Deep Reinforcement Learning

372.2.5910

Ben-Gurion University of the Negev

Lecture Notes

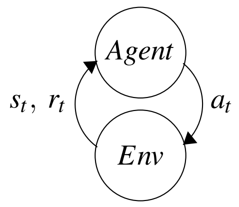
WRITTEN BY: Hadar Sharvit
ALSO AVAILABLE ON GITHUB
CONTACT ME AT: Hadar.Sharvit1@mail.Huji.ac.il
BASED ON: Lectures given by Gilad Katz
CHAPTERS & BOOK COVER BY: Rohit Choudhari/Unsplash

Contents

1. Hello world

R This chapter is provided as a preliminary, and is not part of the course. It is 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

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

R 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.

2. RL basics

2.1 Motivation

Current malware detection platforms often deploy an ensemble of detectors to increase overall performance. This approach creates lots of redundancy, as in most cases one detector is enough, and it of course computationally expensive and time consuming, compared to one detector.
We can come up with a simple improvement - query a subset of detectors and decide based on their classification if more detectors are needed. If we observe our approach under the scope of classification, it may very well be the case that training a model w.r.t to every set of detectors is needed, as we cannot evaluate the performance of a subset of detectors without actually learning how they performed. As this is computationally hard for large detectors, this is not a preferred approach. Instead, we can use RL:

Suppose we use four detectors, and our agent takes as input the vector $[-1,-1,-1,-1] \in \mathbb{R}^4$, which is considered an initial state. The agent will choose a set of detectors/detector configurations, and a classification measurement of either "*malicious*" or "*benign*" will be taken. The decisions of the agent will be based on a reward mechanism that takes uses the values of TP, FP (correctly classified the content as "*malicious*" or "*benign*") and FP, FN (incorrectly classified to "*malicious*" or "*benign*"). We will "punish" using $C(t)$, which is a function that depends on the time it took for the detectors to run. We can see that regardless of how many detectors were used, if we are right - the

Exp. #	TP	TN	FP	FN
1	1	1	-C(t)	-C(t)
2	10	10	-C(t)	-C(t)
3	100	100	-C(t)	-C(t)

Table 2.1: Three suggested reward mechanisms for a malware detection platform

reward is constant (experiment with $1, 10, 100$). On the other hand, if we were wrong, we subtract

$C(t)$ which increases with the time t that has passed. As it is now "painful" to use more detectors, the reward incentives our model to only use more detectors if that addition translated to higher success rates. As our model efficiently scans through the state space, it is able to outperform (at least conceptually) the suggested "check-all" classification approach that was previously introduced.

2.2 When to use RL?

Not all cases adhere to the framework of RL. Here are some rules

- our data should be in the form of trajectories - a set of distinguishable states $s_0 \to s_1 \to \cdots \to s_N$
- need to make a sequence of related decisions - if every decision is independent, like classifying 4 images of cats-vs-dogs, don't use RL.
- the actions we perform result in some feedback - either positive or negative
- Tasks that are more suitable include both learning and planning - we learn our environment and plan an optimal behaviour
- the data is not provided apriori, and its' distribution changes with our action choice. This means that we must make sure our agent effectively explores the entire data, and does not settle in some small subspace.

2.3 Markov Decision Processes (MDP)

consists of the following

- States: that make up the environment. could be either discrete or continuous.
- Actions: by taking an action we transition from one state to another. In a deterministic process, we have that $P(s'|s,a) = 1$
- Reward: taking action $a \in A$ from state $s \in S$ results in reward $R(s,a) \in \mathbb{R}$

In finite MDP, s,a,r are all final.

2.3.1 The Markov Property

the distribution over the future states depends only on the present state and action

$$Pr[s_{t+1}|s_1,a_1,s_2,a_2,...,s_t,a_t] = Pr[s_{t+1}|s_t,a_t] \tag{2.1}$$

In poker, for example, the markovian property does not hold as a player's current hand depends on the actions and hands of previous hands and/or players. A traffic light, on the other hand, is completely markovian as it is based on deterministic rules.
Using the markovian property, we can define the probability to reach a certain state s' with a certain reward r, as

$$Pr[s',r|s,a] \equiv Pr[s_{t+1} = s', r_{t+1} = r | s_t = s, a_t = a] \tag{2.2}$$

The probabilities induced by all event in S and R make up a probability space, hence

$$\forall s \in S \ \forall a \in A : \sum_{s' \in S} \sum_{r \in R} Pr[s',r|s,a] = 1 \tag{2.3}$$

The expected reward for state-action pairs, namely, what should we anticipate (in terms of reward) when performing the action a from the state s is

$$r(s,a) \equiv \mathbb{E}[r_{t+1}|s_t = s, a_t = a] = \sum_{r \in R} r \sum_{s' \in S} Pr[s',r|s,a] \tag{2.4}$$

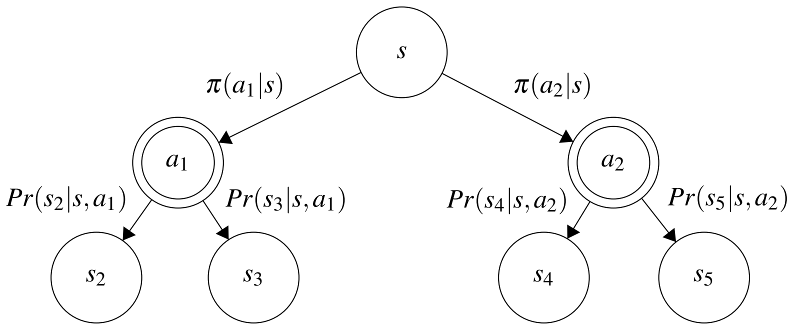
where notice that the probability for a specific reward is the sum over all states, given the specific r (hence the sum over $s' \in S$).
We can also phrase our reward in terms of state-action-next state triplets, namely, what should we anticipate (in terms of reward) when performing the action a that takes us from state s to state s'

$$r(s,a,s') \equiv \mathbb{E}[r_{t+1}|s_t = s, a_t = a, s_{t+1} = s'] = \sum_{r\in R} r \frac{Pr[s',r|s,a]}{Pr[s'|s,a]} \tag{2.5}$$

Where we can think of the probability fraction as the number of events that reach s' (from s after performing a) and provide reward r, out of all the event that reach s' (from s after performing a) given any reward.
MDPs are very flexible

- both states and actions could be either abstract (s="*sad*", s="*happy*", a=*"take a nap"*) or well-defined (like s=sensor readings, or a=turn on a switch).
- the time intervals may not be constant (some transitions are slow while other are fast)
- the setting of an MDP does not need to be an exact copy of the real-world model. For example, a set of sensors may be enough to describe a robotic arm, even though there are many more aspects that the arm is made up of (that are not as relevant).

R At this point is may be helpful to look at what is known as the "*Backup Diagram*", That describes how the states are propagated based on the actions chosen by π and the probabilities induces by the environment.



We can write, for example, the probability to transition from s to s_5 by performing an action a_s as $P(s_5|s,a_2) = \pi(a_2|s)Pr(s_5|s,a_2)$. In general term,, the probability to move to any state by performing any action is the probability to take some action a and sum all probabilities of states s' reachable from s using a, and finally sum over all such actions

$$Pr(\text{reach any state using any action}|s) = \sum_{a\in A} \pi(a|s) \sum_{s'\in S} Pr(s'|s,a) \tag{2.6}$$

Equivalently, we can write the probability to reach any state using any action and receiving any reward

$$Pr(\text{any state any action any reward}|s) = \sum_{a\in A} \pi(a|s) \sum_{s'\in S}\sum_{r\in R} Pr(s',r|s,a) \tag{2.7}$$

2.4 Goals and Rewards

The agent's goal is to maximize the expected return.
For a finite horizon of size T, the undiscounted sum of rewards is

$$G_t = R_{t+1} + R_{t+2} + \ldots + R_{t+T} = \sum_{k=0}^{T} R_{t+k+1} \tag{2.8}$$

For an infinite horizon, we add a discount factor, as otherwise infinite sum would result in an agent that does not really care for the reward mechanism. As previously stated, the intuition here is reward now $>$ future reward.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.9}$$

where $\gamma \in (0,1)$

2.5 Policies,Value-function and Q-function

The policy π defines out strategy, namely, what we choose to do at every step

$$\pi(s,a) = Pr[a_t = a | s_t = s] \tag{2.10}$$

The goal of π is to maximize the value function, which is the cumulative expected return of following π starting from some state s

$$V_\pi(s) = \mathbb{E}_\pi[G_t | s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | s_t = s\right] \tag{2.11}$$

Notice that the expectation is w.r.t π, meaning that after the initial state $s_t = s$, the next states are fully determined by π. We can think of V_π as a measurement of "*How good is* π?", as intuitively, we can choose the policy that provides us the maximal expected return.
We can also define the Q-function, which is the same as V except the fact that we start from s *and* perform an initial action a (that may or may not be one of π's options)

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | s_t = s, a_t = a\right] \tag{2.12}$$

2.6 The Bellman equation

Theorem 2.6.1 The value function can be written as

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t | s_t = s] \\ &= \sum_{a \in A} \pi(a|s) \sum_{s' \in S} \sum_{r \in R} Pr[s', r | s, a][r + \gamma V_\pi(s')] \\ &= \mathbb{E}_{a \in A}[\mathbb{E}_{s',r}[r + \gamma V_\pi(s')]] \end{aligned} \tag{2.13}$$

Proof. [1] The reward and time t can be rewritten as

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2.14}$$

Therefore the can re-write the Value-function as

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | s_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} | s_t = s] + \gamma \mathbb{E}_\pi[G_{t+1} | s_t = s] \end{aligned} \tag{2.15}$$

Focusing on the second term, we will use the law of iterated expectation

$$\mathbb{E}[Y|X=x] = \mathbb{E}[\mathbb{E}[Y|X=x, Z=z]|X=x]$$

with $Y = G_{t+1}, X = S_t, x = s, Z = S_{t+1}$ and $z = s'$, hence

$$\begin{aligned} \mathbb{E}_\pi[G_{t+1} | s_t = s] &= \mathbb{E}[\mathbb{E}[G_{t+1} | S_t = s, S_{t+1} = s']|S_t = s] \\ &= \mathbb{E}[\mathbb{E}[G_{t+1} | S_{t+1} = s']|S_t = s] \\ &= \mathbb{E}[V_\pi(S_{t+1} = s')|S_t = t] \end{aligned} \tag{2.16}$$

In the 2_{nd} transition we removed the inner condition for $S_t = s$ as $G_{t+1} = R_{t+2} + \gamma R_{t+3} + ...$ does not depend on S_t. This is the case as every reward term R_{t+i} is only a function of the current state and action, so since R_t is not present, no term in G_{t+1} is related to S_t (only to S_{t+1}, S_{t+2}...). In the last transition we use the fact that the inner $\mathbb{E}$ term is nothing but the value function for $t \leftarrow t+1$.
Substituting all to 2.15 we have

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[R_{t+1} | s_t = s] + \gamma \mathbb{E}_\pi[G_{t+1} | s_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} | s_t = s] + \gamma \mathbb{E}[V_\pi(S_{t+1} = s')|S_t = t] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1} = s')|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s' \in S} \sum_{r \in R} Pr(s', r|s, a) \left[r(s, a, s') + \gamma V_\pi(S_{t+1} = s') \right] \end{aligned} \tag{2.17}$$

R_{t+1} describes the reward obtained when moving from s to s' using a, so it can be written as $r(s,a,s')$. Furthermore, the expectation $\mathbb{E}_\pi$ is w.r.t to the states, actions and rewards induced by π (so the probability associated with every term is the one introduced in 2.7), and by summing over $r \in R$ we also indicate the fact that the transition $s \rightarrow s'$ could be rewarded with multiple different rewards (more than one option is plausible).

■

The bellman optimality equation is the bellman equation for the optimal Value-function

$$V^*(s) = \max_a \mathbb{E}[r(s,a) + \gamma V^*(s')] \tag{2.18}$$

[1] https://stats.stackexchange.com/questions/243384/deriving-bellmans-equation-in-reinforcement-learning

Theorem 2.6.2 The Q-function can be written as

$$
\begin{aligned}
Q_\pi(s,a) &= \sum_{s',r}[r(s,a)+\gamma\sum_{a'}Q_\pi(s',a')] \\
&= \mathbb{E}_{s',r}\left[r(s,a)+\gamma\mathbb{E}_{a'}[Q^\pi(s',a')]\right]
\end{aligned}
\tag{2.19}
$$

Proof. Not included ■

The bellman optimality equation is the bellman equation for the optimal Q-function

$$Q^*(s,a) = \mathbb{E}[r(s,a)+\gamma\max_{a'}Q^*(s',a)] \tag{2.20}$$

2.7 Policy Iteration

Is a method of finding an optimal policy by performing two steps - evaluation and improvement. After a random initialization of both π and V (Step I), we evaluate the Value-function given some policy π (Step II)[2]. We do this by constantly sampling our environment and updating the value function for every state respectively. The loop stops when the change in value function (for all s) is smaller than some tolerance ε. We use the notation $\pi(a|s)$ to indicate the probability $Pr(\pi(s)=a)$.

Next, in step III, we observe the current Value function and choose an action that maximizes it. This will be considered our new returned action for every single state

Finally, we combine policy evaluation and policy improvement in an iterative process. More specifically, we evaluate V and improve π until a fixed point is reached (for all s, $\pi(s)$ has not changed, possibly up to some margin δ)[3]

[2]The convergence of PE is the result of the Policy evaluation convergence theory. see HERE for more info

[3]I highly recommend checking out THIS implementation, by Denny Britz

Algorithm 1 Policy Iteration

```
Require: tolerance $\varepsilon > 0$ and an MDP $\langle S, A, R, Pr : S \times R \to \mathbb{R}, \gamma \rangle$
  $V, \pi \leftarrow$ Random Initialization $\in \mathbb{R}^{|S|}$        ▷ Step I: Initialization

  while True do                                                          ▷ Step II: Policy Evaluation
      $\Delta \leftarrow 0$
      for $s \in S$ do
          $v \leftarrow V(s)$
          $V(s) \leftarrow \sum_{a \in A} \pi(a|s) \sum_{s' \in S} \sum_{r \in R} Pr[s', r|s, a][r + \gamma V(s')]$    ▷ using thrm. 2.6.1
          $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
      end for
      if $\Delta < \varepsilon$ then break
      end if
  end while

  policy_stable←True                                                     ▷ Step III: Policy Improvement
  for $s \in S$ do
      old_$\pi \leftarrow \pi(s)$
      $\pi(s) \leftarrow \operatorname{argmax}_{a \in A} \sum_{s' \in S} \sum_{r \in R} Pr[s', r|s, a][r + \gamma V_k(s')]$
      if old_$\pi \neq \pi(s)$ then policy_stable←False
      end if
  end for

  if policy_stable then return $V, \pi$
  else go to step II
  end if
```