TD 9 – Learning III Reinforcement Learning

Practical Information

TD Assistant
eponiatowski@clipper.ens.psl.eu

TD Material
https://github.com/esther-poniatowski/2223_UlmM2_ThNeuro

Goals of the TD

Three types of learning algorithms are studied in this series of TDs : unsupervised, supervised, and reinforcement learning.

This TD focuses on **reinforcement learning** algorithms.

Part 1 Markov Decision Process.

Part 2 Several algorithms.

Part 3 Application to maze learning : model-free agent (analytical study), model-based agent (simulation).

1 Markov Decision Process

Markov Decision Process (MDP)

A Markov Decision processes is a mathematical framework for modeling decision making for an agent evolving in an environment. It includes several components :

- **States** $\mathcal{S} = \{s_i\}_i$. It is the set of the environmental conditions that the agent can encounter.
- **Actions** $\mathcal{A} = \{a_i\}_i$. It is the set of the behaviors that the agent can take in any environmental state.
- **Transition probability function** $p_{tr}(s_{t+1}|s_t, a_t)$. It specifies the effects of the agent's choices. If at time t, the agent founds itself in a state s_t and performs an action a_t, then at time $t+1$, it encounters a new state s_{t+1} drawn from the probability distribution $p_{tr}(s_{t+1}|s_t, a_t)$.
- **Reward function** $R(s_t, a_t)$. It determines the reward received by the agent in any state of the environment, depending on the action it performs.
- **Policy** $\pi(a_t|s_t)$. It represents the rule that the agent follows to select its action in any environmental state. This policy may be stochastic, in which case the action performed at time t is drawn from a distribution $\pi(a_t|s_t) = \mathbb{P}(a_t|s_t)$.
- **Return** G_t. This is a random variable defined as the total future reward (possibly temporally discounted) that the agent may obtain, starting from time t.

In order to evaluate and improve its current policy π, the agent builds and updates two functions.

- **State Value function** $V_\pi(s)$. It gives the *subjective value* of each state of the environment *under the policy* π. The value attributed to a state s is the expected return obtained by the agent, assuming that it starts in this state and follows the policy π.
- **State-Action Value function** $Q_\pi(a, s)$. It gives the *subjective value* of each action in each state, *under the policy* π. The value attributed to an action a in a state s is the expected return obtained by the agent assuming that it starts in this state, chooses this action, and then follows the policy π.

① Express the return G_t as a function of the future rewards received by the agent r_{t+k}, $k \in \mathbb{N}^*$ (random variables), with a temporal discount factor $\gamma \in [0, 1]$. Rewrite it in a recursive form as a function of r_{t+1} and G_{t+1}.

② Deduce the value $Q_\pi(a_t, s_t)$ of an action in a state, first as a function of G_t, and then as a function of $R(a_t, s_t)$, $V_\pi(s_{t+1})$ and $p_{tr}(s_{t+1}|a_t, s_t)$.

③ Similarly, express the value of a state $V_\pi(s_t)$, first as a function of G_t, and then as a function of $R(a_t, s_t)$, $V_\pi(s_{t+1})$, $p_{tr}(s_{t+1}|a_t, s_t)$ and $\pi(a_t|s_t)$.

2 Algorithms for Decision-Making

For any Markov Decision Process, there exists an **optimal policy** π^* which allows the agent to maximize its long term expected return.

Several algorithms exist to converge towards the optimal policy, among them :

- **Dynamic Programming** (DP). This algorithm is appropriate when the true functions R and p_{tr} are *known* by the agent.
- **Reinforcement Learning** (RL). This algorithm is relevant when the true functions R and p_{tr} are *unknown* by the agent. At any time, the agent can only perceive the current state s_t and the obtained reward r_t. Thus, it has to *learn from experience* the best actions to take in each state, which entails to randomly explore the environment.

Notations for the value functions associated with the optimal policy π^* :

$$V^*(s) = V_{\pi^*}(s) \qquad Q^*(a, s) = Q_{\pi^*}(a, s)$$

2.1 Dynamic Programming

④ Which kind of distribution characterizes the optimal policy $\pi^*(a|s)$ (distribution over actions in a state s) ?
Justify that for finding the optimal policy π^*, it is sufficient to compute the optimal state-action value function Q^*.

⑤ Give a relation between $V^*(s)$ and $Q^*(a, s)$.
For any pair (a, s), deduce the relation between $Q^*(a, s)$ and $R(a, s)$, γ the values in p_{tr} and Q^*.

⑥ Justify that the optimal state-action value function Q^* can be viewed as the fixed point of a function f operating on matrices, such that $Q^* = f(Q^*)$. Suggest a numerical method for finding the optimal policy.

2.2 Reinforcement Learning

Two reinforcement learning algorithms

- **Temporal Difference Learning** consists in estimating the *state value function* V.
- **Q-Learning** consists in estimating the *state-action value function* Q.

Both algorithms implement **online updates** after each action performed by the agent. The magnitude of one update is proportional to the estimation error δ_t, which is the difference between the reward expected by the agent at this time and the actual reward it receives.

⑦ To which type of reinforcement learning do those algorithm belong ? Justify.

⑧ Explain the difference between both strategies in terms of the number of values estimated by the agent. In which conditions is each value updated ?

(9) *Temporal Difference Learning*

- Express the estimation error δ_t as a function of the actual reward r_t received by the agent at time t, and its current value estimates $V(s_t)$ and $V(s_{t+1})$.
- Propose a formula for the online update of the state value function V.

(10) *Q-Learning*

- Express the estimation error δ_t as a function of the actual reward r_t received by the agent at time t, and its current value estimates in Q.
- Propose a formula for the online update of the state-action value function Q.

Soft-max policy

Reinforcement-learning aims to both updating a model of the environment from experience and collecting rewards nonetheless. This "dilemma" requires a fine-tuned *trade-off between exploitation and exploration* of the environment.

A common policy used during this process is the *soft-max policy* :

$$\pi(a_t|s_t) = \frac{e^{\beta\, Q(a_t,s_t)}}{\sum_{a'} e^{\beta\, Q(a',s_t)}} \text{ with } \beta \text{ a constant}$$

(11) Comment on this formula and its parameters in light of the 'dilemma' faced by the agent.

3 Applications to Maze Learning

3.1 Analytical study – Model-free agent performing Temporal-Difference Learning

In this model :

- The environment is a simple maze, composed of a single corridor. The exit is at the right end.
- At each trial, the agent starts in the middle of the corridor. The trial stops as soon as the agent reaches either the left or the right end of the maze.
- The environment is deterministic, such that a given action performed in a given state always leads to the same outcome.

(12) Propose a model for this situation by defining the states of the environment $s \in \mathcal{S}$, the actions that the agent can perform $a \in \mathcal{A}$, and the reward function.

Initially, the agent's policy is to perform left or right displacements with equal probability $1/2$ at each step.

(13) Determine the value function $V(k)$ (after convergence) as a function of $V(k-1)$ and $V(k-1)$, for any position $k \in [\![-(n-1),(n-1)]\!]$.
Determine the boundary conditions at extreme positions.

(14) Express the discrete Laplacian $\Delta V(k) = V(k-1) - 2V(k) + V(k+1)$.
What is the analogous of the Laplacian in the continuous limit ? What would be the solution ?
Use an ansatz of this form and the boundary conditions to express the solution $V(k)$ as a function of k.

The agents initial estimate of V is a uniform function $V = 1/2$

(15) Describe qualitatively the evolution of the estimated value V during the first learning trial.

3.2 Simulations – Model-based agent

In this model :

- The environment is a more complex maze, composed a grid of 16 cells (4x4). A reward is located in one corner of the grid. Then, it will be possible to add obtacles within this grid.• At each trial, the agent starts in the corner opposite to the reward. The trial stops as soon as the agent reaches the reward.
- The actions are the displacements in the four cardinal directions : $\mathcal{A} = \{N, S, E, W\}$.
- The environment is stochastic, such that a given action performed in a given state may leads to distinct outcomes. More precisely, performing an action a leads to the desired displacement with a probability $7/10$ and to the two neighbor diagonal cells with probability $1/10$ each.

In addition with the state-action value function Q, the **model-based** agent builds an estimated reward function $\widehat{R}$ and an estimated transition probability function $\widehat{P}$.

Those estimates are computed by tracking the number of events of each (state, action) couple N. Updates are performed at each time step as follows :

- Estimated transition probabilities : $\widehat{P}(s'|a_t, s_t) \leftarrow \left(1 - \dfrac{1}{N(a_t, s_t)}\right) \widehat{P}(s'|a_t, s_t) + \dfrac{1}{N(a_t, s_t)} \mathbb{1}_{s'=s_{t+1}}$.
- Estimated reward function : $\widehat{R}(a_t, s_t) \leftarrow \left(1 - \dfrac{1}{N(a_t, s_t)}\right) \widehat{R}(a_t, s_t) + \dfrac{1}{N(a_t, s_t)} r_t$.
- Number of events : $N(a_t, s_t) \leftarrow N(a_t, s_t) + 1$

(16) Propose Python objects/structures appropriate to model the different components of the environment (reward function R, transition probabilities p_{tr}) and the agent (state-action value function Q, estimated reward function $\widehat{R}$, estimated transition probabilities $\widehat{P}$, number of events of each (state, action) couple N).

(17) Generate the true transition probability matrix and the true reward function as global variables.

(18) Initialize the estimated transition probability matrix with a uniform distributions in each state, the estimated reward function as a null matrix, the number of events of each (state, action) couple as a null matrix, and the Q matrix as a null matrix.

(19) Code a function `Evolution_Environment(a,s)` to draw the new state of the environment after an action a is performed in a state s.

(20) Code the soft-max function `Softmax(a,s,Q)` and a function `Decision_Making(a,s,Q)` to select an action under this policy.

(21) Code a function `Update_Model(a,s0,s1,r,hatR,hatP,N)` to update the estimated reward function $\widehat{R}$, the estimated transition probabilities $\widehat{P}$, and the number of events N after an action a had been performed in state s_0 leading to a new state s_1 and a reward r.

(22) Code a function `Update_Values(a,s0,hatR,hatP,Q)` to update the state-action value function Q after an action a had been performed in state s_0.

(23) Implement the algorithm for a number of iteration steps (for instance `ntrials = 10000`).

(24) Display the distribution of the Q-values in the grid at several trials during learning.