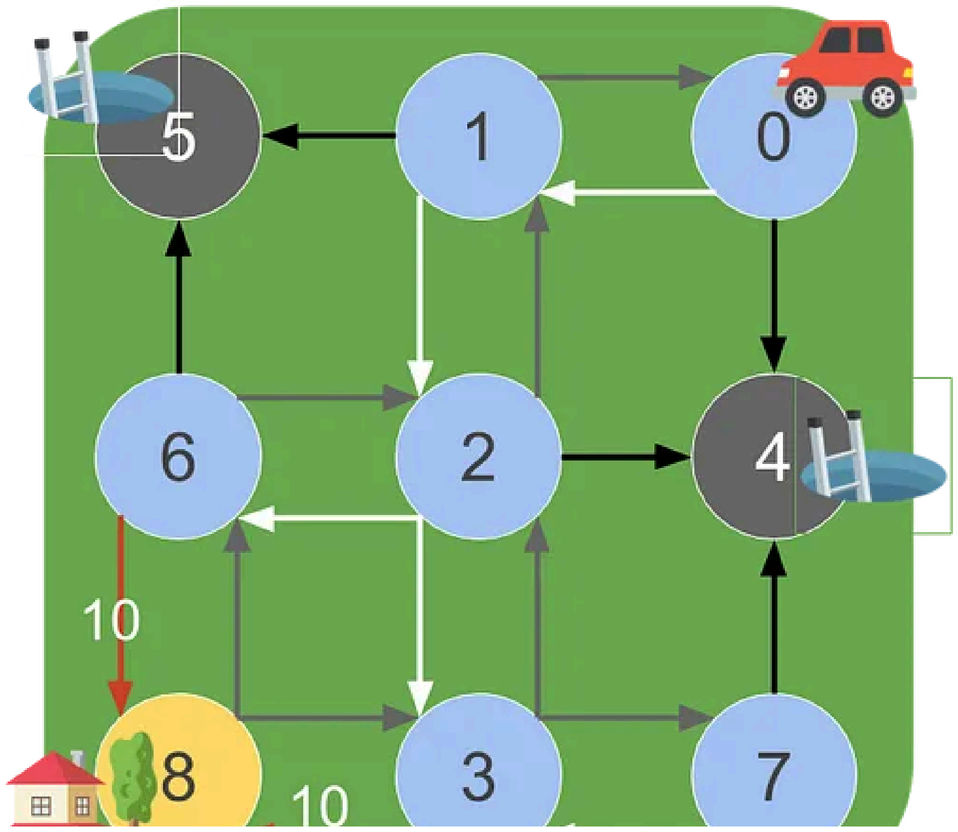
Practical Reinforcement Learning — 02 Getting started with Q-learning

Easiest introduction to Q-Learning with OpenAI Gym. Code in your browser, no installations :)

Shreyas Gite · Follow

Published in Towards Data Science · 4 min read · Apr 4, 2017

263 2



This time we will teach our self driving car to drive us home (orange node). We have to be careful though as some streets are under construction (grey node) and we don't want our car crashing into it.

As you can see we have streets numbered from 0 to 8. This gives us 9 unique **states** (streets). At any given time, our car (agent) can be in one of this 9 states. State 8 is our Goal, also called as a **terminal state.**

Our car can go left, right, up and down. To put it differently, our agent can take four different **actions**. We write it as: *a ∈ A{up,down,left,right}*

The agent receives a **reward 10 for reaching the terminal state**, other than that there are no rewards.

Q-learning and Q-table

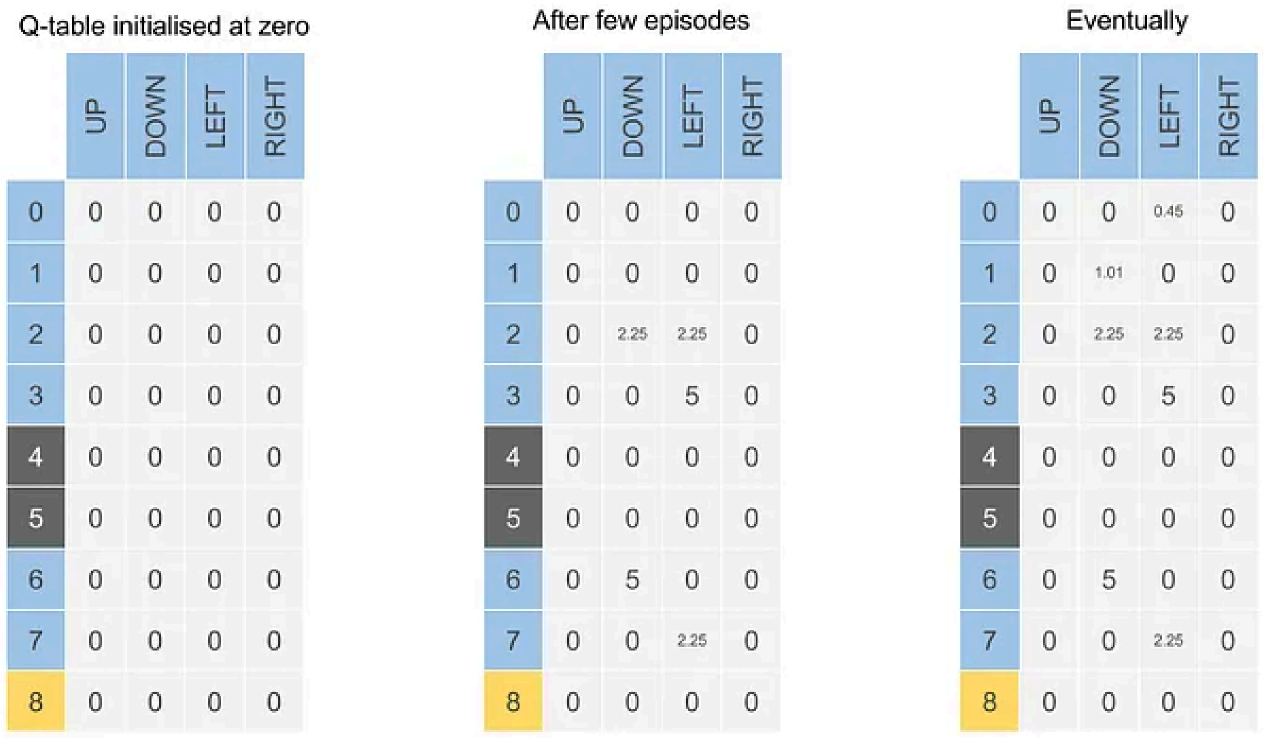
Now we'll create a matrix, "Q" also called as Q-table, this will be the brain of our agent. The matrix Q is initialized to zero, as agent starts out knowing nothing (*just like John Snow;)*). It updates Q-table with new values for state-action pair, as it learns. Here is the formula to calculate the Q[state, action]

> Q[s, a] = Q[s, a] + alpha*(R + gamma*Max[Q(s', A)] - Q[s, a])

Where;

- **alpha** is the **learning rate,**
- **gamma** is the **discount factor**. It quantifies how much importance we give for future rewards. It's also handy to approximate the noise in future rewards. Gamma varies from 0 to 1. If Gamma is closer to zero, the agent will tend to consider only immediate rewards. If Gamma is closer to one, the agent will consider future rewards with greater weight, willing to delay the reward.
- Max[Q(s', A)] gives a **maximum value of Q for all possible actions in the next state**.

The Agent explores different 'state-action' combinations till it reaches the goal or falls into the hole. We will call each of this explorations an **episode**. Each time the agent arrives at goal or is terminated, we start with next episode.



Q-table initialised at zero

	UP	DOWN	LEFT	RIGHT
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0

After few episodes

	UP	DOWN	LEFT	RIGHT
0	0	0	0	0
1	0	0	0	0
2	0	2.25	2.25	0
3	0	0	5	0
4	0	0	0	0
5	0	0	0	0
6	0	5	0	0
7	0	0	2.25	0
8	0	0	0	0

Eventually

	UP	DOWN	LEFT	RIGHT
0	0	0	0.45	0
1	0	1.01	0	0
2	0	2.25	2.25	0
3	0	0	5	0
4	0	0	0	0
5	0	0	0	0
6	0	5	0	0
7	0	0	2.25	0
8	0	0	0	0

Q-table

> *Here is some easy maths to demo how Q-table is updated:*
>
> *take learning rate (alpha) as 0.5 & discount factor (gamma) as 0.9*
>
> *Q[s, a] = Q[s, a] + alpha*(R + gamma*Max[Q(s', A)] — Q[s, a])*
>
> ***Early Episodes***
>
> *Q[3,L] = Q[3,L]+0.5*(10+0.9*Max[Q(8,U),Q(8,D),Q(8,R),Q(8,L)]-Q(3,L))*
>
> *Q[3,L] = 0 + 0.5 * (10 + 0.9 * Max [0, 0, 0, 0] -0)*
>
> *Q[3,L] = 5, Similarly Q[6,D] = 5*
>
> ***Next Episodes***
>
> *Q[2,L] = Q[2,L]+0.5*(0+0.9*Max[Q(6,U),Q(6,D),Q(6,R),Q(6,L)]-Q(2,L))*
>
> *Q[2,L] = 0 + 0.5 * (0 + 0.9 * Max [0, 5, 0, 0] -0)*
>
> *Q[2,L] = 2.25, Similarly Q[2,D] = 2.25 and Q[7,L] = 2.25*
>
> ***Eventually***
>
> *Q[1,D] = 1.0125 and Q[0,L] = 0.455625*

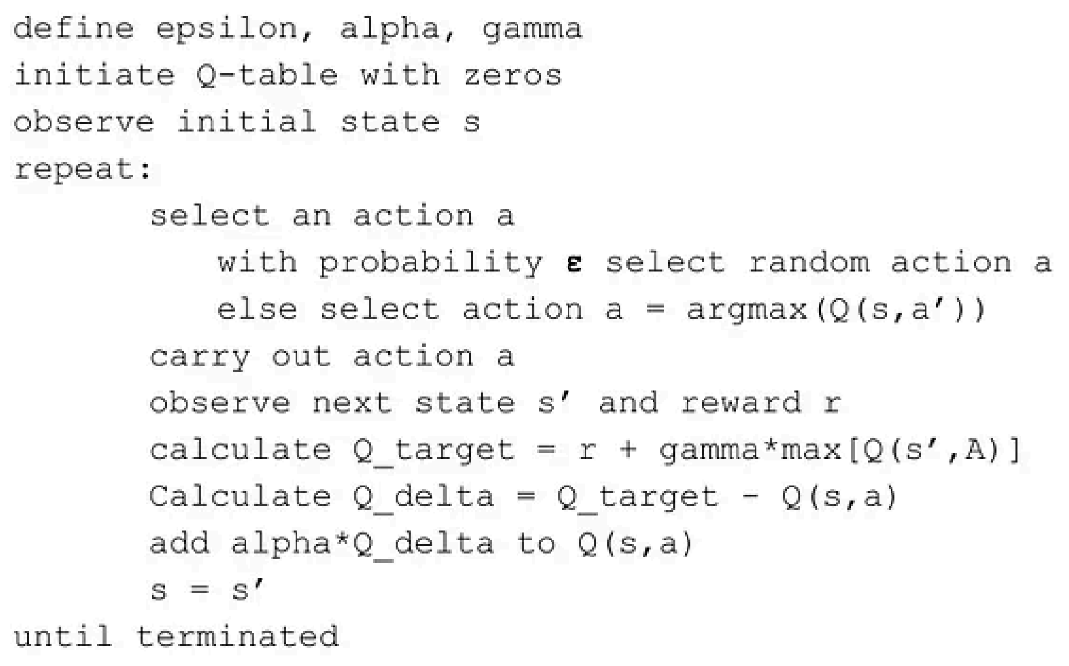
Exploration and Exploitation — Epsilon (ε)

As agent begins the learning, we would want it to take random actions to explore more paths. But as the agent gets better, the Q-function converges to more consistent Q-values. Now we would like our agent to exploit paths with highest Q-value i.e takes greedy actions. This is where epsilon comes in.

The agent takes random actions for probability ***ε*** *and greedy action for probability (1-****ε****).*

Google DeepMind used a decaying ε-greedy action selection. Where ε decays over time from 1 to 0.1 — in the beginning, the system makes completely random moves to explore the state space maximally, and then it settles down to a fixed exploration rate.

Q-learning pseudo code



```
define epsilon, alpha, gamma
initiate Q-table with zeros
observe initial state s
repeat:
      select an action a
         with probability ε select random action a
         else select action a = argmax(Q(s,a'))
      carry out action a
      observe next state s' and reward r
      calculate Q_target = r + gamma*max[Q(s',A)]
      Calculate Q_delta = Q_target - Q(s,a)
      add alpha*Q_delta to Q(s,a)
      s = s'
until terminated
```

Time for some hands-on Q-learning

Now go on and practice Q-learning with OpenAI gym. You could do it in your browser using azure notebooks. or clone this git repo.

Further resources

Reinforcement Learning: An Introduction — Chapter 6: Temporal-Difference Learning
David Silver's RL Course Lecture 4 — Model-Free Prediction
David Silver's RL Course Lecture 5 — Model-Free Control

I hope this tutorial has been helpful to those new to Q-learning and TD-reinforcement learning!

If you'd like to follow my writing on Reinforcement Learning, follow me on Medium Shreyas Gite, or on twitter @shreyasgite.
Feel free to write to me for any questions or suggestions :)

More from my Practical Reinforcement Learning series:

1. Introduction to Reinforcement Learning
2. **Getting started with Q-learning**

Machine Learning Artificial Intelligence Reinforcement Learning OpenAI