Reinforcement Learning

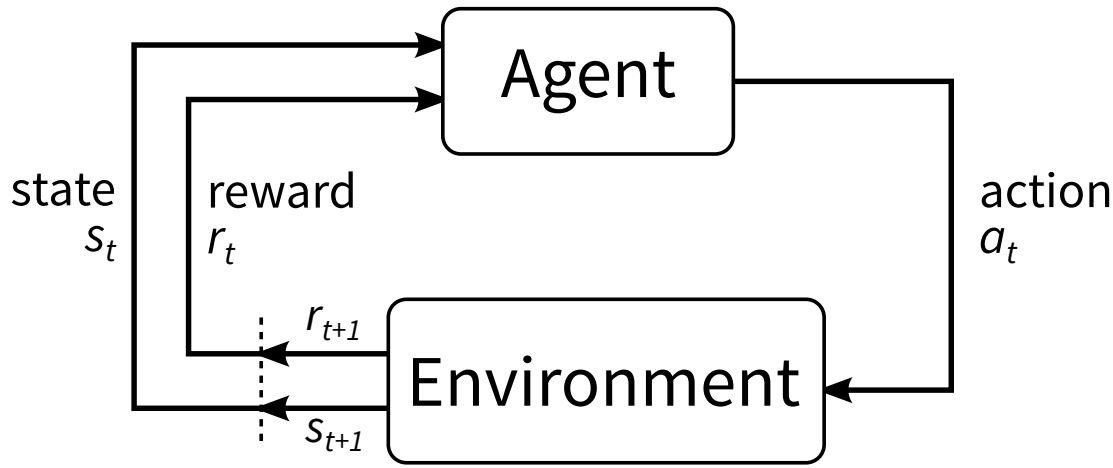
Part II – Methods

Nicolas P. Rougier
INRIA Bordeaux Sud-Ouest
Institute of Neurodegenerative Diseases

3rd Latin America Summer School in Computational Neuroscience
13-31 January 2014, Valparaiso, Chile

Problem

- The learner and decision maker is called the *agent*
- Everything outside is called the *environment*



- The agent and environment interact at discrete time steps $t = 0, 1, 2, \ldots$
- What to do at step t ?

Methods

1. Dynamic Programming

The model of the environment (T and R) must be known.

2. Monte Carlo Methods

No model is required but learning occurs only at the end of an episode.

- On-policy $\rightarrow$ Behavior policy $=$ Estimation policy
- Off-policy $\rightarrow$ Behavior policy $\neq$ Estimation policy

3. Temporal-Difference Learning

Learning occurs at each time step.

- Model based $\rightarrow$ Build a model of T and R
- Model free $\rightarrow$ Build π^* directly

We can use one step ($TD(0)$) or n-step ($TD(\lambda)$) lookups.

Reminder

State-Value function

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\}$$

Action-Value function

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\}$$

Bellman equation

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')\right]$$

Dynamic programming

Policy Evaluation & Improvement

(Dynamic programming)

Policy Evaluation

Require: π the policy to be evaluated
  $V(s) \leftarrow 0$ for all s in S
  repeat
    for each s in S **do**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$
      $\Delta \leftarrow max(\Delta, |v - V(s)|$
    end for
  until $\Delta < \varepsilon$ (small positive number)
Ensure: $V \approx V_\pi$

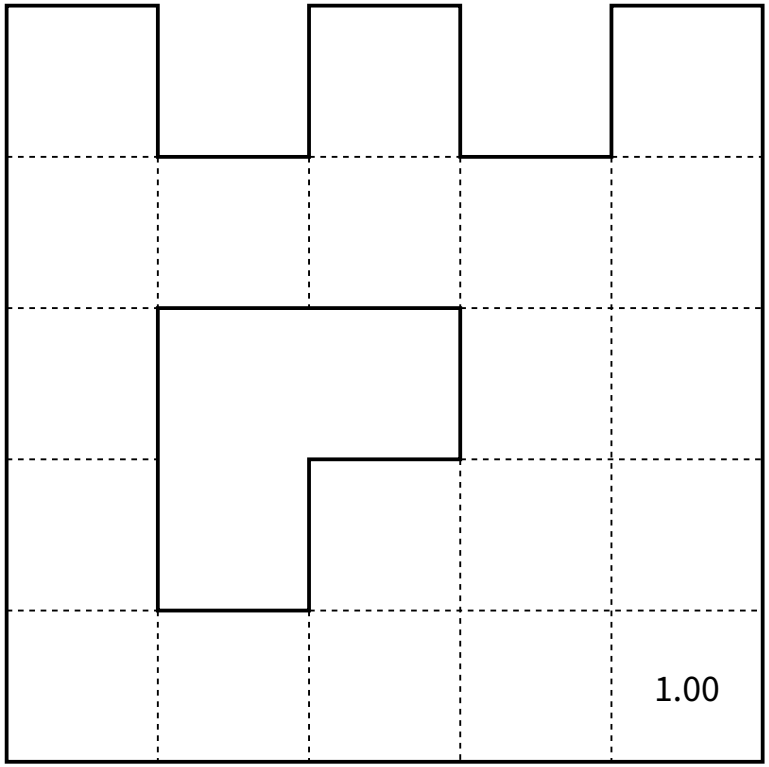
Policy Improvement

Let us modify the policy π anf check if it is better or not.

Policy Iteration

(Dynamic Programming)

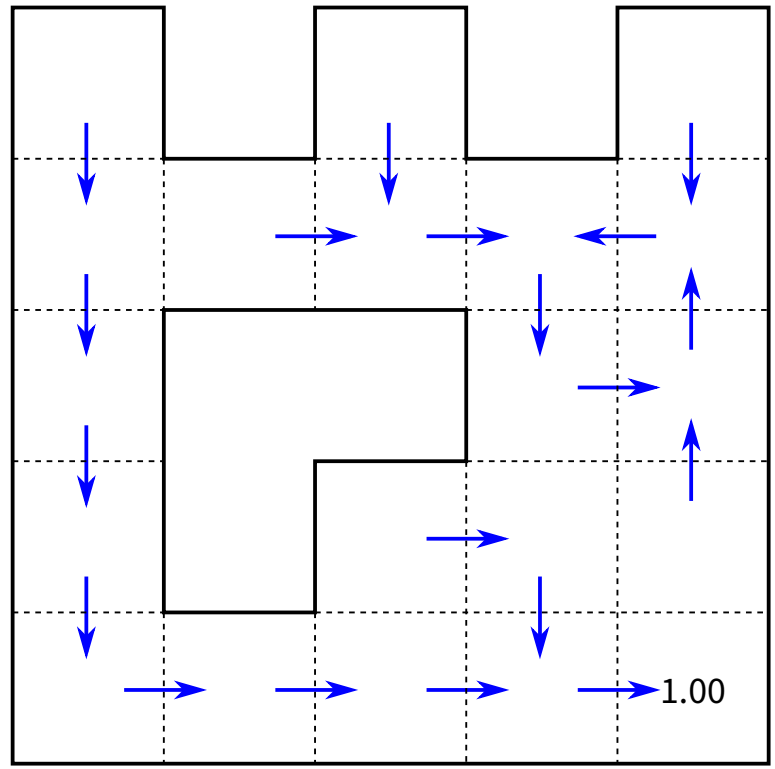
Start froms a random policy and for each state s, choose the best action a' such that $V(\pi(s, a')) \geq V(\pi(s, a))$



Policy Iteration

(Dynamic Programming)

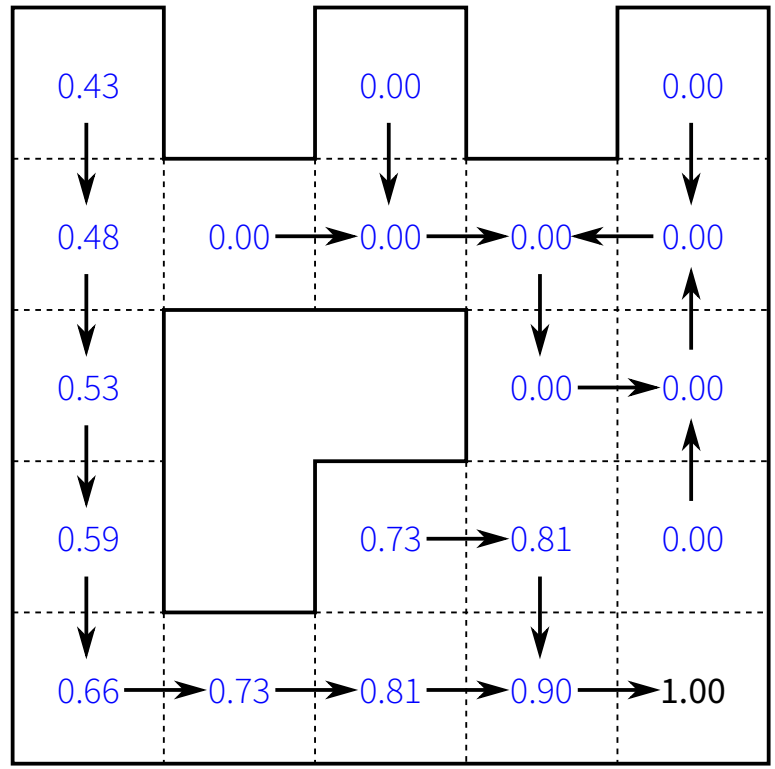
- Pick random policy π_0



Policy Iteration

(Dynamic Programming)

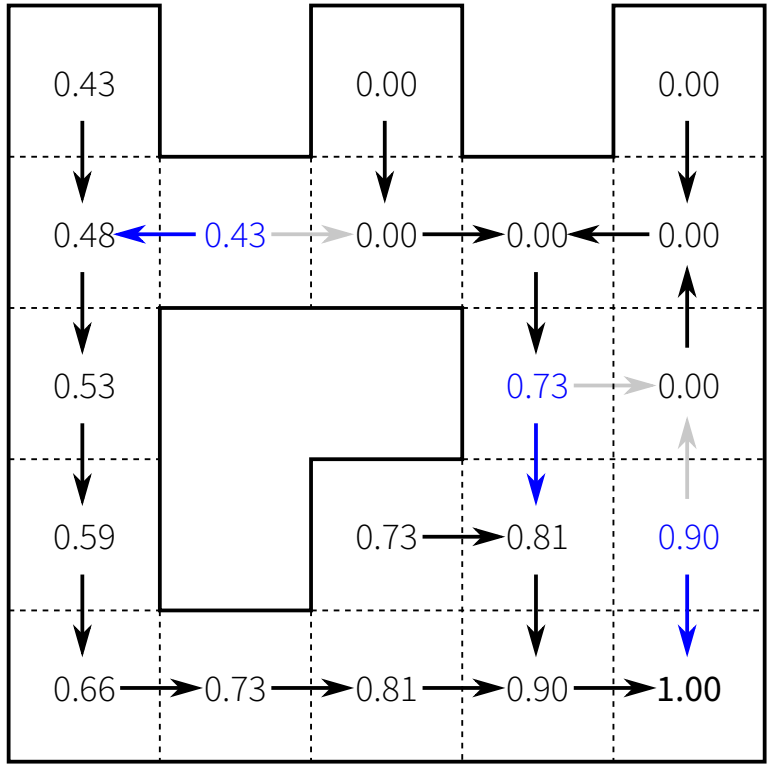
- Pick random policy π_0
- Evaluation of π_0



Policy Iteration

(Dynamic Programming)

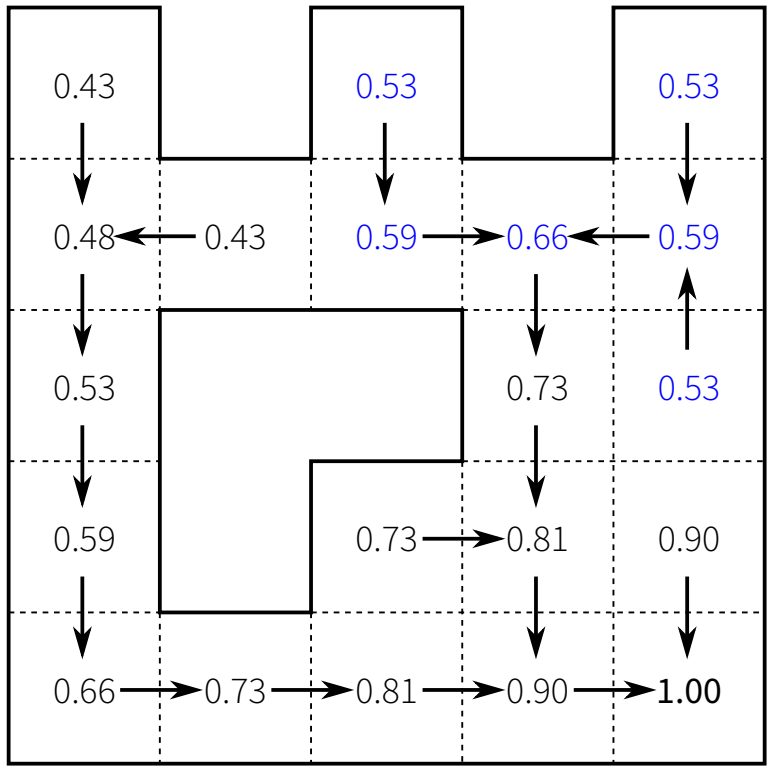
- Pick random policy π_0
- Evaluation of π_0
- Improvement of $\pi_0 \rightarrow \pi_1$



Policy Iteration

(Dynamic Programming)

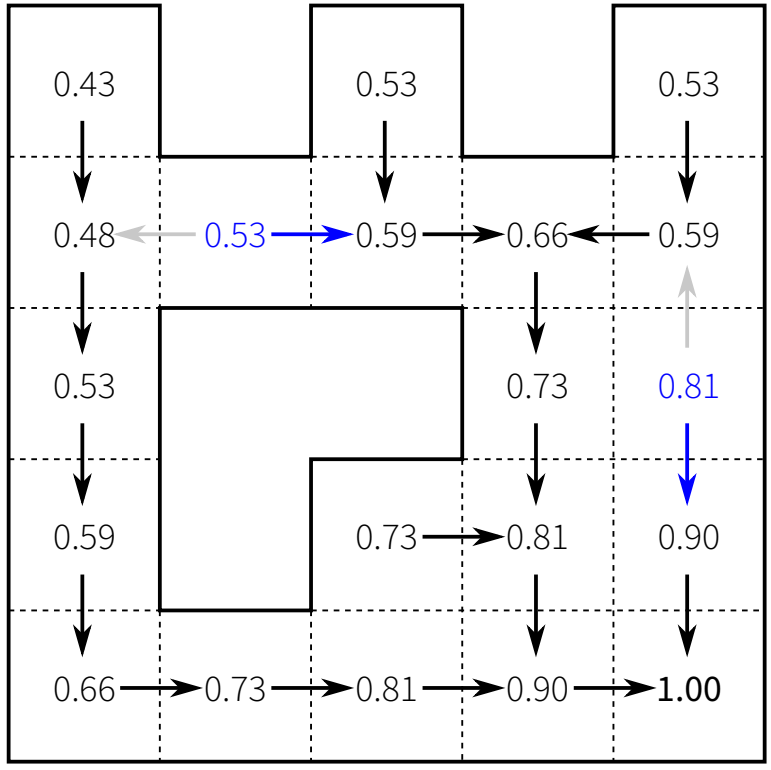
- Pick random policy π_0
- Evaluation of π_0
- Improvement of $\pi_0 \rightarrow \pi_1$
- Evaluation of π_1



Policy Iteration

(Dynamic Programming)

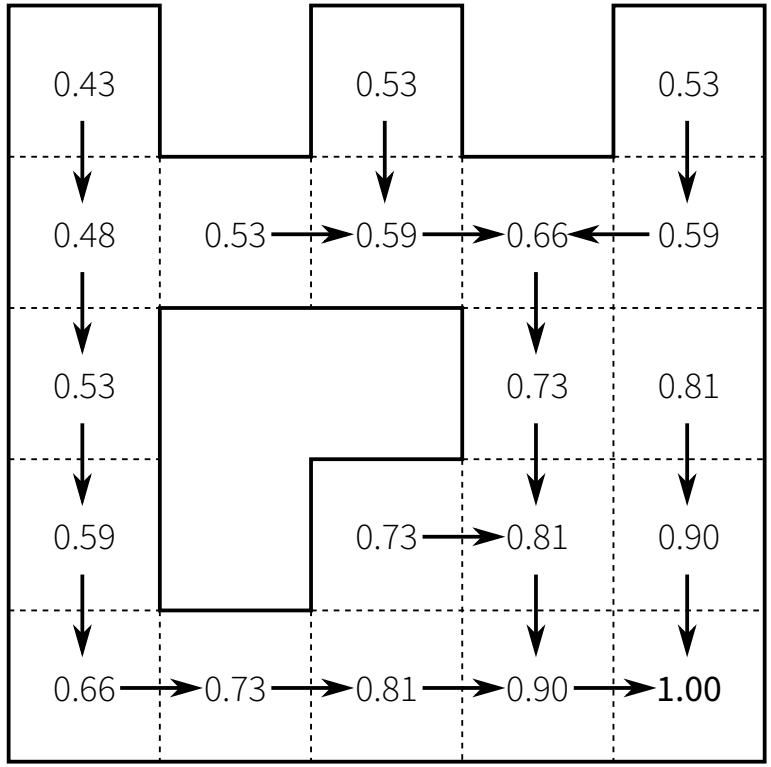
- Pick random policy π_0
- Evaluation of π_0
- Improvement of $\pi_0 \rightarrow \pi_1$
- Evaluation of π_1
- Improvement of $\pi_1 \rightarrow \pi_2$



Policy Iteration

(Dynamic Programming)

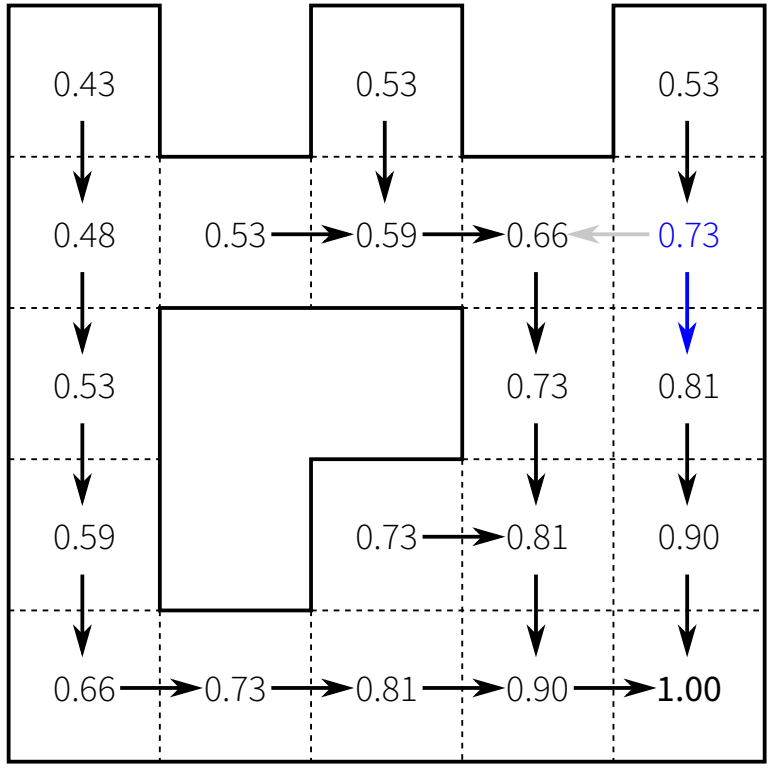
- Pick random policy π_0
- Evaluation of π_0
- Improvement of $\pi_0 \rightarrow \pi_1$
- Evaluation of π_1
- Improvement of $\pi_1 \rightarrow \pi_2$
- Evaluation of π_2



Policy Iteration

(Dynamic Programming)

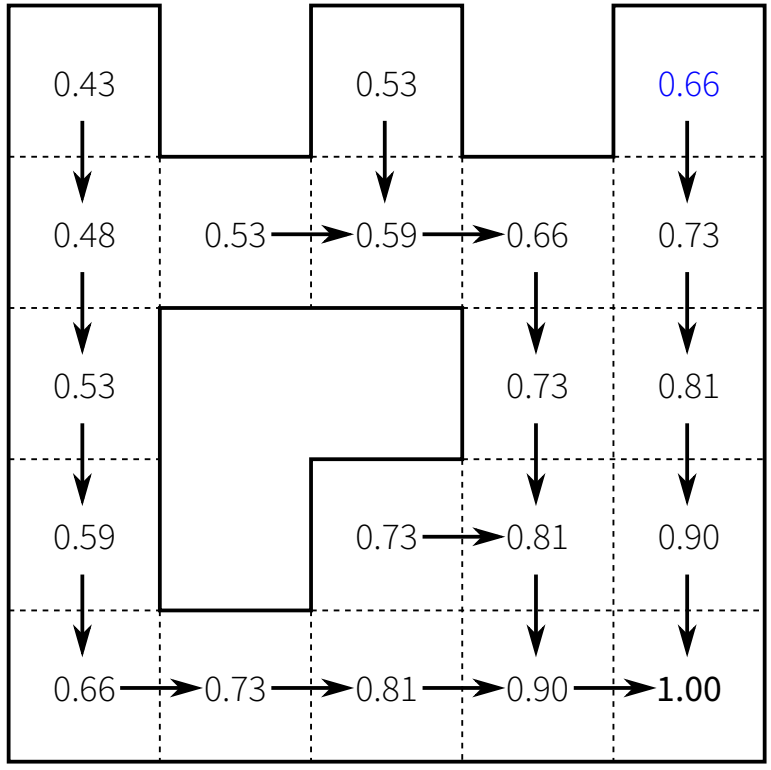
- Pick random policy π_0
- Evaluation of π_0
- Improvement of $\pi_0 \rightarrow \pi_1$
- Evaluation of π_1
- Improvement of $\pi_1 \rightarrow \pi_2$
- Evaluation of π_2
- Improvement of $\pi_2 \rightarrow \pi_3$



Policy Iteration

(Dynamic Programming)

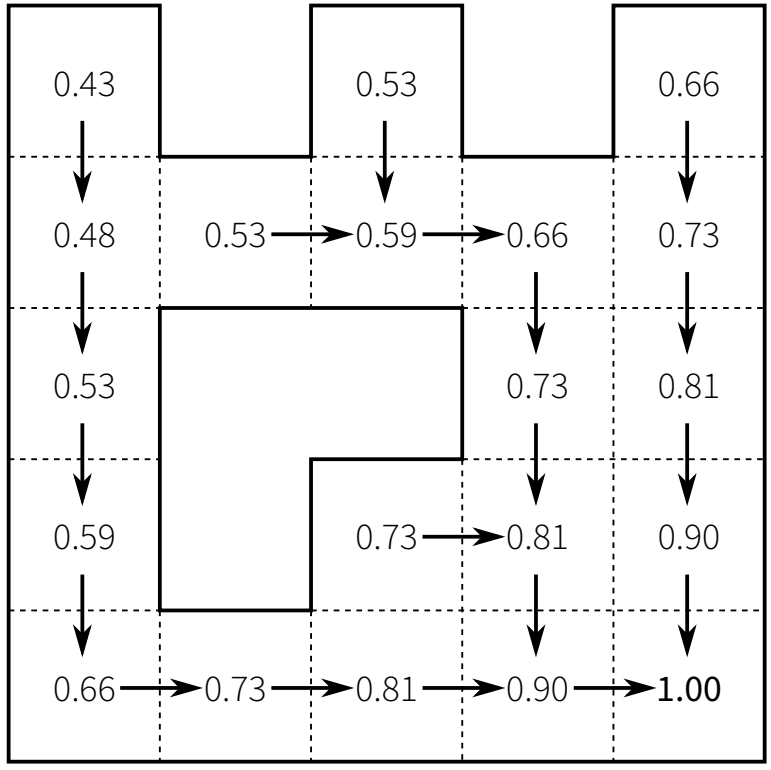
- Pick random policy π_0
- Evaluation of π_0
- Improvement of $\pi_0 \rightarrow \pi_1$
- Evaluation of π_1
- Improvement of $\pi_1 \rightarrow \pi_2$
- Evaluation of π_2
- Improvement of $\pi_2 \rightarrow \pi_3$
- Evaluation of π_3



Policy Iteration

(Dynamic Programming)

- Pick random policy π_0
- Evaluation of π_0
- Improvement of $\pi_0 \rightarrow \pi_1$
- Evaluation of π_1
- Improvement of $\pi_1 \rightarrow \pi_2$
- Evaluation of π_2
- Improvement of $\pi_2 \rightarrow \pi_3$
- Evaluation of π_3
- Done



Value Iteration

(Dynamic Programming)

Require: π the policy to be evaluated

$V(s) \leftarrow 0$ for all s in S

repeat

for each s in S **do**

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$

$\Delta \leftarrow max(\Delta, |v - V(s)|$

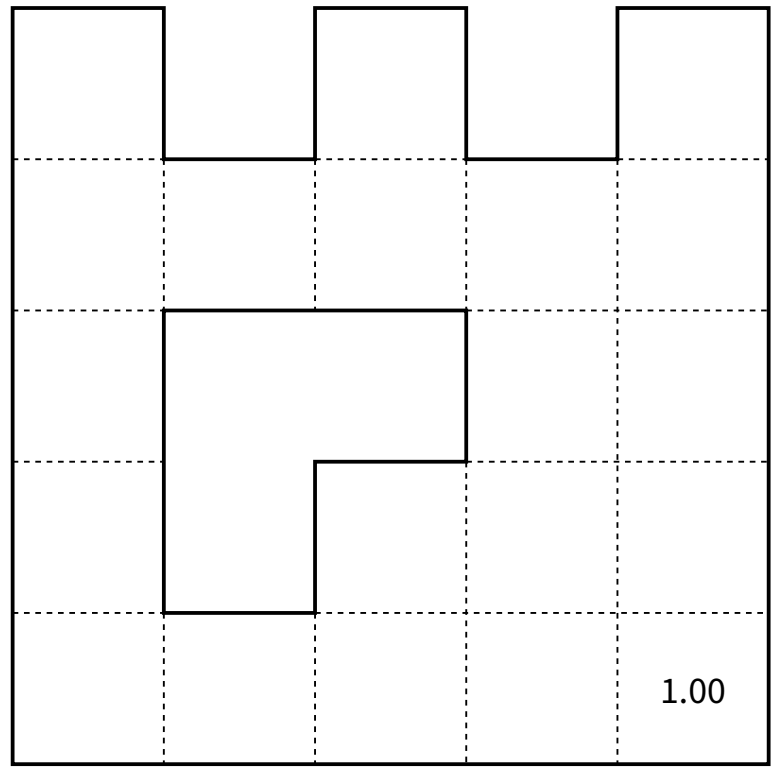
end for

until $\Delta < \varepsilon$ (small positive number)

Then we get $\pi(s) = argmax_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$

Value Iteration

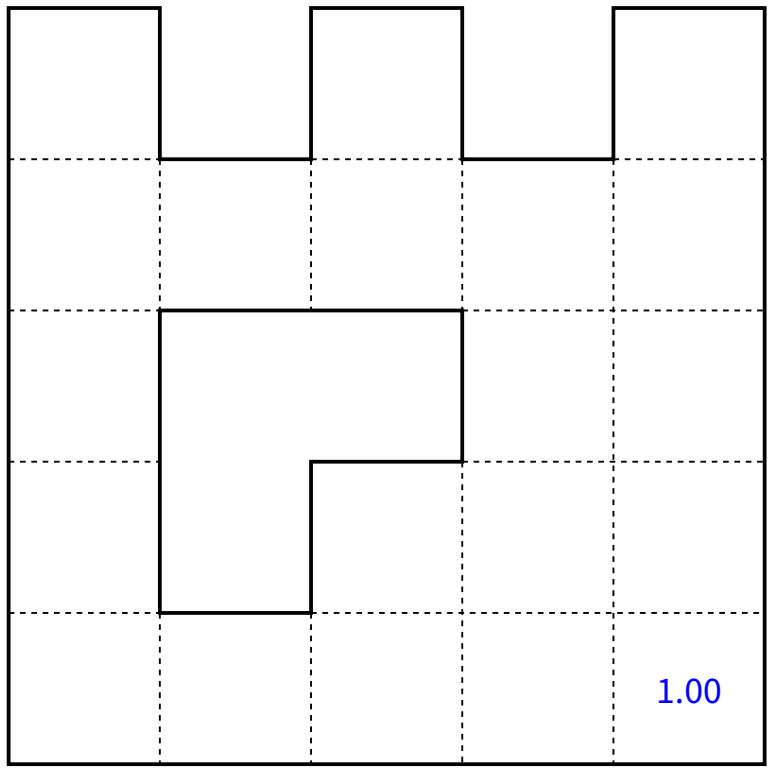
(Dynamic Programming)



Value Iteration

(Dynamic Programming)

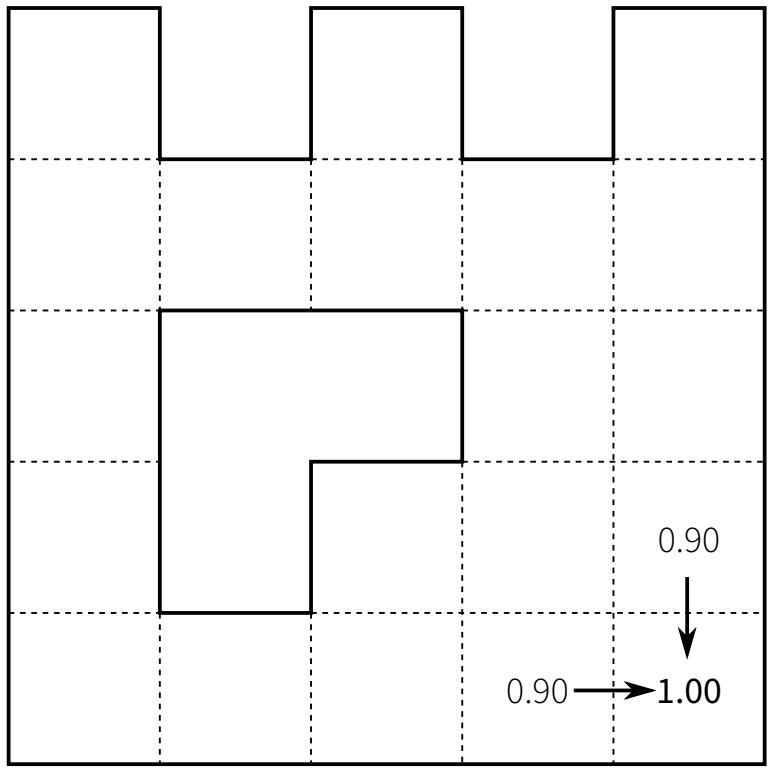
- Start from V = 0 everywhere



Value Iteration

(Dynamic Programming)

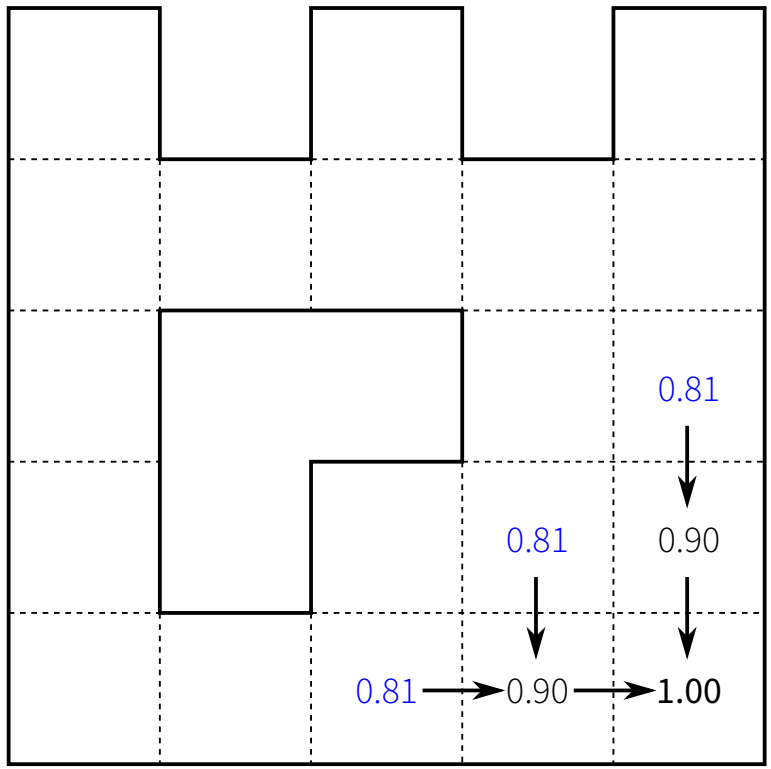
- Start from V = 0 everywhere
- Evaluation (greedy policy)



Value Iteration

(Dynamic Programming)

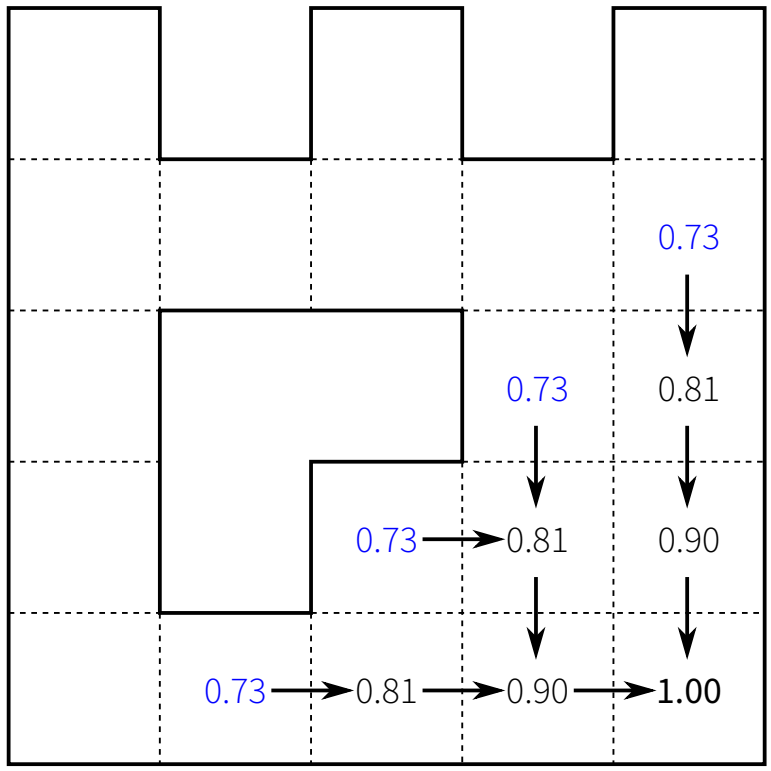
- Start from V = 0 everywhere
- Evaluation (greedy policy)
- Evaluation (greedy policy)



Value Iteration

(Dynamic Programming)

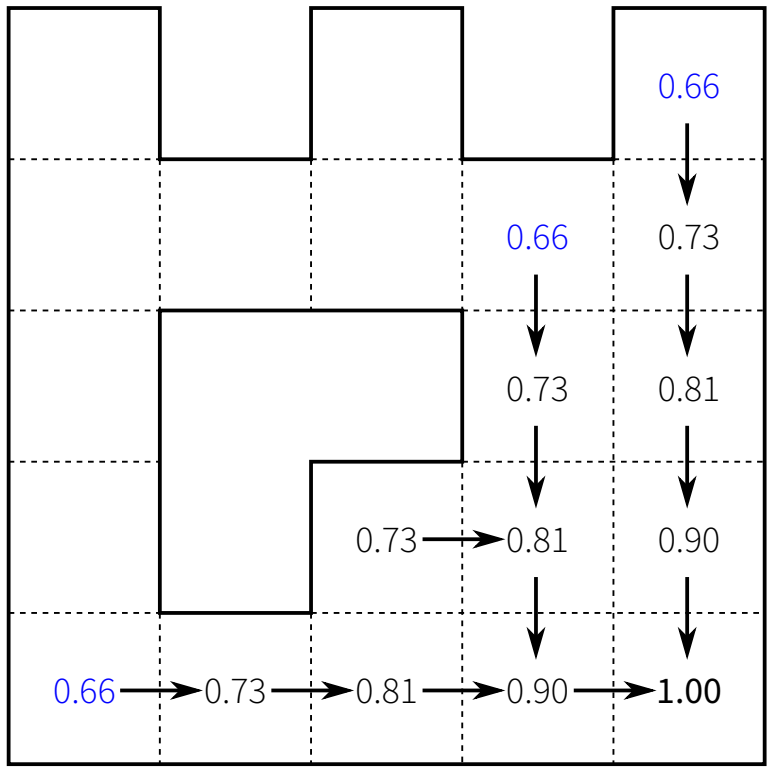
- Start from V = 0 everywhere
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)



Value Iteration

(Dynamic Programming)

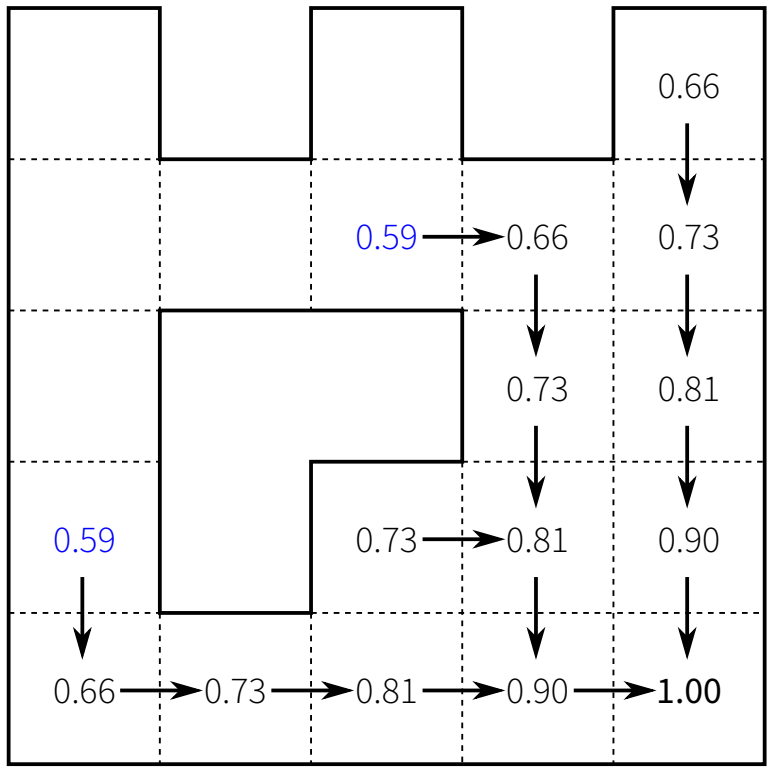
- Start from V = 0 everywhere
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)



Value Iteration

(Dynamic Programming)

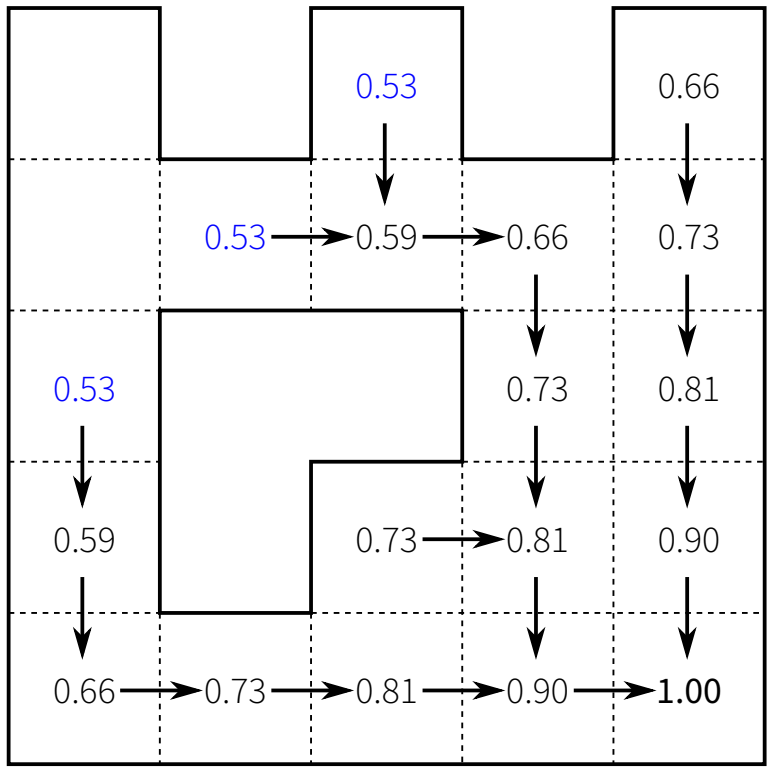
- Start from V = 0 everywhere
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)



Value Iteration

(Dynamic Programming)

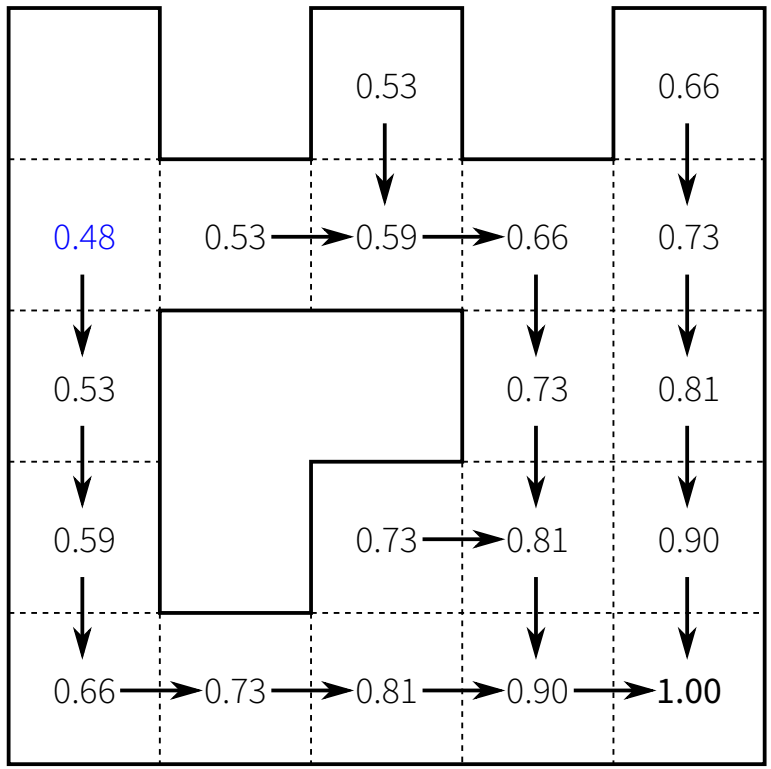
- Start from V = 0 everywhere
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)



Value Iteration

(Dynamic Programming)

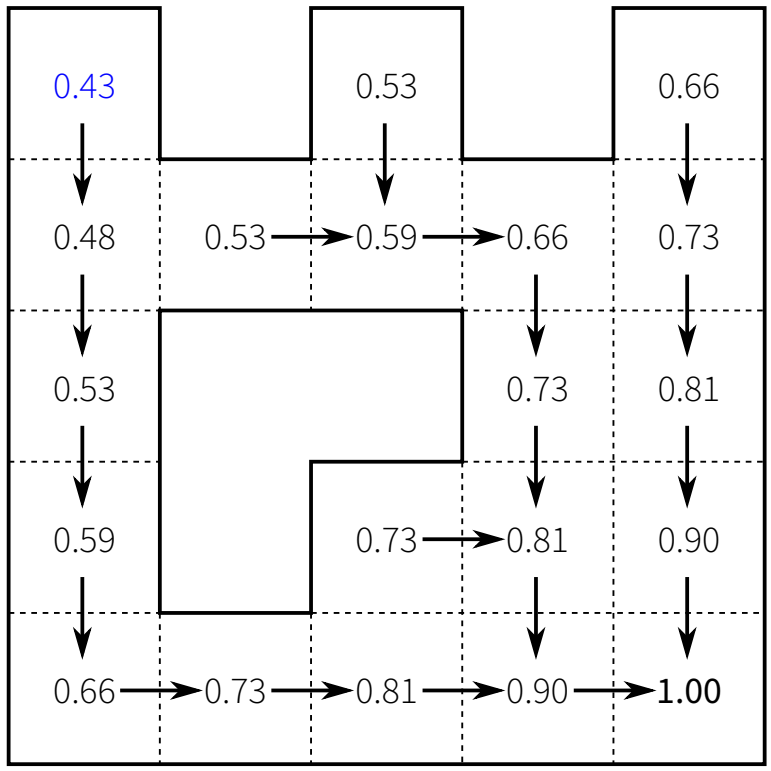
- Start from V = 0 everywhere
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)



Value Iteration

(Dynamic Programming)

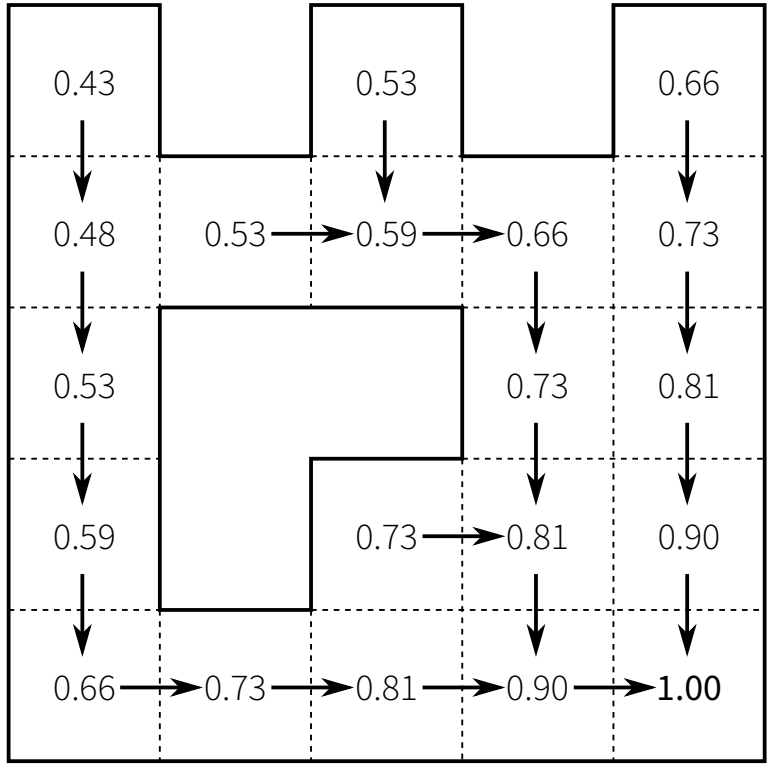
- Start from V = 0 everywhere
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)



Value Iteration

(Dynamic Programming)

- Start from V = 0 everywhere
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Evaluation (greedy policy)
- Done



Monte-Carlo methods

(not covered in this tutorial)

Temporal difference learning

Temporal difference learning

Let's go watch a movie

You're going to the movie theater that happens to be sit at the end of a very long one-way street. There are places to park your car all along the street, but as you approach the theater, it is more and more unlikely you'll find a free place. And if you don't find a free place, you will have to go back to the beginning of the street and you'll be late for the movie. Since you do not want to walk, you want to find a place that is both free and close to the theater.

Where do you choose to park ?

- As soon as you see a free place (exploitation) ?
- Or you feel lucky and try to go a little further (exploration) ?

Exploration vs. Exploitation

Online decision-making involves a fundamental choice between:

Exploitation

- To make the best decision given current information.
- Short-term, immediate, certain benefits.

Exploration

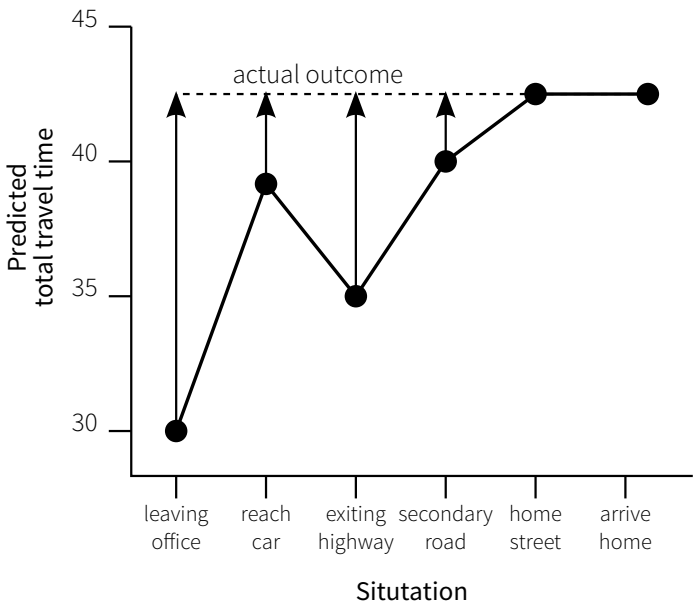
- To gather more information to make a better decision.
- Long-term, risky, uncertain.

To solve the dilemna, a trade-off has to be found between exploration and exploitation (ϵ-greedy, upper-confidence bounds, bayes rules, etc.)

Time of change

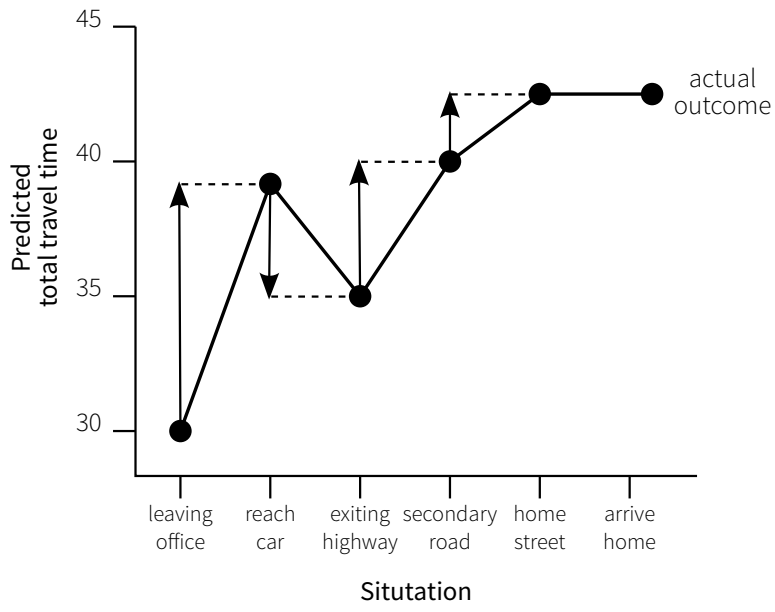
Monte Carlo / Reinforcement learning

Monte Carlo update



Change occurs at the end of an episode.

Reinforcement learning update



Change occurs ater one or several steps (TD(0), TD(λ))

Reinforcement learning

TD methods learn their estimates in part on the basis of other estimates.They learn a guess from a guess–they bootstrap.

And yet, we can still guarantee convergence to the correct answer.

TD(0)

(Simplest method)

TD(0)

Require: π the policy to be evaluated

$V(s) \leftarrow 0$ for all s in S

repeat for each episode

$a \leftarrow \pi(s)$

repeat for each step of episode

Take action a, observe reward r and next state s'

$V(s) \leftarrow (1 - \alpha)V(s) + \alpha(r + \gamma V(s'))$

$s \leftarrow s'$

until s is terminal

until no more episode

Note: we need r and s' to update $V(s)$. We need 1-step backup.

Sarsa

(On-policy, temporal difference)

Sarsa = $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$

Require: Arbitrary Q(s,a)

repeat for each episode

Choose a from s using policy derived from Q (e.g. ϵ-greedy)

repeat for each step of episode

Take action a, observe reward r and next state s'

Choose a' from s' using policy derived from Q (e.g. ϵ-greedy)

$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma Q(s',a'))$

$s \leftarrow s', a \leftarrow a'$

until s is terminal

until no more episodes

Q-Learning

(Off-policy, temporal difference)

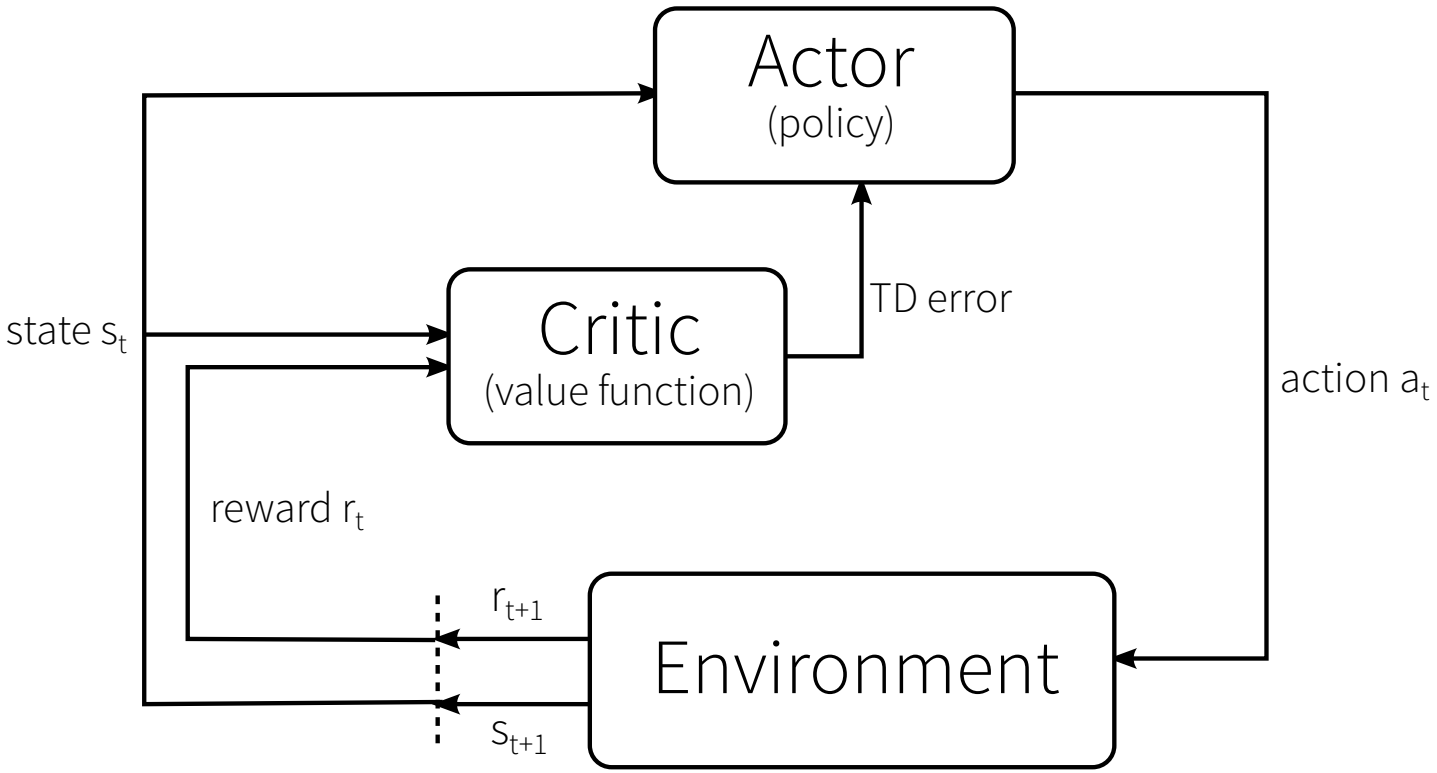
Q-Learning (Watkins, 1989)

Require: Arbitrary Q(s,a)
  repeat for each episode
    repeat for each step of episode
      Choose a from s using policy derived from Q (e.g. ϵ-greedy)
      Take action a, observe reward r and next state s'
      $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma max_a Q(s',a))$
      $s \leftarrow s'$
    until s is terminal
  until no more episodes

Actor-critic methods

(On-policy, temporal difference)

Actor-critic methods are TD methods that have a separate memory structure to explictly represent the policy independent of the value function.



TD(λ)

n-step TD prediction

- Monte-Carlo methods use full episode lookup
- TD(0) methods use one step lookup
- TD(λ) methods use n $-$ ***step*** lookup

n-step backup

In one-step backups the (one step) target $G_t^{(1)}$ is the first reward plus the discounted estimated value of the next state:

$$V(s_t) \leftarrow (1-\alpha)V(s_t) + \alpha(\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{G_t(1)=target})$$

where, by definition:

$$V(s_{t+1}) = r_{t+2} + \gamma V_{t+2}$$

Hence, a two-step target would be:

$$G_t^{(2)} = r_{t+1} + \gamma(r_{t+2} + \gamma V(s_{t+2}))$$

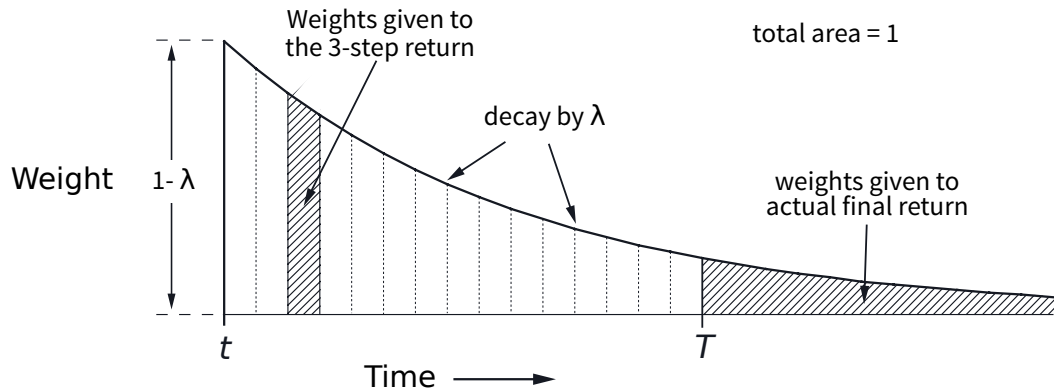
More generally we have:

$$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^n V(s_{t+n})$$

Eligibility traces

TD(λ)

Backups can be done not just toward any n-step return, but toward any average of n-step returns. The $TD(\lambda)$ algorithm is one particular way of averaging n-step backups: $G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$

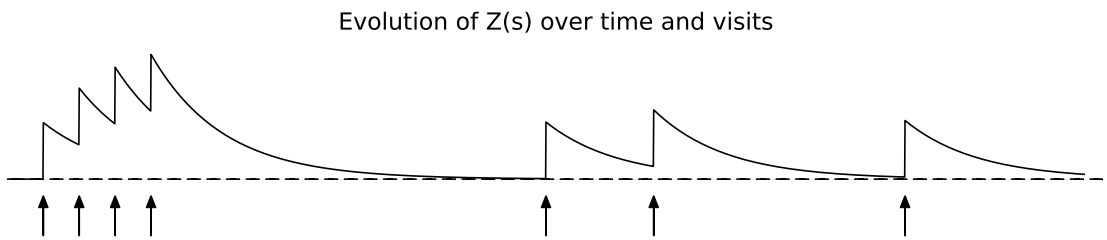


For each state visited, we look forward in time to all the future rewards and decide how best to combine them. This requires knowledge of the future.

Eligibility traces

An eligibility trace is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action.The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. When a TD error occurs, only the eligible states or actions are assigned credit or blame for the error.

$$Z_t(s) = \begin{cases} \gamma\lambda Z_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda Z_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$$



Eligibility traces are a basic mechanism for temporal credit assignment

TD(λ)

$V(s) \leftarrow 0$ for all s in S
repeat for each episode
  $Z(s) \leftarrow 0$ for all s in S
  repeat for each step of episode
    Take action a, observe reward r and next state s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $Z(s) \leftarrow Z(s) + 1$
    for all s in S **do**
      $V(s) \leftarrow V(s) + \alpha\delta Z(s)$
      $Z(s) \leftarrow \gamma\lambda Z(s)$
    end for
    $s \leftarrow s'$
  until s is terminal
until no more episode

Take-Away message

- A Markov Decision Process (MDP) is a tuple (S, A, T, R).
- Different type of learning methods:
  - Dynamic programming (T and R are known)
  - Monte Carlo (learning at end of episode)
  - Temporal Difference (one step, learning at each time step)
  - Eligibility traces (n-step, learing at each time step)
- State-Value function
  $V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\}$
- Action-Value function
  $Q^{\pi}(s, a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\}$

Reference

Reinforcement learning : An Introduction, Richard S. Sutton and Andre G. Barto