Crash Coruse in Reinforcement Learning

Damian Leśniak
Aleksandra Nowak
Igor Sieradzki
Maciej Wołczyk

November 25, 2019

group of machine
gmum
learning research

PART 1: Introduction To Reinforcement Learning

Reinforcement Learning

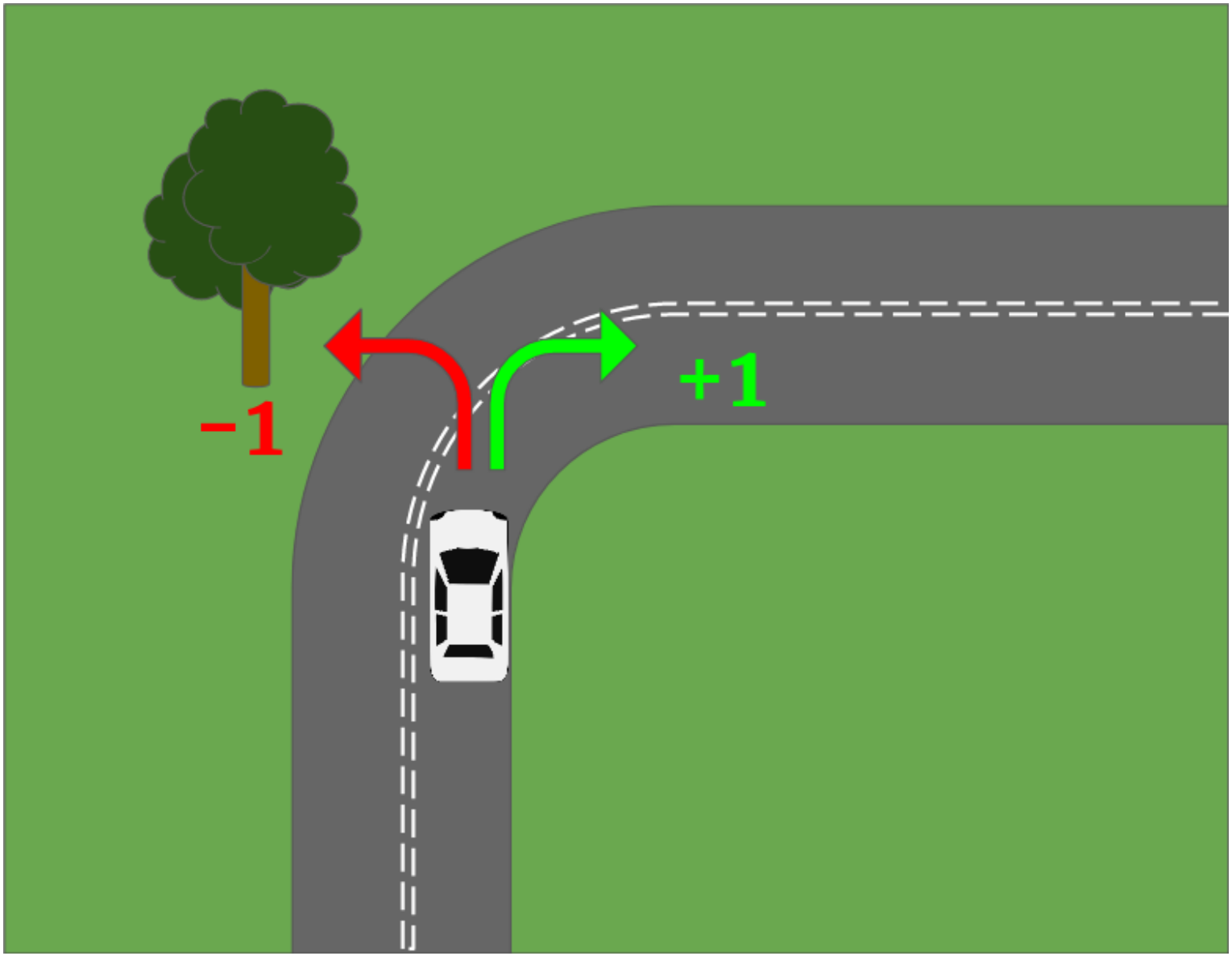
Images (from upper left): https://waymo.com/press/,https://starcraft2.com/pl-pl/media,
https://cloud.google.com/tpu/, https://ai.googleblog.com/2018/06/scalable-deep-reinforcement-learning.html

Agent and Environment

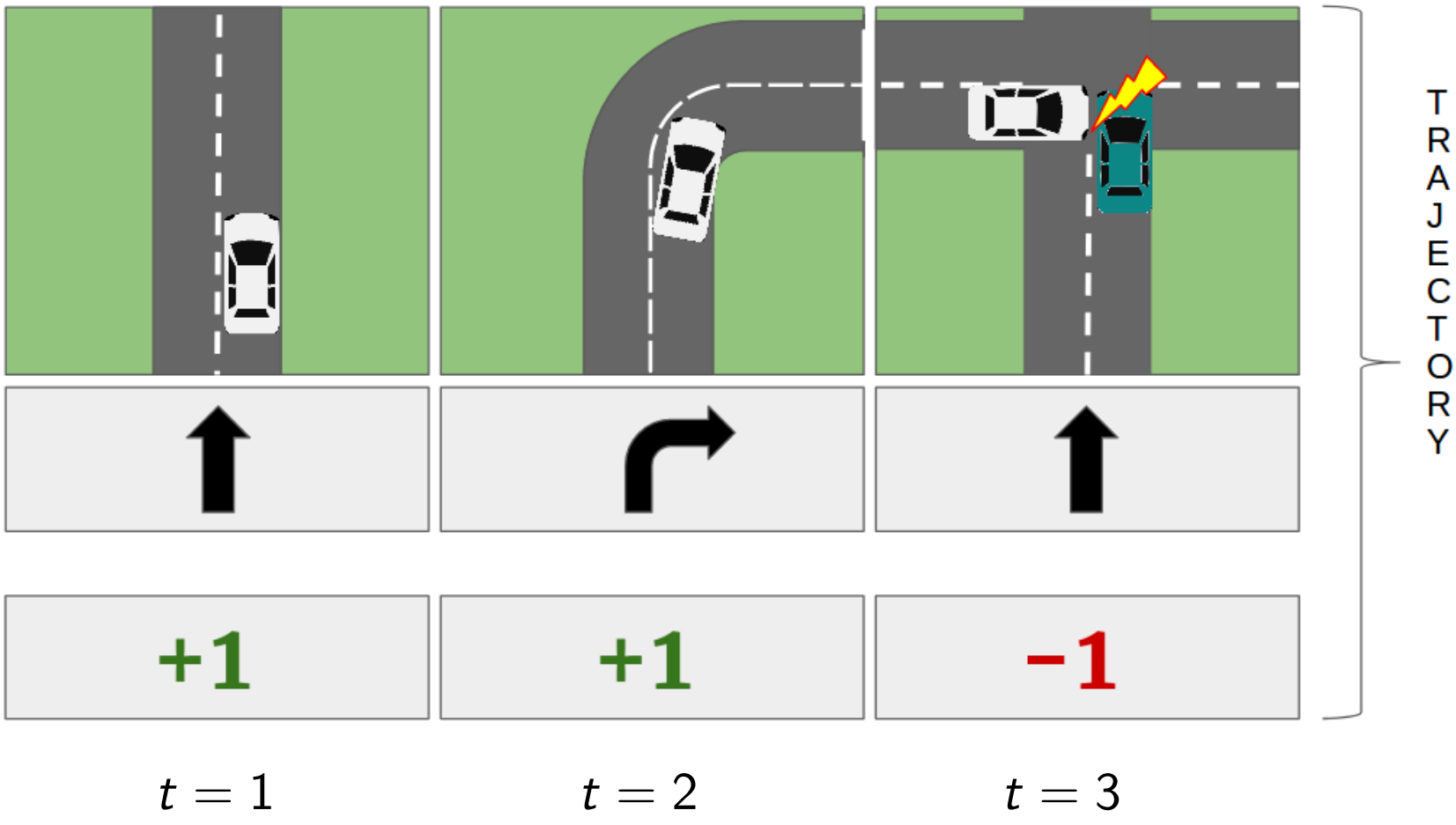
Actions and Observations

Images: https://waymo.com/press/, https://research.mapillary.com/

Rewards



Trajectories



Notation

1. $o \in \mathbb{O}$ is the observation (vector of real numbers), describing what the agent *sees*.
2. $a \in \mathbb{A}$ is the taken action.
3. $r \in \mathbb{R}$ is the reward for performing action a.
4. For simplicity, let us assume that the action space is finite and has A elements

$$\mathbb{A} = \{a^{(1)}, \ldots, a^{(A)}\}$$

Notation

To distinguish between observations, actions and rewards obtained at different (discrete) time t, we write: $o_t \in \mathbb{O}$, $a_t \in \mathbb{A}$, $r_t \in \mathbb{R}$.

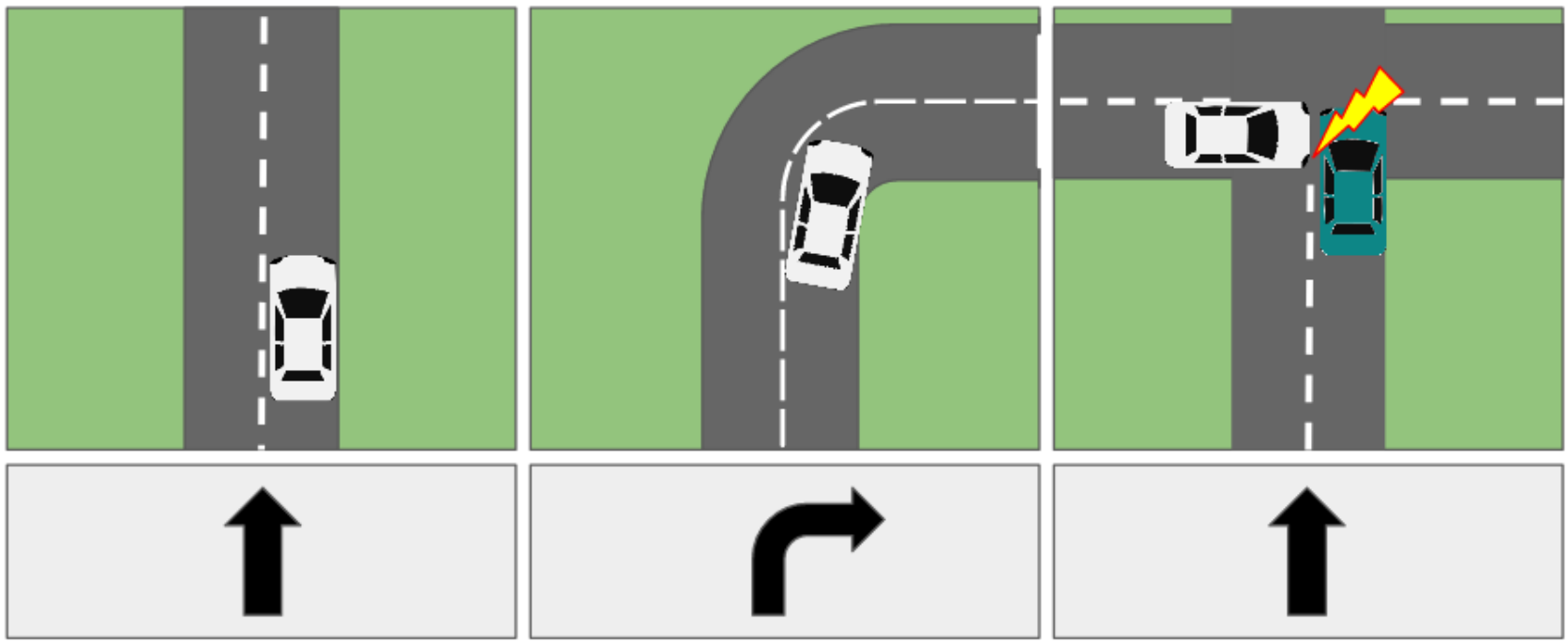
Note the difference between the lower (time) and upper (indicator of the action) indices. In particular:

- $a_3 = a_4$ – action taken at time $t = 3$ is equal to the one taken at time $t = 4$
- $a_4 = a^{(7)}$ – action taken at time $t = 4$ is the seventh possible action (for example: $a^{(7)} =$ turn left).
- $a_7 = a^{(1)}$ – action taken at time $t = 7$ is equal to the first possible action (for example $a^{(1)} =$ go forward). Note that $a_7 \neq a^{(7)}$.

Trajectories

Possible actions:

$\mathbb{A} = \{a^{(1)} = \text{Left} \leftarrow\ ,\ a^{(2)} = \text{Forward} \uparrow\ ,\ a^{(3)} = \text{Right} \rightarrow\}$



$a_1 = a^{(2)}$ $a_2 = a^{(3)}$ $a_3 = a^{(2)}$

Notation – Agent

At time t, the agent is given the observation vector o_t, based on which it performs an action a_t and receives the reward r_t.

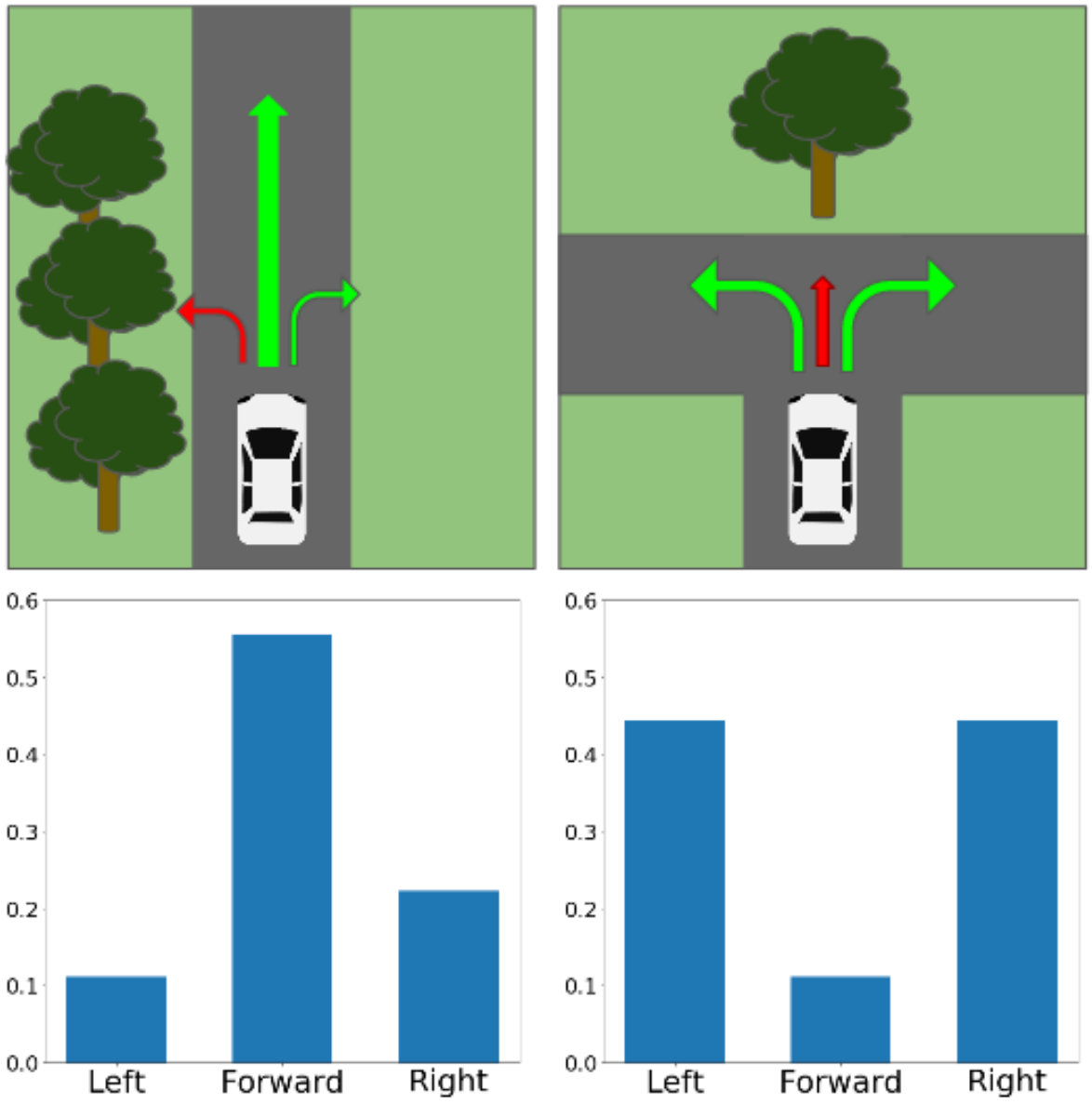
The agent's behaviour for each $o \in \mathbb{O}$ is its *policy*. The agent's policy defines a conditional probability over the possible choices for action a given observation o.
Probability distributions on finite sets can be conveniently represented as vectors, therefore we write:

$$\pi : \mathbb{O} \ni o \mapsto \Big(\pi(a^{(1)} \mid o), \ldots, \pi(a^{(A)} \mid o)\Big) \in \mathbb{R}^A$$

$$0 \leq \pi(a^{(1)} \mid o), \ldots, \pi(a^{(A)} \mid o) \leq 1$$

$$\pi(a^{(1)} \mid o) + \ldots + \pi(a^{(A)} \mid o) = 1$$

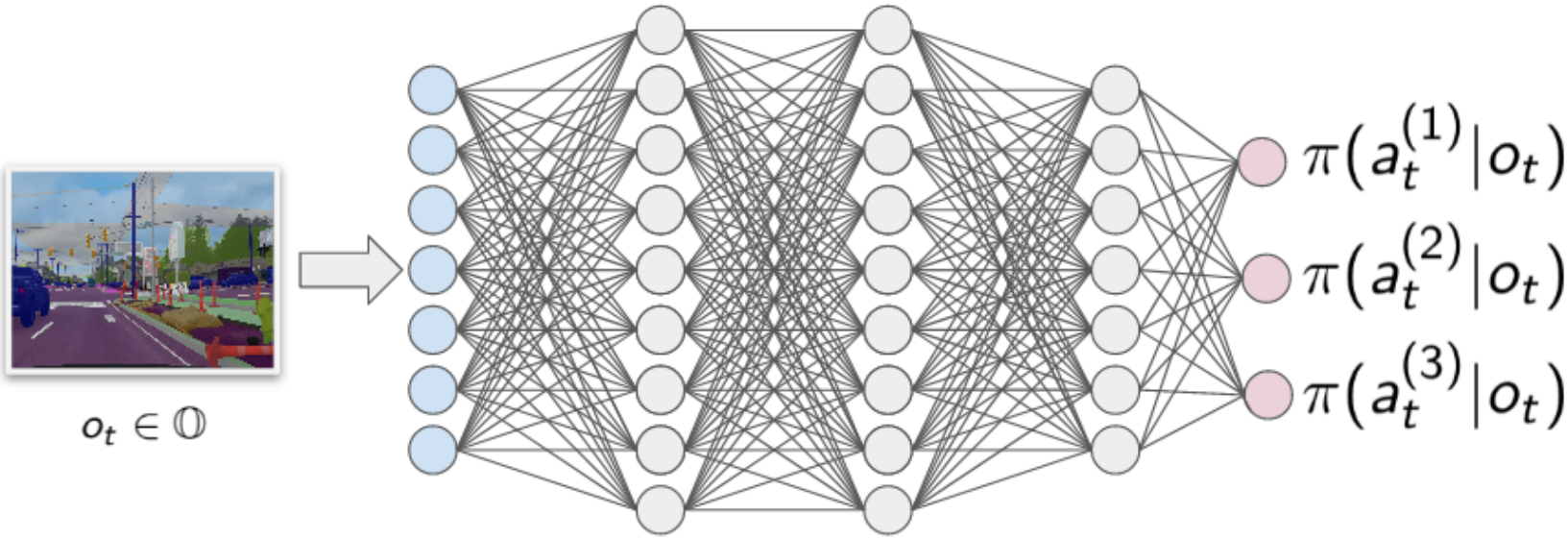
Notation – Agent



$\pi(a_1|o_1)$ $\pi(a_2|o_2)$

Notation – Agent: a Neural Network

The agent may be modeled by a neural network, which defines its policy π_θ with respect to trainable parameters θ:



Note: we need to assign each action its positional number from the set $\{1, 2, \ldots, A\}$, in order to identify the outputs from the network with actions – similar to classification tasks and labels!

Notation – Environment

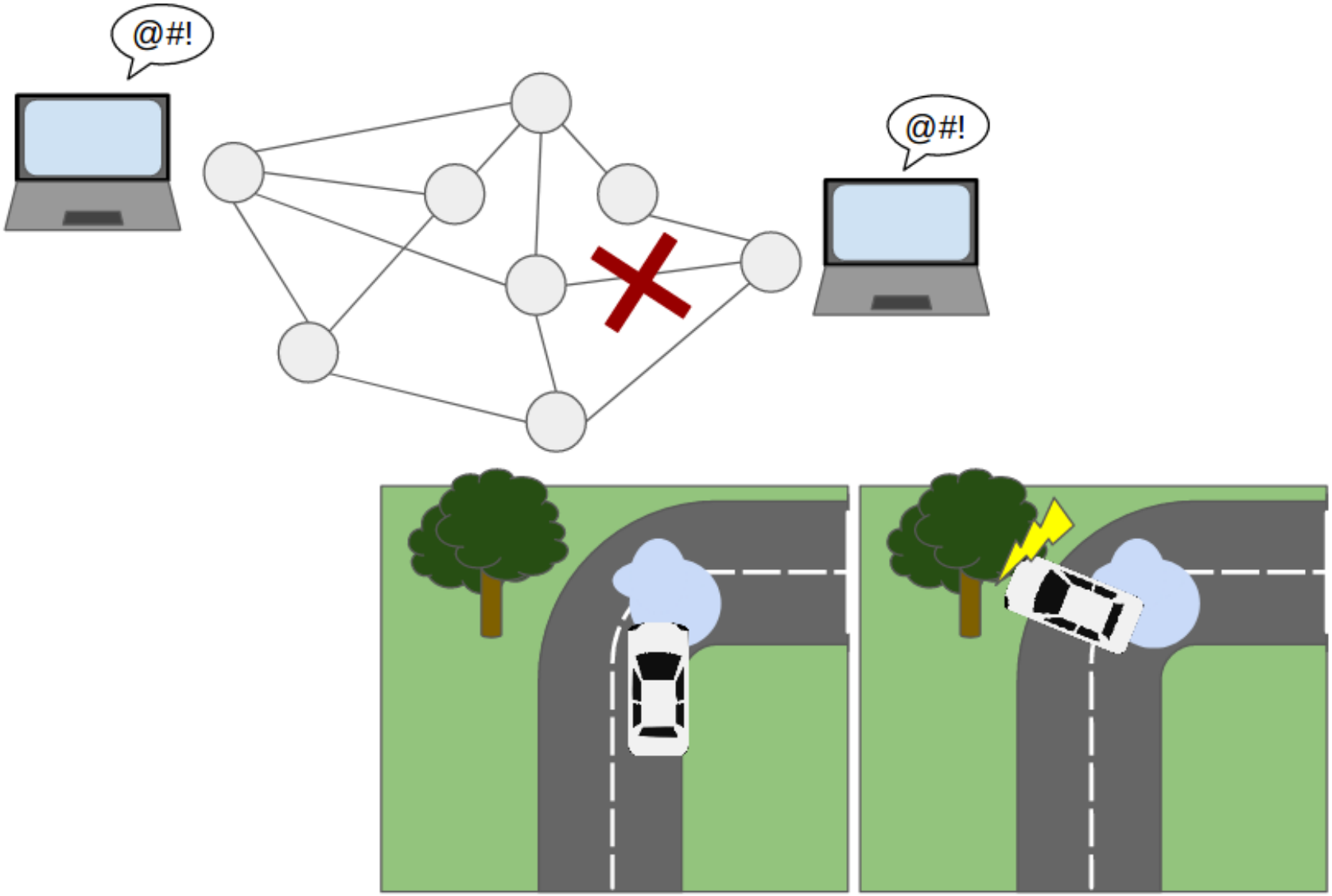
Environment

The environment is the overall model of the world. At any given time step t it is described by a state $s_t \in \mathbb{S}$. After the agent's action, the environment:

1. Emits the rewards for the taken action r_t.
2. Changes the state to s_{t+1}.
3. Emits observations o_{t+1}.

From now on we assume that $o_t = s_t$ for all t. This means that in each step the agent *sees* the full state of the environment. Hereinafter we won't be using the o_t notation anymore.

Environments May Not be Deterministic



Environment – Markov Property

Markov property

The environment changes its state to s_{t+1} accordingly to the *conditional probability distribution* given the current state and the agent action. Note that the future state does not depend on the past, but only on the present – i.e. the state transition satisfies the Markov property:

$$(r_t, s_{t+1}) \sim p(r_t, s_{t+1}|s_t, a_t, r_{t-1}, s_{t-1}, a_{t-1}, \ldots, s_1) = p(r_t, s_{t+1}|s_t, a_t)$$

Notation – Probability Distributions

All probabilities that (even in part) depend on θ will be denoted by π_θ, other by p.

- $p(s_1)$ – initial state (environment)
- $\pi_\theta(a_t \mid s_t)$ – action (agent)
- $p(r_t, s_{t+1} \mid s_t, a_t)$ – state transition (environment)
- $p(r_t, s_{\text{END}} \mid s_t, a_t)$ – end of episode (environment)
- $\pi_\theta(\tau)$ – trajectory (agent + environment)

Notation – Trajectory

A trajectory is a series of pairs of states and taken actions. In this presentation we also include rewards. It's more intuitive and corresponds to the implementation we will use.

Trajectory

$\tau = (s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}})$

The probability of trajectory τ

$\pi_\theta(\tau) = \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}})$

Trajectory – Probability

$$\tau = (s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}})$$

$$\pi_\theta(\tau) = \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}})$$

Trajectory – Probability

$$\tau = (s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}})$$

$$\begin{aligned}\pi_\theta(\tau) &= \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}}) \\ &= \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T, a_T)\ p(r_T, s_{\text{END}} | s_1, a_1, r_1, s_2, \ldots, s_T, a_T)\end{aligned}$$

Trajectory – Probability

$$\tau = (s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}})$$

$$\begin{aligned}
\pi_\theta(\tau) &= \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}}) \\
&= \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T, a_T)\ p(r_T, s_{\text{END}} | s_1, a_1, r_1, s_2, \ldots, s_T, a_T) \\
&\overset{\text{Markov}}{=} \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T, a_T)\ p(r_T, s_{\text{END}} | s_T, a_T)
\end{aligned}$$

Trajectory – Probability

$$\tau = (s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}})$$

$$\begin{aligned}
\pi_\theta(\tau) &= \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}}) \\
&= \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T, a_T)\ p(r_T, s_{\text{END}} | s_1, a_1, r_1, s_2, \ldots, s_T, a_T) \\
&\overset{\text{Markov}}{=} \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T, a_T)\ p(r_T, s_{\text{END}} | s_T, a_T) \\
&= \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T)\ \pi_\theta(a_T | s_{T-1})\ p(r_T, s_{\text{END}} | s_T, a_T)
\end{aligned}$$

Trajectory – Probability

$$\tau = (s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}})$$

$$\begin{aligned}
\pi_\theta(\tau) &= \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}}) \\
&= \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T, a_T)\ p(r_T, s_{\text{END}} | s_1, a_1, r_1, s_2, \ldots, s_T, a_T) \\
&\overset{\text{Markov}}{=} \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T, a_T)\ p(r_T, s_{\text{END}} | s_T, a_T) \\
&= \pi_\theta(s_1, a_1, r_1, s_2, \ldots, s_T)\ \pi_\theta(a_T | s_{T-1})\ p(r_T, s_{\text{END}} | s_T, a_T) \\
&= \ldots \\
&= p(s_1)\ \pi_\theta(a_1 \mid s_1)\ p(r_1, s_2 \mid s_1, a_1) \ldots p(r_T, s_{\text{END}} \mid s_T, a_T)
\end{aligned}$$

Environment – Rewards

For every action a_t the environment returns a reward r_t.

Problem: What is the reward?

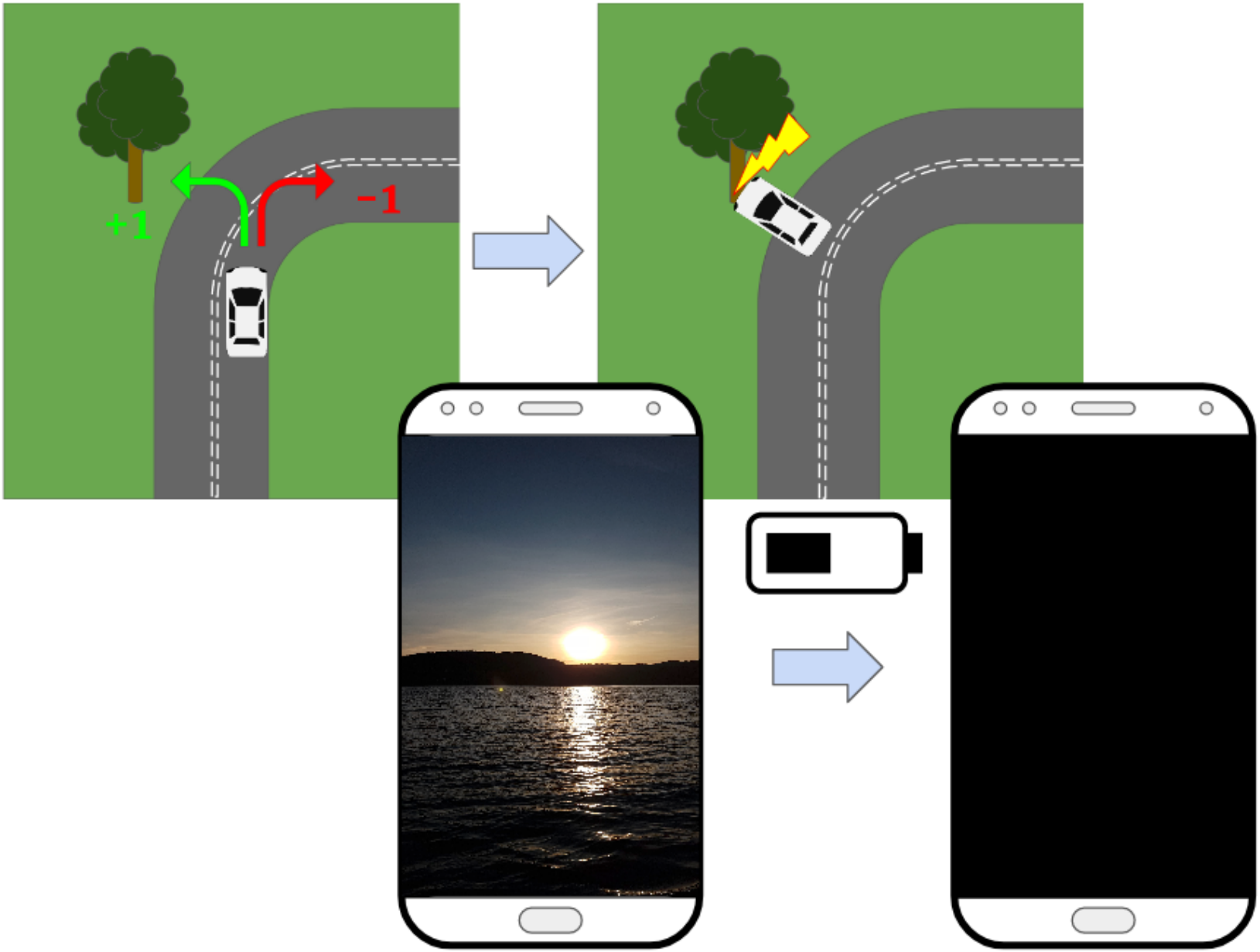
The real (i.e. ideal) reward distribution is usually unknown and task-dependent. In practice, the rewards are defined by the programmer.

While designing the rewards, one has to take into account all possible outcomes of each situation and assign them a real number. It is usually very unclear what that number should be.

Environment – Bad Rewards

One of the key challenges in developing a Reinforcement Learning agent is to correctly design the rewards. Note that bad rewards may lead to wrong or unexpected behaviours.

Environment – Bad Rewards



Return

The return R_t at time t is the sum of all the rewards starting from time t to the end of episode T, obtained by following a given policy π_θ:

$$R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_T = \sum_{j=t}^{T} r_j$$

Therefore the return of trajectory τ is simply:

$$R(\tau) = R(s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}}) = \sum_{j=1}^{T} r_j$$

PART 2: Policy Gradient

Learning Objective

The learning objective is to **maximize** the expected return

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} R(\tau) \approx \frac{1}{N} \sum_{j=1}^{N} R(\tau_j),$$

where $\{\tau_1, \ldots, \tau_N\}$ is the training set of N trajectories generated by policy π_θ.

Gradient

The maximization of the above function (of learnable parameters θ) with gradient-based methods requires the computation of the gradient:

$$\nabla_\theta \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} R(\tau)$$

Then we may update the parameters θ by taking an ascending step in the direction of the gradient.

Gradient

$$\begin{aligned}\nabla_\theta \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} R(\tau) &= \nabla_\theta \int \pi_\theta(\tau) R(\tau) d\tau \\ &= \int \nabla_\theta \pi_\theta(\tau) R(\tau) d\tau \\ &= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} R(\tau)\end{aligned}$$

Gradient

Recall that:

$$\tau = (s_1, a_1, r_1, s_2, \ldots, s_T, a_T, r_T, s_{\text{END}})$$
$$\pi_\theta(\tau) = p(s_1)\ \pi_\theta(a_1 \mid s_1)\ p(r_1, s_2 \mid s_1, a_1) \ldots p(r_T, s_{\text{END}} \mid s_T, a_T)$$

Then:

$$\frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \frac{\nabla_\theta [p(s_1)\ \pi_\theta(a_1 \mid s_1)\ p(r_1, s_2 \mid s_1, a_1) \ldots p(r_T, s_{\text{END}} \mid s_T, a_T)]}{p(s_1)\ \pi_\theta(a_1 \mid s_1)\ p(r_1, s_2 \mid s_1, a_1) \ldots p(r_T, s_{\text{END}} \mid s_T, a_T)}$$

$$= \frac{\nabla_\theta [\pi_\theta(a_1 \mid s_1) \ldots \pi_\theta(a_T \mid s_T)]}{\pi_\theta(a_1 \mid s_1) \ldots \pi_\theta(a_T \mid s_T)}$$

$$= \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)}$$

Gradient

We obtained:

$$\nabla_\theta \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} R(\tau) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} R(\tau) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} R(\tau)$$

Understanding the Gradient of the Expected Return

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} R(\tau)$$

$\pi_\theta(\cdot \mid s_t)$ – output of the policy network on s_t (probability vector)
$\pi_\theta(a_t \mid s_t)$ – The probability of the action a_t that **actually happened** in time t.

$$\pi_\theta(a_t|s_t) = \pi_\theta(a^{(i)}|s_t) = \pi_\theta(\cdot \mid s_t)_i,$$

where $a_t = a^{(i)}$ for some $i \in \{1, \ldots, A\}$.

Understanding the Gradient

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} R(\tau)$$

For a given trajectory $\tau = (s_1, a_1, r_1, \ldots, s_T, a_T, r_T, s_{\text{END}})$, we:

Understanding the Gradient

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} R(\tau)$$

For a given trajectory $\tau = (s_1, a_1, r_1, \ldots, s_T, a_T, r_T, s_{\text{END}})$, we:

- Compute the gradient of the probability of the taken action a_t with respect to parameters θ.

Understanding the Gradient

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} R(\tau)$$

For a given trajectory $\tau = (s_1, a_1, r_1, \ldots, s_T, a_T, r_T, s_{\text{END}})$, we:

- Compute the gradient of the probability of the taken action a_t with respect to parameters θ.
- Multiply it by $R(\tau)$.

Understanding the Gradient

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} R(\tau)$$

For a given trajectory $\tau = (s_1, a_1, r_1, \ldots, s_T, a_T, r_T, s_{\text{END}})$, we:

- Compute the gradient of the probability of the taken action a_t with respect to parameters θ.
- Multiply it by $R(\tau)$.
- Divide it by the value of the probability.

Understanding the Gradient

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} R(\tau)$$

For a given trajectory $\tau = (s_1, a_1, r_1, \ldots, s_T, a_T, r_T, s_{\text{END}})$, we:

- Compute the gradient of the probability of the taken action a_t with respect to parameters θ.
- Multiply it by $R(\tau)$.
- Divide it by the value of the probability.
- Repeat this for all $t = 1, \ldots, T$ and sum the results.

Gradient – $\nabla_\theta \pi_\theta(a_t \mid s_t)$ intuition

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} R(\tau)$$

- The agent in state s_t chose action a_t, by following a policy π_θ.
- We want to update the parameters θ, to obtain a better policy $\pi_{\theta'}$.
- The gradient $\nabla_\theta \pi_\theta(a_t \mid s_t)$ describes the direction of the fastest increase of $\pi_\theta(a_t \mid s_t)$.
- Moving θ in the direction of the gradient results in increasing the $\pi_\theta(a_t \mid s_t)$ for the chosen action a_t and decreasing it for the other actions (as the probability sums to 1).

Gradient – $R(\tau)$ intuition

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} R(\tau)$$

Action a_t contributed to $R(\tau)$, so we want to change the probability of a_t *proportionally* to the return. In particular:

- When $R(\tau) \geq 0$ we want to follow the gradient.
- When $R(\tau) < 0$ we want to decrease $\pi_\theta(a_t \mid s_t)$ – i.e. we want to move in direction opposite to the gradient.

$\nabla_\theta \pi_\theta(a_t \mid s_t) R(\tau)$ means:

"Modify θ to change the probability of a_t proportionally to $R(\tau)$."

Gradient – Dividing by $\pi_\theta(a_t \mid s_t)$

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} R(\tau)$$

Let τ and τ' be two different trajectories. Assume that:

$$R(\tau) = 2 \cdot R(\tau')$$
$$3 \cdot \pi_\theta(\tau) = \pi_\theta(\tau')$$

- The trajectory τ is twice as good as τ', so obviously the agent should prefer τ over τ'.

Gradient – Dividing by $\pi_\theta(a_t \mid s_t)$

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} R(\tau)$$

Let τ and τ' be two different trajectories. Assume that:

$$R(\tau) = 2 \cdot R(\tau')$$

$$3 \cdot \pi_\theta(\tau) = \pi_\theta(\tau')$$

- The trajectory τ is twice as good as τ', so obviously the agent should prefer τ over τ'.
- However τ' happens 3 times more often than τ in the training set, so it has the "effective weight" of $3 \cdot R(\tau')$.

Gradient – Dividing by $\pi_\theta(a_t \mid s_t)$

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} R(\tau)$$

Let τ and τ' be two different trajectories. Assume that:

$$R(\tau) = 2 \cdot R(\tau')$$
$$3 \cdot \pi_\theta(\tau) = \pi_\theta(\tau')$$

- The trajectory τ is twice as good as τ', so obviously the agent should prefer τ over τ'.
- However τ' happens 3 times more often than τ in the training set, so it has the "effective weight" of $3 \cdot R(\tau')$.
- Without the normalization by the probability of the trajectories, the agent would learn to prefer τ' over τ!

Gradient – Summing over t

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} R(\tau)$$

- Take sum over all $t = 1, \ldots, T$ to modify all preferences.
- Note that we do not take the average (i.e. divide by T), because we want trajectories with high $R(\tau)$, even if they are long.

Logarithm of the Gradient

Sometimes it is more desired (for numerical stability) to compute the logarithms (or logits) of the actions probabilities.

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\sum_{t=1}^{T}\frac{\nabla_\theta\pi_\theta(a_t\mid s_t)}{\pi_\theta(a_t\mid s_t)}R(\tau)=\mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\sum_{t=1}^{T}\nabla_\theta\ln\pi_\theta(a_t\mid s_t)\cdot R(\tau)$$

Note that in order to sample the actions we still need to compute the probabilities $\pi_\theta(\cdot\mid s_t)$.

Weighted Gradients

Untill now the gradient of our objective was described by:

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot R(\tau)$$

However, we would like to work with:

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot R_t$$

The difference is that in the second formula we weight the gradients $\nabla_\theta \ln \pi_\theta(a_t \mid s_t)$ only with the rewards obtained **after** the action. We shall later argue that this approach improves the training.

Are we allowed to replace $R(\tau)$ with R_t?

Weighted Gradients – $R(\tau)$

Let's take a closer look at the gradient with $R(\tau)$:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot R(\tau) \\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \Big(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \Big) \cdot \Big(\sum_{t=1}^{T} r_t \Big) \\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t,t'=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot r_{t'}.
\end{aligned}$$

Observe that each gradient $\nabla_\theta \ln \pi_\theta(a_t \mid s_t)$ is weighted by **every** reward of the trajectory.

Weighted Gradients – R_t

Now let's investigate the gradient with R_t:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot R_t \\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \Big(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \Big) \cdot \Big(\sum_{t'=t}^{T} r_{t'} \Big) \\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{1\le t\le t'\le T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot r_{t'}
\end{aligned}$$

Observe that each gradient $\nabla_\theta \ln \pi_\theta(a_t \mid s_t)$ is weighted only by **future** rewards.

Weighted Gradients – R_t

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t,t'=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot r_{t'} \stackrel{?}{=} \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{1\le t\le t'\le T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot r_{t'}$$

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t>t'} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot r_{t'} \stackrel{?}{=} 0$$

The answer is YES. It may be proved that if $t > t'$, then $r_{t'}$, a_t are independent given s_t, and:

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot r_{t'} = 0.$$

Weighted Gradients – R_t

We get that:

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot r_{t'} = 0, \text{ for } t > t'$$

Why is directly excluding the above term from the entire gradient computation helpful?

- The above equality holds on average, but (usually) not on the training set.
- Therefore it only introduces noise.
- Also, leaving those terms is equal to giving $R(\tau)$ as the last reward r_T (with all previous rewards set to 0).
- In consequence, we lose information on when the rewards were obtained.

Practical Notes – Discount Factor

Choose $0 < \gamma < 1$ (usually $0.9 < \gamma < 1$) and replace

$$R_t = r_t + r_{t+1} + r_{t+2} + \ldots + r_T = \sum_{j=t}^{T} r_j$$

with

$$R_t^\gamma = \gamma^0 r_t + \gamma^1 r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{T-t} r_T = \sum_{j=t}^{T} \gamma^{j-t} r_j.$$

Note that:

$$1 = \gamma^0 > \gamma^1 > \gamma^2 > \ldots > \gamma^{T-t}$$

Discount Factor – Intuition

$$R_t^\gamma = \gamma^0 r_t + \gamma^1 r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{T-t} r_T$$

- If $\gamma \to 1$, then $R_t^\gamma \to R_t$
  - We consider all rewards received after time t.
- If $\gamma \to 0$, then $R_t^\gamma \to r_t$
  - We consider only the imminent reward.
- Rewards might be more relevant to recent actions.
  - R_t^γ is a compromise.
  - Early rewards – weighted with ≈ 1.
  - Late rewards – also important, but added with a smaller weight.
- We reduce noise, but introduce error (bias).
  - Still, it's beneficial.

Discount Factor – Horizon

We define the horizon h_γ as:

$$h_\gamma := \frac{1}{1-\gamma}.$$

The horizon indicates how far ahead we look:

- For $\gamma \approx 1$ we have

$$\gamma^{h_\gamma} \approx \frac{1}{e}, \qquad \gamma^{3h_\gamma} \approx 0.05$$

- Examples

$$h_{0.8} = 5, \qquad h_{0.95} = 20, \qquad h_{0.999} = 1000$$

- In practice: rather than choosing γ first decide for the horizon h and then set

$$\gamma = 1 - \frac{1}{h}$$

Practical Notes – Objective Function $J(\theta)$

Let $\{\tau_1, \ldots, \tau_N\}$ is the training set of N trajectories generated by policy π_θ. The approximation of the gradient is:

$$\begin{aligned}
\nabla_\theta \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} R(\tau) &= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta\big(a_t \mid s_t\big) \cdot R_t \\
&\approx \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta\big(a_t \mid s_t\big) \cdot R_t^\gamma \\
&\approx \frac{1}{N} \sum_{j=1}^{N} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta\big(a_t(\tau_j) \mid s_t(\tau_j)\big) \cdot R_t^\gamma(\tau_j)
\end{aligned}$$

Practical notes – Objective function $J(\theta)$

$$\frac{1}{N}\sum_{j=1}^{N}\sum_{t=1}^{T}\nabla_\theta \ln \pi_\theta\big(a_t(\tau_j) \mid s_t(\tau_j)\big)\cdot R_t^\gamma(\tau_j)$$

- Inefficient in practice – we have to separately calculate multiple gradients $\nabla_\theta \ln \pi_\theta\big(a_t \mid s_t\big)$.
- PyTorch and TensorFlow do not like that.
- Observe that we can move the gradient symbol:

$$\nabla_\theta \frac{1}{N}\sum_{j=1}^{N}\sum_{t=1}^{T}\ln \pi_\theta\big(a_t(\tau_j) \mid s_t(\tau_j)\big)\cdot R_t^\gamma(\tau_j) =: \nabla_\theta J(\theta)$$

- Then we only have to calculate $J(\theta)$ and ask PyTorch to compute the gradient and perform the ascent step.

Training loop

1. Sample N trajectories $\tau_1, \ldots, \tau_N$.
2. Calculate the function

$$-J(\theta) = -\frac{1}{N} \sum_{j=1}^{N} \sum_{t=1}^{T} \ln \pi_\theta \big(a_t(\tau_j) \mid s_t(\tau_j) \big) \cdot R_t^\gamma(\tau_j).$$

3. Ask PyTorch to minimize this function by performing gradient descent step.

PART 3: Value Function

PG With Baseline

Environment E

Let E be an environment, with discrete actions $\mathbb{A} = \{a^{(1)}, a^{(2)}, a^{(3)}\}$ and states $s \in \mathbb{S}$. Let s_t be the state visited at time t and R_t the corresponding return.

Environment E'

Now consider an environment E', which is basically a copy of E. The only difference is that, for every trajectory, after reaching the ending state s_{END} we subtract 10 from the trajectories return.

PG With Baseline

Let R_t, r_t, R'_t, r'_t be the returns and rewards for state s_t in environments E and E', respectively. Note that:

- The rewards stay the same: $r'_t = r_t$.
- The **returns** change:

$$R'_t = R_t - 10.$$

- The optimal policy π_{optim} in E is also an optimal policy in E' and *vice versa*.

PG With Baseline

- On **average** there is no difference in how the agent learns in those two environments:

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\frac{\nabla_\theta\pi_\theta(\tau)}{\pi_\theta(\tau)}R'_t] = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\frac{\nabla_\theta\pi_\theta(\tau)}{\pi_\theta(\tau)}(R_t - 10)],$$

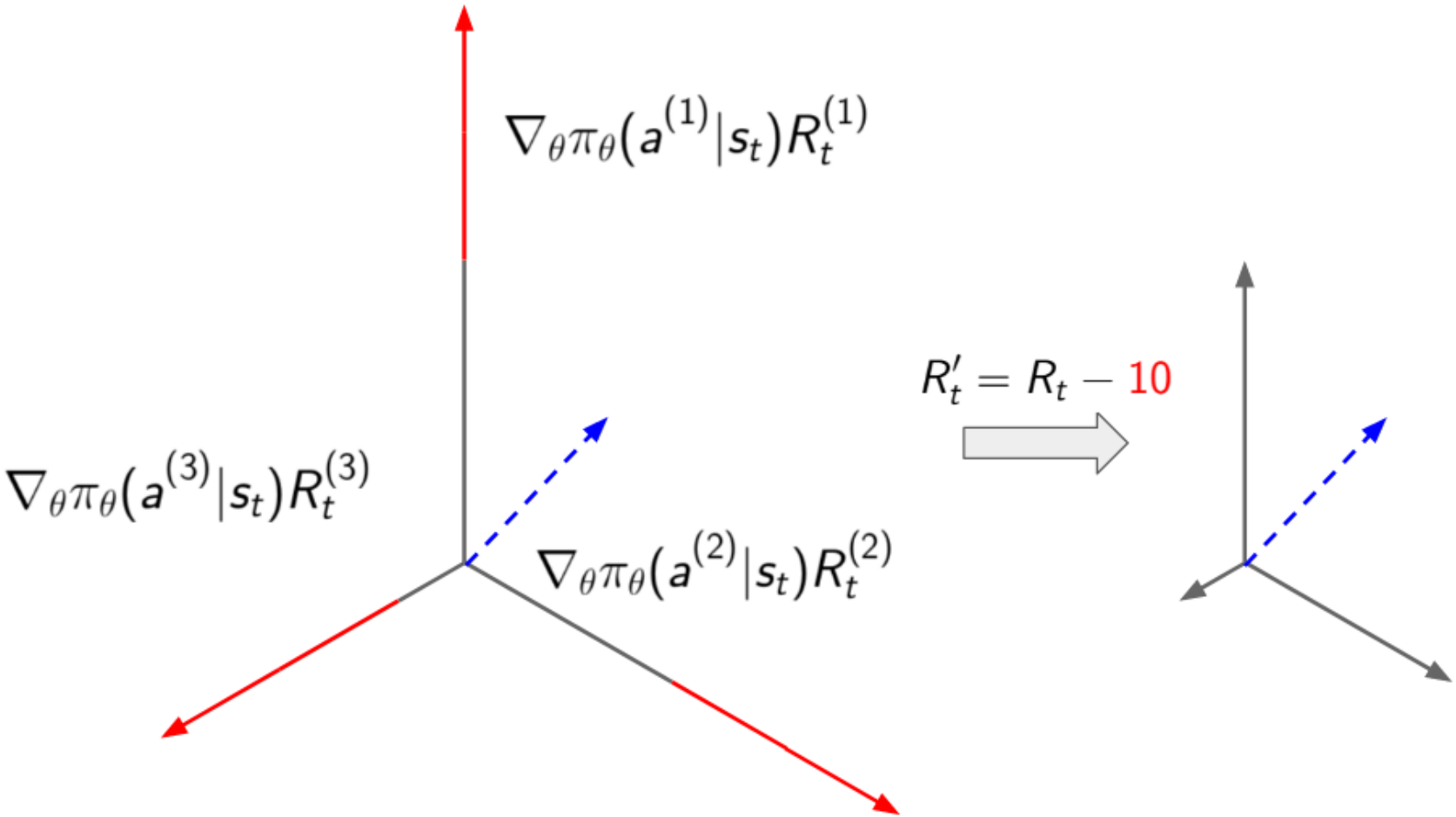
since:

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\frac{\nabla_\theta\pi_\theta(\tau)}{\pi_\theta(\tau)}10] = 10 \cdot \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\frac{\nabla_\theta\pi_\theta(\tau)}{\pi_\theta(\tau)}] =$$

$$10 \cdot \int \nabla_\theta\pi_\theta(\tau)d\tau = 10\nabla_\theta 1 = 0$$

- In **practice** however, learning in E' may be more stable.

PG With Baseline



Baseline Function

It turns out that we may subtract from the returns any **baseline** function $b : \mathbb{S} \to \mathbb{R}$

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot R_t =$$

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot \big(R_t - b(s_t)\big)$$

It may be proved that:

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot b(s_t) = 0$$

The proof is similar to the simple constant case $b(s_t) = 10$.

Baseline Function

Note that we are changing only the **returns**, NOT the rewards.

- Increasing all rewards for a given state s_t would results in agent visiting this state more often. In consequence we would change the objective – BAD IDEA!
- Modifying the return is a trick that always works.
- Note that we can select a different baseline value for each state. For example we can decrease the returns by 10 in one state and increase by 20 in another.
- Unbiased noise reduction – recall replacing $R(\tau)$ with R_t.

How to define a good baseline?

Value Function

The value function $V_{\pi_\theta} : \mathbb{S} \to \mathbb{R}$ is defined as:

$$V_{\pi_\theta}(s) := \mathop{\mathbb{E}}_{\tau \sim \pi_\theta, s_1(\tau)=s} R(\tau),$$

where $s_1(\tau)$ denotes the first state of τ.

The value function depends on *policy* in a complicated way:

- It also involves the environment.
- It cannot be calculated.

Value Function – Markov Property!

The value of a state does not change in time:

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta, s_1(\tau)=s} R(\tau) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta, s_t(\tau)=s} R_t(\tau)$$

Intuition:

- Agent currently is in state s, time t is irrelevant.
- How much rewards in total do we expect from now on?

We can learn the value function from sampled trajectories.

Value Network

We will approximate the true value function V_{π_θ} with a neural network V_ψ with weights ψ:

$$V_\psi \approx V_{\pi_\theta}.$$

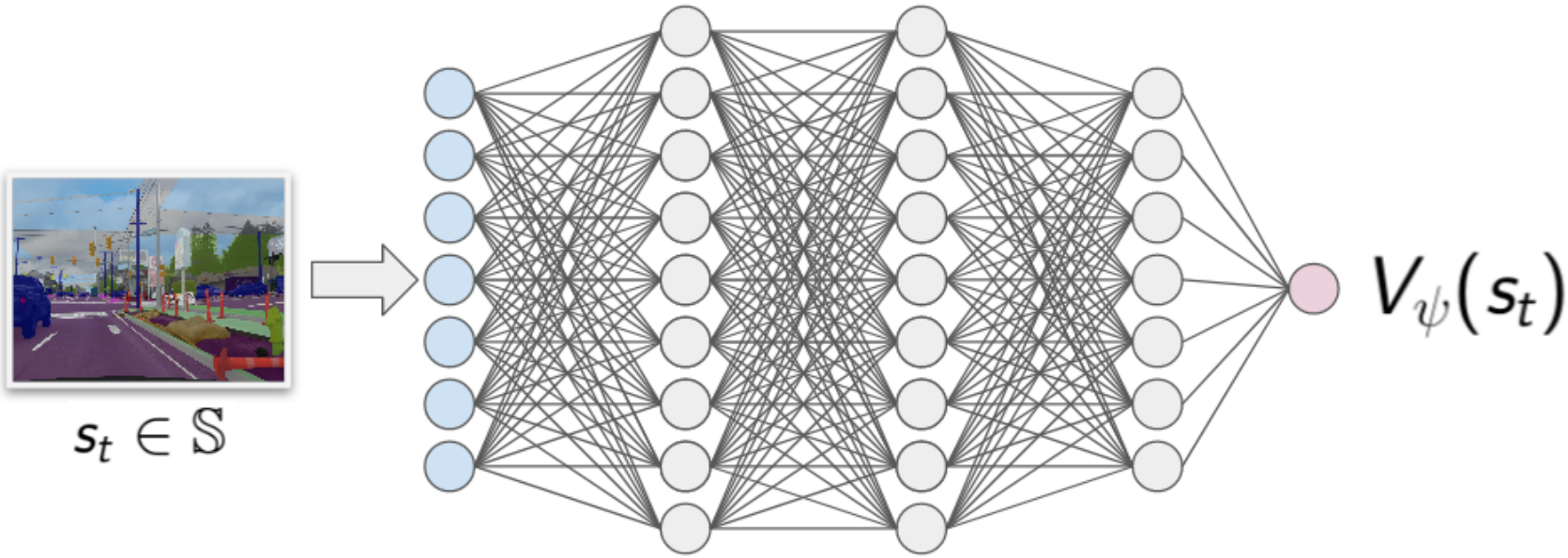
The training set consists of pairs (s_t, R_t):

- May be formed from a single trajectory:

$$\{(s_1, R_1), \ldots, (s_T, R_T)\}$$

- We can also combine multiple trajectories.

Value Network



Value Network: Objective

Learning the value network is a regression task. The cost function of the value network is the standard *Mean Squared Error*.

Recall that the value function depends on the policy! Therefore the value network must be tuned after every update of π_θ:

- Different π_θ means different target V_{π_θ}.
- In contrast to Policy Gradient, we can do multiple updates of ψ without resampling the training set.

Advantage Learning

Let's define the advantage at time t as:

$$A_t := R_t - V_\psi(s_t)$$

Replacing the return R_t in the gradient estimation with A_t is equivalent to using V_ψ as a **baseline**:

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot R_t =$$

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot (R_t - V_\psi(s_t)) =$$

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot A_t$$

$$A_t := R_t - V_\psi(s_t)$$

Note that:

- R_t is the absolute gain. A_t is the gain relative to the average.
- The advantage gives a better estimator for finite number of trajectories.
- Remember – any baseline is correct!
  - V_ψ doesn't even have to be a good approximation of V_{π_θ}.
  - However better approximation gives more stability!

Training Loop – Policy Gradient with Baseline

1. Sample N trajectories $\tau_1, \ldots, \tau_N$.
2. Improve the parameters of our value network by minimizing the loss (apply multiple gradient descent steps):

$$\frac{1}{M} \sum_{j=1}^{N} \sum_{t=1}^{T} \left(R_t(\tau_j) - V_\psi(s_t(\tau_j)) \right)^2,$$

   where M is the total number of samples in all trajectories.
3. Calculate the function

$$J(\theta) = \frac{1}{N} \sum_{j=1}^{N} \sum_{t=1}^{T} \ln \pi_\theta \left(a_t(\tau_j) \mid s_t(\tau_j) \right) \cdot [R_t(\tau_j) - V_\psi(s_t(\tau_j))],$$

4. Ask PyTorch to minimize the function $-J(\theta)$ by performing gradient descent step.

PART 4: Proximal Policy Optimization

Remarks on Some Common Problems

Sampling trajectories

Collecting new trajectories is the most expensive part of RL!

- We would like to reuse trajectories for multiple updates of the agent.
- Recall that this is allowed for V_ψ, but not for π_θ.

Distribution of trajectories changes during the learning

The update of the agent changes the distribution of the trajectories. For example:

- Assume (s_t, a_t) has a positive return. Then after the update $\pi_\theta(a_t \mid s_t)$ may increase.
- This means that the pairs (s_t, a_t) are now **undersampled** in the training set!

Question

Is it possible to make multiple updates of π_θ without resampling trajectories?

Proximal Policy Optimization

Sketch of the algorithm:

1. Copy π_θ to $\pi_{\theta\mathrm{OLD}}$.
2. Sample several trajectories using $\pi_{\theta\mathrm{OLD}}$.
3. Make *multiple updates* of π_θ.

PROBLEM: Until now we assumed $\pi_\theta = \pi_{\theta\mathrm{OLD}}$ and only *one update* of π_θ per training set. Therefore the equations need to be modified!

Proximal Policy Optimization

Probability that (s_t, a_t) will occur in the training set is a product of:

- $\pi_\theta(a_t \mid s_t)$ – probability of taking action a_t after observing s_t.
  - Explicitly given by the policy network.
- $\pi_\theta(s_t)$ – probability of visiting s_t.
  - Depends on the environment and $\pi_\theta(a_{t'} \mid s_{t'})$, where $t' < t$.
  - Different actions *lead to* different states later on.

Proximal Policy Optimization

The Proximal Policy Optimization Assumption

As long as:

$$1 - \epsilon < \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta\text{OLD}}(a_t \mid s_t)} < 1 + \epsilon,$$

for some (small, but significant) ϵ, e.g. $\epsilon = 0.2$, we may assume that $\pi_\theta(s_t)$ is close to $\pi_{\theta\text{OLD}}(s_t)$:

$$\pi_\theta(s_t) \approx \pi_{\theta\text{OLD}}(s_t)$$

Therefore, the following is a good approximation:

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} \approx \mathop{\mathbb{E}}_{\tau\sim\pi_{\theta\text{OLD}}} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_{\theta\text{OLD}}(a_t \mid s_t)}$$

PPO Assumption – Some Intuition

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} \approx \mathop{\mathbb{E}}_{\tau\sim\pi_{\theta\text{OLD}}} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_{\theta\text{OLD}}(a_t \mid s_t)}$$

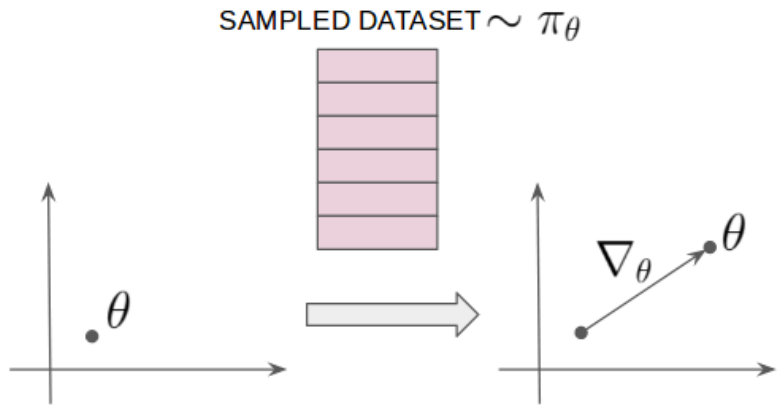
- The left part is what we want.
- However, we must use $\mathbb{E}_{\tau\sim\pi_{\theta\text{OLD}}}$, because training set was sampled from $\pi_{\theta\text{OLD}}$.
- Recall that we always divide by the probability of being in the training set. Therefore we also substitute the denominator. This is an important step, since difference between $\pi_\theta(a_t \mid s_t)$ and $\pi_{\theta\text{OLD}}(a_t \mid s_t)$ is relevant ($\epsilon \not\approx 0$).

PPO Assumption – Some Intuition

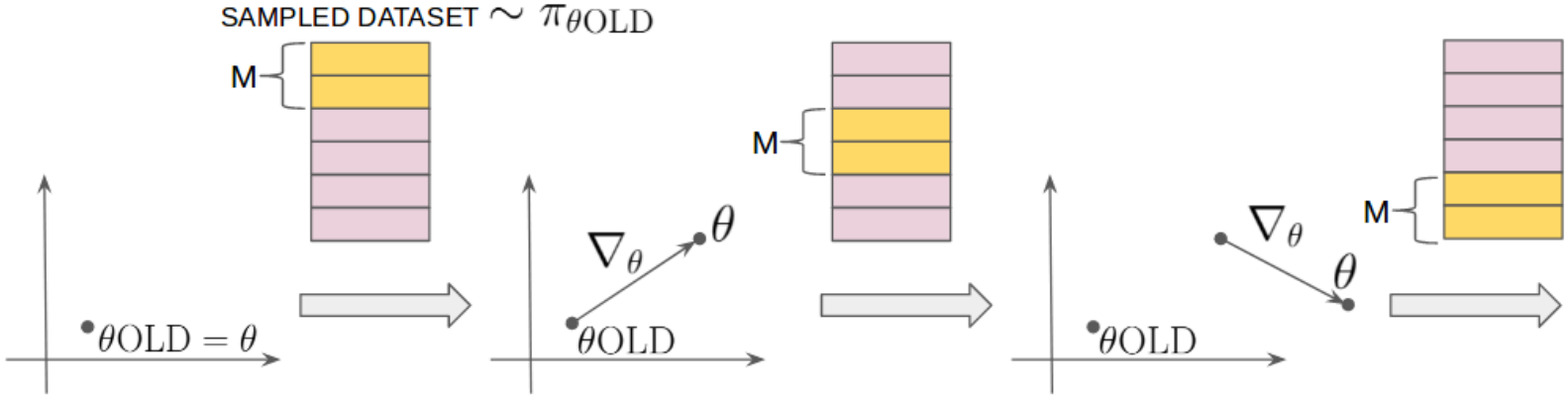
$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} \approx \mathop{\mathbb{E}}_{\tau\sim\pi_{\theta\mathrm{OLD}}} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_{\theta\mathrm{OLD}}(a_t \mid s_t)}$$

- We keep the numerator, because we update π_θ, not $\pi_{\theta\mathrm{OLD}}$.
- We do not have equality because $\pi_\theta(s_t) \neq \pi_{\theta\mathrm{OLD}}(s_t)$. However we assume that this part of error is *relatively* small.
  - We could introduce the necessary corrections, but then the estimator would be very unstable.
  - Analogically to the discounting trick – we introduce error, but gain stability!

Policy Gradient



Proximal Policy Optimization



Proximal Policy Optimization

Let's define ρ_t as:

$$\rho_t := \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta\mathrm{OLD}}(a_t \mid s_t)}.$$

This simplifies the notation (denominator is constant wrt. to θ):

$$\nabla_\theta \rho_t = \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_{\theta\mathrm{OLD}}(a_t \mid s_t)}.$$

For every t we try to keep

$$1 - \epsilon < \rho_t < 1 + \epsilon,$$

so that we can use the formula:

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot A_t \approx \mathop{\mathbb{E}}_{\tau\sim\pi_{\theta\mathrm{OLD}}} \sum_{t=1}^{T} \nabla_\theta \rho_t \cdot A_t$$

Why $\rho_t \neq 1$?

$$\rho_t = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta\text{OLD}}(a_t \mid s_t)}$$

- initially $\pi_\theta(a_t \mid s_t) = \pi_{\theta\text{OLD}}(a_t \mid s_t)$, so $\rho_t = 1$.
- $\pi_{\theta\text{OLD}}(a_t \mid s_t)$ is constant, thus ρ_t changes with $\pi_\theta(a_t \mid s_t)$.

$$\mathop{\mathbb{E}}_{\tau \sim \pi_{\theta\text{OLD}}} \sum_{t=1}^{T} \nabla_\theta \rho_t \cdot A_t$$

- if $A_t > 0$, then ρ_t will usually increase (decrease with $A_t < 0$)
  - Every time it's included in training batch, gradient will try to increase $\pi_\theta(a_t \mid s_t)$.
  - Caution! other training samples will have unknown influence on this particular ρ_t.

Surrogate Objective

For convenience we define a surrogate objective

$$\mathcal{J} := \mathop{\mathbb{E}}_{\tau \sim \pi_{\theta \mathrm{OLD}}} \sum_{t=1}^{T} \rho_t A_t,$$

and we have

$$\nabla_\theta \mathcal{J} = \nabla_\theta \mathop{\mathbb{E}}_{\tau \sim \pi_{\theta \mathrm{OLD}}} \sum_{t=1}^{T} \rho_t \cdot A_t = \mathop{\mathbb{E}}_{\tau \sim \pi_{\theta \mathrm{OLD}}} \sum_{t=1}^{T} \nabla_\theta \rho_t \cdot A_t.$$

Note that we can simply swap ∇_θ and $\mathbb{E}_{\tau \sim \pi_{\theta \mathrm{OLD}}}$, because $\pi_{\theta \mathrm{OLD}}$ does **not** depend on θ.

How to Enforce $1 - \epsilon < \rho_t < 1 + \epsilon$?

- We always keep a copy of $\pi_{\theta\text{OLD}}(a_t \mid s_t)$, so ρ_t may be calculated if needed.
- If ρ_t goes out of bounds, we temporarily remove the corresponding training example from the training set.
- Important change: if $A_t > 0$, then we only enforce $\rho_t < 1 + \epsilon$
  - We *do not* want to exclude examples that went out of range by accident.
  - Analogically for $A_t < 0$ enforce only $\rho_t > 1 - \epsilon$.
- Turns out if we modify our surrogate objective a bit, then we can achieve what we want:

$$\tilde{\mathcal{J}} := \mathop{\mathbb{E}}_{\tau \sim \pi_{\theta\text{OLD}}} \sum_{t=1}^{T} \Big[\min \left(\rho_t \cdot A_t, \mathit{clip}(\rho_t, 1 - \epsilon, 1 + \epsilon) \cdot A_t \right) \Big]$$

The Surrogate Objective – Intuition

$$\min\left(\rho_t \cdot A_t, \mathit{clip}(\rho_t, 1-\epsilon, 1+\epsilon) \cdot A_t\right)$$

Additional intuition taken from the original PPO paper:

- The first part, $\rho_t \cdot A_t$, comes from $\mathcal{J}$.
- "The second term (...) removes the incentive of moving ρ_t outside of the interval $[1-\epsilon, 1+\epsilon]$."
- "We take the minimum of the clipped and unclipped objective, so the final objective is a lower bound (i.e., a pessimistic bound) on the unclipped objective."

Exercise

Prove (on paper) that this formula works as intended

$$\min\left(\rho_t \cdot A_t, \mathit{clip}(\rho_t, 1-\epsilon, 1+\epsilon) \cdot A_t\right).$$

Check all 6 possible cases:

- A_t may be positive or negative.
- ρ_t may be $< 1-\epsilon$, $> 1+\epsilon$, or ≈ 1.

Hints:

- Make two plots of the formula wrt. to ρ_t for positive and negative A_t.
- If ρ_t is out of bounds, then gradient of the second part is equal to zero.

Training loop – PPO

1. Sample N trajectories $\tau_1, \ldots, \tau_N$ from π_θ.
2. Assign $\pi_{\theta\text{OLD}} := \pi_\theta$.
3. Loop over batches $\mathcal{B}$ from gathered trajectories:
   - 3.1 Improve the parameters of our value network by minimizing the loss:
   $$\frac{1}{|\mathcal{B}|} \sum_{(s_i, R_i) \in \mathcal{B}} \Big[R_i - V_\psi(s_i) \Big]^2.$$
   - 3.2 Calculate the function:
   $$\tilde{J}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{(s_i, a_i, R_i) \in \mathcal{B}} \Big[\min \left(\rho_i \cdot A_i, \mathit{clip}(\rho_i, 1-\epsilon, 1+\epsilon) \cdot A_i \right) \Big].$$
   - 3.3 Ask PyTorch to minimize the function $-\tilde{J}(\theta)$ by performing gradient descent step.

Further readings

1. "Reinforcement Learning: An Introduction"
   (Sutton, Barto)
2. UCL Course on RL
   (Silver, videos)
3. Deep Reinforcement Learning course at UC Berkeley
   (Levine, videos)
4. Proximal Policy Optimization
   (Schulman, Wolski, Dhariwal, Radford, Klimov)
5. OpenAI Resources page
6. The Policy of Truth
   (Recht, "policy gradient is nothing more than random search dressed up in mathematical symbols and lingo")