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 : supervised, unsupervised and reinforcement learning.

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

Part **1** *introduces the framework of the Markov Devision Process* to describe the behavior of an agent in an environment.

Part **2** *presents several algorithms allowing an agent to optimize its behavior.*

Part **3** *aims to apply those results to maze learning, through the analytical study of a model-free agent in a simple corridor, and a simulation study of a model-based agent in a more complex environment.*

1 Markov Decision Process

The Markov Decision process model consists of an agent evolving in an environment.

The environment can be in any **state** from a set $\mathcal{S} = \{s\}$. The agent can perform any **action** from a set $\mathcal{A} = \{a\}$.

Actions change the state of the world through a **transition probability function** $p_{tr}(s_{t+1}|s_t, a_t)$.

If at time t, the state of the world is s_t and the agent performs action a_t, then at time $t+1$, the new state of the world s_{t+1} is drawn from the probability distribution $p_{tr}(s_{t+1}|s_t, a_t)$.

Actions also allow the agent to obtain rewards through a **reward function** $R(s_t, a_t)$.

It specifies the reward received by the agent depending on the current state the world and the action performed.

The agent selects its actions through a **policy** $\pi(a_t|s_t)$.

This policy may be stochastic, in which case the action performed at time t is drawn from a distribution $\pi(a_t|s_t) = \mathbb{P}(a_t|s_t)$.

For any Markov Decision Process, there exists an optimal policy π^* which allows the agent to maximize its long term expected return. The **return** at time t is a random variable G_t defined as the total future reward (possibly temporally discounted).

The goal of the agent is to converge towards this optimal policy in order to maximize its long term return.

To do so, the agent builds and updates two functions associated with its current policy π.

- **State Value function** $V_\pi(s)$

It gives the *subjective values* of the different *states* of the environment under the policy π. The value ascribed to a state s is the expected return obtained by the agent assuming that it starts in this state and follows its policy.

- **State-Action Value function** $Q_\pi(a, s)$

It gives the *subjective values* of the different *actions in each state*, under the policy π. The value ascribed 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 its policy.

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

2 Algorithms for Decision-Making

Several algorithms exist to determine the optimal policy, which will be detailed in next sections.

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

2.1 Dynamic Programming

④ Characterize the optimal policy in terms of the distribution $\pi^*(a|s)$ over actions in a state s. Justify that it is sufficient to compute the optimal state-action value function Q^* and then to deduce the optimal policy π^*.

⑤ Characterize the optimal policy with a relation between $V^*(s)$ and $Q^*(a, s)$. Justify that the optimal state-action value function Q^* can be viewed as the fixed point of a fonction f operating on matrices, such that $Q^* = f(Q^*)$.

⑥ Suggest a numerical method for finding the optimal policy.

2.2 Reinforcement Learning

⑦ What are the two types of reinforcement learning models by which the agent can learn to choose its actions ?

Two common *model-free* reinforcement learning methods are Temporal Difference Learning and Q-Learning.

- **Temporal Difference Learning** consists in estimating the *state value function* V for a *fixed policy*.
- **Q-Learning** consists in estimating the *state-action value function* Q with a *soft-max policy*. It is a *policy learning algorithm* which allows for fine-tuning the trade-off between exploitation and exploration.

For instance, an agent can start exploring the environment with a Temporal Difference Learning algorithm. Once the agent has found an accurate enough estimate of V, it can switch to a Q-Learning algorithm.

Both algorithms implement online updates after each action. The magnitude of one update is proportional to the Temporal Difference Error in the value estimate made at that time. More precisely, the estimation error is the difference between the expected value of the state and the actual value perceived after the current iteration.

⑧ Suggest an expression for the Temporal-Difference 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 $V(s_{t+1})$. Give an alternative interpretation of this variable in terms of an error in the *expected reward* at time t.

⑨ Deduce a formula for the online update of the value function V in Temporal Difference Learning which can be implemented every time the agent performs an action.

⑩ Express the error and the update in the Q-learning algorithm for the state-action value function Q. What is the difference between both strategies ?

⑪ Recall the expression of the soft-max policy. 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

The environment is a simple maze, composed of a single corridor.

The agent starts in the middle and has to find the exit, which is at the right end of the maze. An exploration trial stops as soon as the agent reaches either the left or the right end of the maze.

The environment is considered to be 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 would be the solution in the continuous limit ? 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

The environment is a more complex maze, composed a grid of 16 cells (4x4).

The agent starts in one corner and has to find the reward located at the opposite corner. An exploration 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 considered to be 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 $\widehat{P}$. This is done by tracking the number of events of each (state, action) couple N. The updates are performed as follows at each time step :

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