

Deep RL course

Deep RL team

September 2022

- 1 Introduction
- 2 Reminders of Dynamic Programming
 - Markov Decision Processes and Policies
 - Returns and goal of Reinforcement Learning
 - Bellman equations and computing V and Q functions
- 3 Deep Q-Networks
- 4 Deep Policy Gradient Algorithms
 - Motivation and principle
 - Reinforce
 - Reducing variance with baselines
 - Using finite trajectories with bootstrapping
- 5 Regularisation in Deep Reinforcement Learning
- 6 Off-Policy Reinforcement Learning

Introduction

What is RL?

A general paradigm for **learning** via **trial and error**, when an agent **interacts** with an environment and seeks to **maximize a reward signal over time**.

When should I use RL?

Extremely general framework in its formulation.

Pros:

- Most problems can be formalized as RL problems.
- Interesting paradigm when very little hypothesis can be made about the problems being tackled.

Cons:

- Makes very little hypothesis about the underlying problem.
- Nearly always applicable, rarely the best solution.

What is Deep Reinforcement Learning?

Adaptation of the reinforcement learning framework to make it amenable to the use of deep learning based function approximators in order to be able to tackle complex environments.

Successes of deep reinforcement learning

Applied to various domains

- Board games,
- Video games,
- Robotics, continuous control.

Key ingredient: possibility to simulate the environment at a (relatively) low cost.

So far has been state of the art in:

- BackGammon, Atari games, Go, Chess, Shogi, StarCraft II, Dota 2....

What we will do in this course?

What this course is about:

- Having a broad understanding of the theoretical foundations of Reinforcement Learning.
- Obtaining insights on how to bridge the gap between standard/tabular reinforcement learning and deep reinforcement learning.
- Implementing various simple deep reinforcement learning algorithms, and being confident in ones ability to implement more complex ones.

What this course is not about:

- Providing an in depth mathematical foundation for reinforcement learning.
- Studying the convergence properties of well mathematically founded reinforcement learning algorithms.
- Providing theoretical guarantees for the use of deep learning in conjunction with reinforcement learning.

Program of the lectures

- Reminders of Markov Decision Processes and Dynamic Programming.
- Deep Q Networks as an instantiation of Approximate Value Iteration.
- Deep Policy Gradient algorithms.
- Regularization in deep reinforcement learning.
- Off-policy corrections.

Program of the practicals

- Introduction to JAX and Haiku.
- Tabular reinforcement learning.
- Implementing DQN.
- Implementing deep policy gradient based algorithms and their extensions.

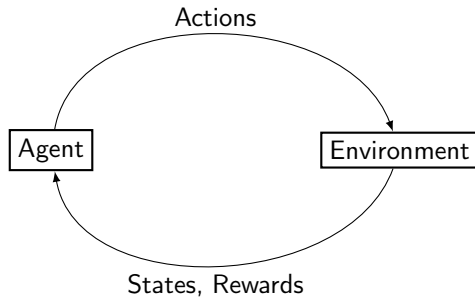
Do Not Hesitate to

- Ask questions.
- Ask for more explanation.
- Ask for examples.
- The course should be interactive and is made for you! Do not hesitate to post comments/remarks/questions on slack.

Course evaluation

- We will provide you with an environment of our choice. You will be asked to reach the highest possible score on this environment with a limited compute budget. You will be asked to hand out a report alongside the your code. Grades will be based on your rank and the quality of your report.
- Groups or 4 or 5 (depending on the number of students).

Reminders of Dynamic Programming



Markov Decision Process (MDP)

Formally, a finite Markov Decision Process (MDP) is a tuple $(\mathcal{X}, \mathcal{A}, p, \rho, r, \gamma)$ with:

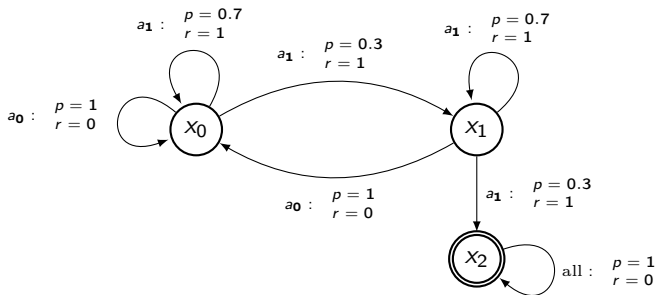
- \mathcal{X} : the finite state space of cardinal $|\mathcal{X}|$.
- \mathcal{A} : the finite action space of cardinal $|\mathcal{A}|$.
- p : the dynamics is a transition kernel from $\mathcal{X} \times \mathcal{A}$ to \mathcal{X} . $p(y|x, a)$ is the probability to transition to state $y \in \mathcal{X}$ by doing action $a \in \mathcal{A}$ in state $x \in \mathcal{X}$.
- $\rho \in \mathbb{R}^{\mathcal{X}}$: the initial state distribution and $\rho(x)$ is the probability to start in state x .
- r : the reward function is a mapping from state-action to real number, $r \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$. The reward $r(x, a)$ represents the local benefit of doing action $a \in \mathcal{A}$ in state $x \in \mathcal{X}$.
- $\gamma \in [0, 1)$: the scalar discount factor.

Intuitions on MDPs

- **Markov assumption:** Next state and reward only depend on the current state and action. This implies:
 - **States** are fully representative of the environment. There are no hidden variables.
 - States provide all the information required to select actions. No need to remember the past.
- **Sequential decision making:** Action not only influence immediate reward, but also future state. Need to *plan* several steps ahead.

Example: GarbageBot

- **Actions:** 0. charge battery 1. look for garbage
- **States:** 0. high battery 1. low battery 2. battery depleted



Reinforcement Learning Goal

The goal of Reinforcement Learning algorithms is to find a **policy** (a behaviour) that maximises a notion of **cumulative return**.

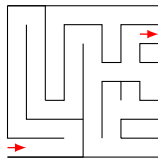
Goal of the GarbageBot

For the GarbageBot, the goal is to find a **policy** that allows it to look for garbage as long as possible without depleting the battery.

Maze environment

Fixed maze template, agent starts in the bottom left corner and must go to the top right corner.

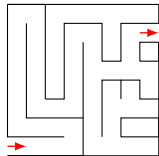
- **Reward:** +1 when getting out of the maze.
- **Actions:** One of the four directions.
- **States:** Position in the maze



Maze environment

Fixed maze template, agent starts in the bottom left corner and must go to the top right corner.

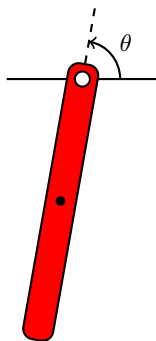
- **Reward:** +1 when getting out of the maze.
- **Actions:** One of the four directions.
- **States:** Position in the maze



Yes

Pendulum environment:

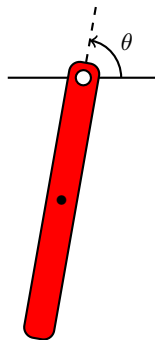
- **Reward:** +1 if pendulum is upward, 0 otherwise.
- **Actions:** Torque +1 or -1 (i.e. angular acceleration to the left or to the right).
- **States:** Angle θ and angular velocity $\dot{\theta}$.



Quiz time: Is this an MDP? (2)

Pendulum environment:

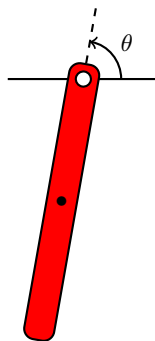
- **Reward:** +1 if pendulum is upward, 0 otherwise.
- **Actions:** Torque +1 or -1 (i.e. angular acceleration to the left or to the right).
- **States:** Angle θ and angular velocity $\dot{\theta}$.



Yes

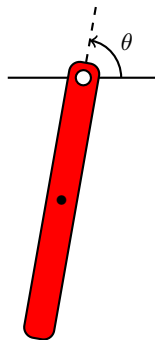
Pendulum environment:

- **Reward:** +1 if pendulum is upward, 0 otherwise.
- **Actions:** Torque +1 or -1 (i.e. angular acceleration to the left or to the right).
- **States:** Angle θ .



Pendulum environment:

- **Reward:** +1 if pendulum is upward, 0 otherwise.
- **Actions:** Torque +1 or -1 (i.e. angular acceleration to the left or to the right).
- **States:** Angle θ .



No

Policy

We consider *Markovian* stochastic policies π , i.e. state dependent probability distributions over actions. $\pi(a|x)$ denotes the probability of choosing action $a \in \mathcal{A}$ in state $x \in \mathcal{X}$. The set of markovian stochastic policies is denoted by Π .

Intuitions on the policy

- A policy defines what action a the agent will perform in state x .
- The action selection process can incorporate some randomness: a policy associate a **distribution over actions** $\pi(\cdot|x)$ to each state x .
- Deterministic policies can be recovered when $\forall x, \exists a$ s.t. $\pi(a|x) = 1$.

Trajectories generated by a policy π

We define recursively trajectories $\tau^\pi = (X_t, A_t, R_t)_{t \in \mathbb{N}^+}$ as:

$$\begin{aligned} X_0 &\sim \rho \\ \forall t \geq 0, \quad A_t &\sim \pi(\cdot | X_t), \quad R_t = r(X_t, A_t), \quad X_{t+1} \sim p(\cdot | X_t, A_t). \end{aligned}$$

$\mathcal{T}(\pi)$ is the distribution of such trajectories, and $\mathcal{T}(x, \pi)$ (resp. $\mathcal{T}(x, a, \pi)$) is the distribution of trajectories conditioned on $X_0 = x$ (resp. $X_0 = x, A_0 = a$).

Intuitions on the trajectory

Informally, a trajectory is obtained by:

- Picking an initial state according to distribution ρ .
- Iterating:
 - Picking an action according to the policy.
 - Drawing a new state and a reward, according to the transition kernel of the environment.

A simple custom policy for our GarbageBot

- Charges the battery as soon as they are low: $\pi(a_1|x_0) = 1$
- Otherwise looks for garbage: $\pi(a_0|x_1) = 1$ and $\pi(a_0|x_2) = 1$

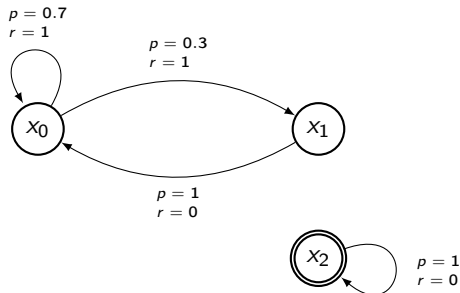


Figure: Markov Reward Process (Markov Chain) associated to policy π on the GarbageBot environment.

StarCraft II against a fixed opponent

- **Reward:** +1 for winning, -1 for losing.
- **Actions:** Keyboard and mouse controls.
- **States:** The state of the screen (+ audio).

Quizz time: Is this an MDP? (4)

StarCraft II against a fixed opponent

- **Reward:** +1 for winning, -1 for losing.
- **Actions:** Keyboard and mouse controls.
- **States:** The state of the screen (+ audio).

No

Chess against a fixed opponent

- **Reward:** +1 for winning, -1 for losing.
- **Actions:** Pieces standard moves.
- **States:** The state of the board.
- **Opponent policy:** $\pi_{\text{fixed}}(\text{move}|\text{board state})$.

Chess against a fixed opponent

- **Reward:** +1 for winning, -1 for losing.
- **Actions:** Pieces standard moves.
- **States:** The state of the board.
- **Opponent policy:** $\pi_{\text{fixed}}(\text{move}|\text{board state})$.

Yes

Chess against Magnus Carlsen

- **Reward:** $+1$ for winning, -1 for losing.
- **Actions:** Pieces standard moves.
- **States:** The state of the board.
- **Opponent policy:** What Magnus Carlsen would play.

Chess against Magnus Carlsen

- **Reward:** +1 for winning, -1 for losing.
- **Actions:** Pieces standard moves.
- **States:** The state of the board.
- **Opponent policy:** What Magnus Carlsen would play.

Probably not

Cumulative Return

We define the cumulative return of a trajectory, $Z(\tau^\pi)$ as the discounted sum of rewards over the trajectory:

$$Z(\tau^\pi) = \sum_{t \geq 0} \gamma^t R_t.$$

Intuitions on Cumulative Return

- The cumulative return takes into account the rewards obtained at **all steps of the trajectory**.
- The discount factor $\gamma < 1$ quantifies a preference for the present:
 - A unit of reward at time 0 is worth $\frac{1}{\gamma}$ units of reward at time 1.
 - With a discount of γ , future rewards will typically become negligible after $O(\frac{1}{1-\gamma})$ timesteps.
 - Undiscounted returns can be defined, but are harder to work with.

Important Remark

Cumulative returns Z^π are stochastic (they are random variables), due to **randomness in both the policy and the environment**. We want to define (and optimize) averages of these quantities over trajectories.

Expected Cumulative Returns

$$J^\pi = \mathbb{E}_{\tau^\pi \sim \mathcal{T}(\pi)} [Z(\tau^\pi)],$$

$$V^\pi(x) = \mathbb{E}_{\tau^\pi \sim \mathcal{T}(x, \pi)} [Z(\tau^\pi)],$$

$$Q^\pi(x, a) = \mathbb{E}_{\tau^\pi \sim \mathcal{T}(x, a, \pi)} [Z(\tau^\pi)],$$

$$V^\pi(x) = \mathbb{E}_{a \sim \pi(\cdot|x)} [Q^\pi(x, a)] \text{ and } J^\pi = \mathbb{E}_{x \sim \rho} [V^\pi(x)].$$

More Intuitions on the Expected Returns, Value and Q-functions

- J^π is the average cumulative returns over all trajectories. It provides a single number to attest for **how good a policy performs**.
- $V^\pi(x)$, the value (or V-)function, is the average cumulative returns over all trajectories that start in state x . It provides an indication of **how good one state is**, compared to another, **under the current policy**. It can also be used to **compare the performance of two policies**, starting in the same initial state x .
- $Q^\pi(x, a)$, the Q-function, is the average cumulative returns over all trajectories that start with action a being performed in state x . It provides an indication of **how good one action is in one state**, compared to another, **under the current policy**.

Formal Definition of The RL Goal

The goal of a reinforcement learning algorithm, in a given MDP, is to find an **optimal policy** $\pi^* \in \Pi$, such that for all $x \in \mathcal{X}$, and for any policy π , $V^{\pi^*}(x) \geq V^{\pi}(x)$.

Remark on Optimal Policy.

An optimal policy is thus simply a **policy that performs better than any other policy**, for all initial state x .

Question: How do we compute the optimal policy?

Bellman equation on V

The value function verifies the following **Bellman** equation:

$$V^\pi(x) = \sum_{a \in \mathcal{A}} \pi(a|x) r(x, a) + \gamma \sum_{y \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(a|x) p(y|x, a) V^\pi(y)$$

What the Bellman equation means

The expected return can be decomposed into two parts:

- **An instantaneous part**, how much rewards am I going to receive at the next step, following the policy.
- **A future part**. This future part looks at all states that are reachable from x , computes the V function on each of those states (i.e. the expected return one will get from this state), and averages all those values.

Bellman equation on V: Sketch of Proof

$$\begin{aligned}
 V^\pi(x) &= \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r(X_t, A_t) | X_0 = x\right] \\
 &= \mathbb{E}[r(X_0, A_0) + \sum_{t \geq 1} \gamma^t r(X_t, A_t) | X_0 = x] \\
 &= \mathbb{E}_{A_0 \sim \pi(\cdot | x)}[r(x, A_0)] + \gamma \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r(X_{t+1}, A_{t+1}) | X_0 = x\right] \\
 &= \mathbb{E}_{a \sim \pi(\cdot | x)}[r(x, a)] + \gamma \mathbb{E}\left[\mathbb{E}\left[\sum_{t \geq 0} \gamma^t r(X_{t+1}, A_{t+1}) | X_1\right] | X_0 = x\right] \\
 &= \mathbb{E}_{a \sim \pi(\cdot | x)}[r(x, a)] + \gamma \mathbb{E}[V^\pi(X_1) | X_0 = x] \\
 &= \sum_a \pi(a|x) r(x, a) + \gamma \sum_{y \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(a|x) p(y|x, a) V^\pi(y).
 \end{aligned}$$

Bellman equation on Q

The Q function similarly verifies an **equivalent Bellman equations**:

$$Q^\pi(x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{X}} \sum_{b \in \mathcal{A}} p(y|x, a) \pi(b|y) Q^\pi(y, b).$$

Affine Property of the Bellman equations

One can remark that $V^\pi(x)$ and $Q^\pi(x, a)$ can be expressed as affine functions of $V^\pi(x)$ and $Q^\pi(x, a)$ respectively. Therefore it is possible to express them with matricial notations when the state and action spaces are finite.

Matricial Notations for V^π

Let us define $r^\pi(x) = \sum_{a \in \mathcal{A}} \pi(a|x)r(x, a)$ and $P_{\mathcal{X}}^\pi$ the stochastic square matrix of size $|\mathcal{X}|$ such that:

$$P_{\mathcal{X}}^\pi(x, y) = \sum_{a \in \mathcal{A}} p(y|x, a)\pi(a|x).$$

Then:

$$V^\pi = r^\pi + \gamma P_{\mathcal{X}}^\pi V^\pi,$$

where r^π and V^π are seen as column vectors.

Matricial Notations for Q^π

Let us define $P_{\mathcal{X} \times \mathcal{A}}^\pi$ the stochastic square matrix of size $|\mathcal{X}||\mathcal{A}|$ such that:

$$P_{\mathcal{X} \times \mathcal{A}}^\pi((x, a), (y, b)) = p(y|x, a)\pi(b|y).$$

Then:

$$Q^\pi = r + \gamma P_{\mathcal{X} \times \mathcal{A}}^\pi Q^\pi$$

Exercise on the GarbageBot

Using the custom policy:

- Charges the battery as soon as they are low: $\pi(a_1|x_0) = 1$
- Otherwise looks for garbage: $\pi(a_0|x_1) = 1$ and $\pi(a_0|x_2) = 1$

compute the following quantities:

- Q^π
- V^π

What we know

- V^π verifies a Bellman equation.
- Q^π verifies a Bellman equation.

What we will show

- V^π is the unique function that verifies the Bellman equation.
- We will propose the optimality Bellman equation with unique solution V^* .
- The function V^* dominates all the V^π .
- There is a set of policies such that $V^\pi = V^*$.

What do we need

- Evaluation Operator.
- Optimality Operator.

Evaluation Operator

The evaluation operator $B^\pi \in \mathbb{R}^{\mathcal{X} \times \mathcal{X}}$ is defined as follows:

$$B^\pi[V](x) = \sum_{a \in \mathcal{A}} \pi(a|x) \left[r(x, a) + \gamma \sum_{y \in \mathcal{X}} p(y|x, a) V(y) \right].$$

Remark

$$B^\pi[V] = r^\pi + \gamma P_{\mathcal{X}}^\pi V.$$

Optimality Operator

The optimality operator $B^* \in \mathbb{R}^{\mathcal{X} \times \mathcal{X}}$ is defined as follows:

$$B^*[V](x) = \max_{a \in \mathcal{A}} \left[r(x, a) + \gamma \sum_{y \in \mathcal{X}} p(y|x, a) V(y) \right].$$

Evaluation Operator

The evaluation operator $T^\pi \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}} \rightarrow \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$ is defined as follows:

$$T^\pi[Q](x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{X}} p(y|x, a) \sum_{b \in \mathcal{A}} \pi(b|y) Q(y, b).$$

Remark

$$T^\pi[Q] = r + \gamma P_{\mathcal{X} \times \mathcal{A}}^\pi Q.$$

Optimality Operator

The optimality operator $T^* \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}} \rightarrow \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$ is defined as follows:

$$T^*[Q](x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{X}} p(y|x, a) \max_{b \in \mathcal{A}} Q(y, b).$$

Monotonicity

Let $V_1 \in \mathbb{R}^X$ and $V_2 \in \mathbb{R}^X$ such that $V_1 \leq V_2$ and let $\pi \in \Pi$, then

$$B^\pi[V_1] \leq B^\pi[V_2],$$

$$B^*[V_1] \leq B^*[V_2].$$

The same goes with evaluation and optimality operators for Q -functions.

Domination of the Optimal Operator

Let $V \in \mathbb{R}^X$ and $\pi \in \Pi$, then:

$$B^\pi V \leq B^* V.$$

The same goes with evaluation and optimality operators for Q -functions.

Exercise

Prove those properties.

Th-1: Fixed-points of the evaluation operators.

B^π is a γ -contraction with unique fixed-point V^π . Similarly, T^π is a γ -contraction with unique fixed-point Q^π :

$$\begin{aligned}B^\pi[V^\pi] &= V^\pi, \\T^\pi[Q^\pi] &= Q^\pi.\end{aligned}$$

Proof that T^π is a contraction.

Let $Q_1 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$ and $Q_2 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$, then $\forall (x, a) \in \mathcal{X} \times \mathcal{A}$ we have:

$$\begin{aligned}T^\pi[Q_1](x, a) - T^\pi[Q_2](x, a) &= \gamma \sum_{y \in \mathcal{X}} \sum_{b \in \mathcal{A}} p(y|x, a) \pi(b|y) [Q_1(y, b) - Q_2(y, b)], \\&\leq \gamma \max_{b \in \mathcal{A}, y \in \mathcal{X}} [Q_1(y, b) - Q_2(y, b)], \\&\leq \gamma \|Q_1 - Q_2\|_\infty.\end{aligned}$$

Therefore $\|T^\pi[Q_1] - T^\pi[Q_2]\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty$

Th-2: Fixed-points of the optimality operators.

B^* is a γ -contraction, its unique fixed-point is noted V^* and called the optimal value function. Similarly, T^* is a γ -contraction, its unique fixed-point is noted Q^* and called the optimal action-value function:

$$\begin{aligned} B^*[V^*] &= V^*, \\ T^*[Q^*] &= Q^*. \end{aligned}$$

Proof

Prove the theorem as an exercise. This consists in showing that T^* and B^* are contractions.

Th-3: Majorants of the Value functions.

V^* is a majorant of $\{V^\pi\}_{\pi \in \Pi}$ and Q^* is a majorant of $\{Q^\pi\}_{\pi \in \Pi}$:

$$\forall \pi \in \Pi, V^\pi \leq V^*,$$

$$\forall \pi \in \Pi, Q^\pi \leq Q^*.$$

Proof for Q functions.

By domination of the optimality operator, we have that:

$$Q^\pi = T^\pi Q^\pi \leq T^* Q^\pi.$$

By monotonicity of the optimality operator, we have that:

$$T^* Q^\pi \leq (T^*)^2 Q^\pi \leq (T^*)^3 Q^\pi \dots \leq (T^*)^n Q^\pi.$$

Therefore at the limit $T^* Q^\pi \leq (T^*)^\infty Q^\pi = Q^*$ which concludes the proof.

Th-4: Existence and construction of optimal policies.

A policy $\pi \in \Pi$ is optimal if and only if $V^\pi = V^*$. Optimal stochastic policies exist and can be constructed from Q^* or Q^π :

$$\begin{aligned} V^\pi = V^* &\iff \forall x \in \mathcal{X}, \text{Supp}[\pi(.|x)] \subset \arg \max_{a \in \mathcal{A}} [Q^*(x, a)], \\ &\iff \forall x \in \mathcal{X}, \text{Supp}[\pi(.|x)] \subset \arg \max_{a \in \mathcal{A}} [Q^\pi(x, a)], \end{aligned}$$

$\text{Supp}[\pi(.|x)] = \{a \in \mathcal{A} | \pi(a|x) > 0\}$ is the support of $\pi(.|x)$.

Proof: Exercise

To do so remark that for a function $Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$, for a policy π and for $x \in \mathcal{X}$:

$$\text{Supp}[\pi(.|x)] \subset \arg \max_{a \in \mathcal{A}} [Q(x, a)] \iff \sum_{a \in \mathcal{A}} \pi(a|x) Q(x, a) = \max_{a \in \mathcal{A}} Q(x, a).$$

This means that expectation and maximum operators are equivalent.

Remark on Greediness

For a given function $Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$, the set of policies:

$$\mathcal{G}(Q) = \{\pi \in \Pi \mid \forall x \in \mathcal{X}, \text{Supp}[\pi(\cdot|x)] \subset \arg \max_{a \in \mathcal{A}} [Q(x, a)]\},$$

is called the greedy set with respect to Q . If $\pi \in \mathcal{G}(Q)$, we say that π is a greedy policy with respect to Q . Therefore, a policy π is optimal if and only if it is greedy with respect to Q^* or Q^π :

$$\begin{aligned} V^\pi = V^* &\iff \pi \in \mathcal{G}(Q^\pi), \\ &\iff \pi \in \mathcal{G}(Q^*). \end{aligned}$$

Th-5: The Greedy Policy Improvement.

Let $\pi \in \Pi$ be a stochastic policy, then the greedy policy with respect to Q^π noted $\pi_{\mathcal{G}} \in \mathcal{G}(Q^\pi)$ is such that:

$$\forall \pi \in \Pi, \quad Q^\pi \leq Q^{\pi_{\mathcal{G}}}.$$

In addition:

$$Q^\pi = Q^{\pi_{\mathcal{G}}} \iff Q^\pi = Q^* \implies \pi_{\mathcal{G}} \text{ is optimal.}$$

Bonus Point Alert!

If you prove the 5-Theorems by the end of the week, +2 points on your mark!

Lessons of the 5 Fundamental Theorems:

- They show how we can explicitly construct an optimal policy from Q^* .
- Q^* is the fixed point of a γ -contraction operator.

Value Iteration

$$\text{Value Iteration: } \begin{cases} Q_0 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}} & \text{Initialisation} \\ \forall k \in \mathbb{N}, \quad Q_{k+1} = T^* Q_k & \text{Recurrence} \end{cases}$$

As T^* is a γ -contraction operator, we have $\lim_{k \rightarrow \infty} Q_k = Q^*$.

Remark On Greediness

With the precedent remark on greediness, it is trivial to show that :

$$\pi \in \mathcal{G}(Q) \implies T^*Q = T^\pi Q.$$

Therefore, one can rewrite Value Iteration:

$$\text{Value Iteration: } \left\{ \begin{array}{ll} Q_0 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}} & \text{Initialisation} \\ \forall k \in \mathbb{N}, \pi_k \in \mathcal{G}(Q_k) & \text{Greedy-step} \\ \forall k \in \mathbb{N}, Q_{k+1} = T^{\pi_k} Q_k & \text{Evaluation} \end{array} \right.$$

Value Iteration Performance Bound

$$\|Q^* - Q^{\pi_k}\|_\infty \leq \frac{2\gamma^k \|r\|_\infty}{(1-\gamma)^2},$$

where $\|\cdot\|_\infty$ is the infinite norm.

Policy Iteration

It relies mainly on the greedy policy improvement theorem.

$$\text{Policy Iteration: } \left\{ \begin{array}{ll} Q_0 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}} & \text{Initialisation} \\ \forall n \in \mathbb{N}, \pi_k \in \mathcal{G}(Q_k) & \text{Greedy-step} \\ \forall n \in \mathbb{N}, Q_{k+1} = (T^{\pi_k})^\infty Q_k & \text{Evaluation} \end{array} \right.$$

Full Evaluation

In PI, the evaluation is a full evaluation $Q_{k+1} = (T^{\pi_k})^\infty Q_k$ which means that $Q_{k+1} = Q^{\pi_k}$.

Improvement Property

Because of the greedy policy improvement theorem, we have:

$$Q_1 = Q^{\pi_0} \leq Q_2 = Q^{\pi_1} \leq Q_3 = Q^{\pi_2} \dots$$

where each step is strictly improving unless optimality is reached.

Convergence Property

The PI scheme finishes in a finite number of steps:

$$\exists K \in \mathbb{N}, \text{ such that } \pi_K \text{ is optimal.}$$

Modified Policy Iteration

It generalizes VI and PI in one common framework.

$$\text{MPI: } \left\{ \begin{array}{ll} Q_0 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}} & \text{Initialisation} \\ \forall k \in \mathbb{N}, \pi_k \in \mathcal{G}(Q_k) & \text{Greedy-step} \\ \forall n \in \mathbb{N}, Q_{k+1} = (T^{\pi_k})^m Q_k & \text{Evaluation} \end{array} \right.$$

Links between PI, VI and MPI

Here the evaluation is $Q_{k+1} = (T^{\pi_k})^m Q_k$ with $m \in \overline{\mathbb{N}^*}$. This trivially generalizes to VI with $m = 1$ and to PI with $m = \infty$.

Three Major Difficulties

- Locality problem: States can't be trivially reached. Therefore information about the dynamics and the reward can't be trivially collected. This calls for the need of some underlying process, called exploration, to get this necessary information to solve the task.
- Sampling problem: Dynamics are not fully known. This puts an even bigger burden on the exploration process.
- Representation problem: State and action spaces can be so large that associated information can't be stored (nor updated) but need to be learned through functional approximations. Objects of interest such as the Q-values will be parameterised by $\theta \in \mathbb{R}^N$ where $N \in \mathbb{N}$ is the number of parameters. When functional approximation is done with deep neural networks, we are in the territory of Deep Reinforcement Learning.

What Information we will have access to

In most practical applications, an agent observes at time t the state x_t choose action a_t , receives the reward $r_t = r(x_t, a_t)$ and transition to state $x_{t+1} = y_t$ with probability $p(y_t|x_t, a_t)$. The agent has no access to the local probability $p(y_t|x_t, a_t)$ nor the global information of dynamics p , reward r , but only to the transition (x_t, a_t, r_t, y_t) .

What we did today

- Introduced MDPs, a standard setup for RL.
- Introduced the **goal of reinforcement learning**: to find a policy that **maximizes the discounted cumulated reward** in an MDP.
- Introduced algorithms to achieve this goal, **when we know everything about the MDP**, including the transition kernel.

Readings and other courses

- **Sutton and Barto**, chapters 1 to 4.
- **David Silver's RL course** lectures 1, 2 and 3.

Deep Q-Networks

Milestones: From Value Iteration to DQN

VI in Stochastic Games (Shapley 1953) and MDP (Bellman 1957)

- Computes the optimal value of an MDP.
- The proof relies on Banach fixed-point contraction theorem (1922).

Approximate Value Iteration

- Bounds in infinity norm: Bertsekas and Tsilikis (1996).
- Bounds in L_p norm: Munos (2007).

Fitted-Q Algorithms

- Fitted-Q with random forests: Ernst (2005)
- Neural fitted-Q: Riedmiller (2005)

Atari as an Environment

- Arcade Learning Environment: Bellemare (2012)

Deep Q-Network

- Scaling Neural fitted-Q to Atari games: Bellemare (2013)

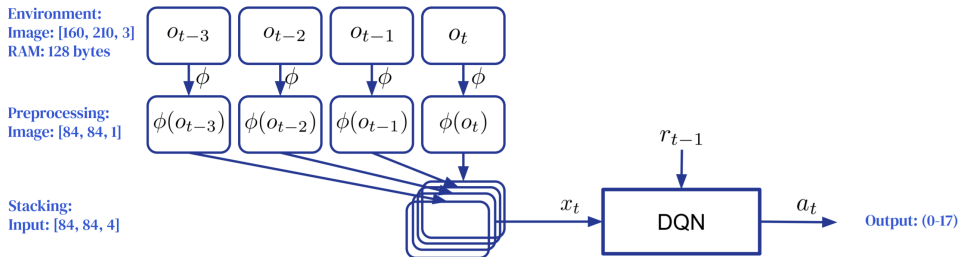
Details

- Around 50 Atari games
- Raw observations: 128 bytes (0-255) of RAM + $160 \times 210 \times 3$ (100800 bytes) of 2-D RGB image.
- Raw actions: 18 discrete actions (0-17)
- Rewards: deltas of the score of the games
- Frequency: 60Hz
- Length of a game: up to 30 minutes.

Why is it interesting?

- Huge state space
- No canonical meaningful features
- Very long optimisation horizon

Preprocessing of the ALE observations



Preprocessing

- Preprocessing ϕ of each observation is done in this order (Reduce the computation):
 - No RAM state in the observation, keep only the image: [160, 210, 3]
 - Take the maximum for each pixel of image at time t and $t - 1$: [160, 210, 3]
 - Extract the luminance from the RGB: [160, 210, 1]
 - Downscale the image to 84x84: [84, 84, 1]
- Stacking of the 4 previous preprocessed observation (Reduce the partial observability): [84, 84, 4]
- The stack of the preprocessed observation is the input of the DQN algorithm.
- The output of DQN is the action at 15 Hz which means that there is an action repeat of 4.

Approximation of the Value Iteration scheme

- In practice, it is not possible to apply T^* to a function $Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$
- Therefore, $Q_{k+1} = T^* Q_k$ is going to be approximated by $Q_{k+1} = \widehat{T^* Q_k}$
- The approximation error at step k is noted $\epsilon_k = T^* Q_k - \widehat{T^* Q_k}$

Approximate Value Iteration

$$\text{Approximate Value Iteration: } \begin{cases} Q_0 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}} & \text{Initialisation} \\ \forall k \in \mathbb{N}, \quad Q_{k+1} = T^* Q_k + \epsilon_k & \text{Recurrence} \end{cases}$$

Error Propagation Bound

Let $\pi_k \in \mathcal{G}(Q_k)$, then one can show that:

$$\|Q^* - Q^{\pi_k}\|_{\infty} \leq \frac{2\gamma}{(1-\gamma)^2} \left((1-\gamma) \sum_{j=1}^k \gamma^{k-j} \|\epsilon_j\|_{\infty} \right) + \frac{2\gamma^k \|r\|_{\infty}}{(1-\gamma)^2}.$$

How we can get a good approximation $\widehat{T^*Q_k}$ of T^*Q_k ?

- We have transitions of the type $\xi = (x \in \mathcal{X}, a \in \mathcal{A}, r(x, a), y \sim p(\cdot|x, a))$.
- We can compute the target $t(\xi, Q) = r(x, a) + \gamma \max_{b \in \mathcal{A}} Q(y, b)$ which is an unbiased estimate of $T^*Q(x, a)$.
- As from transitions we can build unbiased estimates of T^*Q , we can apply any regression algorithm to get an approximation $\widehat{T^*Q}$ that fits T^*Q .

Regression Algorithm at step k

- We have a set $\mathcal{D} = \{\xi_n = (x_n, a_n, r_n = r(x_n, a_n), y_n)\}_{n=1}^N$.
- We build the targets $t(\xi_n, Q_k)$ (unbiased estimates of $T^*Q_k(x_n, a_n)$)
- We define $\widehat{T^*Q_k}$ as the output of this general L_2 -regression algorithm:

$$\widehat{T^*Q_k} = \arg \min_{f \in \mathcal{F}} \sum_{n=1}^N (t(\xi_n, Q_k) - f(x_n, a_n))^2,$$

where $\mathcal{F} \subset \mathcal{R}^{\mathcal{X} \times \mathcal{A}}$ is a functional space.

Optimization

In practice, getting the arg min is quite hard but we can get the opt min which is simply the result obtained by the optimization technique chosen:

$$\widehat{T^*Q_k} = \operatorname{opt} \min_{f \in \mathcal{F}} \sum_{n=1}^N (t(\xi_n, Q_k) - f(x_n, a_n))^2.$$

Instantiation of AVI

$$\left\{ \begin{array}{ll} Q_0 \in \mathcal{F} & \text{Initialisation} \\ \forall k \in \mathbb{N}, \left\{ \begin{array}{ll} \mathcal{D} = \{\xi_n\}_{n=1}^N & \text{Data Collection} \\ Q_{k+1} = \operatorname{opt} \min_{f \in \mathcal{F}} \sum_{n=1}^N (t(\xi_n, Q_k) - f(x_n, a_n))^2 & \text{Regression} \end{array} \right. \end{array} \right.$$

Deep RL Lingo

In deep RL, the data collection step is often called the acting and the regression step is called the learning.

High Level Properties

At high-level Neural Fitted-Q is an instantiation of AVI without data collection and parameterize by a neural network

- Consists in learning a good policy from a fixed data set $\mathcal{D} = \{\xi_n\}_{n=1}^N$: the data collection step (the acting) is not existent here.
- The functional space chosen is a neural network $\mathcal{F}_\theta = \{Q_\theta | \theta \in \mathbb{R}^{\mathcal{N}}\}$ where θ is the vector of weights and $\mathcal{N} \in \mathbb{N}$ is the number of weights.

Neural Fitted-Q:

$$\left\{ \begin{array}{ll} \mathcal{D} = \{\xi_n\}_{n=1}^N & \text{Fixed Data} \\ \theta_0 \in \mathbb{R}^{\mathcal{N}} & \text{Initialisation} \\ \forall k \in \mathbb{N}, Q_{\theta_{k+1}} = \text{opt min}_{Q_\theta \in \mathcal{F}_\theta} \sum_{n=1}^N (t(\xi_n, Q_{\theta_k}) - Q_\theta(x_n, a_n))^2, & \text{Regression} \end{array} \right.$$

The Learning: Regression Step

The regression step, also called learning, is practically implemented with stochastic gradient descent.

Algorithm 1: Regression step of Neural Fitted Q.

Input : online weights: $\theta \in \mathbb{R}^{\mathcal{N}}$, target weights: $\theta_k \in \mathbb{R}^{\mathcal{N}}$, Dataset of transitions: \mathcal{D} , update period: $I \in \mathbb{N}$, learning rate: $\alpha \in \mathbb{R}$, batch size: $B \in \mathbb{N}$, discount factor: γ

Output: θ_{k+1}

```
1 for  $i \in \{0, \dots, I - 1\}$  do
2   Draw uniformly a batch  $\mathcal{B} = \{(x_j, a_j, r_j, y_j)\}_{j=1}^B$  from  $\mathcal{D}$ 
3   Compute the targets:  $\forall 1 \leq j \leq B, \quad t_j \leftarrow r_j + \gamma \max_{b \in \mathcal{A}} Q_{\theta_k}(y_j, b)$ 
4   Compute the loss:  $\mathcal{L}(\theta) \leftarrow \sum_{j=1}^B (Q_{\theta}(x_j, a_j) - t_j)^2$ 
5   Update the online weights:  $\theta \leftarrow \theta - \alpha \nabla_{\theta} \mathcal{L}(\theta)$ 
6 end
7  $\theta_{k+1} = \theta$ 
```

Important Observation

One observation is that only two set of weights is necessary to perform the regression described in Algorithm 1. A fixed set of weights $\theta' = \theta_k$ that allows to generate the targets and a set of weights θ that allows to optimize the loss. The fixed set of weights is called the target network and the one used to optimize the loss is the online network. After I steps of stochastic gradient descent, we consider the regression finished and we can output the online weights θ as θ_{k+1} and set $\theta' = \theta_{k+1}$.

Neural Fitted-Q in a nutshell

Neural Fitted-Q is simply K steps of the previous regression. More precisely, a target network $\theta' \in \mathbb{R}^{\mathcal{N}}$ and an online network $\theta \in \mathbb{R}^{\mathcal{N}}$ are initialized by the neural network architecture and θ' plays the role of θ_k in the regression, then at the end of each regression step the target network is updated by the weights of the online network and so on.

Algorithm 2: Neural Fitted-Q.

Input : Neural network architecture: \mathcal{F}_θ , dataset of transitions: \mathcal{D} , learning steps: $K \in \mathbb{N}$, update period: $I \in \mathbb{N}$, learning rate: $\alpha \in \mathbb{R}$, batch size: $B \in \mathbb{N}$, discount factor: γ

Output: θ_{NFQ}

```
1 Initialize online weights:  $\theta \leftarrow \text{Init}(\mathcal{F}_\theta)$ 
2 Initialize target weights:  $\theta' \leftarrow \text{Init}(\mathcal{F}_\theta)$ 
3 for  $k \in \{0, \dots, K-1\}$  do
4   for  $i \in \{0, \dots, I-1\}$  do
5     Draw uniformly a batch  $\mathcal{B} = \{(x_j, a_j, r_j, y_j)\}_{j=1}^B$  from  $\mathcal{D}$ 
6     Compute the targets:  $\forall 1 \leq j \leq B, \quad t_j \leftarrow r_j + \gamma \max_{b \in \mathcal{A}} Q_{\theta'}(y_j, b)$ 
7     Compute the loss:  $\mathcal{L}(\theta) \leftarrow \sum_{j=1}^B (Q_\theta(x_j, a_j) - t_j)^2$ 
8     Update the online weights:  $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$ 
9   end
10  Update the target weights:  $\theta' \leftarrow \theta$ 
11 end
12  $\theta_{\text{NFQ}} = \theta$ 
```

What is DQN?

- Deep Q-Network (DQN) is an instantiation of AVI with a neural network used as functional space for the regression step (learning).
- Contrary to Neural Fitted-Q, the data collection (acting) is a continuous process that happens in parallel of the learning.
- Those two processes share the weights θ of the online network as well as the collected data set \mathcal{D} .

Data Collection

The data set \mathcal{D} (also called the replay buffer) is implemented as a First-In First-Out (FIFO) queue and the data inserted in \mathcal{D} is collected by interacting with the environment with an ϵ -greedy policy $\pi_{\theta, \epsilon}$:

$$\pi_{\theta, \epsilon} = (1 - \epsilon)\pi_{\theta} + \epsilon\pi_{\text{U}},$$

where $\pi_{\theta} \in \mathcal{G}(Q_{\theta})$ and π_{U} is the uniform policy.

Algorithm 3: Acting process of DQN.

Input : replay buffer: \mathcal{D} , environment: \mathcal{E}

Shared: online weights: $\theta \in \mathbb{R}^{\mathcal{N}}$

Output: None

```
1  $x \leftarrow \text{Init}(\mathcal{E})$ 
2 while True do
3    $a \leftarrow \text{Sample}(\pi_{\theta, \epsilon}(\cdot|x))$ 
4    $r(x, a), y \leftarrow \text{Step}(\mathcal{E}, a)$ 
5   Put  $(x, a, r(x, a), y)$  in  $\mathcal{D}$ 
6    $x \leftarrow y$ 
7 end
```

Algorithm 4: Learning process of DQN.

Input : Neural network architecture: \mathcal{F}_θ learning steps: $K \in \mathbb{N}$, update period: $I \in \mathbb{N}$, learning rate: $\alpha \in \mathbb{R}$, batch size: $B \in \mathbb{N}$, discount factor: γ

Shared: replay buffer: \mathcal{D}

Output: θ_{DQN}

```

1 Initialize online weights:  $\theta \leftarrow \text{Init}(\mathcal{F}_\theta)$ 
2 Initialize target weights:  $\theta' \leftarrow \text{Init}(\mathcal{F}_\theta)$ 
3 for  $k \in \{0, \dots, K-1\}$  do
4   for  $i \in \{0, \dots, I-1\}$  do
5     Draw uniformly a batch  $\mathcal{B} = \{(x_j, a_j, r_j, y_j)\}_{j=1}^B$  from  $\mathcal{D}$ 
6     Compute the targets:  $\forall 1 \leq j \leq B, \quad t_j \leftarrow r_j + \gamma \max_{b \in \mathcal{A}} Q_{\theta'}(y_j, b)$ 
7     Compute the loss:  $\mathcal{L}(\theta) \leftarrow \sum_{j=1}^B (Q_\theta(x_j, a_j) - t_j)^2$ 
8     Update the online weights:  $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$ 
9   end
10  Update the target weights:  $\theta' \leftarrow \theta$ 
11 end
12  $\theta_{\text{DQN}} = \theta$ 

```

Architecture

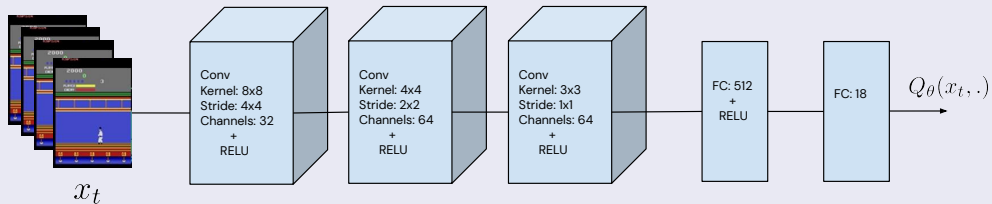


Figure: DQN's Neural Architecture.

Algorithmic Improvements

- Double DQN (DDQN) aims to solve the overestimation of the targets [Van Hasselt et al., 2016]
- Prioritized Experience Replay aims to sample transitions with higher TD-errors [Schaul et al., 2015]
- Distributional Reinforcement Learning aims to learn not only the expected returns but the entire distributions of returns via a categorical approach [Bellemare et al., 2017] or a quantile approach [Dabney et al., 2018]

Architectural Improvements

- Dueling architecture allows an agent to have less-biased estimates of unused actions [Wang et al., 2016].
- Distributed settings allows to collect more diverse data for the learning process [Nair et al., 2015, Horgan et al., 2018, Kapturowski et al., 2018] and can be combine with different prioritization mechanisms.

Memory Additions

- Working memories such as Long-Short Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] or Gated-Recurrent Unit (GRU) [Cho et al., 2014, Chung et al., 2014] have been added to the vanilla DQN agent to better handle partial-observability [Hausknecht and Stone, 2015].
- Combining LSTM and distributed actors is done in the Recurrent Replay Distributed DQN (R2D2) agent [Kapturowski et al., 2018]. Episodic memory [Badia et al., 2020] has also been added to help exploration.

Meta Controllers

Tuning some hyper-parameters while learning can also be beneficial. This can be done with bandits [Schaul et al., 2019], via meta-gradients [Xu et al., 2018, Xu et al., 2020] or via population-based training [Jaderberg et al., 2017].

Deep Policy Gradient Algorithms

So far

- Computing approximate optimal Q function.
- Maximize approximate optimal Q to derive optimal policy.

Idea of policy gradient algorithms

Choose a **family of parameterized policies** π_θ , $\theta \in \mathbb{R}^p$. Directly find the policy that maximizes the return,

$$\theta^* = \operatorname{argmax}_\theta J^{\pi_\theta}$$

by **gradient ascent**.

Question: How do we compute the gradient of the return?

Gradient ascent

Let $f: \mathbb{R}^p \rightarrow \mathbb{R}$ be a sufficiently smooth function (e.g. continuously differentiable), then the sequence of $(x_n)_{n \geq 0}$ defined by recurrence as

$$x_0 = x$$

$$x_{n+1} = x_n + \alpha_n \nabla_x f(x_n)$$

converges to a local maximum of f , under mild conditions on the α 's.

Stochastic gradient ascent

Often, no direct access to $\nabla_x f$, but to some unbiased estimate, i.e. to a random variable $G(x)$, such that $\mathbb{E}[G(x)] = \nabla_x f(x)$.

Under reasonable hypothesis on the α 's, and on the variance of the $G(x)$'s, replacing $\nabla_x f$ with G in gradient ascent converges to a local maximum of f .

PG algorithm in a nutshell

Most policy gradient (PG) algorithms proceed as follows

- Initialize $\theta \leftarrow \theta_0$
- Iterate
 - Compute a stochastic estimate of $\nabla_{\theta} J^{\pi_{\theta}}$, $G(\theta)$
 - Update the current parameters by gradient ascent

$$\theta \leftarrow \theta + \alpha G(\theta)$$

Insight: PG algorithms mostly differ in how they approx. $\nabla_{\theta} J^{\pi_{\theta}}$.

Reinforce gradient estimate

The following quantity

$$G(\theta) = \sum_{t \geq 0} \gamma^t \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \sum_{t' \geq t} \gamma^{t'-t} r(S_{t'}, A_{t'})$$

is an **unbiased estimate** of $\nabla_{\theta} J^{\pi_{\theta}}$, i.e. $\mathbb{E}[G(\theta)] = \nabla_{\theta} J^{\pi_{\theta}}$.

Log-trick

Let p_θ be a **parametric probability distribution**, i.e. a function defined on a finite set \mathcal{X} such that $\sum_x p_\theta(x) = 1$ and $p_\theta(x) \geq 0$ for all x .

For any function R of x ,

$$\nabla_\theta \mathbb{E}_{x \sim p_\theta} [R(x)] = \mathbb{E}_{x \sim p_\theta} [\nabla_\theta \log p_\theta(x) R(x)].$$

Insights on the log-trick

- **Very general trick** to get Monte-Carlo estimate of the gradient of an average quantity w.r.t. the parameters of the probability with which we are taking the average.
- **Often high variance:** It is general, but is often sub-optimal. Lower variance estimates should often be preferred.
- **Very simple intuitively:** to increase the expectation, one should increase the probability of samples when the associated value is high, and reduce it when the value is low.

Gradient of the log probability of a trajectory

For any trajectory $\tau = (xa)_{t \geq 0}$,

$$\nabla_{\theta} \log \mathcal{T}(\tau, \pi_{\theta}) = \sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(a_t | x_t)$$

Exercise

Prove this equality.

First step towards Reinforce

$$\nabla_{\theta} J^{\pi_{\theta}} = \mathbb{E}_{\tau \sim \mathcal{T}(\cdot, \pi_{\theta})} \left[\left(\sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \right) \left(\sum_{t' \geq 0} \gamma^{t'} r(S_{t'}, A_{t'}) \right) \right]. \quad (1)$$

Exercise

Prove this equality using the log trick and the equality on the previous slide.

Gradient of log probability and independant variable

Let X be a random variable independant from A_t knowing S_t ,

$$\mathbb{E} [\nabla_{\theta} \log \pi_{\theta}(A_t|S_t)X|S_t] = 0 . \quad (2)$$

Exercise

Prove this equality using the independance property and the fact that a probability distribution sums to one

Unbiasedness of Reinforce

$$\nabla_{\theta} J^{\pi_{\theta}} = \mathbb{E}_{\tau \sim \mathcal{T}(\cdot, \pi_{\theta})} \left[\sum_{t \geq 0} \gamma^t \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \sum_{t' \geq t} \gamma^{t'-t} r(S_{t'}, A_{t'}) \right]$$

Exercise

Prove unbiasedness of reinforce by decomposing (1), and using (2), after noticing that A_t is independent from S_{t-k} ($k > 0$) knowing S_t .

Remarks

Things to remember about reinforce

- + Makes very **little use of the Markov property**. In fact can be slightly adapted to work even in the non-Markov case.
- **High variance**, may require a lot of time to converge.
- **Requires full trajectories**, impossible to use for infinite horizon MDPs.

Baselined estimator

For any sequence of random variables X_t , such that, for all t , X_t is independant from A_t knowing S_t ,

$$\nabla_{\theta} J^{\pi_{\theta}} = \mathbb{E}_{\tau \sim \mathcal{T}(\cdot, \pi_{\theta})} \left[\sum_{t \geq 0} \gamma^t \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \left(\sum_{t' \geq t} \gamma^{t'-t} r(S_{t'}, A_{t'}) - X_t \right) \right]$$

Exercise

Prove this property using (2).

On optimal baselines

Ideally, the sequence of X_t 's should be chosen to **minimize the variance of the estimator**, while **maintaining the independance property**. Finding such baselines is difficult/impossible, and we often resort to suboptimal solutions.

Value baseline

$$\nabla_{\theta} J^{\pi_{\theta}} = \mathbb{E}_{\tau \sim \mathcal{T}(\cdot, \pi_{\theta})} \left[\sum_{t \geq 0} \gamma^t \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \left(\sum_{t' \geq t} \gamma^{t'-t} r(S_{t'}, A_{t'}) - V^{\pi_{\theta}}(S_t) \right) \right]$$

Exercise

Prove this equation.

Value baseline vs. optimal baseline

Using $X_t = V^{\pi_{\theta}}(S_t)$ is **suboptimal**. However, in most cases, it still provides **some variance reduction**, since the value function is the solution of

$$V^{\pi} = \operatorname{argmin}_V \operatorname{Var}_{\tau \sim \mathcal{T}(\cdot, \pi_{\theta})} \left[\sum_{t \geq 0} \gamma^t r(S_t, A_t) - V(S_0) \right],$$

and thus minimizes the variance of the second term of the gradient estimator.

Insight

- All our gradient estimates must **sample infinite trajectories**: impossible in practice.
- Could we **replace returns with average returns**, i.e. value or Q functions?

Value bootstrapping

- Q-function gradient estimator

$$\nabla_{\theta} J^{\pi_{\theta}} = \mathbb{E}_{\tau \sim \mathcal{T}(\cdot, \pi_{\theta})} \left[\sum_{t \geq 0} \gamma^t \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) (Q^{\pi_{\theta}}(S_t, A_t) - V^{\pi_{\theta}}(S_t)) \right].$$

- N-steps value gradient estimator

$$\nabla_{\theta} J^{\pi_{\theta}} = \mathbb{E}_{\tau \sim \mathcal{T}(\cdot, \pi_{\theta})} \left[\sum_{t \geq 0} \gamma^t \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \left(\sum_{n=0}^N \gamma^n r(S_{t+n}, A_{t+n}) + V^{\pi_{\theta}}(S_{t+N+1}) - V^{\pi_{\theta}}(S_t) \right) \right]$$

Exercise

Prove the Q-function and N-steps value gradient estimator unbiasedness. To do this, decompose the gradient into a sum, then perform partial conditioning inside the main expectation.

Remarks

- More generally, one can use practically any **unbiased estimate of the return** to replace the empirical return and reduce the variance (e.g. λ -returns, importance reweighted estimates, ...).
- Most often, one only has access to **an approximate value or Q function**. Using an approximation will bias the estimate. There is thus a **bias/variance trade-off** when using something other than the empirical return.

Approximating the state distribution

Ideal policy gradient estimator:

$$\nabla_{\theta} J^{\pi_{\theta}} = \mathbb{E}_{\tau \sim \mathcal{T}(\cdot, \pi_{\theta})} \left[\sum_{t \geq 0} \gamma^t \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \left(\sum_{t' \geq t} \gamma^{t'-t} r(S_{t'}, A_{t'}) \right) \right]$$

In practice,

$$\nabla_{\theta} \tilde{J}^{\pi_{\theta}} = \mathbb{E}_{\tau \sim \mathcal{T}(\cdot, \pi_{\theta})} \left[\sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \left(\sum_{t' \geq t} \gamma^{t'-t} r(S_{t'}, A_{t'}) \right) \right]$$

Why are we doing this?

- Not doing this requires maintaining an absolute counter on the number of steps elapsed, which might be impractical.
- Adding this discount reduces the effect of latter part of trajectories on the gradient. This is an artifact of working with discounts $\gamma < 1$, which makes our objective depend on the initial state distribution. In many practical use cases, adding this correction will actually hurt performance for the undiscounted return, which is often what we care about.

N-step returns in practice

In practice, N -step bootstrapped returns often provide better results, compared to 1-step bootstrapped. Different explanations can back up this experimental finding:

- As mentioned earlier, estimating returns using N -step bootstraps yields higher variance, but lower bias estimates. When one can draw many samples from the environment, variance might be a lesser problem than bias.
- When working with non fully-observable (MDP) environment, even when using the optimal state based value function, the PG update remains biased, while the full return update is unbiased. In that case, extending the temporal horizon for bootstrap will help mitigate the inherent error in the bootstrapped return.

Implementing N -steps return

To ease implementing N -step returns, most implementation actually sample a trajectory of size $T \geq N$, and run a window of size N on the trajectory of size T to compute the N -steps returns. This typically mean that the last elements in the trajectory don't have access to N elements in the future, and in that case only those elements are considered for bootstrap.

Shortcomings of PG

- The PG methods we've talked about all require sampling trajectories from the exact same policy that the one we're using to act: Vanilla PG is on-policy.
- Combines poorly with how people usually use Deep Learning: collect a lot of data, then train a model on those data, by performing several steps of gradient optimization on the data.
- Here, in theory, only one pass of gradient on all the data collected at once is allowed.
- In what comes next, how to mitigate this need for fully on policy data: Regularized Reinforcement learning methods and off-policy corrections!

Policy Gradient in a Nutshell

The high-level idea is to improve the value function V^{π_θ} of a parameterized policy π_θ via gradient ascent:

$$\theta \leftarrow \theta + \alpha \partial_\theta V^{\pi_\theta}.$$

To implement such an idea it is essential to compute: $\partial_\theta V^{\pi_\theta}$.

Preliminary Conditions and Notations

We will consider a general set of policies Π_θ parameterize by $\theta \in \mathbb{R}^{\mathcal{N}}$:

$$\Pi_\theta = \{\pi_\theta \in \Pi | \theta \in \mathbb{R}^{\mathcal{N}}\}.$$

We make the assumption that the following quantities exist:

$$\begin{aligned} \forall \theta \in \mathbb{R}^{\mathcal{N}}, \forall (x, a) \in \mathcal{X} \times \mathcal{A}, \quad \partial_\theta \pi_\theta(a|x) \in \mathbb{R}^{\mathcal{N}}, \\ \forall \theta \in \mathbb{R}^{\mathcal{N}}, \forall x \in \mathcal{X}, \quad \partial_\theta V^{\pi_\theta}(x) \in \mathbb{R}^{\mathcal{N}}. \end{aligned}$$

Policy Gradient Theorem

Let Π_θ be a set of policies. Then $\forall \theta \in \mathbb{R}^{\mathcal{N}}, \forall x \in \mathcal{X}$:

$$\partial_\theta V^{\pi_\theta}(x) = \mathbb{E}_{\tau^{\pi_\theta} \sim \mathcal{T}(x, \pi_\theta)} \left[\sum_{t \geq 0} \gamma^t \sum_{a \in \mathcal{A}} Q^{\pi_\theta}(X_t, a) \partial_\theta \pi_\theta(a|X_t) \right].$$

Exercise

Prove the policy gradient theorem! Use the unicity of the solution of Bellman operator.

Proof (1)

Let θ_i be one dimension of the vector of parameters θ :

$$\begin{aligned} V^{\pi_\theta}(x) &= \sum_{a \in \mathcal{A}} \pi_\theta(a|x) Q^{\pi_\theta}(x, a), \\ \partial_{\theta_i} V^{\pi_\theta}(x) &= \sum_{a \in \mathcal{A}} \partial_{\theta_i} \pi_\theta(a|x) Q^{\pi_\theta}(x, a) + \sum_{a \in \mathcal{A}} \pi_\theta(a|x) \partial_{\theta_i} Q^{\pi_\theta}(x, a). \end{aligned}$$

Let's take a closer look at $\partial_{\theta_i} Q^{\pi_\theta}(x, a)$:

$$\begin{aligned} \partial_{\theta_i} Q^{\pi_\theta}(x, a) &= \partial_{\theta_i} (r(x, a) + \gamma \sum_{y \in \mathcal{X}} p(y|x, a) V_\theta^\pi(y)), \\ &= \gamma \sum_{y \in \mathcal{X}} p(y|x, a) \partial_{\theta_i} V_\theta^\pi(y). \end{aligned}$$

Proof (2)

Therefore:

$$\partial_{\theta_i} V^{\pi_\theta}(x) = \sum_{a \in \mathcal{A}} \partial_{\theta_i} \pi_\theta(a|x) Q^{\pi_\theta}(x, a) + \gamma \sum_{a \in \mathcal{A}} \sum_{y \in \mathcal{X}} \pi_\theta(a|x) p(y|x, a) \partial_{\theta_i} V_\theta^\pi(y).$$

We recognize a Bellman equation where the quantity $\sum_{a \in \mathcal{A}} \partial_{\theta_i} \pi_\theta(a|x) Q^{\pi_\theta}(x, a)$ plays the role of the reward $r^\pi(x)$, therefore we have by uniqueness (we recall that $V^\pi(x) = \mathbb{E}_{\tau^\pi \sim \mathcal{T}(x, \pi)} \left[\sum_{t \geq 0} \gamma^t r^\pi(X_t) \right]$):

$$\partial_{\theta_i} V^{\pi_\theta}(x) = \mathbb{E}_{\tau^{\pi_\theta} \sim \mathcal{T}(x, \pi_\theta)} \left[\sum_{t \geq 0} \gamma^t \sum_{a \in \mathcal{A}} Q^{\pi_\theta}(X_t, a) \partial_{\theta_i} \pi_\theta(a|X_t) \right].$$

This is true for every dimension θ_i so we conclude that:

$$\partial_\theta V^{\pi_\theta}(x) = \mathbb{E}_{\tau^{\pi_\theta} \sim \mathcal{T}(x, \pi_\theta)} \left[\sum_{t \geq 0} \gamma^t \sum_{a \in \mathcal{A}} Q^{\pi_\theta}(X_t, a) \partial_\theta \pi_\theta(a|X_t) \right].$$

Value Trick or Baseline Trick (Variance Reduction)

$\forall \theta \in \mathbb{R}^{\mathcal{N}}, \forall V \in \mathbb{R}^{\mathcal{X}}, \forall x \in \mathcal{X}$:

$$\partial_{\theta} V^{\pi_{\theta}}(x) = \mathbb{E}_{\tau^{\pi_{\theta}} \sim \mathcal{T}(x, \pi_{\theta})} \left[\sum_{t \geq 0} \gamma^t \sum_{a \in \mathcal{A}} (Q^{\pi_{\theta}}(X_t, a) - V(X_t)) \partial_{\theta} \pi_{\theta}(a|X_t) \right].$$

In particular we can choose $V = V^{\pi_{\theta}}$ and then have:

$$\partial_{\theta} V^{\pi_{\theta}}(x) = \mathbb{E}_{\tau^{\pi_{\theta}} \sim \mathcal{T}(x, \pi_{\theta})} \left[\sum_{t \geq 0} \gamma^t \sum_{a \in \mathcal{A}} (Q^{\pi_{\theta}}(X_t, a) - V^{\pi_{\theta}}(X_t)) \partial_{\theta} \pi_{\theta}(a|X_t) \right].$$

Exercise

Prove the Value Trick or Baseline Trick. Use the fact that $\sum_{a \in \mathcal{A}} \pi_{\theta}(a|x) = 1$.

The Log-Trick (One-action Evaluation)

Under the hypothesis $\forall x \in \mathcal{X}, \text{Supp } \pi_\theta(\cdot|x) = \mathcal{A}$, we have $\forall \theta \in \mathbb{R}^{\mathcal{N}}, \forall x \in \mathcal{X}$:

$$\partial_\theta V^{\pi_\theta}(x) = \mathbb{E}_{\tau^{\pi_\theta} \sim \mathcal{T}(x, \pi_\theta)} \left[\sum_{t \geq 0} \gamma^t Q^{\pi_\theta}(X_t, A_t) \partial_\theta \ln(\pi_\theta(A_t|X_t)) \right].$$

Exercise

Prove the Log trick. Remember what is the gradient of the \ln function.

The Sample Trick (Uses only Values)

$$\forall \theta \in \mathbb{R}^{\mathcal{N}}, \forall x \in \mathcal{X},$$

$$\partial_{\theta} V^{\pi_{\theta}}(x) = \mathbb{E}_{\tau^{\pi_{\theta}} \sim \mathcal{T}(x, \pi_{\theta})} \left[\sum_{t \geq 0} \gamma^t (r(X_t, A_t) + \gamma V^{\pi_{\theta}}(X_{t+1})) \partial_{\theta} \ln(\pi_{\theta}(A_t | X_t)) \right].$$

Exercise

Prove the Sample Trick.

Combination of Value, Log and Sample Tricks

$$\forall \theta \in \mathbb{R}^{\mathcal{N}}, \forall x \in \mathcal{X},:$$

$$\partial_{\theta} V^{\pi_{\theta}}(x) = \mathbb{E}_{\tau^{\pi_{\theta}} \sim \mathcal{T}(x, \pi_{\theta})} \left[\sum_{t \geq 0} \gamma^t (r(X_t, A_t) + \gamma V^{\pi_{\theta}}(X_{t+1}) - V^{\pi_{\theta}}(X_t)) \partial_{\theta} \ln(\pi_{\theta}(A_t | X_t)) \right].$$

Some Practical Considerations

In practice, we can't compute the full gradient $\partial_{\theta} V^{\pi_{\theta}}(x)$ for mainly two reasons:

- we can't perfectly estimate the value function $V^{\pi_{\theta}}$,
- we can't have access to full trajectories $\tau^{\pi_{\theta}} \sim \mathcal{T}(x, \pi_{\theta})$ of infinite length.

On-Policy Data

We have access to a batch $(\hat{\tau}_i^{\pi_{\theta}})_{i=1}^B$ of B partial trajectories of length H collected by the policy π_{θ} :

$$(\hat{\tau}_i^{\pi_{\theta}})_{i=1}^B = \left((X_{t,i}, A_{t,i}, R_{t,i}, X_{t+1,i})_{t=0}^{H-1} \right)_{i=1}^B :$$

$$X_{0,i} \sim \rho$$

$$A_{t,i} \sim \pi_{\theta}(\cdot | X_{t,i}),$$

$$R_{t,i} = r(X_{t,i}, A_{t,i}),$$

$$X_{t+1,i} \sim p(\cdot | X_{t,i}, A_{t,i}).$$

Neural Networks

- The policy (actor) network: \mathcal{N}_θ . The policy π_θ is simply going to be a softmax over the last activation layer of \mathcal{N}_θ .
- The value (critic) network: \mathcal{N}_ϕ . The value V_ϕ is simply going to be the last activation layer of \mathcal{N}_ϕ .

The Losses

The actor network loss is:

$$\mathcal{L}_A(\theta, (\hat{\tau}_i^{\pi_\theta})_{i=1}^B, \phi) = -\frac{1}{BH} \sum_{i=1}^B \sum_{t=0}^{H-1} (R_{t,i} + \gamma V_\phi(X_{t+1,i}) - V_\phi(X_t)) \log(\pi_\theta(A_{t,i}|X_{t,i})),$$

The critic network loss is:

$$\mathcal{L}_C((\hat{\tau}_i^{\pi_\theta})_{i=1}^B, \phi) = \frac{1}{BH} \sum_{i=1}^B \sum_{t=0}^{H-1} (R_{t,i} + \gamma V_\phi(X_{t+1,i}) - V_\phi(X_t))^2$$

Gradient of the Actor Network Loss

The gradient of the actor network loss is:

$$\partial_{\theta} \mathcal{L}_A(\theta, (\hat{\tau}_i^{\pi_{\theta}})_{i=1}^B, \phi) = -\frac{1}{BH} \sum_{i=1}^B \sum_{t=0}^{H-1} (R_{t,i} + \gamma V_{\phi}(X_{t+1,i}) - V_{\phi}(X_t)) \partial_{\theta} \log(\pi_{\theta}(A_{t,i}|X_{t,i})),$$

Remark

One can remark that if the value network V_{ϕ} perfectly estimates $V^{\pi_{\theta}}$ and we have infinite data $(B, H) = (\infty, \infty)$, then:

$$\partial_{\theta} \mathcal{L}_A(\theta, (\hat{\tau}_i^{\pi_{\theta}})_{i=1}^B, \phi) = -\mathbb{E}_{X \sim \rho} [\partial_{\theta} V^{\pi_{\theta}}(X)].$$

Therefore, the actor network gradient is a correct approximation of the average gradient over the starting distribution ρ , under the conditions of proper estimation of the value of the policy π_{θ} and enough data being collected.

Algorithm 5: Vanilla advantage actor-critic policy gradient.

Input : Actor neural network architecture: \mathcal{N}_θ , critic neural network architecture \mathcal{N}_ϕ , learning steps: $K \in \mathbb{N}$, learning rate: $\alpha \in \mathbb{R}$, batch size: $B \in \mathbb{N}$, discount factor: γ

Output: θ_{PG}

```
1 Initialize actor weights:  $\theta \leftarrow \text{Init}(\mathcal{N}_\theta)$ 
2 Initialize target weights:  $\phi \leftarrow \text{Init}(\mathcal{N}_\phi)$ 
3 for  $k \in \{0, \dots, K - 1\}$  do
4     Collect a batch of on-policy trajectories  $(\hat{\tau}_i^{\pi_\theta})_{i=1}^B$ 
5     Compute the actor loss:  $\mathcal{L}_A(\theta, (\hat{\tau}_i^{\pi_\theta})_{i=1}^B, \phi)$ 
6     Compute the critic loss:  $\mathcal{L}_C((\hat{\tau}_i^{\pi_\theta})_{i=1}^B, \phi)$ 
7     Update actor weights:  $\theta \leftarrow \theta - \alpha \partial_\theta \mathcal{L}_A(\theta, (\hat{\tau}_i^{\pi_\theta})_{i=1}^B, \phi)$ 
8     Update critic weights:  $\phi \leftarrow \phi - \alpha \partial_\phi \mathcal{L}_C((\hat{\tau}_i^{\pi_\theta})_{i=1}^B, \phi)$ 
9 end
10  $\theta_{\text{PG}} = \theta$ 
```

The Collapse Problem

The vanilla PG algorithm has the tendency to collapse quite fast to bad deterministic policies if the initial policy is not providing enough diverse trajectories. For instance, it is possible to rapidly find a local optimum and never move from there if there is no mechanism to explore new actions in known states. This is the classical trade-off between exploration and exploitation.

Entropy Regularization to avoid collapse

In policy-gradient algorithms, entropy regularization is simply introduced in the form of an additional loss function which goal is to maximize entropy of the policy along the collected trajectories:

$$\mathcal{L}_E(\theta, (\hat{\tau}_i^{\pi_\theta})_{i=1}^B) = -\mathcal{H}(\pi^\theta, (\hat{\tau}_i^{\pi_\theta})_{i=1}^B) = \frac{1}{BH|\mathcal{A}|} \sum_{i=1}^B \sum_{t=0}^{H-1} \sum_{a \in \mathcal{A}} \pi(a|X_{t,i}) \log(\pi(a|X_{t,i})).$$

Algorithm 6: Advantage actor-critic policy gradient.

Input : Actor neural network architecture: \mathcal{N}_θ , critic neural network architecture \mathcal{N}_ϕ , learning steps: $K \in \mathbb{N}$, learning rate: $\alpha \in \mathbb{R}$, batch size: $B \in \mathbb{N}$, discount factor: γ

Output: θ_{PG}

```
1 Initialize actor weights:  $\theta \leftarrow \text{Init}(\mathcal{N}_\theta)$ 
2 Initialize target weights:  $\phi \leftarrow \text{Init}(\mathcal{N}_\phi)$ 
3 for  $k \in \{0, \dots, K - 1\}$  do
4     Collect a batch of on-policy trajectories  $(\hat{\tau}_i^{\pi_\theta})_{i=1}^B$ 
5     Compute the actor loss:  $\mathcal{L}_A(\theta, (\hat{\tau}_i^{\pi_\theta})_{i=1}^B, \phi)$ 
6     Compute the critic loss:  $\mathcal{L}_C((\hat{\tau}_i^{\pi_\theta})_{i=1}^B, \phi)$ 
7     Compute the entropy regularization loss:  $\mathcal{L}_E(\theta, (\hat{\tau}_i^{\pi_\theta})_{i=1}^B)$ 
8     Compute the state-action value loss:  $\mathcal{L}_Q(\phi) \leftarrow \sum_{j=1}^B (Q_\phi(x_j, a_j) - t_j)^2$ 
9     Compute the policy loss:  $\mathcal{L}_{\text{KL}}(\theta) \leftarrow \sum_{j=1}^B \text{KL}(\pi_\theta(\cdot | x_j) || \pi_j)$ 
10    Update actor weights:  $\theta \leftarrow \theta - \alpha \partial_\theta (\mathcal{L}_A(\theta, (\hat{\tau}_i^{\pi_\theta})_{i=1}^B, \phi) + \mathcal{L}_E(\theta, (\hat{\tau}_i^{\pi_\theta})_{i=1}^B))$ 
11    Update critic weights:  $\phi \leftarrow \phi - \alpha \partial_\phi \mathcal{L}_C((\hat{\tau}_i^{\pi_\theta})_{i=1}^B, \phi)$ 
12 end
13  $\theta_{\text{PG}} = \theta$ 
```

Policy Gradient Improvements

Several Improvements can be made to the classical advantage Actor Critic PG algorithm:

- Asynchronous data collection: [A3C algorithm](#).
- Use of off-policy data: [VTrace algorithm](#).
- More sophisticated regularization schemes: [KL regularization](#).

Regularisation in Deep Reinforcement Learning

Motivations

- The learning is generally more stable and leads sometimes to higher performance at convergence.
- Regularisation is in fact a sound approach that benefits from the same theoretical guarantees than classical Dynamic Programming.
- General theoretical approach on regularisation will help us better understand the existing deep RL algorithms.

Choice of Regularisation

Penalize policies $\pi \in \Pi$ that derive too much from a baseline policy $\mu \in \Pi$. To do so, we choose the concept of Kulback-Leibler divergence:

$$\forall x \in \mathcal{X}, \text{KL}(\pi(\cdot|x) || \mu(\cdot|x)) = \sum_{a \in \mathcal{A}} \pi(a|x) \ln \left(\frac{\pi(a|x)}{\mu(a|x)} \right),$$

where by convention we choose that $0 \ln(0) = 0$ and $\ln(0) = -\infty$.

A new reward function

For each policy $\pi \in \Pi$, we define $r_{\mu,\lambda} \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$ such that:

$$\forall (x, a) \in \mathcal{X} \times \mathcal{A}, r_{\mu,\lambda}(x, a) = r(x, a) - \lambda \ln \left(\frac{\pi(a|x)}{\mu(a|x)} \right).$$

It is important to note that this reward depends also on the policy π . We can also define a regularised state-dependent reward $r_{\mu,\lambda}^\pi \in \mathbb{R}^{\mathcal{X}}$:

$$\begin{aligned} \forall x \in \mathcal{X}, r_{\mu,\lambda}^\pi(x) &= \sum_{a \in \mathcal{A}} \pi(a|x) \left[r(x, a) - \lambda \ln \left(\frac{\pi(a|x)}{\mu(a|x)} \right) \right], \\ &= \sum_{a \in \mathcal{A}} \pi(a|x) (r(x, a) - \lambda \sum_{a \in \mathcal{A}} \pi(a|x) \ln \left(\frac{\pi(a|x)}{\mu(a|x)} \right)), \\ &= r^\pi(x) - \lambda \text{KL}(\pi(\cdot|x) \parallel \mu(\cdot|x)). \end{aligned}$$

Constraint on the policy space

As $\text{KL}(\pi(\cdot|x)||\mu(\cdot|x))$ can be unbounded, we impose:

$$\forall x \in \mathcal{X}, \quad \text{Supp}(\pi(\cdot|x)) \subset \text{Supp}(\mu(\cdot|x)).$$

The set of policies that verify the previous condition is noted Π_μ and we have:

$$\forall \pi \in \Pi_\mu, \quad \|r_{\mu,\lambda}\| < \infty \text{ and } \|r_{\mu,\lambda}^\pi\| < \infty.$$

Regularised Discounted Cumulative returns

Regularised state discounted cumulative return:

$$Z_{\mathcal{X}}^{\mu,\lambda}((X_t)_{t \in \mathbb{N}}) = \sum_{t \geq 0} \gamma^t r_{\mu,\lambda}^\pi(X_t),$$

Regularised state-action discounted cumulative return:

$$Z_{\mathcal{X} \times \mathcal{A}}^{\mu,\lambda}((X_t, A_t)_{t \in \mathbb{N}}) = r(X_0, A_0) + \sum_{t \geq 1} \gamma^t r_{\mu,\lambda}^\pi(X_t, A_t).$$

How do we introduce the regularisation?

Expected cumulative returns

We define the regularised value (or V-)function and

$$V_{\mu,\lambda}^{\pi}(x) = \mathbb{E}_x^{\pi} \left[Z_{\mathcal{X}}^{\mu,\lambda}((X_t)_{t \in \mathbb{N}}) \right] = \mathbb{E}_x^{\pi} \left[\sum_{t \geq 0} \gamma^t r_{\mu,\lambda}^{\pi}(X_t) \right],$$

and the regularised state-action value (or Q-)function as:

$$Q_{\mu,\lambda}^{\pi}(x, a) = \mathbb{E}_{x,a}^{\pi} \left[Z_{\mathcal{X} \times \mathcal{A}}^{\mu,\lambda}((X_t, A_t)_{t \in \mathbb{N}}) \right] = \mathbb{E}_{x,a}^{\pi} \left[r(X_0, A_0) + \sum_{t \geq 1} \gamma^t r_{\mu,\lambda}^{\pi}(X_t, A_t) \right].$$

$V_{\mu,\lambda}^{\pi}(x)$ is the regularised discounted and expected cumulative return starting from state x and then following policy π . Similarly, $Q_{\mu,\lambda}^{\pi}(x, a)$ is the regularised discounted and expected cumulative return starting from state-action couple (x, a) and then following policy π .

Well-defined quantities

Prove that the quantities $V_{\mu,\lambda}^{\pi}(x)$ and $Q_{\mu,\lambda}^{\pi}(x, a)$ are well defined.

Theorem

For a given policy $\pi \in \Pi_\mu$, the value function $V_{\mu,\lambda}^\pi$ verifies:

$$\begin{aligned} V_{\mu,\lambda}^\pi(x) &= \sum_{a \in \mathcal{A}} \pi(a|x) r_{\mu,\lambda}(x, a) + \gamma \sum_{y \in \mathcal{X}} p_{\mathcal{X}}^\pi(y|x) V_{\mu,\lambda}^\pi(y), \\ &= \sum_{a \in \mathcal{A}} \pi(a|x) r_{\mu,\lambda}(x, a) + \gamma \sum_{y \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(a|x) p(y|x, a) V_{\mu,\lambda}^\pi(y), \end{aligned}$$

and the action value function $Q_{\mu,\lambda}^\pi$ verifies:

$$Q_{\mu,\lambda}^\pi(x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{X}} \sum_{b \in \mathcal{A}} p(y|x, a) \pi(b|y) \left[Q_{\mu,\lambda}^\pi(y, b) - \lambda \ln \left(\frac{\pi(b|y)}{\mu(b|y)} \right) \right].$$

Proof

Prove the theorem as an exercise.

Regularised evaluation operator for V -functions

The regularised evaluation operator $B_{\mu,\lambda}^{\pi} \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$ is defined as follows:

$$\begin{aligned} B_{\mu,\lambda}^{\pi}[V](x) &= \sum_{a \in \mathcal{A}} \pi(a|x) \left[r_{\mu,\lambda}(x, a) + \gamma \sum_{y \in \mathcal{X}} p(y|x, a) V(y) \right], \\ &= \sum_{a \in \mathcal{A}} \pi(a|x) \left[r(x, a) - \lambda \ln \left(\frac{\pi(a|x)}{\mu(a|x)} \right) + \gamma \sum_{y \in \mathcal{X}} p(y|x, a) V(y) \right]. \end{aligned}$$

Regularised evaluation operator for Q -functions

The regularised evaluation operator $T_{\mu,\lambda}^{\pi} \in \mathbb{R}^{\mathcal{X} \times \mathcal{A} \times \mathcal{X} \times \mathcal{A}}$ is defined as follows:

$$T_{\mu,\lambda}^{\pi}[Q](x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{X}} p(y|x, a) \sum_{b \in \mathcal{A}} \pi(b|y) \left[Q(y, b) - \lambda \ln \left(\frac{\pi(b|y)}{\mu(b|y)} \right) \right].$$

How should we define the optimality operators?

Optimality Operator Properties

Create $T_{\mu,\lambda}^*$ that verifies the same properties than T^* :

- Monotonicity.
- Domination over the evaluation operator.
- Reachability: for a given $Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$, there exists $\pi \in \Pi$ such that $T^\pi Q = T^* Q$ ($\pi \in \mathcal{G}(Q)$ verifies this).
- γ -contraction.

Strategy

The domination and reachability properties of T^* comes from the fact that:

- the mean operation $\sum_{b \in \mathcal{A}} \pi(b|y) Q(y, b)$ in T^π is replaced by a max operation $\max_{b \in \mathcal{A}} Q(y, b)$ in T^*

We are going to follow the same strategy and replace:

- the mean operation $\sum_{b \in \mathcal{A}} \pi(b|y) \left[Q(y, b) - \lambda \ln \left(\frac{\pi(b|y)}{\mu(b|y)} \right) \right]$ by a max operation $\max_{\delta \in \Delta_{\mathcal{A}}} \sum_{b \in \mathcal{A}} \delta(b) \left[Q(y, b) - \lambda \ln \left(\frac{\delta(b)}{\mu(b|y)} \right) \right]$.

How should we define the optimality operators?

Explaining the strategy.

We decided to take the maximum over the set of probability distributions over \mathcal{A} and not only over the set of actions because it is a more general concept adapted to stochastic policies. Indeed, in the regularised case, maximum over the set of probability distributions over \mathcal{A} is not always equal to the maximum over the set of actions which is not true for the classical case where you have:

$$\forall Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}, \forall x \in \mathcal{X}, \quad \max_{a \in \mathcal{A}} Q(x, a) = \max_{\delta \in \Delta_{\mathcal{A}}} \sum_{a \in \mathcal{A}} \delta(a) Q(x, a).$$

Remark.

This explains why we can always find deterministic policies which are optimal in the classical case. This will not be true anymore in the regularised case.

Regularised optimality operator for Q -functions

$$T_{\mu,\lambda}^*[Q](x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{X}} p(y|x, a) \max_{\delta \in \Delta_A} \sum_{b \in \mathcal{A}} \delta(b) \left[Q(y, b) - \lambda \ln \left(\frac{\delta(b)}{\mu(b|y)} \right) \right].$$

Regularised optimality operator for V -functions

$$B_{\mu,\lambda}^*[V](x) = \max_{\delta \in \Delta_A} \sum_{a \in \mathcal{A}} \delta(a) \left[r(x, a) - \lambda \ln \left(\frac{\delta(a)}{\mu(a|y)} \right) + \gamma \sum_{y \in \mathcal{X}} p(y|x, a) V(y) \right].$$

Notations.

To each function $V \in \mathbb{R}^{\mathcal{X}}$ we associate a function $Q_V \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$ such that:

$$Q_V(x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{X}} p(y|x, a) V(y).$$

Moreover, we have for $\pi \in \Pi$:

$$\sum_{a \in \mathcal{A}} \pi(a|x) Q_V(x, a) = T^\pi[V](x),$$

One can also remark that $Q_{V^\pi} = Q^\pi$.

Notations.

In addition, let \mathcal{S} be a finite space, $f \in \mathbb{R}^{\mathcal{S}}$ and $g \in \mathbb{R}^{\mathcal{S}}$, we define the dot product on \mathcal{S} as:

$$\langle f, g \rangle_{\mathcal{S}} = \sum_{s \in \mathcal{S}} f(s)g(s).$$

Regularised evaluation operator for V -functions.

$$\begin{aligned} B_{\mu,\lambda}^{\pi}[V](x) &= \sum_{a \in \mathcal{A}} \pi(a|x) \left[r(x, a) - \lambda \ln \left(\frac{\pi(a|x)}{\mu(a|x)} \right) + \gamma \sum_{y \in \mathcal{X}} p(y|x, a) V(y) \right], \\ &= \sum_{a \in \mathcal{A}} \pi(a|x) Q_V(x, a) - \lambda \text{KL}(\pi(\cdot|x) || \mu(\cdot|x)), \\ &= \langle \pi(\cdot|x), Q_V(x, \cdot) \rangle_{\mathcal{A}} - \lambda \text{KL}(\pi(\cdot|x) || \mu(\cdot|x)), \end{aligned}$$

Regularised optimality operator for V -functions.

$$B_{\mu,\lambda}^*[V](x) = \max_{\delta \in \Delta_{\mathcal{A}}} [\langle \delta, Q_V(x, \cdot) \rangle_{\mathcal{A}} - \lambda \text{KL}(\delta || \mu(\cdot|x))].$$

Regularised evaluation operator for Q -functions.

$$\begin{aligned}T_{\mu,\lambda}^{\pi}[Q](x,a) &= r(x,a) + \gamma \sum_{y \in \mathcal{X}} p(y|x,a) \sum_{b \in \mathcal{A}} \pi(b|y) \left[Q(y,b) - \lambda \ln \left(\frac{\pi(b|y)}{\mu(b|y)} \right) \right], \\&= r(x,a) + \gamma \sum_{y \in \mathcal{X}} p(y|x,a) [\langle \pi(\cdot|y), Q(y, \cdot) \rangle_{\mathcal{A}} - \lambda \text{KL}(\pi(\cdot|y) || \mu(\cdot|y))].\end{aligned}$$

Regularised optimality operator for Q -functions.

$$T_{\mu,\lambda}^{\star}[Q](x,a) = r(x,a) + \gamma \sum_{y \in \mathcal{X}} p(y|x,a) \max_{\delta \in \Delta_{\mathcal{A}}} [\langle \delta, Q(y, \cdot) \rangle_{\mathcal{A}} - \lambda \text{KL}(\delta || \mu(\cdot|y))].$$

Property

Monotonicity. The regularised evaluation and optimality operators for V-functions are monotonous. Let $V_1 \in \mathbb{R}^X$ and $V_2 \in \mathbb{R}^X$ such that $V_1 \leq V_2$ and let $\pi \in \Pi_\mu$, then

$$\begin{aligned} B_{\mu,\lambda}^\pi[V_1] &\leq B_{\mu,\lambda}^\pi[V_2], \\ B_{\mu,\lambda}^*[V_1] &\leq B_{\mu,\lambda}^*[V_2]. \end{aligned}$$

The same goes for Q-functions.

Property

Domination of the regularised optimality operators. The optimality operator for V-functions dominates the evaluation operator. Let $V \in \mathbb{R}^X$ and $\pi \in \Pi$, then:

$$B_{\mu,\lambda}^\pi V \leq B_{\mu,\lambda}^* V.$$

The same goes for Q-functions.

Theorem

Fixed-points of the regularised evaluation operators.

The operator $B_{\mu,\lambda}^\pi$ is a γ -contraction with unique fixed-point $V_{\mu,\lambda}^\pi$. Similarly, the operator $T_{\mu,\lambda}^\pi$ is a γ -contraction with unique fixed-point $Q_{\mu,\lambda}^\pi$:

$$\begin{aligned} B_{\mu,\lambda}^\pi[V_{\mu,\lambda}^\pi] &= V_{\mu,\lambda}^\pi, \\ T_{\mu,\lambda}^\pi[Q_{\mu,\lambda}^\pi] &= Q_{\mu,\lambda}^\pi. \end{aligned}$$

Theorem

Fixed-points of the regularised optimality operators.

The operator $B_{\mu,\lambda}^$ is a γ -contraction, its unique fixed-point is noted $V_{\mu,\lambda}^*$ and called the optimal value function. Similarly, the operator $T_{\mu,\lambda}^*$ is a γ -contraction, its unique fixed-point is noted $Q_{\mu,\lambda}^*$ and called the optimal action-value function:*

$$\begin{aligned} B_{\mu,\lambda}^*[V_{\mu,\lambda}^*] &= V_{\mu,\lambda}^*, \\ T_{\mu,\lambda}^*[Q_{\mu,\lambda}^*] &= Q_{\mu,\lambda}^*. \end{aligned}$$

Corollary

Relations between $V_{\mu,\lambda}^\pi$ and $Q_{\mu,\lambda}^\pi$ and between $V_{\mu,\lambda}^$ and $Q_{\mu,\lambda}^*$.*

For a given policy $\pi \in \Pi_\mu$,

$$\forall x \in \mathcal{X}, V_{\mu,\lambda}^\pi(x) = \sum_{a \in \mathcal{A}} \pi(a|x) Q_{\mu,\lambda}^\pi(x, a) - \lambda KL(\pi(\cdot|x) || \mu(\cdot|x)),$$

and:

$$\forall x \in \mathcal{X}, V_{\mu,\lambda}^*(x) = \max_{\delta \in \Delta_{\mathcal{A}}} [\langle \delta, Q_{\mu,\lambda}^*(x, \cdot) \rangle_{\mathcal{A}} - \lambda KL(\delta || \mu(\cdot|x))].$$

Theorem

Majorants of the value functions.

$V_{\mu,\lambda}^$ is a majorant of the set $\{V_{\mu,\lambda}^\pi\}_{\pi \in \Pi_\mu}$, respectively $Q_{\mu,\lambda}^*$ is a majorant of the set $\{Q_{\mu,\lambda}^\pi\}_{\pi \in \Pi_\mu}$:*

$$\forall \pi \in \Pi_\mu, V_{\mu,\lambda}^\pi \leq V_{\mu,\lambda}^*,$$

$$\forall \pi \in \Pi_\mu, Q_{\mu,\lambda}^\pi \leq Q_{\mu,\lambda}^*.$$

Remark

Regularised maximum and argmaximum

Let S be a finite state, $q \in \mathbb{R}^S$ and $\eta \in \mathbb{R}^S$, then the following regularised maximum $\max_{\delta \in \Delta_S} [\langle \delta, q \rangle_S - KL(\delta, \eta)]$ is such that:

$$\max_{\delta \in \Delta_S} [\langle \delta, q \rangle_S - \lambda KL(\delta, \eta)] = \lambda \ln \left(\sum_{s \in S} \eta(s) \exp \left(\frac{q(s)}{\lambda} \right) \right),$$

and the argmaximum $\mathcal{G}_{\eta, \lambda}(q) = \arg \max_{\delta \in \Delta_S} [\langle \delta, q \rangle_S - KL(\delta, \eta)]$ is unique and $\mathcal{G}_{\eta, \lambda}(q) \in \Delta_S$ is such that:

$$\forall s \in S, \quad \mathcal{G}_{\eta, \lambda}(q)(s) = \frac{\eta(s) \exp \left(\frac{q(s)}{\lambda} \right)}{\sum_{s' \in S} \eta(s') \exp \left(\frac{q(s')}{\lambda} \right)}.$$

Proof

Prove the remark's statement as an exercise.

Theorem

Existence and construction of optimal stochastic policies.

A stochastic policy $\pi \in \Pi_\mu$ is optimal if and only if $V_{\mu,\lambda}^\pi = V_{\mu,\lambda}^$. A unique regularised optimal stochastic policy exists and can be explicitly constructed from $Q_{\mu,\lambda}^*$ or $Q_{\mu,\lambda}^\pi$:*

$$V_{\mu,\lambda}^\pi = V_{\mu,\lambda}^* \iff \forall x \in \mathcal{X}, \pi(\cdot|x) = \arg \max_{\delta \in \Delta_{\mathcal{A}}} [\langle \delta, Q_{\mu,\lambda}^*(x, \cdot) \rangle_{\mathcal{A}} - \lambda KL(\delta || \mu(\cdot|y))] ,$$

$$\iff \forall (x, a) \in \mathcal{X} \times \mathcal{A}, \pi(a|x) = \frac{\mu(a|x) \exp \left(\frac{Q_{\mu,\lambda}^*(x,a)}{\lambda} \right)}{\sum_{b \in \mathcal{A}} \mu(b|x) \exp \left(\frac{Q_{\mu,\lambda}^*(x,b)}{\lambda} \right)}$$

$$\iff \forall x \in \mathcal{X}, \pi(\cdot|x) = \arg \max_{\delta \in \Delta_{\mathcal{A}}} [\langle \delta, Q_{\mu,\lambda}^\pi(x, \cdot) \rangle_{\mathcal{A}} - \lambda KL(\delta || \mu(\cdot|y))] ,$$

$$\iff \forall (x, a) \in \mathcal{X} \times \mathcal{A}, \pi(a|x) = \frac{\mu(a|x) \exp \left(\frac{Q_{\mu,\lambda}^\pi(x,a)}{\lambda} \right)}{\sum_{b \in \mathcal{A}} \mu(b|x) \exp \left(\frac{Q_{\mu,\lambda}^\pi(x,b)}{\lambda} \right)} .$$

Remark

For a function $Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$, $\mathcal{G}_{\mu, \lambda}(Q)$ is the unique regularised greedy policy:

$$\forall x \in \mathcal{X}, \mathcal{G}_{\mu, \lambda}(Q)(\cdot|x) = \arg \max_{\delta \in \Delta_A} [\langle \delta, Q(x, \cdot) \rangle_{\mathcal{A}} - \lambda KL(\delta || \mu(\cdot|y))].$$

In addition, we have:

$$\begin{aligned} & \langle \mathcal{G}_{\mu, \lambda}(Q)(\cdot|x), Q(x, \cdot) \rangle_{\mathcal{A}} - \lambda KL(\mathcal{G}_{\mu, \lambda}(Q)(\cdot|x) || \mu(\cdot|y)), \\ &= \max_{\delta \in \Delta_A} [\langle \delta, Q(x, \cdot) \rangle_{\mathcal{A}} - \lambda KL(\delta || \mu(\cdot|y))] \end{aligned}$$

Therefore by definition :

$$\pi = \mathcal{G}_{\mu, \lambda}(Q) \implies T_{\mu, \lambda}^* Q = T_{\mu, \lambda}^{\pi} Q.$$

Finally, the optimality theorem becomes: π is optimal in the regularised case $\iff \pi = \mathcal{G}_{\mu, \lambda}(Q^*)$
 $\iff \pi = \mathcal{G}_{\mu, \lambda}(Q^{\pi}).$

Theorem

The Regularised Greedy Policy Improvement.

Let $\pi \in \Pi_\mu$ be a stochastic policy, then the regularised-greedy policy with respect to $Q_{\mu,\lambda}^\pi$ noted $\pi_{\mathcal{G}_{\mu,\lambda}} = \mathcal{G}_{\mu,\lambda}(Q_{\mu,\lambda}^\pi)$ is such that:

$$Q_{\mu,\lambda}^\pi \leq Q_{\mu,\lambda}^{\pi_{\mathcal{G}_{\mu,\lambda}}}.$$

In addition:

$$Q_{\mu,\lambda}^\pi = Q_{\mu,\lambda}^{\pi_{\mathcal{G}_{\mu,\lambda}}} \iff Q_{\mu,\lambda}^\pi = Q_{\mu,\lambda}^* \implies \pi_{\mathcal{G}_{\mu,\lambda}} \text{ is optimal in the regularised case.}$$

Conclusion of the 5 theorems

We can explicitly construct a regularised optimal policy from the regularised optimal action-value function $Q_{\mu,\lambda}^*$. In addition, as $Q_{\mu,\lambda}^*$ is the fixed point of a γ -contraction operator, there is also an explicit iteration scheme to build $Q_{\mu,\lambda}^*$ called Regularised Value Iteration (RVI).

Regularised Value Iteration

$$\text{Regularised Value Iteration: } \left\{ \begin{array}{ll} Q_0 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}} & \text{Initialisation} \\ \forall k \in \mathbb{N}, \quad Q_{k+1} = T_{\mu,\lambda}^* Q_k & \text{Recurrence} \end{array} \right.$$

As $T_{\mu,\lambda}^*$ is a γ -contraction operator, we have $\lim_{k \rightarrow \infty} Q_k = Q_{\mu,\lambda}^*$.

Rewriting of RVI

Moreover, knowing that $\pi = \mathcal{G}_{\mu,\lambda}(Q) \implies T_{\mu,\lambda}^* Q = T_{\mu,\lambda}^\pi Q$, one can rewrite Regularised Value Iteration:

$$\text{Regularised Value Iteration: } \left\{ \begin{array}{ll} Q_0 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}} & \text{Initialisation} \\ \forall k \in \mathbb{N}, \quad \pi_{k+1} = \mathcal{G}_{\mu,\lambda}(Q_k) & \text{Greedy-step} \\ \forall k \in \mathbb{N}, \quad Q_{k+1} = T_{\mu,\lambda}^{\pi_{k+1}} Q_k & \text{Evaluation} \end{array} \right.$$

Convergence of RVI

One can easily show that $\lim_{k \rightarrow \infty} Q^{\pi_k} = Q^*$ and even more precisely:

$$\forall k \geq 1, \quad \|Q_{\mu,\lambda}^* - Q^{\pi_k}\|_\infty \leq \frac{2\gamma^k \|Q_{\mu,\lambda}^*\|_\infty}{(1 - \gamma)},$$

under the assumption that we start from $Q_0 = 0$.

Regularised Policy Iteration

$$\text{Regularised Policy Iteration: } \left\{ \begin{array}{ll} Q_0 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}} & \text{Initialisation} \\ \forall k \in \mathbb{N}, \pi_{k+1} = \mathcal{G}_{\mu, \lambda}(Q_k) & \text{Greedy-step} \\ \forall k \in \mathbb{N}, Q_{k+1} = (T_{\mu, \lambda}^{\pi_{k+1}})^{\infty} Q_k & \text{Evaluation} \end{array} \right.$$

In PI, the evaluation is a full evaluation $Q_{k+1} = (T_{\mu, \lambda}^{\pi_{k+1}})^{\infty} Q_k$ which means that $Q_{k+1} = Q_{\mu, \lambda}^{\pi_{k+1}}$. Because of the greedy policy improvement theorem, we have:

$$Q_1 = Q_{\mu, \lambda}^{\pi_1} \leq Q_2 = Q_{\mu, \lambda}^{\pi_2} \leq Q_3 = Q_{\mu, \lambda}^{\pi_3} \dots$$

where each step is strictly improving unless optimality is reached.

Regularised Modified Policy Iteration

$$\text{RMPI: } \left\{ \begin{array}{ll} Q_0 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}} & \text{Initialisation} \\ \forall k \in \mathbb{N}, \quad \pi_{k+1} = \mathcal{G}_{\mu, \lambda}(Q_k) & \text{Greedy-step} \\ \forall k \in \mathbb{N}, \quad Q_{k+1} = (T_{\mu, \lambda}^{\pi_k})^m Q_k & \text{Evaluation} \end{array} \right.$$

Here the evaluation is $Q_{k+1} = (T_{\mu, \lambda}^{\pi_{k+1}})^m Q_k$ with $m \in \overline{\mathbb{N}^*}$. This trivially generalizes to RVI with $m = 1$ and to RPI with $m = \infty$. One can show that $\lim_{k \rightarrow \infty} Q^{\pi_k} = Q_{\mu, \lambda}^*$.

Why?

Regularised DP schemes converge towards $Q_{\mu,\lambda}^*$ which is not Q^* . Therefore, we would like to have an algorithm that still benefits from the regularisation properties and converge towards Q^* .

Idea

One possible avenue to build such an algorithm is to change the baseline policy μ to a policy closer to the online policy π_{k+1} along the learning. Such an algorithm is an adaptive regularised DP scheme. The idea is pretty simple and simply consists in changing μ to π_k at each step k of Regularised Modified Policy Iteration.

Mirror Descent Modified Policy Iteration

It is initialised with $Q_0 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$ and $\pi_0 = \mathcal{U}$ where \mathcal{U} is the uniform policy.

$$\text{MDMPI: } \left\{ \begin{array}{ll} Q_0 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}, \quad \pi_0 = \mathcal{U} & \text{Initialisation} \\ \forall k \in \mathbb{N}, \quad \pi_{k+1} = \mathcal{G}_{\pi_k, \lambda}(Q_k) & \text{Greedy-step} \\ \forall k \in \mathbb{N}, \quad Q_{k+1} = (T_{\pi_k, \lambda}^{\pi_{k+1}})^m Q_k & \text{Evaluation} \end{array} \right.$$

Convergence Guarantees

For $m = 1$, one can show that:

$$\forall k \geq 1, \quad \|Q^* - Q^{\pi_k}\|_{\infty} \leq \frac{4\|r\|_{\infty}}{(1 - \gamma)^2 k},$$

Approximate MDMP

$$\text{Approximate MDMP: } \left\{ \begin{array}{ll} Q_0 \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}, & \pi_0 = \mathcal{U} \\ \forall k \in \mathbb{N}, & \pi_{k+1} = \mathcal{G}_{\pi_k, \lambda}(Q_k) + \epsilon'_k \\ \forall k \in \mathbb{N}, & Q_{k+1} = T_{\pi_k, \lambda}^{\pi_{k+1}} Q_k + \epsilon_k \end{array} \right. \quad \begin{array}{l} \text{Initialisation} \\ \text{Greedy-step} \\ \text{Evaluation} \end{array}$$

High Level Idea

We approximate the Greedy and Evaluation steps with well-known Machine Learning algorithms. We already know that we can approximate the evaluation step with a regression algorithm. We approximate the greedy-step with a KL minimization (or cross-entropy minimization) algorithm. It is important to remark that contrarily to AVI, the greedy-step at step k can't be executed perfectly because even if it is closed-form it depends on the policy at step $k - 1$. Therefore, we need a target policy network to store the policy at step $k - 1$ to build the targets to learn the policy at step k that is approximated by an online policy network.

Policy Update

Let us suppose that, at step k , we have a target state-action value network $Q_{\phi'}$, a target policy network $\pi_{\theta'}$ and a replay buffer D . Our goal is to train the online policy network π_{θ} to be as close as possible to our closed-form target $\mathcal{G}_{\pi_{\theta'}, \lambda}(Q_{\phi'})$. We use KL-divergence to build our loss on a batch of states sampled from the replay buffer, $\mathcal{B} = \{(x_j)\}_{j=1}^B$ with $B \in \mathbb{N}^*$ the batch size:

$$\begin{aligned}\mathcal{L}_{\text{KL}}(\theta) &= \sum_{j=1}^B \text{KL}(\pi_{\theta}(\cdot|x_j) || \mathcal{G}_{\pi_{\theta'}, \lambda}(Q_{\phi'}) (\cdot|x_j)), \\ &= \sum_{j=1}^B \text{KL} \left(\pi_{\theta}(\cdot|x_j) || \frac{\pi_{\theta'}(\cdot|x_j) \exp\left(\frac{Q_{\phi'}(x_j, \cdot)}{\lambda}\right)}{\sum_{b \in \mathcal{A}} \pi_{\theta'}(b|x_j) \exp\left(\frac{Q_{\phi'}(x_j, b)}{\lambda}\right)} \right).\end{aligned}$$

Policy Evaluation

The evaluation step use the targets computed with the operator $T_{\pi_{\theta'}, \lambda}^{\mathcal{G}_{\pi_{\theta'}, \lambda}(Q_{\phi'})}$.

Algorithm 7: Greedy Step of Approximate MDMPI.

Input : online policy weights: $\theta \in \mathbb{R}^{\mathcal{N}_\theta}$, target policy weights: $\theta' \in \mathbb{R}^{\mathcal{N}_\theta}$, target state-action value network weights $\phi' \in \mathbb{R}^{\mathcal{N}_\phi}$, Dataset of transitions: \mathcal{D} , update period: $I \in \mathbb{N}$, learning rate: $\alpha \in \mathbb{R}$, batch size: $B \in \mathbb{N}$, discount factor: γ

Output: θ, θ'

```

1 for  $i \in \{0, \dots, I - 1\}$  do
2   Draw uniformly a batch  $\mathcal{B} = \{(x_j)\}_{j=1}^B$  from  $\mathcal{D}$ 
3   Compute the targets:  $\forall 1 \leq j \leq B, \quad \pi_j \leftarrow \frac{\pi_{\theta'}(\cdot | x_j) \exp\left(\frac{Q_{\phi'}(x_j, \cdot)}{\lambda}\right)}{\sum_{b \in \mathcal{A}} \pi_{\theta'}(b | x_j) \exp\left(\frac{Q_{\phi'}(x_j, b)}{\lambda}\right)}$ 
4   Compute the loss:  $\mathcal{L}_{\text{KL}}(\theta) \leftarrow \sum_{j=1}^B \text{KL}(\pi_\theta(\cdot | x_j) || \pi_j)$ 
5   Update the online weights:  $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\text{KL}}(\theta)$ 
6 end
7  $\theta' \leftarrow \theta$ 

```

Algorithm 8: Learning process of Approximate MDMPI.

Input : Policy network architecture: \mathcal{N}_θ , State-action value network architecture: \mathcal{N}_ϕ , learning steps: $K \in \mathbb{N}$, update period: $l \in \mathbb{N}$, learning rate: $\alpha \in \mathbb{R}$, batch size: $B \in \mathbb{N}$, discount factor: γ

Shared: replay buffer: \mathcal{D}

Output: θ_{MDMPI}

```
1 Initialize online policy weights:  $\theta \leftarrow \text{Init}(\mathcal{N}_\theta)$ 
2 Initialize target policy weights:  $\theta' \leftarrow \text{Init}(\mathcal{N}_\theta)$ 
3 Initialize online state-action value weights:  $\phi \leftarrow \text{Init}(\mathcal{N}_\phi)$ 
4 Initialize target state-action value weights:  $\phi' \leftarrow \text{Init}(\mathcal{N}_\phi)$ 
5 for  $k \in \{0, \dots, K-1\}$  do
6   for  $i \in \{0, \dots, l-1\}$  do
7     Draw uniformly a batch  $\mathcal{B} = \{(x_j, a_j, r_j, y_j)\}_{j=1}^B$  from  $\mathcal{D}$ 
8     Compute the state-action value targets:  $\forall 1 \leq j \leq B, \quad t_j \leftarrow r_j + \gamma \lambda \ln \left( \sum_{b \in \mathcal{A}} \pi_{\theta'}(b|y_j) \exp \left( \frac{Q_{\phi'}(y_j, b)}{\lambda} \right) \right)$ 
9     Compute the policy targets:  $\forall 1 \leq j \leq B, \quad \pi_j \leftarrow \frac{\pi_{\theta'}(\cdot|x_j) \exp \left( \frac{Q_{\phi'}(x_j, \cdot)}{\lambda} \right)}{\sum_{b \in \mathcal{A}} \pi_{\theta'}(b|x_j) \exp \left( \frac{Q_{\phi'}(x_j, b)}{\lambda} \right)}$ 
10    Compute the state-action value loss:  $\mathcal{L}_Q(\phi) \leftarrow \sum_{j=1}^B (Q_\phi(x_j, a_j) - t_j)^2$ 
11    Compute the policy loss:  $\mathcal{L}_{\text{KL}}(\theta) \leftarrow \sum_{j=1}^B \text{KL}(\pi_\theta(\cdot|x_j) || \pi_j)$ 
12    Update the online state-action weights:  $\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}_Q(\phi)$ 
13    Update the online policy weights:  $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\text{KL}}(\theta)$ 
14  end
15  Update the target state-action value weights:  $\phi' \leftarrow \phi$ 
16  Update the target policy weights:  $\theta' \leftarrow \theta$ 
17 end
18  $\theta_{\text{MDMPI}} = \theta \quad \phi_{\text{MDMPI}} = \phi$ 
```

Off-Policy Reinforcement Learning

Motivations

Learning from any agent that collects transitions/trajectories through interactions with the environment.

DQN

DQN is an example of an off-policy reinforcement learning algorithm.

Why aren't we satisfied with DQN?

- DQN is a one-step RL algorithm. DQN uses only the reward at the current step and the bootstrapped action-value at the next state to compute its targets. This means that the targets are probably very far off from the expected-cumulative return of the current greedy policy, at least at the beginning of the learning
- With a one-step algorithm, it may take a long time to back-propagate this information. In that regard, DQN is often referred as an algorithm with high bias.

Reducing Bias but Increasing Variance

- To reduce this bias, an obvious choice is to consider an n -step RL algorithm where an n -step discounted cumulative reward is computed before adding the bootstrapped action-value at step $n + 1$.
- As n increases, the less biased the algorithm is going to be. However, the problem with n -step algorithms is that they are by nature on-policy, as the n -step reward depends on the policy that collected it, and their variance increases as n increases.

Goal of the chapter

Present new operators and algorithms that tackle the bias-variance trade-off and can handle off-policiness. In particular we will primarily focus on Retrace [Munos et al., 2016].

One-step Bellman Operator T^π

$$\forall Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}, \quad T^\pi Q = r + \gamma P_{\mathcal{X} \times \mathcal{A}}^\pi Q.$$

To make notations more concise, we note in the remaining of this chapter $P_{\mathcal{X} \times \mathcal{A}}^\pi = P^\pi$, therefore:

$$\forall Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}, \quad T^\pi Q = r + \gamma P^\pi Q.$$

n -step Bellman Operator, $(T^\pi)^n$ with $n \in \mathbb{N}^*$

$$\forall Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}, \quad (T^\pi)^n Q = \sum_{k=0}^{n-1} (\gamma P^\pi)^k r + (\gamma P^\pi)^n Q.$$

Remark

Estimates of $T^\pi Q$ are in expectation far off Q^π but with low-variance and estimates of $(T^\pi)^n Q$ are closer in expectation to Q^π but with high variance.

Geometric-averaged Bellman operator T_λ^π , with parameter $\lambda \in [0, 1]$

$$\forall \lambda \in [0, 1), \quad \forall Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}, \quad T_\lambda^\pi Q = (1 - \lambda) \sum_{n \geq 0} \lambda^n (T^\pi)^{n+1} Q.$$

Theorem

For all $\lambda \in [0, 1)$ and all $Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$, the quantity $T_\lambda^\pi Q = (1 - \lambda) \sum_{n \geq 0} \lambda^n (T^\pi)^{n+1} Q$ is well defined and such that:

$$\forall \lambda \in [0, 1), \quad \forall Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}, \quad T_\lambda^\pi Q = Q + (I - \lambda \gamma P^\pi)^{-1} (T^\pi Q - Q).$$

Remark

An interesting point is that even if the quantity $T_\lambda^\pi Q = Q + (I - \lambda \gamma P^\pi)^{-1} (T^\pi Q - Q)$ was not originally defined for $\lambda = 1$, we can still define it as:

$$T_{\lambda=1}^\pi Q = Q + (I - \gamma P^\pi)^{-1} (T^\pi Q - Q).$$

Theorem

For all $Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$:

$$T_{\lambda=1}^{\pi} Q = (T^{\pi})^{\infty} Q = Q^{\pi}.$$

Remark

Therefore the operator T_{λ}^{π} ranges from $T_{\lambda=0}^{\pi} = T^{\pi}$ to $T_{\lambda=1}^{\pi} = (T^{\pi})^{\infty}$ and otherwise mixes the different n -step operators with the weights $(1 - \lambda)\lambda^n$. Moreover, as the other Bellman operators, T_{λ}^{π} is a γ -contraction with fixed point Q^{π} .

Theorem

For all $\lambda \in [0, 1)$, Q^{π} is the unique fixed point of T_{λ}^{π} .

Markovian Form

$$T_{\lambda}^{\pi} Q(x, a) = Q(x, a) + \mathbb{E}_{x,a}^{\pi} \left[\sum_{t \geq 0} (\lambda \gamma)^t [T^{\pi} Q(X_t, A_t) - Q(X_t, A_t)] \right].$$

Lemma

For all $Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$ and for all $\lambda \in [0, 1)$:

$$T_{\lambda}^{\pi} Q(x, a) = Q(x, a) + \mathbb{E}_{x,a}^{\pi} \left[\sum_{t \geq 0} (\lambda \gamma)^t \left[r(X_t, A_t) + \gamma \sum_{b \in \mathcal{A}} \pi(b|X_{t+1}) Q(X_{t+1}, b) - Q(X_t, A_t) \right] \right].$$

Remark

The operator T_{λ}^{π} is an on-policy multi-step operator that mixes different n -step operators. In the next section, we are going to see how we can generalize this operator to make it off-policy.

Importance Sampling

The main ingredient to build off-policy operators is to rely on the simplest tricks in the books! Let \mathcal{S} be a finite state space, $f \in \mathbb{R}^{\mathcal{S}}$, $\eta \in \Delta_{\mathcal{S}}$ and $\rho \in \Delta_{\mathcal{S}}$, then:

$$\mathbb{E}_{\rho}[f] = \mathbb{E}_{\eta} \left[\frac{\rho}{\eta} f \right],$$

under the hypothesis that $\text{Supp } \rho \subset \text{Supp } \eta$.

Application to Markov Chain

Let $(X_t, A_t)_{t \in \mathbb{N}}$ be the state-action process under policy π and starting from $(x, a) \in (\mathcal{X} \times \mathcal{A})$, μ a policy and $r \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$, then:

$$\mathbb{E}_{x,a}^{\pi} \left[\sum_{t \geq 0} \gamma^t r(X_t, A_t) \right] = r(x, a) + \mathbb{E}_{x,a}^{\mu} \left[\sum_{t \geq 1} \gamma^t \prod_{s=1}^t \frac{\pi(A_s | X_s)}{\mu(A_s | X_s)} r(X_t, A_t) \right],$$

under the constraint that $\pi \in \Pi_{\mu}$

Retrace Operator

The Retrace operator $\mathcal{R}_c^{\pi, \mu}$ is a multi-step off-policy operator that is a generalization of T_λ^π . Let $\pi \in \Pi$, $\mu \in \Pi$ and $c \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$, then $\forall Q \in \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$ and $\forall (x, a) \in \mathcal{X} \times \mathcal{A}$:

$$R_c^{\pi, \mu} Q(x, a) = Q(x, a) + \mathbb{E}_{x, a}^\mu \left[\sum_{t \geq 0} \gamma^t \prod_{s=1}^t c(X_s, A_s) \left[r(X_t, A_t) + \gamma \sum_{b \in \mathcal{A}} \pi(b|X_{t+1}) Q(X_{t+1}, b) - Q(X_t, A_t) \right] \right],$$

where we define $\prod_{s=1}^0 c(X_s, A_s)$ to be 1.

Remark

The operator $R_c^{\pi, \mu}$ differs from T_λ^π in two aspects. First, the on-policy expectation $\mathbb{E}_{x, a}^\pi$ is replaced by an off-policy expectation $\mathbb{E}_{x, a}^\mu$ and the coefficients $(\lambda^t)_{t \in \mathbb{N}}$ are replaced by the coefficients $(\prod_{s=1}^t c(X_s, A_s))_{t \in \mathbb{N}}$.

Convergence?

The natural question to ask is under which conditions on the function c , Q^π becomes the unique fixed-point of the operator $R_c^{\pi, \mu}$.

Answer

- The Importance Sampling (IS) operator with $c(X_s, A_s) = \frac{\pi(A_s|X_s)}{\mu(A_s|X_s)}$ guarantees that Q^π is the unique fixed point. However, this operator is known to be high variance.
- The Off Policy operator with $c(X_s, A_s) = \lambda$ avoids the high-variance but does not guarantee that Q^π is the unique fixed point.
- The Tree-back-up operator with $c(X_s, A_s) = \lambda\pi(A_s|X_s)$ avoids the high-variance and guarantees that Q^π is the unique fixed point. However, this operator is too conservative. Indeed, it is not efficient in the near on-policy case (where μ and π are similar) as it unnecessarily cuts the traces, preventing it to make use of full returns.
- The Retrace operator with $c(X_s, A_s) = \lambda \min(1, \frac{\pi(A_s|X_s)}{\mu(A_s|X_s)})$ avoids the high-variance and guarantees that Q^π is the unique fixed point. In addition, it is less conservative than the Tree-back-up operator because $\min(1, \frac{\pi(A_s|X_s)}{\mu(A_s|X_s)}) \geq \pi(A_s|X_s)$.

Theorem

Under the condition that:

$$\forall (x, a) \in \mathcal{X} \times \mathcal{A}, \quad 0 \leq c(x, a) \leq \frac{\pi(a|x)}{\mu(a|x)},$$

where $c(x, a) = 42$ when $\mu(a|x) = 0$, Q^π is the unique fixed point of the operator $R_c^{\pi, \mu}$.

From Transitions to Traces

In DQN, the replay contains transitions ξ :

$$\xi = (x, a, r, y).$$

In Retrace, the replay contains traces τ :

$$\tau = (x_t, a_t, \mu_t, r_t, x_{t+1}, a_{t+1}, \dots, x_{t+k}),$$

where $\mu_t = \mu(A_t = a_t | X_t = x_t)$.

Targets

We follow the Retrace operator to define the targets:

$$t(\tau, Q_{\theta'}) = Q_{\theta'}(x_t, a_t) + \sum_{h=t}^{t+k} \gamma^{h-t} \prod_{s=t+1}^{t+i} c(x_s, a_s) \left[r(x_h, a_h) + \gamma \sum_{b \in \mathcal{A}} \pi(b | x_{h+1}) Q_{\theta'}(x_{h+1}, b) - Q_{\theta'}(x_h, a_h) \right],$$

where $c(x_s, a_s) = \lambda \min(1, \frac{\pi_s}{\mu_s})$ with $\pi_s = \pi(A_s = a_s | X_s = x_s)$.

Acting Process


Same as DQN.

Learning Process

We use the targets $t(\tau, Q_{\theta'})$ instead of $t(\xi, Q_{\theta'})$.

Homework

Take a look at the Vtrace algorithm (Impala paper [Espeholt et al., 2018]).

 Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., et al. (2020).

Never give up: Learning directed exploration strategies.

arXiv preprint arXiv:2002.06038.

 Bellemare, M. G., Dabney, W., and Munos, R. (2017).

A distributional perspective on reinforcement learning.

In International Conference on Machine Learning, pages 449–458. PMLR.

 Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014).

On the properties of neural machine translation: Encoder-decoder approaches.

arXiv preprint arXiv:1409.1259.

 Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014).


Empirical evaluation of gated recurrent neural networks on sequence modeling.

arXiv preprint arXiv:1412.3555.

 Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018).

Implicit quantile networks for distributional reinforcement learning.

In International conference on machine learning, pages 1096–1105. PMLR.

 Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. (2018).

Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures.
In International Conference on Machine Learning, pages 1407–1416. PMLR.

 Hausknecht, M. and Stone, P. (2015).


Deep recurrent q-learning for partially observable mdps.
In 2015 aaai fall symposium series.

 Hochreiter, S. and Schmidhuber, J. (1997).

Long short-term memory.
Neural computation, 9(8):1735–1780.

 Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., Van Hasselt, H., and Silver, D. (2018).

Distributed prioritized experience replay.
arXiv preprint arXiv:1803.00933.

 Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al. (2017).

Population based training of neural networks.
arXiv preprint arXiv:1711.09846.

-  Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. (2018).
Recurrent experience replay in distributed reinforcement learning.
In International conference on learning representations.
-  Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. G. (2016).
Safe and efficient off-policy reinforcement learning.
arXiv preprint arXiv:1606.02647.
-  Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., et al. (2015).
Massively parallel methods for deep reinforcement learning.
arXiv preprint arXiv:1507.04296.
-  Schaul, T., Borsa, D., Ding, D., Szepesvari, D., Ostrovski, G., Dabney, W., and Osindero, S. (2019).
Adapting behaviour for learning progress.
arXiv preprint arXiv:1912.06910.
-  Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015).
Prioritized experience replay.
arXiv preprint arXiv:1511.05952.



Van Hasselt, H., Guez, A., and Silver, D. (2016).

Deep reinforcement learning with double q-learning.

In Proceedings of the AAAI conference on artificial intelligence, volume 30.



Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016).

Dueling network architectures for deep reinforcement learning.

In International conference on machine learning, pages 1995–2003. PMLR.



Xu, Z., van Hasselt, H., Hessel, M., Oh, J., Singh, S., and Silver, D. (2020).

Meta-gradient reinforcement learning with an objective discovered online.

arXiv preprint arXiv:2007.08433.



Xu, Z., van Hasselt, H., and Silver, D. (2018).

Meta-gradient reinforcement learning.

arXiv preprint arXiv:1805.09801.