informations ex: $\langle \text{coord} : (x, y, z), \text{heading} : 10\,\text{deg} \rangle$. $\mathbf{s}_t$ is a shorthand for $S_t = \mathbf{s}$ at time step $t \in [1, \mathsf{T}]$
$a, \mathbf{a}, a_t,$ or $\mathbf{a}_t$	Action $a, \mathbf{a} \in \mathcal{A}$. $\mathbf{a}$ is a action compose of multiple informations ex: $\langle x : +1, y : +2, z : 0,$ say:hello$\rangle$. $\mathbf{s}_t$ is a shorthand for $S_t = \mathbf{s}$ at time step $t \in [1, \mathsf{T}]$
r or r_t	the reward at time step t $r, r_t \in \mathcal{R}$. r_t is a shorthand for $R_t = r$ at time step $t \in [1, \mathsf{T}]$
$p(\mathbf{s}' \vert \mathbf{s}, \mathbf{a})$	unknown transition dynamic - 3 argument (aka transition fct of unknown model) model of the world express in term of conditional probability of getting from $(\mathbf{s}_t, \mathbf{a}_t)$ to $\mathbf{s}_{t+1}$
$r(\mathbf{s}_t, \mathbf{a}_t)$	expected immediate reward from state $\mathbf{s}$ after action $\mathbf{a}$ $r : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$;
$J(\theta)$	Objective function (optimisation) the function to optimize, sometime synonimous with loss/cost/error function
$\mathbb{E}_{\tau \sim \pi_\theta(\tau)}[X]$	Expectation of X with respect to the trajectory τ conditionned on the policy π_θ
γ	discount factor penality to uncertainty of futur rewards; $0 < \gamma \leq 1$
G_t or r_t^γ	return or total discounted futur reward $G_t = r_t^\gamma = \sum_{k=t}^{\infty} \gamma^{k-t}\, r(\mathbf{s}_k, \mathbf{a}_k)$

The big picture

Context:

- We have:
- A reward function
- A environment from which we can do many rollout with few restriction

- We dont have: Prior knowledge of the system dynamic

Key idea: "Formalize de notion of trial & error"

Goal: Find a optimal policy

How: Run the policy $\longrightarrow$ weight the run $\longrightarrow$ compute the gradient of the policy π_θ

Strenghts:
- Work in discreet & continuous space
- Dont need the system dynamic $p(\mathbf{s}'|\mathbf{s},\mathbf{a})$ to optimise the policy π_θ
- Can be use in partially observable MDP without modification

Weaknesses:
- High variance
- is ON-Policy

Improvements:
- Variance reduction:
  - True reward to GO
  - Baseline: average reward or optimal baseline

Policy gradient key component

Objective function: $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[r(\tau)] = \int \pi_\theta(\tau)\, r(\tau)\, d\tau$

Gradient of the objective: $\nabla_\theta J(\theta) = \int \nabla_\theta\, \pi_\theta(\tau)\, r(\tau)\, d\tau = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)\, r(\tau)]$

Sample based estimate: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau_i)\, r(\tau_i)$

Recall
- The reward function over a trajectory is given by: $r(\tau) = \sum_{t=1}^{\mathsf{T}} r(\mathbf{s}_t, \mathbf{a}_t)$
- The grad log policy over a particular trajectory is given by: $\nabla_\theta \log \pi_\theta(\tau) = \sum_{t=1}^{\mathsf{T}} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

Goal of Policy gradient

We seek the optimal parameter $\theta^\star$:

$$\begin{aligned}\theta^\star &= \arg\max_\theta J(\theta)\\ &= \arg\max_\theta \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[r(\tau)]\\ &= \arg\max_\theta \sum_{t=1}^{\mathsf{T}} \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\pi_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)]\end{aligned}$$

⟨ finite horizon case; See Actor-Critic for the infinite horizon case ⟩

with respect to trajectory τ conditionned on $\pi_\theta(\tau)$:

$$\pi_\theta(\tau) = \pi_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T) = p(\mathbf{s}_1)\prod_{t=1}^{\mathsf{T}} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$$

Building bloc of basic policy gradient

■ Finding $\nabla_\theta J(\theta)$: *Direct policy differentiation*

Problem: We want to solve $\nabla_\theta J(\theta)$ so we can climb it in order to improve π_θ

Let's do some math!

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \left(\mathbb{E}_{\tau\sim\pi_\theta(\tau)} [\, r(\tau) \,] \right) && \langle \text{ The reinforcement learning objective } \rangle \\
&= \int \nabla_\theta \, \pi_\theta(\tau) \, r(\tau) \, d\tau \\
& && \left\langle \text{Identity: } \quad \nabla_\theta \, \pi_\theta(\tau) = \tfrac{\pi_\theta(\tau)}{\pi_\theta(\tau)} \nabla_\theta \, \pi_\theta(\tau) = \pi_\theta(\tau) \nabla_\theta \, \log \pi_\theta(\tau) \right\rangle \\
&= \int \pi_\theta(\tau) \nabla_\theta \, \log \pi_\theta(\tau) \, r(\tau) \, d\tau \\
&= \mathbb{E}_{\tau\sim\pi_\theta(\tau)} [\, \nabla_\theta \log \, \pi_\theta(\tau) \, r(\tau) \,] \\
& && \left\langle \; \pi_\theta(\tau) \;\; = \;\; p(\mathbf{s}_1) \, \textstyle\prod_{t=1}^{\mathsf{T}} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \, p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \; \right\rangle \\
&= \mathbb{E}_{\tau\sim\pi_\theta(\tau)} \left[\nabla_\theta \log \left(p(\mathbf{s}_1) \prod_{t=1}^{\mathsf{T}} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \, p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \right) r(\tau) \right] \\
& && \left\langle \text{ Identity: } \; \log(A*B) = \log A + \log B \right\rangle \\
&= \mathbb{E}_{\tau\sim\pi_\theta(\tau)} \left[\nabla_\theta \left(\log p(\mathbf{s}_1) + \sum_{t=1}^{\mathsf{T}} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \right) r(\tau) \right] \\
& && \left\langle \text{terms who do not depend on } \theta \text{ get canceled out} \right\rangle \\
&= \mathbb{E}_{\tau\sim\pi_\theta(\tau)} \left[\nabla_\theta \left(\sum_{t=1}^{\mathsf{T}} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \right) r(\tau) \right] \\
& && \left\langle \text{ Distributive propertie: } \; \nabla(A+B) = \nabla A + \nabla B \right\rangle \\
&= \mathbb{E}_{\tau\sim\pi_\theta(\tau)} \left[\sum_{t=1}^{\mathsf{T}} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \; r(\tau) \right]
\end{aligned}
$$

Solution: Turns out **we dont need the transition dynamic $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ in order to compute $\nabla_\theta J(\theta)$**. We only need the policy π_θ and the reward function $r(\tau)$

■ Before we can optimize θ, we need to evaluate $J(\theta)$

Problem: Those trajectory τ are very high dimensionnal so **we can't evaluate τ exactly!**

Solution: We need to approximate τ by sampling from it's distribution.
$\Longrightarrow$ **Do a *sample based estimate*.**

How: By rolling out the policy $\pi_\theta(\tau)$. Ex: execute trajectories in a simulator, run the robot

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=1}^{\mathsf{T}} r(\mathbf{s}_t,\mathbf{a}_t)\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{i=1}^{\mathsf{T}} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) \qquad \langle \text{ with } N \text{ the number of samples } \rangle
\end{aligned}
$$

Using the ***direct policy differentiation*** **result** we get

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=1}^{\mathsf{T}} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\, r(\tau)\right] \\
&= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{t=1}^{\mathsf{T}} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{\mathsf{T}} r(\mathbf{s}_t,\mathbf{a}_t)\right)\right] \\
&\qquad \langle \textbf{estimate the gradient} \text{ over } N \text{ samples} \rangle \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{\mathsf{T}} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{\mathsf{T}} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right) \\
&= \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau_i)\, r(\tau_i)
\end{aligned}
$$

(Key idea) Building bloc of basic policy gradient

Use direct policy differentiation:

$$
\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=1}^{\mathsf{T}} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\; r(\tau)\right]
$$

$\Longrightarrow$

We dont need the transition dynamic in order to compute $\nabla_\theta J(\theta)$

Do a sample based estimate

$\Longleftrightarrow$

Rollout the policy π_θ to estimate τ

$$
J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{i=1}^{\mathsf{T}} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) \qquad \langle \text{ with } N \text{ the number of samples } \rangle
$$

Combining **Direct policy differentiation** with **sample based estimate**

$$
\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{\mathsf{T}} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{\mathsf{T}} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau_i)\, r(\tau_i)
$$

🔁 Algorithm anatomy: Basic policy gradient

What it does:

It gives a **conditionnal distribution** over actions given a state

$$\nabla_\theta J(\theta) \quad \approx \quad \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{\mathsf{T}}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{\mathsf{T}} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

It **weights each sample** by how good they are

Meaning: It give more importance to good trajectories and less importance to bad ones.

*" Policy gradient simply formalizes the notion of **trial & error** "*

Sergey Levine, CS 294–112 *Deep Reinforcement Learning*: lecture 5

Algorithm 1: *REINFORCE*

repeat

1. **sample** $\{\tau_i\}$ **from** $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ — "run the policy"
2. $\nabla_\theta J(\theta) \quad \approx \quad \frac{1}{N}\sum_i^N \left(\sum_t^{\mathsf{T}} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_t^{\mathsf{T}} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$ — "estimate the return"
3. $\theta \longleftarrow \theta + \alpha\,\nabla_\theta J(\theta)$ — "improve the policy"

until $i = N$

– **Markov property:**
Past state are never used in the optimisation phase
$\Longrightarrow$ The markov property is not used at all
$\Longrightarrow$ We can use basic policy gradient in partially observed MDP out of the box 👍

– **High variance:**
Each sample are weighted according to there *goodness/badness*
$\Longrightarrow$ Each samples **needs to be be compared to something** in order to measure it's quality.

❗ But right now the algorithm uses the *return* as an indicator of sample quality, **assuming it is a well defined measure**:

- Is it a strictly positive reward function?
- What if the best sample has a reward of 0?

This assumption leads to **high variance** 👎
which will give poor results if not adressed (See section on Improvment).

– **Is on-policy**
$\Longrightarrow$ At each iteration (after step 3 has been performed), new samples need to be generated since π_θ has changed. Training can be very inefficient 👎.
But it's possible to implement a off-policy PG version. More on this later in the semester.

🔃 (Key idea) Algorithm - Basic Policy Gradient

Formalize the notion of trial & error :	The algorithm weights each sample by how good they are using the return.
Markov property is not used :	$\Longrightarrow$ The algorithm can be used in POMDP out of the box.
Suffer from high variance:	This must be mitigated by applying diverse techniques otherwise the algorithm will poorly perform.
Is on-policy:	$\Longrightarrow$ Since we are optimizing a neural net, training can be very inefficient, but it's possible to implement an off-policy PG version.

Improvment: Reducing variance

▶ Trick 1: *The reward to GO*

Idea: The causality principle $\Longrightarrow$ The futur does not affect the past.

Which mean we can re-write the gradient of the objective like the following:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{\mathsf{T}}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{\mathsf{T}} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

$\langle$ Distribute the reward inside the sum over policy. $\rangle$

$$= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{\mathsf{T}}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(\sum_{t'=1}^{\mathsf{T}} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right)$$

$\langle$ Apply the causality principle by changing the start index of the sum of rewards $\rangle$

$$\geq \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{\mathsf{T}}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(\sum_{t'=t}^{\mathsf{T}} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right)$$

$$= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{\mathsf{T}}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\,\underbrace{\widehat{Q}_{i,t}}_{\text{This is the reward to GO}}$$

Why dose it work: Removing the past reward **make the sum smaller**

$\Longrightarrow$

the variance $\mathbb{V} = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right]$ become smaller to.

Pro:

- It's easy to implement
- It always work

Algorithm 2: *REINFORCE with reward to go*

repeat

1. **sample** $\{\tau_i\}$ **from** $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ "run"
2. $\nabla_\theta J(\theta) \geq \frac{1}{N}\sum_i^N \sum_t^{\mathsf{T}} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum_{t'=t}^{\mathsf{T}} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right)$ "estimate"
3. $\theta \longleftarrow \theta + \alpha\,\nabla_\theta J(\theta)$ "improve"

until $i = N$

▶ Trick 2: *Baselines*

Idea: Policy gradient seeks to formalize the notion of trial & error by making good policy appear more likely. So it needs a proper way to compare them to something that is a good comparaison point and do it in a way that can be measured.

Let's compare each run to the average reward of all rollout:

$$\nabla_\theta J(\theta) \quad \approx \quad \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \; (r(\tau) - \underbrace{b}_{\frac{1}{N}\sum_{i=1}^{T} r(\tau)})$$

Why does it work?

$$\mathbb{E}\Big[\nabla_\theta \log \pi_\theta(\tau)\,(\,r(\tau) - b\,)\Big] \quad = \quad \overbrace{\mathbb{E}\Big[\nabla_\theta \log \pi_\theta(\tau)\, r(\tau)\Big]}^{\text{We already know how to compute this term!}} - \underbrace{\mathbb{E}\Big[\nabla_\theta \log \pi_\theta(\tau)\, b\Big]}_{\text{What about this one?}}$$

$$\begin{aligned}
\mathbb{E}\Big[\nabla_\theta \log \pi_\theta(\tau)\, b\Big] &= \int \pi_\theta(\tau)\, \nabla_\theta \log \pi_\theta(\tau)\, b \; d\tau \\
&\qquad \Big\langle \text{Identity:} \quad \nabla_\theta\, \pi_\theta(\tau) = \tfrac{\pi_\theta(\tau)}{\pi_\theta(\tau)}\, \nabla_\theta\, \pi_\theta(\tau) = \pi_\theta(\tau)\, \nabla_\theta\, \log \pi_\theta(\tau) \Big\rangle \\
&= \int \nabla_\theta \pi_\theta(\tau)\, b \; d\tau \\
&= b\, \nabla_\theta \int \pi_\theta(\tau) \; d\tau \\
&\qquad \Big\langle \text{Integrale of a probability distribution sum to 1} \Big\rangle \\
&= b\, \nabla_\theta\, 1 \\
&\qquad \Big\langle \text{Gradient of a constant is 0} \Big\rangle \\
&= 0
\end{aligned}$$

This implies that

$$\mathbb{E}\Big[\nabla_\theta \log \pi_\theta(\tau)\,(\,r(\tau) - b\,)\Big] \quad = \quad \mathbb{E}\Big[\nabla_\theta \log \pi_\theta(\tau)\, r(\tau)\Big] - 0 \tag{1}$$

wich mean that comparing trajectories to a baseline wont change it's expectation even if it's variance changes. Leveraging this idea could lead to reduce variance (provided we chose a good baseline) and it's exactly the improvment that we want.

▶ Choosing & Computing a baseline:

There is multiple way of computing a baseline but let's look at the 2 base cases:
the **simpliest** one and the **optimal** one.

1: The simpliest baseline:

The average reward

$$b = \frac{1}{N}\sum_{i=1}^{N} r(\tau)$$

- It's the simpliest one to implement 👍;
- It's not the best one, but it give relatively good result.

2: The optimal baseline:

What is the best baseline? The one with the optimal variance (the lowest one).

How do we find it? It's a optimisation problem:

Step 1. Write the equation for variance;
Step 2. Take it's derivative;
Step 3. Set the derivative to zero.

The weighted expected reward

$$b = \frac{\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\, r(\tau)\right]}{\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\right]} \qquad \langle \text{ with } g(\tau) = \nabla_\theta \log \pi_\theta(\tau) \rangle$$

This is the expected reward weighted by gradient magnitude

- Tricky to implement 👎;
- The gain does not always outweight the trouble of implementing it;

Demonstration:
Since $\nabla_\theta J(\theta)$ is an expectation $\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\nabla_\theta \log \pi_\theta(\tau)\,(r(\tau)-b)\right]$, we can write it's variance like the following:

Step 1: Write the equation for variance

$$\begin{aligned}
\mathbb{V}[X] &= \mathbb{E}\left[X^2\right] - \mathbb{E}\left[X\right]^2 \\
&\qquad \langle \; X \coloneqq \nabla_\theta \log \pi_\theta(\tau)\,(r(\tau)-b) \; \rangle \\
\mathbb{V}[\nabla_\theta \log \pi_\theta(\tau)\,(r(\tau)-b)] &= \mathbb{E}\left[\left(\nabla_\theta \log \pi_\theta(\tau)\,(r(\tau)-b)\right)^2\right] - \mathbb{E}\left[\nabla_\theta \log \pi_\theta(\tau)\,(r(\tau)-b)\right]^2 \\
&\qquad \langle \text{ baselines } b \text{ are unbias in expectation (1) } \rangle \\
&= \mathbb{E}\left[\left(\nabla_\theta \log \pi_\theta(\tau)\,(r(\tau)-b)\right)^2\right] - \mathbb{E}\left[\nabla_\theta \log \pi_\theta(\tau)\, r(\tau)\right]^2 \\
&\qquad \langle \; g(\tau) \coloneqq \nabla_\theta \log \pi_\theta(\tau), \text{ and } (ab)^2 = a^2b^2 \; \rangle \\
&= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\,(r(\tau)-b)^2\right] - \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)\right]
\end{aligned}$$

Step 2: Take it's derivative

$$\begin{aligned}
\frac{d\mathbb{V}}{db} &= \frac{d}{db}\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\left(r(\tau)-b\right)^2\right] - \frac{d}{db}\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)\right] \\
&= \frac{d}{db}\left(\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\, r(\tau)^2\right] - 2\,\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\, r(\tau)\, b\right] - b^2\,\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\right]\right) - 0 \\
&= 0 - 2\frac{d}{db}\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\, r(\tau)\, b\right] + \frac{d}{db}\left(b^2\,\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\right]\right) \\
&= -2\,\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\, r(\tau)\right] + 2b\,\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\right]
\end{aligned}$$

Step3: Set the derivative to zero

$$\begin{aligned}
\frac{d\mathbb{V}}{db} &= \cdots = 0 \\
&= -2\,\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\, r(\tau)\right] + 2b\,\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\right] = 0 \\
&= b = \frac{\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\, r(\tau)\right]}{\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\right]}
\end{aligned}$$

This means that baseline $b = \dfrac{\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\, r(\tau)\right]}{\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[g(\tau)^2\right]}$ is the optimal baseline.

Key improvment - Reducing variance of Basic Policy Gradient

The reward to GO: Compute the return by starting at timestep t and sum until the horizon end.

" The futur does not affect the past "

$$\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$$

The baseline Instead of evaluating the goodness of each rollout using the return as a measure, do it by comparing each rollout to a baseline b and use their difference as a measure .

- We can think of a baseline as an average return over all rollout.
- There is several way to compute a baseline: the average reward is a simple and good enough one.

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{T}\nabla_\theta \log \pi_\theta(\tau)\left[r(\tau) - b\right]$$

Basic policy gradient - In Practice

›_ From theory to practice

Implementing an algorithm is a true test of it's comprehention. Situation where abstract ideas clashes with concrete limitations arises and are a great source of learning opportunities. Also, subtle details that we might have overlooked during theory can suddenly be the difference between a broken or a working algorithm.

Like collecting the *right observation* $\mathbf{s}_t$ that trigered the action $\mathbf{a}_t$, ... here t is a critical detail. I say the *right observation* because collecting accidently the *observation* $\mathbf{s}_{t+1}$ will cause the algorithm to map a policy $(\text{action} \times \text{observe reaction}) \longrightarrow [0, 1]$ instead of $(\text{observation} \times \text{action}) \longrightarrow [0, 1]$ and the agent wont learn. This mistake is very easy to make when implementing if we are not carreful with detail.

</> Implementing REINFORCE

For this assignement, I have chosen to implement *REINFORCE with reward to go*, one of the most basic version of policy gradient. I have limited it's developement to tackle discrete environnement, leaving the continous case for the next project.

Main effort: Build the basic component that would make a good foundation for later developement & experimentation. With this in mind, I have focused on implementing the following:

- key component of RL algorithm (I call them bloc) with a focus on reusability;
- the REINFORCE agent learning loop;
- the REINFORCE computation graph in TensorFlow;
- a sample container & collector responsible of agregating and processing sampled trajectories;
- a consol printer that summarizes relevant training metric during training;

The simulator: I've use OpenAI Gym with the environment 'CartPole-v0'. I've chosen that environement because of it's simplicity and the speed at which it execute trajectories. Those qualities permit fast iteration between implementing and getting results from training, thus maximizing feedback. This is essential to quickly asses if what you did is wrong and move foward.

</> Sampling consideration: Batch size & shape matters

Why? Because sampling will naturally produce trajectories of different lenghts. This must be taken into consideration otherwise it will prevent the computation graph from executing certain operation required to minimize the loss.

Two option:

1. constrain each batch to the same dimension, cutting exceding timesteps of certain trajectories;
2. or tweak the loss fonction with a *Q-values mask placeholder* of constrained shappe padded with zeroes;

In any case, this implementation detail was not obvious to me at first when studying DRL litterature in general and is one of the important details I've learned from passing from theory to practice in the context of DRL.

›_ Lesson learned & best practice

As S. Zayd Enam have so eloquently put it in his essay on Why is machine learning 'hard'?

> *" What is unique about machine learning is that it is exponentially harder to figure out what is wrong when things don't work as expected "*

He argued that from the two usual dimensions where things can go wrong in standard sofware engineering (algoritmic and implementation), machine learning add two extra dimensions where things can go wrong (the model and the data).

Add to this the fact that RL algorithm extends this complexity by adding the concept of time. Surely, RL software engineering face additionnal problems. One of the main problems arises from the lack of immediate feedback when the algorithm does not work properly or worst, which mean no feedback at all. It might compile, it might act and look like it works but in reality it does not learn or it does not learn well enough.

Usually, it's really hard to pin point the origin of the problem because of this lack of information: after all, your code compiles, no error message, nothing is red, ... so where do we look? What do we look for? Do we read the full code base? What do we change? When this happens, things can quickly spiral out of control, shooting in every direction tinking your aiming at the right thing.

</> The art of debugging reinforcement learning algorithm

> *" ... broken RL code almost always fails silently, ... "*
>
> Josh Achiam, OpenAI Spinning Up

What I've learned is that working the problem methodically is the most effective way to solve it. But in order to do that, we need first to build some tools to make our code talk to us:

Compute relevant metric: It will give you precious indication on the behavior of the agent. Is he learning sommething? Is he surviving the environment for a longer periode? Does the loss function get computed at all?

Write relevant unit-test: It will give you confidence on your code base and greatly limit the scope of your search during debuging. Make sure to test interaction between parts and test their behavior.

Assertion statement are your friend: Use them to chalenge the expected input/output of each component of your code. Make those assertions informative with explicit error message. But be careful to put those assertion in piece of code that does not get executed millions of time during training if you are developing in Python. It's a interpreted language, not a compiled one. Alternatively, make sure to be able to turn them off when you don't need them.

</> Integration test for DRL

Goal: Narrow the scope of your debuging quest.

One way I've founded to get myself out of a "silent" broken implementation was to write a integration test leveraging a working implementation as reference. I've written the test so that it PASSED when the references algortihm reached a certain metric goal: in my case a return of 200 in less then 50 epoch, otherwise the test FAILED.

Procedure:

1. **Transform the reference** implementation as a integration test which asserts a condition on a relevant metric;

2. Then **proceed by elimination**, by substituting each part of the reference algorithm, one part at the time;

   Insert one part $\longrightarrow$ execute the integration test

```
86
87      # make loss function whose gradient, for the right data, is policy gradient
88      # weights_ph = tf.placeholder(shape=(None,), dtype=tf.float32)          # ////// Original bloc //////
89      # act_ph = tf.placeholder(shape=(None,), dtype=tf.int32)                # ////// Original bloc //////
90      # action_masks = tf.one_hot(act_ph, n_acts)                             # ////// Original bloc //////
91      # log_probs = tf.reduce_sum(action_masks * tf.nn.log_softmax(logits), axis=1)  # ////// Original bloc //////
92      # loss = -tf.reduce_mean(weights_ph * log_probs)                        # ////// Original bloc //////
93      # loss = BLOC.discrete_pseudo_loss(log_p_all, act_ph, weights_ph, playground)  # \\\\\\    My bloc    \\\\\\
94
95      reinforce_policy = BLOC.REINFORCE_policy(obs_ph, act_ph,                # \\\\\\    My bloc    \\\\\\
96                                               weights_ph, exp_spec, playground)  # \\\\\\    My bloc    \\\\\\
97      (actions, _, loss) = reinforce_policy                                   # \\\\\\    My bloc    \\\\\\
```

3. **Repeat until it FAILS** the integration test;

```
Tests failed: 1, ignored: 1 of 2 tests – 2 m 18 s 131 ms
E       AssertionError:
E
E         :: The agent FAILED to learned enough in 50 epoch
E             - Test run over 2 run
E             - Env: CartPole-v0 with NN hidden [62]
E             - Required mean return 200.0
E
E               Run 0
E                 | Loss:           0.037
E                 | Mean return:    9.412 < 200.0 !!
E               Run 1
E                 | Loss:           0.052
E                 | Mean return:    9.459 < 200.0 !!
E
```

4. When it does, you will now be able to **assess new information**:

   - where to look for to solve your quest;
   - and what piece of code are probably good to go;

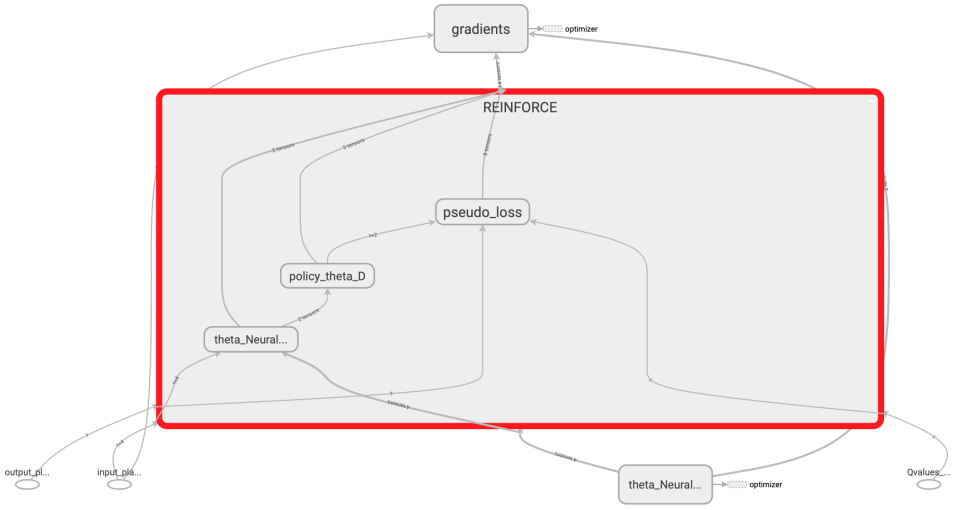
   thus **narrowing down the scope of your search quest** 👍.

Those are precious new information and can save you a lot of effort going in the wrong direction.

Important note: Take into account the stochastic aspect of DRL algorithm, you need to make your test PASS multiple times in order to have a green light or run it against multiple random seeds.

</> TensorFlow best practice (In my opinion)

Name key component of your computation graph: Use *name scope* or *variable scope* to give meaningful name to your operation, then use TensorBoard to visualize your computation graph. It's much easier to assess graph construction problem when you can inspect it visualy and navigate your way through it. TensorBoard gives you access to a lot of information, like which tensor pipe data into which tensor and the shape of the tensor. You can also visualize collected summary operations information.



Closing thoughts

This project did not turne out the way I expected and I certainly did not expect those challenges ... particularly dealing with 2 different silent errors back to back. Even if it was 10 times more time consuming than anticipated, it has been a learning oportunity filled with reward and lessons learned. I have now gained a priceless knowledge regarding practical consideration in the Deep Reinforcement Learning field, one that I would not have been able to gained had I not failed at the first attempt.

I have now a greater understanding of every moving parts of *basic policy gradient* algorithm, a better knowledge of some critical details to pay attention for when implementing DRL algorithm and new skills on how to solve certain problems specific to the DRL world.

</> Implementation detail to tackle for futur projects:

- Implement the policy for continuous environment;
- Refactor the agent logic as a class to maximize reusability;
- Implement TensorFlow summary operations to make use of TensorBoard to visualize learning metric;

BIBLIOGRAPHY

1. Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* **8,** 229–256. ISSN: 0885-6125 (1992).
2. Sutton, R. S. & Barto, A. G. *Reinforcement learning: An introduction* 2nd ed. (ed MIT Press) ISBN: 978-0262039246. `http://incompleteideas.net/book/RLbook2018.pdf` (Cambridge, MA, 2018).