



@AndrzejWasowski@scholar.social

Andrzej Wąsowski

Advanced

Programming

Reinforcement Learning: A programming project

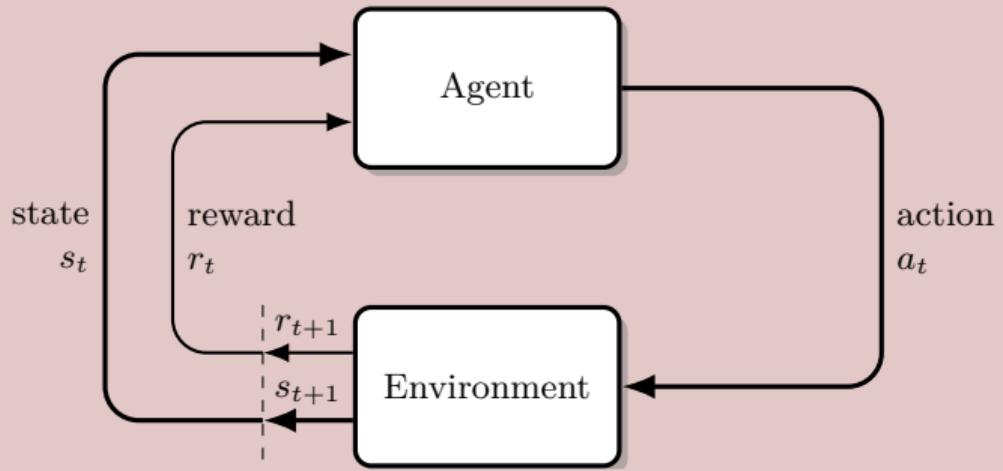
IT UNIVERSITY OF COPENHAGEN

SOFTWARE
QUALITY
RESEARCH

What is Reinforcement Learning?

- **Reinforcement Learning** (RL) is a kind of Machine Learning, where we learn a behavior policy for an agent
- Here is a simple (continuous control) example:
<https://www.youtube.com/shorts/hgjsLmFSkxo>
- An agent can take actions and the environment responds with **state** changes and **rewards**
- A reinforcement learning algorithm allows the agent to try many different behaviors and remembers which were best.
- A **strategy**, or a behavioral **policy** maps states to best actions
- RL is an **optimization** method that optimizes the policy to maximize total expected reward
- We implement a simple kind of RL, **Q-learning** where the state and actions are discrete (like in automata)

RL Setup



- S_t — system state at time t
- A_t — action chosen by the agent at time t
- R_{t+1} — reward received after taking action A_t
- S_{t+1} — new state after taking action A_t

- The agent looks at the state and decide the best action a_t
- In our case, the action choice is **deterministic** (the best action is always chosen)
- The environment updates the state and gives a **reward** r_t probabilistically
- The process repeats in a loop until agent reaches a **terminal state** (end of **episode**). We run many episodes.

Q-Learning

The update rule

- The agent maintains a **Q-table**, which maps state-action pairs to an estimation of expected future reward.
- $Q(S, A)$ estimates the expected cumulative reward of taking action A in state S and following the best policy thereafter.
- Initially the Q -table contains random values (or zeroes).
- The agent updates the table after taking each action. At time t :

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right) \text{ where}$$

- $\alpha \in (0; 1]$ is the **learning rate** constant
(a hyper-parameter, defining how quickly we learn and forget)
- $\gamma \in [0; 1]$ is the **discount factor** constant
(a hyper-parameter defining importance of future rewards vs the immediate reward)
- $\max_a Q(S_{t+1}, a)$ is the best estimated Q-value for the next state S_{t+1}

Exploration-Exploitation trade-off in RL

- Updating Q -table along the best gradient might get stuck in **local optima**
- To avoid this, we allow the agent to explore—try random actions sometimes
- The **exploration-exploitation** ratio ϵ allows us to control between **exploitation** (choosing the best-known action) and **exploration** (trying new actions to discover their rewards)
- A common strategy is called ϵ -greedy:
 - **Exploit:** choose the action with the highest Q-value with probability $1 - \epsilon$
 - **Explore:** choose a random action with probability ϵ
 - $\epsilon \in [0; 1]$ is another hyperparameter
- The value of ϵ can be decayed over time to shift from exploration to exploitation as learning progresses

Q-Learning

The algorithm

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$S \leftarrow S'$

 until S is terminal

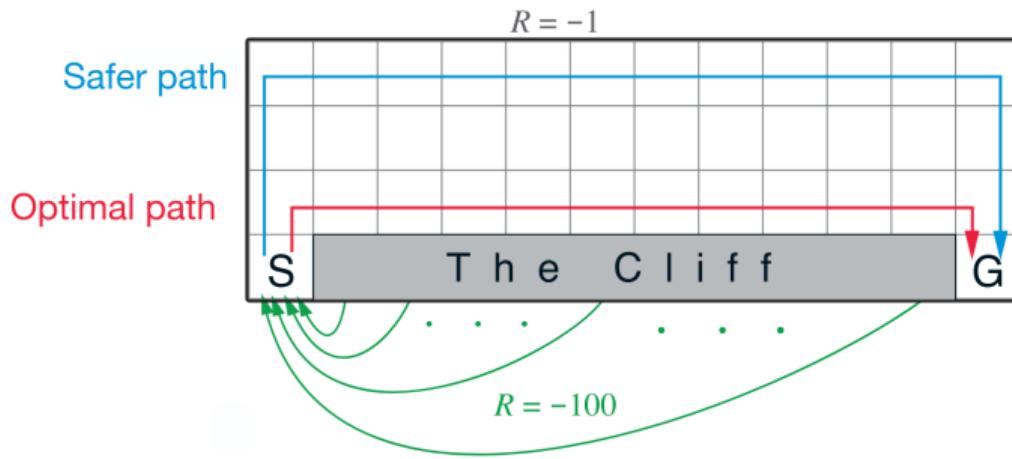
Notes: Initialize S either randomly or fix the **initial** state. Selection action A using ϵ -greedy strategy.

Exploitation = selecting the action maximizing $Q(S, A)$. **Note loops and assignments! Impure use of rand!**

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd edition. MIT Press, 2018.

The Cliff Walking problem

as used in the project



Notes: The agent starts at the **S** cell and must reach the **G** cell. The cells at the bottom row (the cliff) give a large negative reward (-100) if stepped on, otherwise each step gives a small negative reward (-1). The goal is to learn a policy that maximizes total reward (minimizes total penalty).

Questions: What is the state space? What are the actions? What are the rewards?

The project

Assorted notes

- Implement a Q-learning setup in pure Scala, using the course abstractions
- Your implementation should be generic in A and S (it works for different problems).
- Instantiate the setup for the Cliff Walking agent
- Q-tables can be represented as values of type `Map[S, Map[A, Double]]`
- Print output like this to represent a policy (N is the number of episodes tried):

```
````  
a=0.1 ε=0.1 γ=1.0 N=400
↓↓↑→↓↑→→↓→→↓
→↓↓→→→↓↓↓↓
→→→→→→→→→↓
↑.....
````
```

More info in README.md

In the next episode ...

- Let's talk a bit about the exam format!