Value Iteration
Reinforcement Learning Review

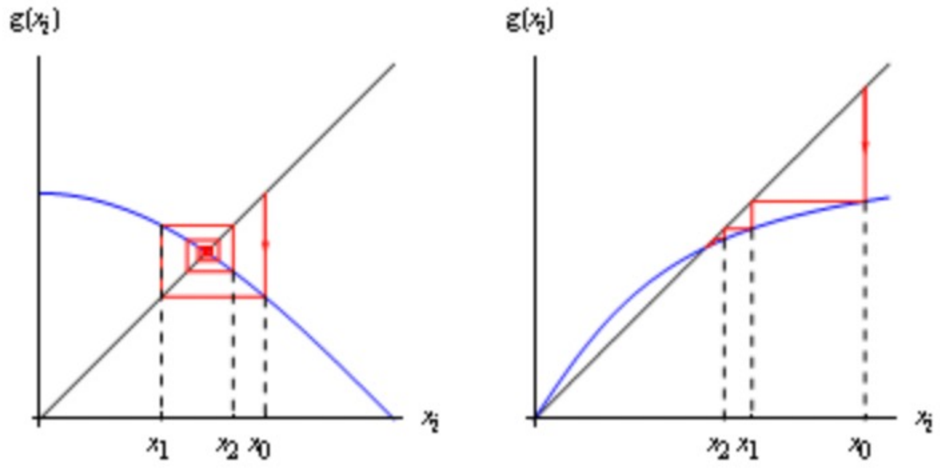
Based on Prof. Oh's Reinforcement Learning Lectures

Suinne Lee

Dynamic Programming: find the optimal policy (model based: known state-transition probability)

- Value Iteration
  - use Bellman Optimality Equation
- Policy Iteration
  - use Bellman Expectation Equation

Value Iteration – Basic Idea



Start with an estimator and iterate

Value Iteration – Basic Idea

- Uses the Bellman Optimality Equation

Optimal State Value Function $$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')]$$

(gives how to compute the optimal state value function without computing for the entire epoch)

- Uses it how? Iteratively!

$$V_{k+1}(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_k(s')]$$

- Initialize → Iterate → if it converges, we find the optimal value function.

Value Iteration – Basic Idea

2. Iterate.

$$V_{k+1}(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_k(s')]$$

We find the next (iterated) value function estimator $V_{k+1}(s)$

by finding the maximum value of $\sum_{s',r} p(s',r|s,a)[r + \gamma V_k(s')]$

V_k(s) over all possible states s' and action a. (Action taken at state s')

Value Iteration – Basic Idea

2. Iterate.

$$V_{k+1}(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_k(s')]$$

If this converges after multiple iterations, we are likely to have found the maximum value function for state s and an action a at each s' that achieves it.

This gives the optimal policy and this gives the optimal value function.

Compute the optimal policy π_* (one-step lookahead)

$$\pi_*(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V^*(s')]$$

Value Iteration – Basic Idea

3. Upon convergence, we find the optimal policy:

$$\pi_*(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V^*(s')]$$

Ways to iterate?

- Synchronous

  Compute for all states simultaneously.

  i.e. in each iteration, we compute V_k+1(s) for all s.

  We use the previous values V_k(s') for all s' to compute V_k+1.

- Asynchronous

  Compute for one state, update.

  We use the updated V_k+1(s) to compute for other states.

Can choose which states to update, and make the computation more compact.

$$V_{k+1}(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_k(s')]$$

Advantages

- Conceptually simple
- **Memory** efficient (only need to store and update value functions) (but as we will see, this does not necessarily mean it is computationally low cost.)

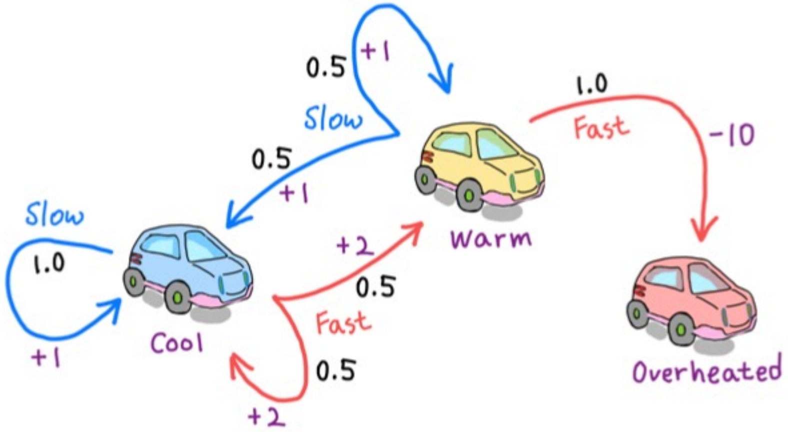
Disadvantages

- Takes long to converge / policy converges faster than value in a lot of cases
- In exploitation vs exploration, the value iteration leans more towards exploitation. Policy iteration is better for exploring the whole of the state space effectively. https://en.wikipedia.org/wiki/Exploration-exploitation_dilemma
- Computationally expensive (updates all the value functions for the states at each iteration) O(SS^2A)

Example 1 - Car

A robot car wants to travel far and quickly.

- 3 states: cool, warm, overheated
- 2 actions: slow, fast
- rewards: slow = 1, fast = 2 (but −10 when overheated)

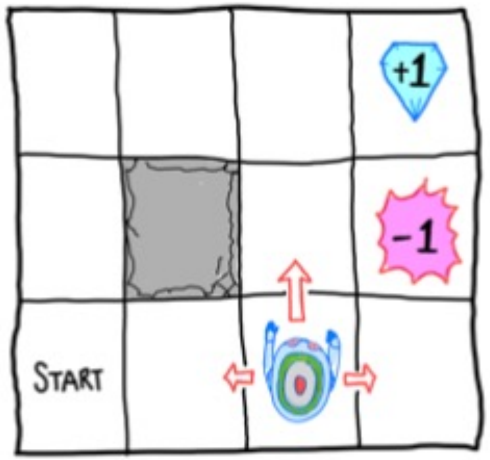


$V(s)$	cool	warm	over
V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0

$$V_2(\text{Warm}) = \max_a \begin{Bmatrix} 0.5(1+2) + 0.5(1+1) \\ 1.0(-10+0) \end{Bmatrix}$$

$$V_{k+1}(s) \leftarrow \max_a \sum_{s',r} p(s',r \mid s,a)[r + \gamma V_k(s')]$$

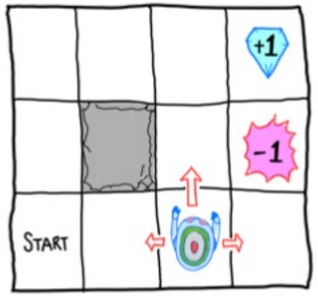
Example 2 – Grid World



noise = 0.2, discount = 0.9, living reward c = 0

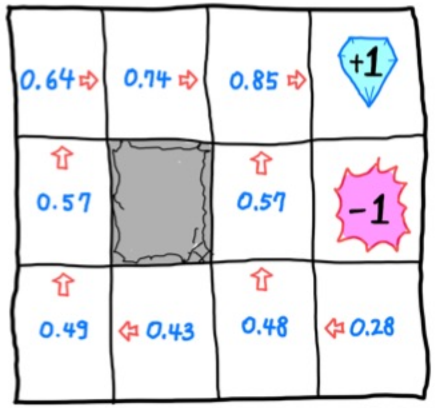
https://gibberblot.github.io/rl-notes/single-agent/value-iteration.html

Example 2 – Grid World

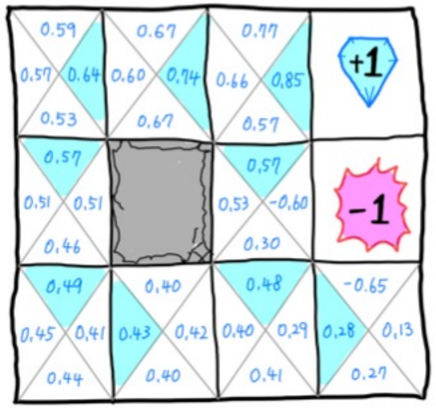


noise = 0.2, discount = 0.9, living reward c = 0

Values after 100 iterations



Q-values after 100 iterations



Q-table

actions

states

	⇑	⇓	⇐	⇒
1	0.49	0.44	0.45	0.41
2	0.40	0.40	0.43	0.42
3	0.48	0.41	0.40	0.29
⋮	⋮	⋮	⋮	⋮
9	0.77	0.57	0.66	0.85

Q-values after 100 iterations

$$\pi_*(s) = \arg\max_a \textstyle\sum_{s',r} p(s',r \mid s,a)[r + \gamma V^*(s')]$$
$$= \arg\max_a Q^*(s,a)$$

$$V_{k+1}(s) \leftarrow \max_a \textstyle\sum_{s',r} p(s',r \mid s,a)[r + \gamma V_k(s')]$$

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

Hyperparameter: small threshold $\epsilon > 0$ for the convergence check
Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$, except $V(terminal) = 0$

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

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