

Bellman Equation(s)

<https://www.youtube.com/watch?v=14BfO5lMiuk&list=PLWzQK00nc192L7UMJyTmLXaHa3KcO0wBT>

Dynamic Programming

- A class of algorithms
- Which seek to simplify complex problems
- By breaking them up into sub-problems
- And solving the sub-problems recursively (by a function that calls itself)

by recursively (递归) that means a function that calls itself over and over again until it comes up with the right solution.

逐步深入的调用函数自身来逐步地解决复杂问题。

$$f(x) = (\dots f(x))$$

What Question Does the Bellman Equation Answer?

- Given the state I'm in, assuming I take the best possible action now and at each subsequent step, what long-term reward can I expect?
In other words.
- What is the VALUE of the STATE?
- Helps us evaluate the expected reward relative to the advantage or disadvantage of each state

Value is a key word that the equation is solving for.

Why is this important? We want to making decisions based on the best it can do given the state it's in. so if a robot falls over and manages to get up, the getting back up is something we want to reinforce even though the reward for having fallen over may be lower even negative. so in other words, the bellman equation helps us evaluate the expected reward relative to the advantage or disadvantage of each state that we find ourselves in.

Bellman方程要回答的问题:

假设在当前状态 s 下, 取最优[1]的动作, 并在所有随后的状态, 都取最优的动作,

那么, 在当前的状态 s 下, 能获得的最优的long-term reward是多少?

[1]一个状态下的最优动作, 就是能获得maximized long-term reward的动作。

综上, 给定一个状态 s 作为bellman方程的输入, **bellman会告诉我们, agent在这个状态下, 尽最大努力后, 能够获得的long-term maximum reward**, 也就是对这个状态 s 的评价。

之前学过的state-value function也是对状态的评价, 但是定义却不同。State-value function是从 a_t 开始, 对long-term cumulative discounted reward求关于所有动作和状态的期望, **state-value function相当于agent在状态 s 下, 能获得的long-term expected reward**。

Bellman equation:

Maximum reward
in state s

$$V(s) = \max_a (R(s, a) + \gamma V(s'))$$

The value of a given state is equal to

Max action means that for all the actions available in the state we're in, we pick the action which is going to maximize the value. And then, we use the max value.

Every time we take an action we get back to the next state which is *s prime*

The function is where dynamic programming comes in, because it's **recursive**.

So, we take an *action* in a *state*, and we get a *reward* back, and we get *s Prime*. so now we take that *s Prime* and we put it into the V value function, and we continue that until we **hit the terminal state** and the episode is over. Then from there, we know **the value of the state we're in, assuming we choose the optimal action**.

如果agent能够知道在每个状态下的V(s), 那么agent就能够进行决策 — 只要顺着V(s)上升的方向做动作就行。那么, 如何求解在每个状态下的bellman方程的解?

brute forcing (backward) solution for bellman

In a state, **we have to know what the optimal action is at each step**, which will maximize the value of the expected long-term discounted rewards. In the days before the deep learning revolution, this was traditionally done by trying out all possible actions.

Let's look at a practical real-world example of **brute forcing (backward)** the Bellman equation to avoid lava pits and rescue the princess. A simplest possible scenario and assume our actions are completely **deterministic-if we're at the starting point and we choose move up is our action 100% of the time**.

Generally in reinforcement learning, we will play through an entire episode storing our state-action-reward-s prime transitions in a list, then we work **backwards** starting from the end. EXECUTING Bellman: we add the reward obtained from the current action and add the calculated value of the next step multiplied by the discount factor.

S	A	R	S'
(1,3)	right	+1	(1,4) Terminal
(1,3)	down	0	(2,3)
(1,3)	left	0	(1,2)
(1,2)	right	0	(1,3)
(1,2)	left	0	(1,1)

$$\gamma = 0.9$$

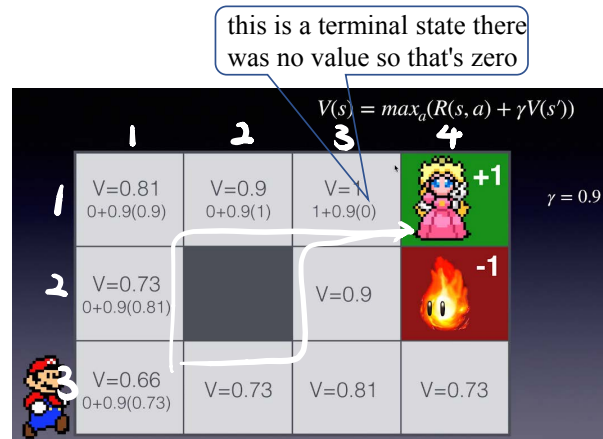
$$V_{1,3} = +1 + \gamma \cdot 0 = 1$$

$$V_{1,2} = +0 + \gamma \cdot 1 = 0.9$$

$$V_{1,1} = +0 + \gamma \cdot 0.9 = 0.81$$

⋮

(2, 3)	up	0	(1, 3)
(2, 3)	right	-1	(2, 4) Terminal
(2, 3)	down	0	(3, 3)
(1, 1)	right	0	(1, 2)
(1, 1)	down	0	(2, 1)
(3, 3)	up	0	(2, 3)
(3, 3)	right	0	(3, 4)
(3, 3)	left	0	(3, 2)
	⋮		



Once we calculate the values, then **the policy: we're going to take the action that's going to bring us into the state with the highest value.** so we have a very easy way to see path here or we could follow around this path.

不用上面的暴力法，可不可以求 $V(s)$? 如果你能知道在 s 及其之后状态的最优动作，就能求。
但问题是，如果我都知道最优动作，就相当于有policy了，还要贝尔曼方程指导干啥?

Instead of brute force in every possible action every possible state, the network can simply estimate the value of the state and intelligently guess which action will maximize it.

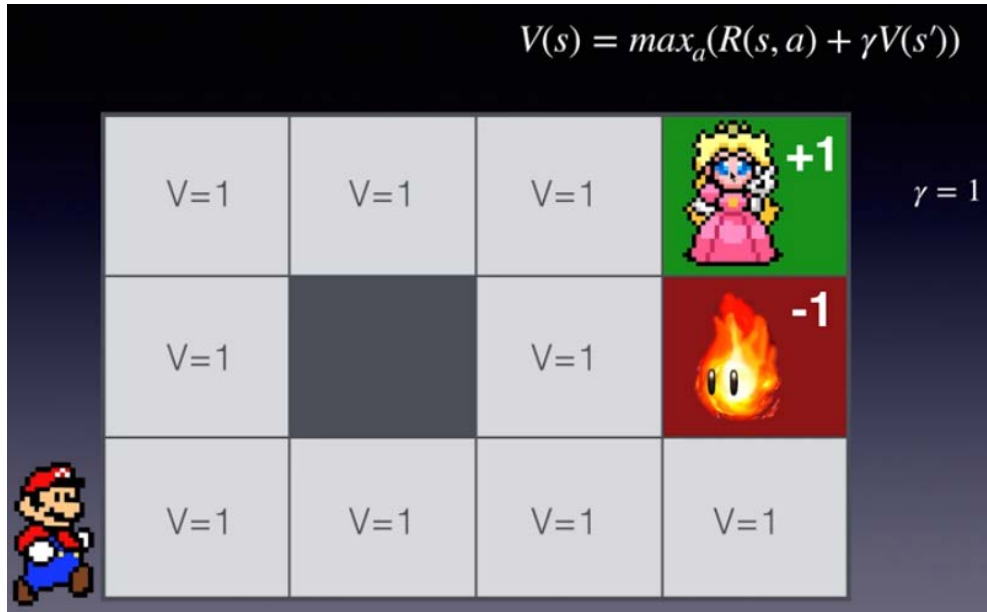
Gamma Tips

- It is important to tune this hyperparameter to get optimum results
- Successful values range between 0.9 and 0.99.
- A lower value encourages short-term thinking
- A higher value emphasizes long-term rewards

$$V(s) = \max_a (R(s, a) + \gamma V(s'))$$

Gamma越小，当前的奖励越重要；越大，长远的奖励越重要。

Why the discount factor gamma is so important? If gamma=1:



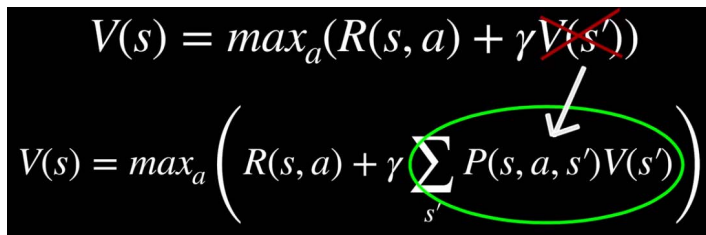
you can basically see how we basically end up wandering around aimlessly until we accidentally someone to the goal.

action \rightarrow state 具有随机性情况下的bellman方程

What is a stochastic Markov decision process?

We're in a state, we choose an action and now there are several possible s-prime states we could end up in, based on random probability.

Each possible state transition from an action has an exact probability and all the probabilities add up to 1. Remember the Markov property: the probability of future transitions depends only on the present and the past doesn't matter.

$$V(s) = \max_a (R(s, a) + \gamma \cancel{V(s')})$$
$$V(s) = \max_a \left(R(s, a) + \gamma \sum_{s'} P(s, a, s') V(s') \right)$$


- 1) Loop through every possible next state
- 2) Multiply the value of that state by its probability of occurring
- 3) Sum them all together!

仅作参考

Bellman Backup

<https://www.youtube.com/watch?v=WnI-Qh2UHGg&t=1402s>

Value Functions

► Definitions (review):

1. The expected return obtained upon performing action a in state s and subsequently following policy π is denoted $q_{\pi}(s, a)$ and is given by

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi} [r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \mid s_0 = s, a_0 = a]$$

Called Q-function or state-action-value function
or action-value function

2. $V^{\pi}(s) = \mathbb{E}_{\pi} [r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \mid s_0 = s]$ 相比于Q, V的Expectation
将action a_0 也包含进去了
 $= \mathbb{E}_{a \sim \pi} [Q^{\pi}(s, a)]$ 也可以理解成, 将Q再次求关于 a_0 的期望, a_0

Called state-value function

服从policy $\pi(a|s)$ 分布。

3. $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$
Called advantage function

Bellman Equations for Q^π

- ▶ Bellman equation for Q^π

$$\begin{aligned} Q^\pi(s_0, a_0) &= \mathbb{E}_{s_1 \sim P(s_1 | s_0, a_0)} [r_0 + \gamma V^\pi(s_1)] \\ &= \mathbb{E}_{s_1 \sim P(s_1 | s_0, a_0)} [r_0 + \gamma \mathbb{E}_{a_1 \sim \pi} [Q^\pi(s_1, a_1)]] \end{aligned}$$

- ▶ We can write out Q^π with k -step empirical returns

$$\begin{aligned} Q^\pi(s_0, a_0) &= \mathbb{E}_{s_1, a_1 | s_0, a_0} [r_0 + \gamma V^\pi(s_1, a_1)] \\ &= \mathbb{E}_{s_1, a_1, s_2, a_2 | s_0, a_0} [r_0 + \gamma r_1 + \gamma^2 Q^\pi(s_2, a_2)] \\ &= \mathbb{E}_{s_1, a_1, \dots, s_k, a_k | s_0, a_0} \left[r_0 + \gamma r_1 + \dots + \gamma^{k-1} r_{k-1} + \gamma^k Q^\pi(s_k, a_k) \right] \end{aligned}$$

Bellman Backups

- ▶ From previous slide:

$$Q^\pi(s_0, a_0) = \mathbb{E}_{s_1 \sim P(s_1 | s_0, a_0)} [r_0 + \gamma \mathbb{E}_{a_1 \sim \pi} [Q^\pi(s_1, a_1)]]$$

- ▶ Define the Bellman backup operator (operating on Q -functions) as follows

$$[\mathcal{T}^\pi Q](s_0, a_0) = \mathbb{E}_{s_1 \sim P(s_1 | s_0, a_0)} [r_0 + \gamma \mathbb{E}_{a_1 \sim \pi} [Q(s_1, a_1)]]$$

- ▶ Then Q^π is a *fixed point* of this operator

$$\mathcal{T}^\pi Q^\pi = Q^\pi$$

- ▶ Furthermore, if we apply \mathcal{T}^π repeatedly to any initial Q , the series converges to Q^π

$$Q, \mathcal{T}^\pi Q, (\mathcal{T}^\pi)^2 Q, (\mathcal{T}^\pi)^3 Q, \dots \rightarrow Q^\pi$$