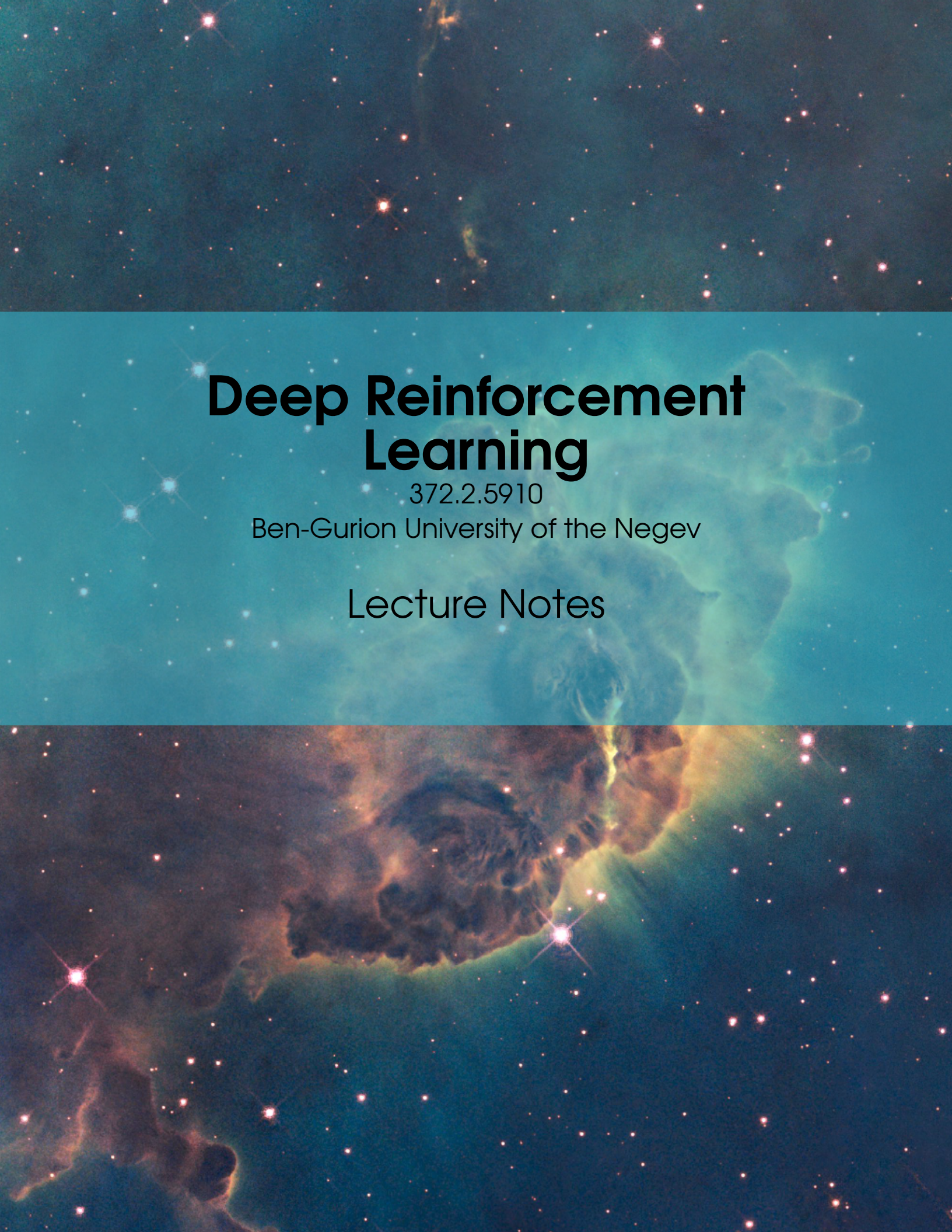


Deep Reinforcement Learning

372.2.5910

Ben-Gurion University of the Negev

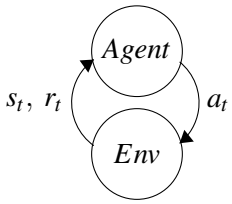
Lecture Notes

WRITTEN BY: Hadar Sharvit
ALSO AVAILABLE ON GITHUB:
CONTACT ME AT: Hadar.Sharvit1@mail.Huji.ac.il
BASED ON: Lectures given by Gilad Katz
LAST UPDATE: October 26, 2022

Contents

1. Hello world

Based on OpenAI's Spinning Up docs (For further references see here).
Reinforcement Learning (RL) is the study of agents and how they learn by trial and error. The two main components of RL are the *agent* and the *environment* - The agent interacts with the environment (also known as taking a "step") by seeing a (sometimes partial) *observation* of the environment's *state*, and then decides which *action* should be. The agent also perceives a reward from the environment, which is essentially a number that tells the agent how good the state of the world is, and the agent's goal is to maximize the *cumulative reward*, called *return*.



1.1 Terminology

let's introduce some additional terminology

1.1.1 State & Observation

The complete description of the environment/world's state is the *state s*, and an *observation o* is a partial description of *s*. We usually work with an observation, but wrongly denote it as *s* - we are going to stick with this convention.

R if $o = s$ we say that the environment is *fully observed*. Otherwise, it is *partially observed*.

1.1.2 Action spaces

Is the set of all valid actions in the environment. The action space could be discrete (like the action to move in one of the direction $\{\uparrow,\downarrow,\leftarrow,\rightarrow\}$) or continuous (like the action to move the motor with $\alpha \in \mathbb{R}$ Newtons of force)

1.1.3 Policy

Is a set of rules used by our agent to decide on the next action. It can be either deterministic of stochastic $a_t \sim \pi(\cdot|s_t)$. Under the scope of deep RL, the policy is a parameterized function, i.e it is a mapping with parameters θ that should be learned in some optimization process. A deterministic Policy could be implemented, for example, using some basic MLP architecture. For a stochastic policy, the two most common types are *Categorical* policy (for discrete action space) and *Diagonal-Gaussian* policy (for continuous action space)

Categorical (stochastic) Policy

Is essentially a classifier, mapping discrete states to discrete actions. For example, you could build a basic NN that takes in the observation and outputs action probabilities (after applying softmax). Denoting the last layer as $P_\theta(s)$, we can treat the actions as indices so the log-likelihood for action a is

$$\log \pi_\theta(a|s) = \log [P_\theta(s)]_a \tag{1.1}$$

Given $P_\theta(s)$, we can also sample from the distribution (one can use PyTorch `Categorical` to sample from a probability vector)

Diagonal-Gaussian (stochastic) Policy

Is a policy that can be implemented using a neural network that maps observations to *mean* actions, under the assumption that the action probability space can be represented by some multivariate Gaussian with diagonal covariance matrix, which can be represented in two ways

- we use $\log diag(\Sigma) = \log \sigma$ which is *not* a function of the state s (σ is a vector of standalone parameters)
- we use a NN that maps from $s \rightarrow \log \sigma_\theta(s)$

we use $\log \sigma$ and not σ as the log takes any value $\in (-\infty,\infty)$, unlike σ that only takes values in $[0,\infty)$, making it harder to train.
Once the mean action $\mu_\theta(s)$ and the std $\sigma_\theta(s)$ are obtain, the action is sampled as $a = \mu_\theta(s) + \sigma_\theta(s) \otimes z$, where $z \sim N(0,1)$ and $\otimes$ is element-wise multiplication (This is similar to VAEs).
The log-likelihood of a k-dimensional action $a \in \mathbb{R}^k$ for a diagonal-Gaussian with mean μ_θ and std σ_θ can be simplified if we remember that when Σ is diagonal, a k-multivariate Gaussian's PDF is equivalent to the product of k one-dimension Gaussian PDF, hence

$$\log [\pi_\theta(a|s)] = \log \left[\frac{\exp\left[-\frac{1}{2}(a-\mu)^T \Sigma^{-1}(a-\mu)\right]}{\sqrt{(2\pi)^k |\Sigma|}} \right] = \ldots = -\frac{1}{2}\left[\sum_{i=1}^{k} \left(\frac{(a_i-\mu_i)^2}{\sigma_i^2} + 2\log \sigma_i \right) + k \log 2\pi \right] \tag{1.2}$$

1.1.4 Trajectories

we denote the trajectory as $\tau = (s_0, a_0, s_1, a_1, \ldots)$ where the first state s_0 is randomly sampled from some start-state distribution $s_0 \sim \rho_0$. A new state is obtained from the previous state and action in either a stochastic or deterministic process.

R τ is also noted as "episode" or "rollout"

1.1.5 Reward

The reward r_t is some function of our states and action, and the goal of the agent is to maximize the cumulative reward over some trajectory τ.

- Finite-horizon un-discounted return: $R(\tau) = \sum_{t=0}^{T} r_t$
- Infinite-horizon discounted return: $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$, $\gamma \in (0,1)$

Not adding a converging term γ^t means that our infinite sum may diverge, but also it manifests the concept of "reward now $>$ reward later"

1.1.6 The goal of RL

We always wish to find a policy π^* which maximizes the expected return when the agent acts according to it. Under the assumption of stochastic environment and policy, we can write the probability to obtain some trajectory τ of size T, given a policy π as

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} \underbrace{P(s_{t+1}|s_t, a_t)}_{\substack{\text{Pr. to reach } s_{t+1} \\ \text{from } s_t \text{ when applying } a_t.}} \cdot \underbrace{\pi(a_t|s_t)}_{\substack{\text{Pr. to choose action } a_t \\ \text{when in state } s_t.}} \tag{1.3}$$

The expected return is by definition the sum of returns given all possible trajectories, weighted by their probabilities

$$J(\pi) = \int_{\tau} P(\tau|\pi) R(\tau) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] \tag{1.4}$$

and w.r.t to this objective, we wish to find

$$\pi^* = \underset{\pi\in\Pi}{\operatorname{argmax}} J(\pi) \tag{1.5}$$

1.1.7 Value function

We can think of the expected return given some specific state, or some specific state-action pair as the "value" of the state, or the state-action pair. Those are simply the expected return conditioned with some initial state or action

- On-policy Value function $V^\pi(s)$: if you start from s and act according to π, the expected reward is

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s]$$

- On-policy Action-Value function $Q^\pi(s)$: if you start from s, take an action a (which may or may not come from π) and only then act according to π, the expected reward is

$$Q^\pi(s,a) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]$$

when finding a value function or an action-value function that maximizes the expected reward, we scan various policies and extract $V^*(s) = \max_{\pi \in \Pi} V^\pi(s)$ or $Q^*(s,a) = \max_{\pi \in \Pi} Q^\pi(s,a)$.
We can also find a relation between V and Q:

$$\begin{aligned}
V^\pi(s) &= \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s] \\
&= \sum_{\tau\sim\pi} Pr[R(\tau)|s_0 = s]R(\tau) \\
&= \sum_{\tau\sim\pi}\sum_{a\sim\pi} Pr[R(\tau), a|s_0 = s]R(\tau) \text{ [Total prob.]} \\
&= \sum_{a\sim\pi} Pr[a|s_0 = s] \sum_{\tau\sim\pi} Pr[R(\tau)|s_0 = s, a_0 = a]R(\tau) \\
&= \sum_{a\sim\pi} Pr[a|s_0 = s]\mathbb{E}_{\tau\sim\pi} Pr[R(\tau)|s_0 = s, a_0 = a]R(\tau) \\
&= \sum_{a\sim\pi} Pr[a|s_0 = s]Q^\pi(s,a) \\
&= \mathbb{E}_{a\sim\pi} Q^\pi(s,a)
\end{aligned} \tag{1.6}$$

where in the 4'th line we used the fact that the probability of both $R(\tau)$ and a is the same as summing over all possible a and conditioning the probability $Pr[R(\tau)]$ given a. The above also means that

$$\begin{aligned}
V^*(s) &= \max_{\pi\in\Pi} V^\pi(s) \\
&= \max_{\pi\in\Pi} \mathbb{E}_{a\sim\pi} Q^\pi(s,a) \\
&= \mathbb{E}_{a\sim\pi} \max_{\pi\in\Pi} Q^\pi(s,a) \\
&= \mathbb{E}_{a\sim\pi} Q^*(s,a) \\
&= Q^*(s,a)
\end{aligned} \tag{1.7}$$

1.1.8 The optimal Q-Function and the optimal action

$Q^*(s,a)$ gives the expected return for starting in s and taking action a, and then acting according to the optimal policy. As the optimal policy will select, when in s, the action that maximizes the expected return for when the initial state is s, we can obtain the optimal action a^* by simply maximizing over all values of Q^*

$$a^*(s) = \operatorname*{argmax}_a Q^*(s,a) \tag{1.8}$$

We also note that if there are many optimal actions, we may choose one randomly

1.1.9 Bellman Equations

An important idea for all value functions is that the value of your starting point is the reward you expected to get from being there + the value of wherever you land next

$$V^\pi(s) = \mathbb{E}_{a\sim\pi, s'\sim P}\left[r(s,a) + \gamma V^\pi(s')\right] \tag{1.9}$$

$$Q^\pi(s,a) = \mathbb{E}_{s'\sim P}\left[r(s,a) + \gamma \mathbb{E}_{a'\sim\pi}[Q^\pi(s',a')]\right] \tag{1.10}$$

and optimality is obtained for

$$V^*(s) = \max_{a\sim\pi} \mathbb{E}_{s'\sim P}\left[r(s,a) + \gamma V^*(s')\right] \tag{1.11}$$

$$Q^*(s,a) = \mathbb{E}_{s'\sim P}\left[r(s,a) + \gamma \max_{a'\sim\pi}[Q^*(s',a')]\right] \tag{1.12}$$

1.1.10 Advantage function

Sometimes we only care if an action is better than others on average, and do not care as much for its' value on its own. The advantage function $A^\pi(s,a)$ describes how better is taking action a (that can be from π or not) when in s compared to selecting some random action $a' \sim \pi$, assuming you act according to π afterwards.

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{1.13}$$

1.2 Kinds of RL Algorithms

1.2.1 Model-Free vs Model-Based RL

In some cases we have a closed form of how our environment behaves. For example, we may know the probability space $P[s'|s,a]$, i.e we know that probability to transition from some s to some other s' given arbitrary action a. It may also be the case that our our model can be described using some equation of motion. Either way, we can use this knowledge to formulate an optimal solution, which in many cases translate to some greedy approach of scanning various states and thinking ahead.
Some problems are that such model may not even be available (or even known), as for example, I do not have a model of how my chess opponent may play. Furthermore, greedy approaches usually mean brute-forcing all possible solutions.
In Model-Free RL, the model is not available, and we are trying to understand how the environment behaves by exploring it in some non-exhaustive manner. This means that model-free RL are likely to be not sample-efficient, though they are usually easier to implement

1.2.2 What do we learn in RL

After going through the taxonomy, we can ask whether we wish to learn the Q-function, the value-function, the policy or the environment model itself.

Under model-free RL

- Policy optimization: we parameterize the policy $\pi_\theta(a|s)$ and find optimum w.r.t the return $J(\pi_\theta)$. Such optimization is usually *on-policy*, meaning that the data used in the training process is only data given while acting according to the most recent version of the policy. In policy optimization we also find an approximator value function $V_\phi(s) \approx V^\pi(s)$. Some examples are *A2C,A3C,PPO*.
- Q-Learning: approximate $Q_\theta(s,a) \approx Q^*(s,a)$. Usually the objective is some form of the bellman equation. Q-Learning is usually *off-policy*, meaning that we use data from any point during training. Some examples are *DQN, C51*.

Compared to Q-Learning, that approximates Q^*, policy optimization finds exactly what we wish for - how to act optimally in the environment. Also, there are models that combine the two approaches, as *DDPG* for example, which learns both a Q function and an optimal policy.

Under model-based RL

cannot be clustered as easily, though some of the (many) approaches include methods of planning techniques to select actions that are optimal w.r.t to the model.

1.3 Intro to policy optimization

We aim to maximize the expected return $J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)]$, and we assume the finite-horizon undiscounted return (∞-horizon is nearly identical). Our goal is to optimize π_θ with a gradient step

$$\theta_{k+1} = \theta_k + \alpha \underbrace{\nabla_\theta J(\pi_\theta)}_{\text{Policy gradient}} |_{\theta_k} \tag{1.14}$$

and to do so, we must find a numerical expression for the policy gradient. As $J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] = \int_\tau P(\tau|\theta)R(\tau)$, we might as well write down a term for the probability of a trajectory

$$P(\tau|\theta) = \rho_0(s_0)\prod_{t=0}^{T} P(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t) \tag{1.15}$$

Using the log-derivative trick, $\frac{d}{dx}\log x = \frac{1}{x}$, meaning that $x\frac{d\log x}{dx} = 1$. rewrite 1 as $\frac{d}{dx}x$, Substitute $x \leftrightarrow P(\tau|\theta)$ and $\frac{d}{dx} \leftrightarrow \nabla_\theta$ and we have that $P(\tau|\theta)\nabla_\theta \log P(\tau|\theta) = \nabla_\theta P(\tau|\theta)$. We will use this later. Now, lets expand the log term

$$\log P(\tau|\theta) = \log\rho_0(s_0) + \sum_{t=0}^{T}[\log P(s_{t+1}|s_t,a_t) + \log\pi_\theta(a_t|s_t)] \tag{1.16}$$

When deriving w.r.t θ, we are only left with the last term (the others only depend on the environment and not our agent), hence

$$\nabla_\theta \log P(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T} \log\pi_\theta(a_t|s_t) = \sum_{t=0}^{T} \nabla_\theta \log\pi_\theta(a_t|s_t) \tag{1.17}$$

Notice the use of linearity in the second transition. Consequently, we re-write the expected return using eq 1.17 -

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) \\
&= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) \\
&= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau) \\
&= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau|\theta)R(\tau)] \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log\pi_\theta(a_t|s_t)R(\tau)\right]
\end{aligned} \tag{1.18}$$

In the 3rd transition we used the log-derivative trick, and in the last transition we used the expression from 1.17.

The last term is an expectation, hence can be estimated using mean - given a collected set $D =$

$\{\tau_1, \tau_2, ..., \tau_N\}$ of trajectories obtained by letting our agent act in the environment using π_θ we can write

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \tag{1.19}$$

It should be stated that this "Loss" term is not really a "loss" like we know from supervised learning. First of all, it does not depend on a fixed data distribution - here, the data is sampled from the recent policy. More importantly, it does not measure performance! the only thing it makes sure of is that given the *current* parameters, it has the negative gradient of performance. After this first step of gradient descent, there is no more connection to performance. This means that the loss minimization has no guarantee to improve expected return. This should come as a warning to when we look at the loss going down thinking that all is well - in policy gradients, this intuition is wrong, and we should only look at the average return.