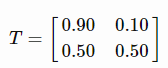
Reinforcement Learning

A **Markov Chain** has a set of **states** S={s0,s1,...,sm}S={s0,s1,...,sm}and a **process** that can move successively from one state to another. Each move is a single **step** and is based on a **transition model** T. You should make some effort in remembering the keywords in bold because we will use them extensively during the rest of the article. To summarize a Markov chain is defined by:

1. Set of possible States: S={s0,s1,...,sm}
2. Initial State: s0
3. Transition Model: T(s,s′)
4. Let’s suppose we have a chain with only two states s0 and s1, where s0 is the initial state. The process is in s0 90% of the time and it can move to s1 the remaining 10% of the time. When the process is in state s1it will remain there 50% of the time. Given this data we can create a **Transition Matrix** TT as follow:

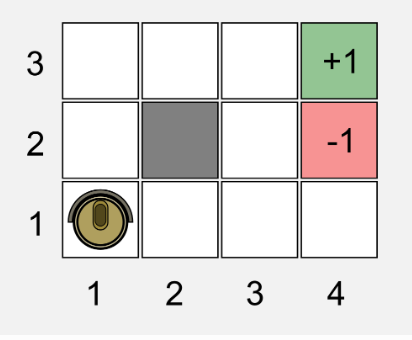


The transition matrix is always a square matrix, and since we are dealing with probability distributions all the entries are within 0 and 1

* **Problem** the agent has to maximize the reward avoiding states which return negative values and choosing the one which return positive values.
* **Solution** finds a **policy** π(s) which returns the action with the highest reward.

The agent can try different policies but only one of those can be considered an **optimal policy**, denoted by  π∗, which yields to the highest expected utility.

Let suppose we have a **cleaning robot** which has to reach a charging station. Our simple world is a 4x3 matrix where the starting point s0 is at (1,1), the charging station at (3,4), dangerous stairs at (4,2), and an obstacle at (2,2). **The robot has to find the best way to reach the charging station** (Reward +1) **and to avoid falling down the flight of stairs** (Reward -1). Every time the robot takes a decision it is possible to have the interference of a stochastic factor (ex. the ground is slippery, an evil cat is stinging the robot), which makes the robot diverge from the original path 20% of the time. If the robot decides to go ahead in 10% of the cases it will finish on the left and in 10% of the cases on the right state.



Set of actions=u\*T(actions from state 0 to 4)

And adding actions of all states (up, down, left, right)

Utility=reward + gamma \*(max values of actions)

States defined as 3\*4 matrixes

