CS 4100: Introduction to AI

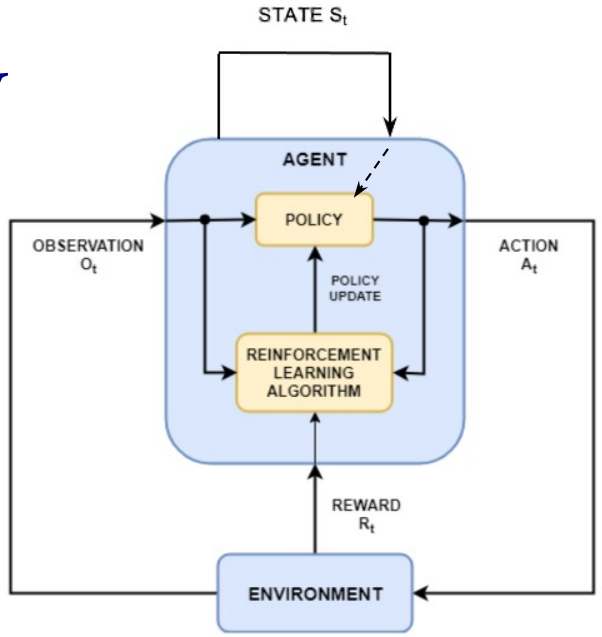
Wayne Snyder
Northeastern University

Lecture 18: Reinforcement Learning: Markov Decision Processes; The Q Algorithm

Plan for Today:

- Markov Decision Processes
- Q-Algorithm: Dynamic Programming to learn an optimal policy

Reinforcement Learning: Review



Recall that Reinforcement Learning (RL) involves an Agent taking actions in an environment to maximize its rewards. It can be considered to be a form of unsupervised ML.

The principal components of an RL system are the following:

1. Environment: "physical space" in which agent moves; discrete/continuous, deterministic/stochastic, fully/partially observable, may contain other agents.
2. State S_t : "Memory" of agent preserving information about progress through environment.
3. Action A_t : Agent performs actions to "move" through environment and get rewards.
4. Reward R_t : Scalar representing how much of the overall goal or purpose has been achieved.
5. Value $V(S_t)$: How much reward is expected starting from a particular state.
6. Policy $\pi(S_t, O_t)$: Given a state and possible observations, what action to take. Learning an optimal policy is the entire purpose of the RL system.
7. Agent: The algorithm that organizes all the activity.
8. Model: Representation of environment stored in agent.

ALSO:

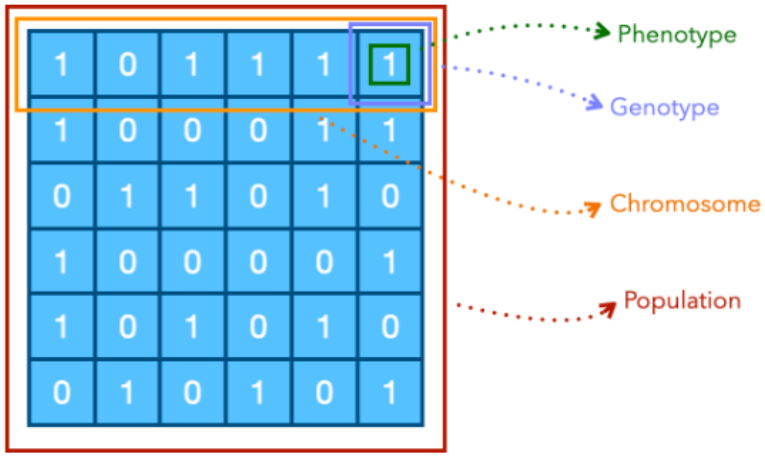
There is a tradeoff for the agent between exploiting what is has already learned to gain predictable rewards and exploring the environment to find better rewards with possibly some cost.

Genetic Algorithms as RL

1. Environment
2. State
3. Action
4. Reward
5. Value
6. Policy
7. Agent
8. Model

Genetic Algorithms are a special subclass of RL algorithms with the following characteristics:

No model
No Value function
Possibly multiple agents
Environment could consist of other agents
Policy = array of numbers
Search is used to create "offspring" from the Agent, by analogy with biological mutation and crossover



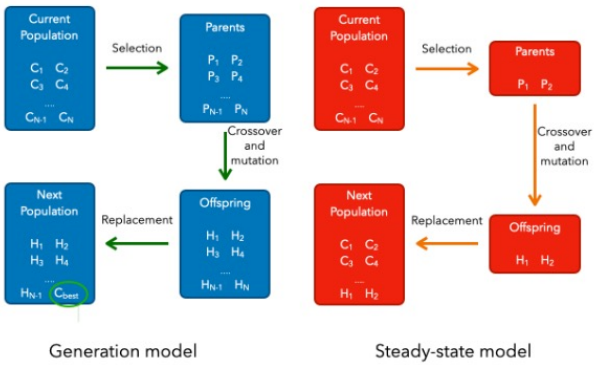
Genetic Algorithms as RL

Framework for learning:

Multiple agents (arrays of numbers) compete with each other, and receive immediate rewards.

On the basis of the rewards, the agents can

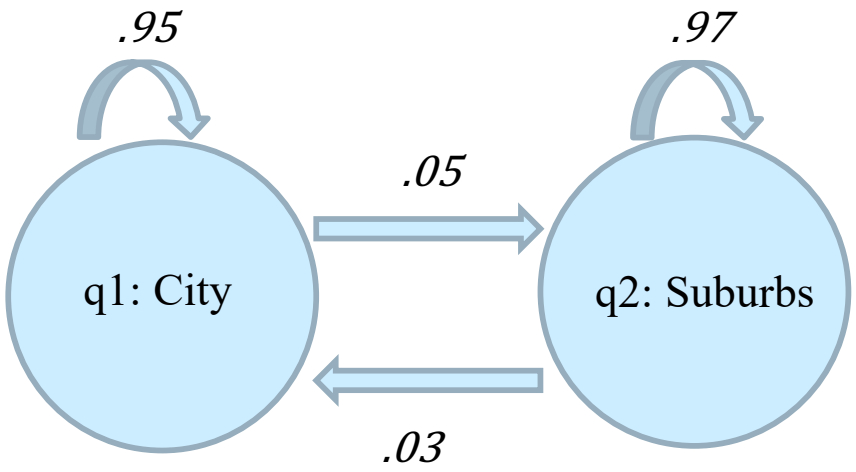
- Die
- Produce offspring through:
  - Mutation: Make small (usually random) changes in the numbers
  - Crossover: Breed with other successful agents



Markov Decision Processes

Recall: Markov Chains are directed graphs defined by (Q, A, π):

$Q = q_1 q_2 \ldots q_N$	a set of N states
$A = a_{11} a_{12} \ldots a_{N1} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$



A

	From City	From Suburbs
To City	.95	.03
To Suburbs	.05	.97

π = [.6, .4]

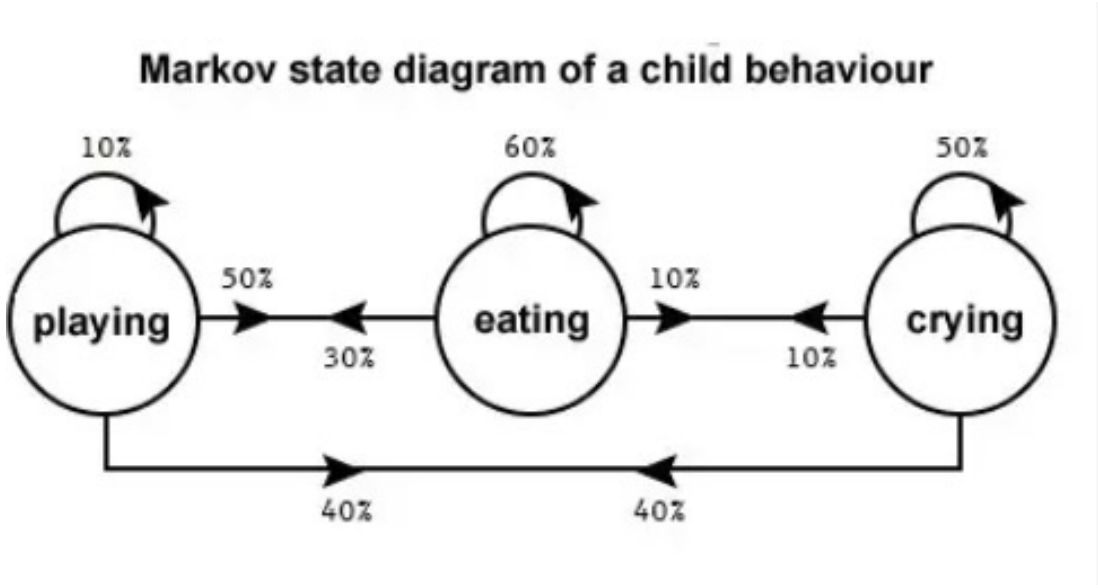
OR:

π = [600000, 400000]

Markov Decision Processes

The most important things to remember about Markov Chains are:

- An agent moves from state to state with some probability; and
- The decision about which state to move to is based entirely on the current state: the agent is "memory-less."



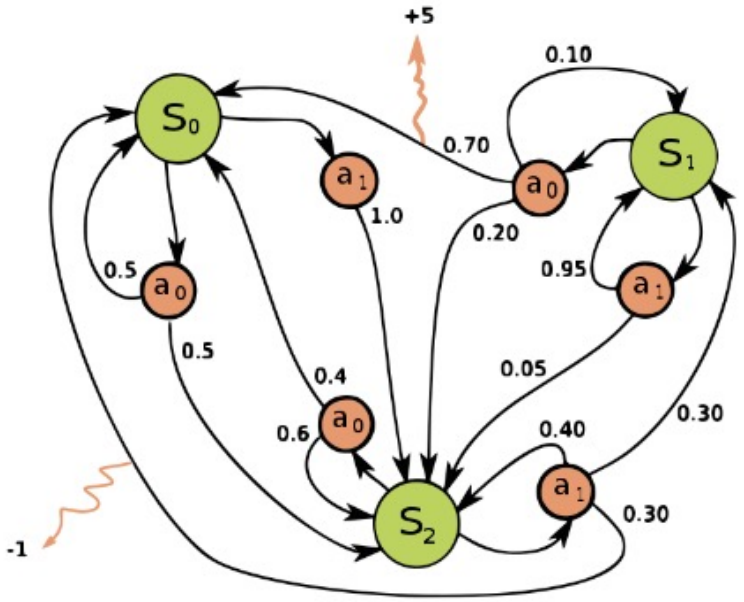
Markov Decision Processes

In general, a Markov Decision Process is "a discrete-time stochastic control process. It provides a mathmatical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful studying optimization problems solved via dynamic programming." (Wikipedia)

We may translate this into RL terms as:

At time t, an agent is in state S_t and following its policy π chooses an action A_t which moves it to a new state S_{t+1} and results in a reward R_t. The choice of an action, and the next state resulting from this action may be stochastic (i.e., involve randomness).

The goal of the agent is to maximize its total rewards.

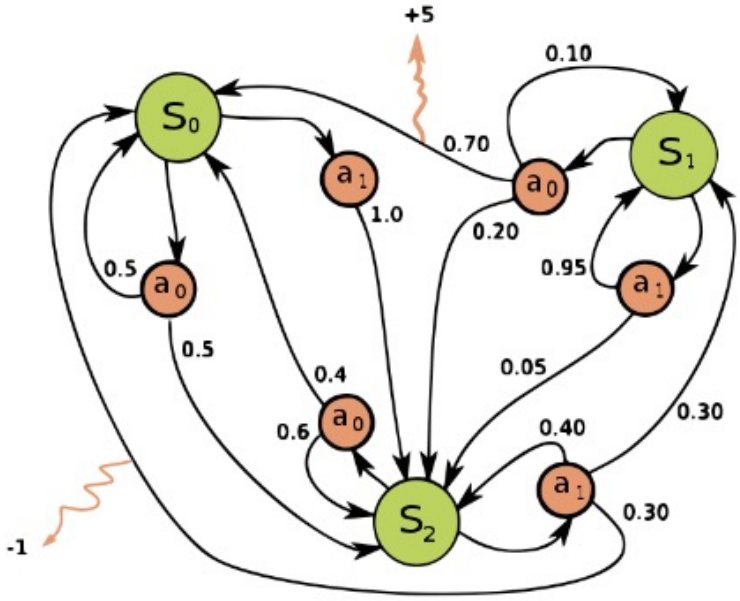


States are green circles; actions are red circles; rewards are orange wiggly arrows attached to actions. Here, the state resulting from an action is determined randomly as shown. The choice of action may be deterministic or stochastic.

Markov Decision Processes

Formally, a Markov Decision Process is represented by four components:

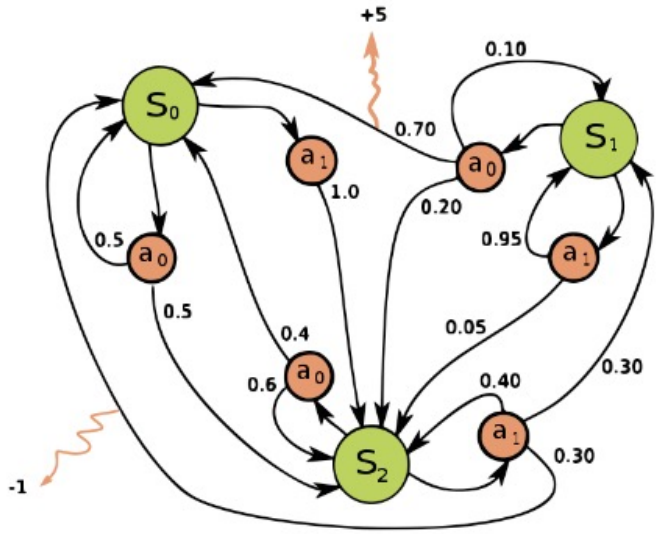
- $S = \{S_0, S_1, \ldots\}$ is the state space
- $A = \{A_0, A_1, \ldots\}$ is the action space (we may also speak of A_s = the set of actions available from state s)
- $P(a,s) = Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that an action a in state s at time t leads to state s' at time t + 1.
- $R(s,a,s')$ = the reward received by taking action a in state s and transitioning to state s'.



This is the most general framework, and many RL systems do not use MDPs in their full generality, particularly as regards the role of randomness.

Thus: the "Markov" in the name does not necessarily mean that state transitions are made stochastically, but rather refers to the memoryless property.

Markov Decision Processes



In diagrams, you may see rewards attached to actions, OR to states (when an action leads to a unique state, so an action = choose the next state).

Important considerations:

- Actions may be chosen stochastically, and state transitions (given a current state and the action chosen) *may be* stochastic or deterministic.

- A common approach for choosing actions stochastically, called ε – greedy, is to set a probability ε and choose
  - an exploitative action (the one with the highest known reward) with probability 1- ε; or
  - an exploratory action (with unknown or less-certain rewards) with probability ε.

- Rewards may be
  - Immediate – Each transition from a state s to a state s' by means of action a gives an immediate reward. The optimization goal is to maximize the cumulative reward by the end.
  - Sparce – Rewards may only be given in a small number of state/action transitions. In the worst case, a reward may only be given at the end of the entire experiment. In this case, it is usual to do "reward shaping" by using a value function (learned during the experiment, to predict future rewards from the current state and action).

Markov Decision Processes

Let's look at a simple example, the well-known "cliff walking" problem.

The agent is in some starting location, and wants to get to a goal location by walking. The shortest path is to walk along the cliff edge. What is the best policy?

Markov Decision Processes

A simple "grid world" environment might look like this:

Actions: R

L

U

D

States: (row, col) in grid

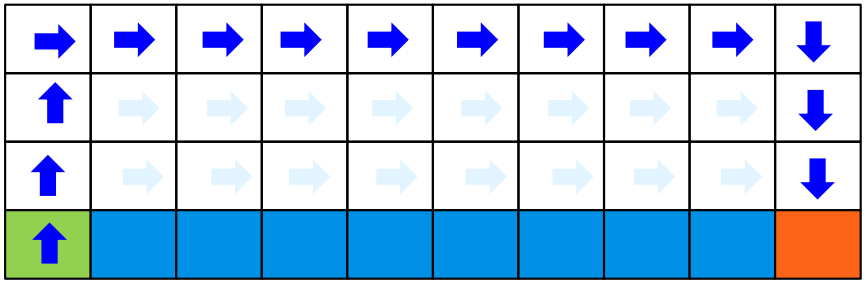
Rewards are attached to states because everything is deterministic: agent gets reward when it moves to a state.

Delayed reward: shortest path yields the optimal reward.

Markov Decision Processes

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
A	-100	-100	-100	-100	-100	-100	-100	-100	100

A policy can be represented by a mapping from states (locations in grid) to actions, and could be shown by giving the action to take in each state:



Reward of the "safest" policy: 100 – 15 = 85.

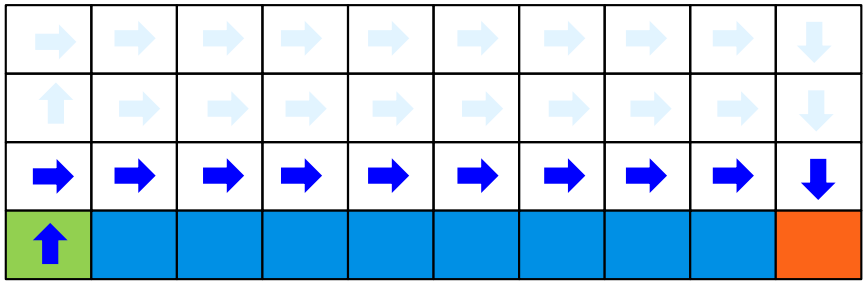
(Assume that if the agent walks off the cliff, the experiment is terminated and starts over.)

I goofed: this is supposed to have 12 columns instead of 10. We'll use 12 later.....

Markov Decision Processes

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
A	-100	-100	-100	-100	-100	-100	-100	-100	100

The optimal policy has a reward of 100 – 10 = 90.



How to find the optimal policy?

Run multiple experiments using ε – greedy algorithm, starting with $\varepsilon = 1.0$ (completely random) and decreasing by a factor of λ in each iteration of the experiment.

How to make this more efficient?

We will "back up" rewards using "rewards shaping" and a value function to tell us how good each state is in terms of getting a future reward.

Markov Decision Processes: Q-Learning

The Q-Learning Algorithm (Q = Quality) uses dynamic programming to find the optimal (highest reward) path.

The main data structure is a 2D matrix holding the value function (here called the Q-Table). It is a mapping from states and actions to projected rewards (called Q-Values):

$Q(S_t,a)$ -> Q-values (floats)

and in our cliff-walking example, it would have

12 * 4 = states

4 actions

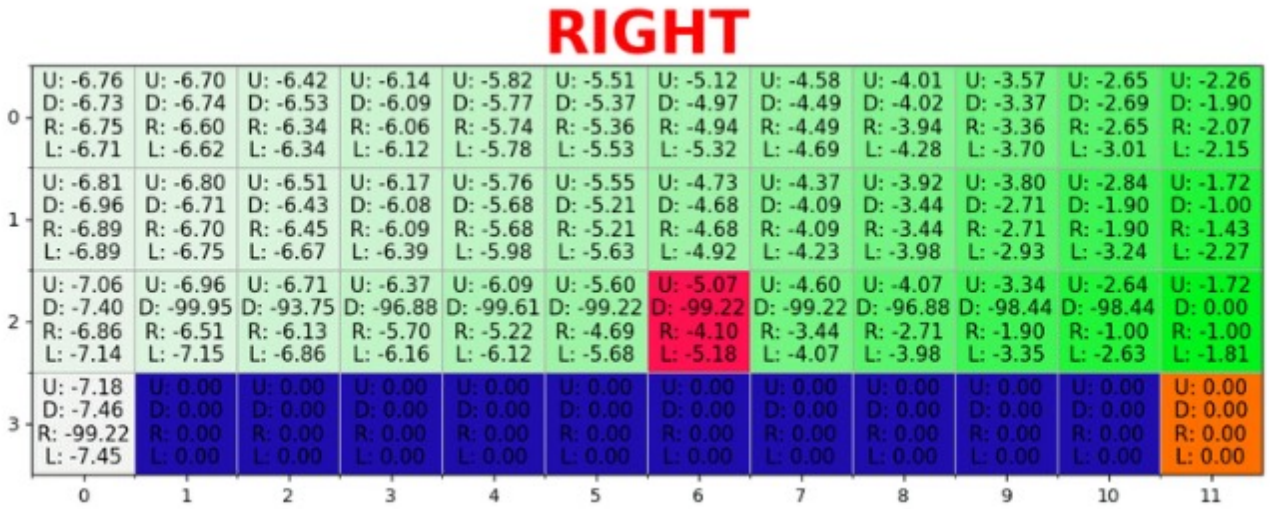
so the Q-Table would be a 48 x 4 matrix.

States	up	down	right	left
0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0

Q-values of the first five states.

Markov Decision Processes: Q-Learning

Here is an example of the Q-Table which might give an optimal strategy after multiple experiments. At each time step, take the action with the highest Q-Value:



Markov Decision Processes: Q-Learning

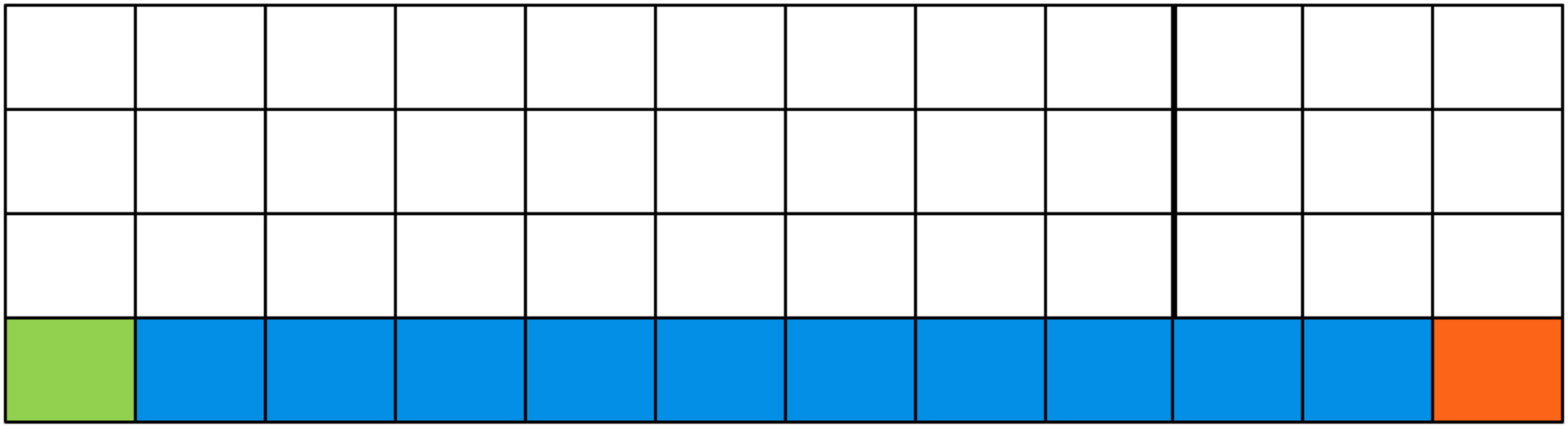
Calculating the Q-Values

A naive approach to finding Q-values would look like this:

Remember that the agent will also keep track of the cumulative rewards gained so far.....

1. Initialize all Q-values to -1;
2. For each run:

   Choose actions randomly, and if you find the goal or the cliff, indicate the Q-Values of the next-to-last state (which would not be estimates, but the actual reward!).



Markov Decision Processes: Q-Learning

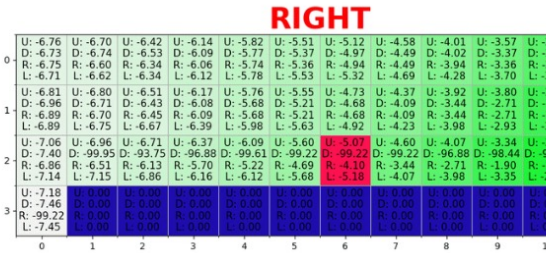
The Q Algorithm makes a number of improvements on this naive approach.

The Q-Table is initilized with all 0 Q-Values, or perhaps with random values.

Multiple experiments are run using the epsilon-greedy strategy, using a fixed ε or a diminishing sequence

$$\varepsilon, \gamma\varepsilon, \gamma^2\varepsilon, \gamma^3\varepsilon, \ldots$$

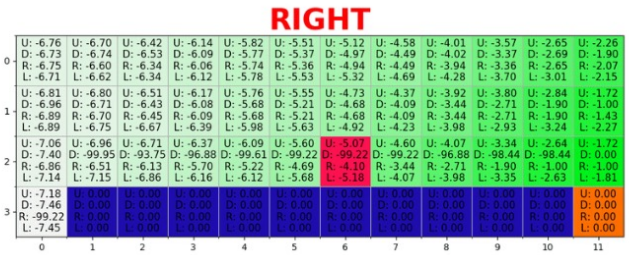
(for example, we might use $\varepsilon = 1.0$ and $\gamma = 0.99$.)



The epsilon-greedy approach here would consist of the following two possible choices of the next action (=transition to a new state):

- Exploitative: Take the action which has the largest Q-Value; and
- Exploration: Choose equiprobably from among D, U, L, R.

Markov Decision Processes: Q-Learning



The most important component of the Q-Algorithm is that dynamic programming is used to update the Q-values, following the

Bellman Equation (without learning rate):

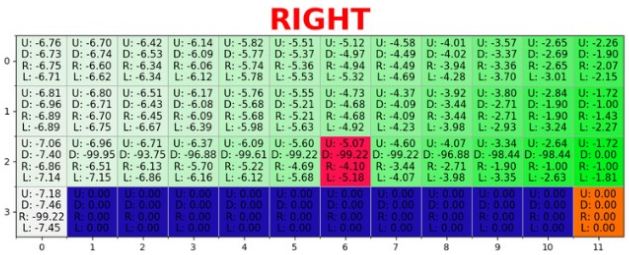
$$Q(s,a) = R + \gamma \max_{a'} Q(s',a')$$

where R is the current cumulative reward gained so far and s' and a' are the possible next actions and states.

This equation basically says that you should choose the action which results in the maximum increment in the cumulative reward.

The discount factor γ, with $0 \leq \gamma \leq 1$, expresses how confident we are in the estimate: as we project into the future, our estimate will be used less and less. Typical values are between 0.9 and 0.99.

Markov Decision Processes: Q-Learning



Bellman Equation (with learning rate):

In some cases (e.g., Deep Q-Learning using NNs), it is necessary to add a "learning rate" $0 \leq \alpha \leq 1$ to the calculation to smooth out the approximation of the values:

$$Q(s,a) = \alpha\,(R + \gamma\, \max_{a'} Q(s',a'))\; +\; (1-\alpha)\, Q(s,a)$$

which is usually expressed in the compact form:

$$Q(s,a) = \alpha\,[\, R + \gamma\, \max_{a'} Q(s',a')\; - Q(s,a)\,]$$

Markov Decision Processes: Q-Learning

The final version of the Q-Algorithm is therefore as follows:

Choose parameters α and ε and (optionally) γ;

Initialize Q-Table with fixed values (e.g., 0) or random values;

Run repeated experiments until some terminal criterion is met:

- s = Start state
- R = 0
- Choose action a $\in A_s$ using epsilon-greedy strategy to choose between:
  - explotation: use action with maximum Q-Value
  - exploration: choose random action
- Take action a;
- Add immediate reward to R;
- Update Q-table:

$$Q(s,a) = \alpha\,(R + \gamma\, \max_{a'} Q(s',a')) \;+\; (1-\alpha)\, Q(s,a)$$

- s = s';

A visualization of this algorithm (with very cheesy music) may be found here:

https://youtu.be/Vto8n9C7DSQ

Deep Q-Learning

So what's the problem??

How would this work for, say, Connect-4?

How big is the Q-Table?

Naive approximation is $3^{64} = 3.43 * 10^{30}$ (!!!)

Therefore, a neural network is used to store/approximate/learn the Q-Table.

Most significant applications of reinforcement learning are now done with deep learning.....