Reinforcement Learning Lesson - 2

Recall

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],
\end{aligned}
$$

$$
\mathit{NewEstimate} \leftarrow \mathit{OldEstimate} \;+\; \mathit{StepSize}\;\Big[\mathit{Target} - \mathit{OldEstimate}\Big].
$$

Homework

You are given six dice.

Assume you cannot see the die faces.

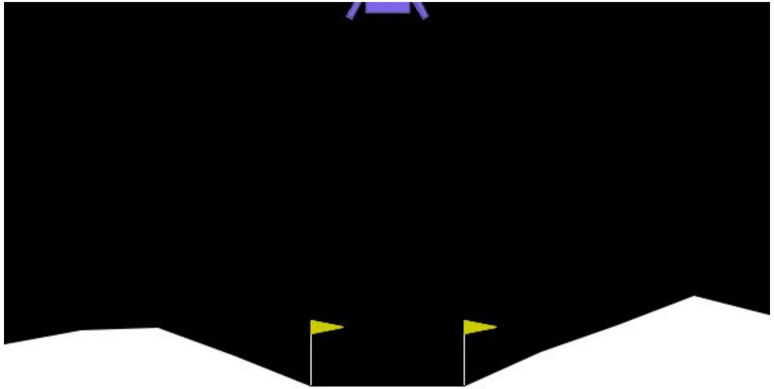
And the probability is not uniform.

Find the strategy to get maximum sum in 1000 rolls.

How Do We Solve It?

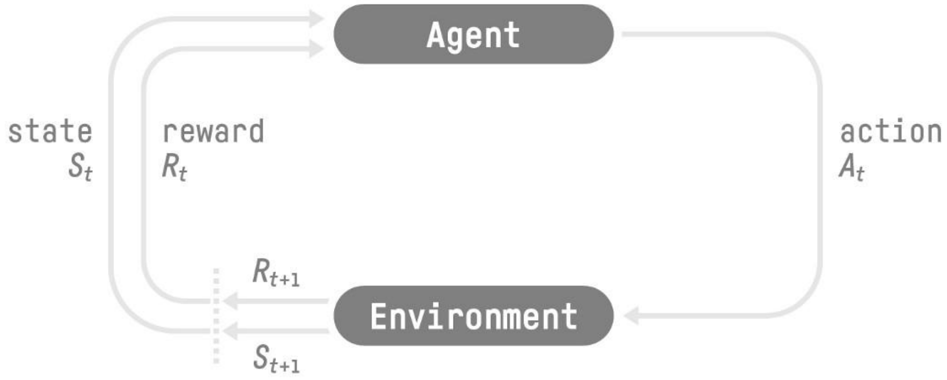
1. Transform the problem statement.
2. Find the die that will give us the largest expected value and keep rolling.

Reinforcement Learning Environments



https://gymnasium.farama.org/

Reinforcement Learning Framework



Goal - **Maximize** its cumulative reward, called the expected return.

https://huggingface.co/learn/deep-rl-course/unit1/rl-framework

Observation Space

State s:
A **complete description** of the state of the world (no hidden information)

Observation o:
A **partial description** of the state of the world.

https://huggingface.co/learn/deep-rl-course/unit1/rl-framework

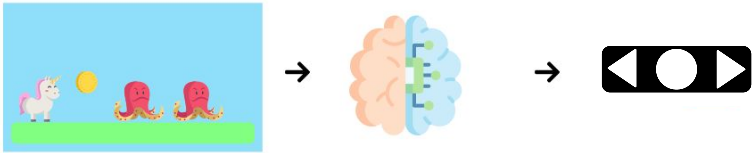
Action Space

Discrete space
The number of possible action is finite.

Continuous space
The number of possible action is infinite.

https://huggingface.co/learn/deep-rl-course/unit1/rl-framework

Policy



State → π(State) → Action

State s → $\pi(a|s)$ → [Left: 0.1, Right: 0.7, Jump: 0.2]

State s_0 → $\pi(s_0)$ → a_0 = Right

Stochastic: output a probability distributions over actions.

$$\pi(a|s) = \boxed{P[A|s]}$$

Probability Distribution over the set of actions given the state

https://huggingface.co/learn/deep-rl-course/unit1/two-methods

Return

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

Return

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \; = \; \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the *discount rate*.

Value Function of State

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a)\Big[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s']\Big]
\end{aligned}
$$

Dynamic Programming

Dynamic programming refers to a problem-solving approach, in which we precompute and store simpler, similar subproblems, in order to build up the solution to a complex problem.

Dynamic Programming Example

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

Input: n = 2 Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step

2. 2 steps

Example 2:

Input: n = 3 Output: 3

Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step

2. 1 step + 2 steps

3. 2 steps + 1 step

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
       $\Delta \leftarrow 0$
       Loop for each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
       old-action $\leftarrow \pi(s)$
       $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$
       If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\mathit{terminal}) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$