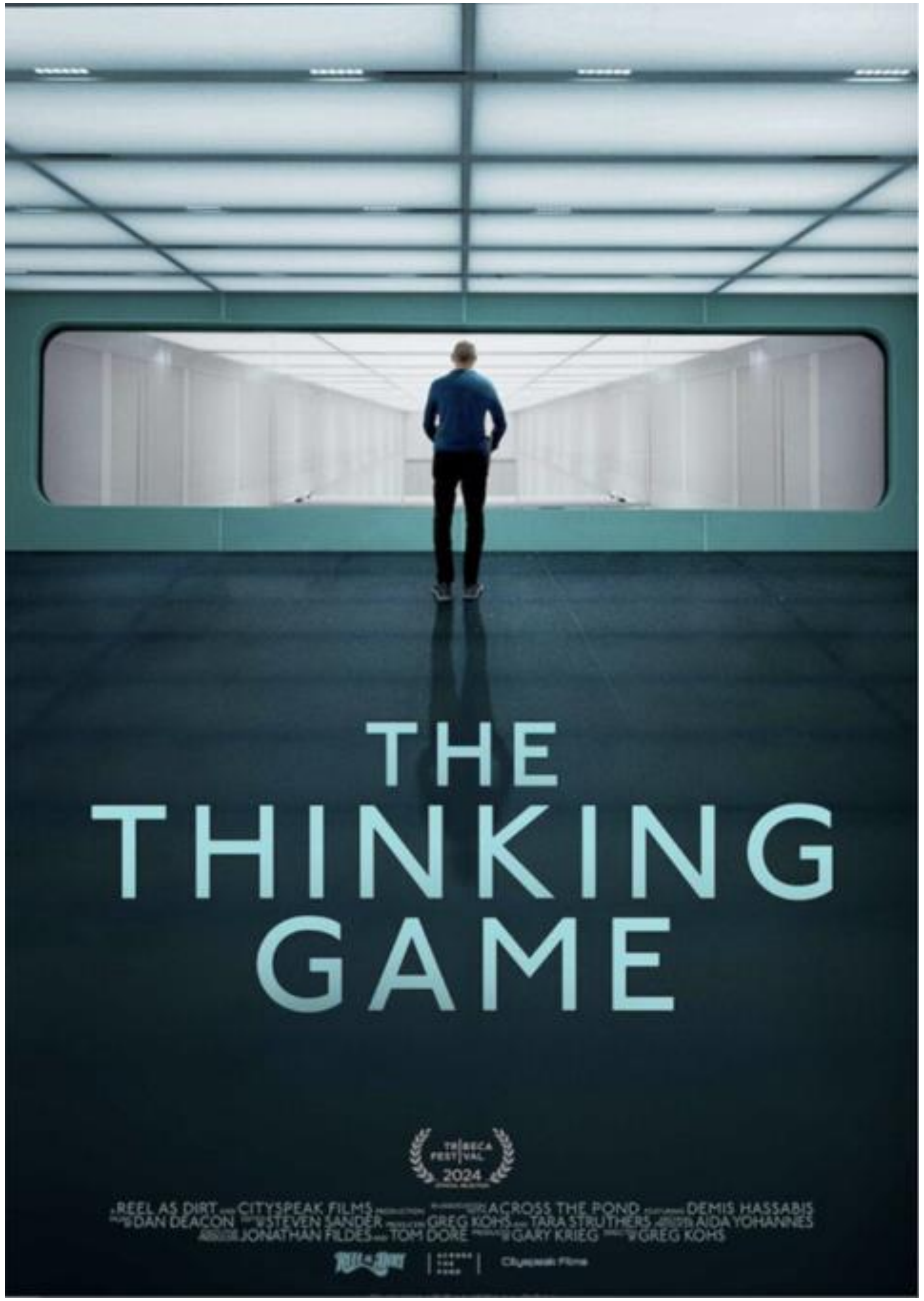
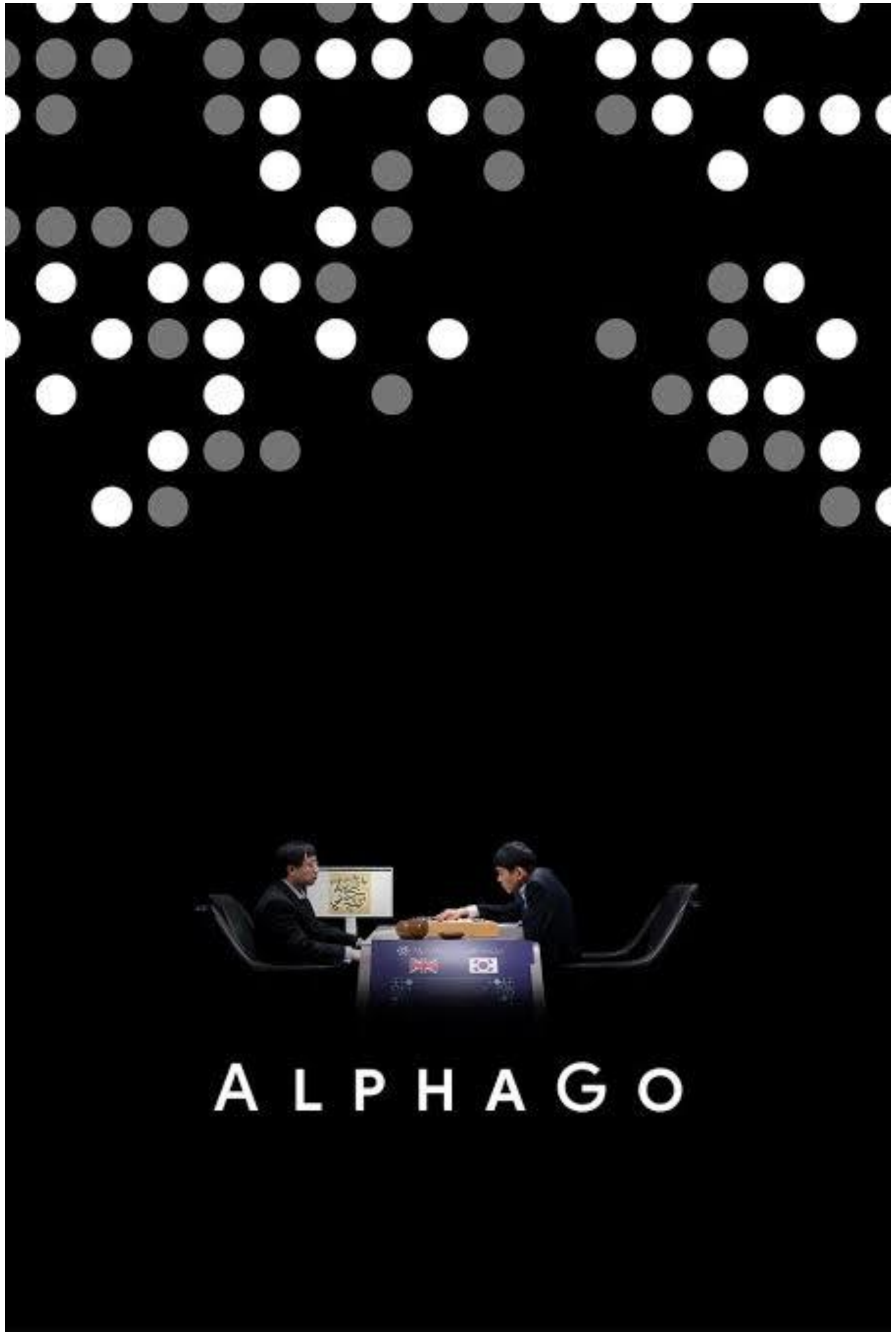
UNIVERSITY
SCHOOL OF SCIENCE & TECHNOLOGY

Lecture 4
Dynamic Programming

jmanero@faculty.ie.edu


THE
THINKING
GAME
TRIBECA FESTIVAL
2024
REEL AS DIRT
CITYSPEAK FILMS
ACROSS THE POND
DEMIS HASSABIS
DAN DEACON
STEVEN SANDER
GREG KOHS
TARA STRUTHERS
AIDA YOHANNES
JONATHAN FILDES
TOM DORE
GARY KRIEG
GREG KOHS
Cityspeak Films

Must Watch



Lee Sedol against AlphaGo

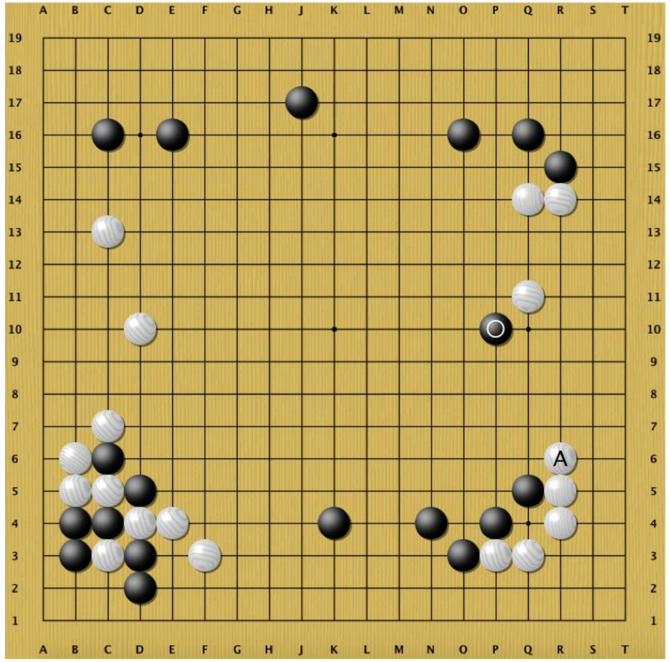
https://www.wired.com/2016/03/two-moves-alphago-lee-sedol-redefined-future/

Movement 37 2nd game

https://www.youtube.com/watch?v=HT-UZkiOLv8

Lee Sedol against AlphaGo

AlphaGo Move 37 in second game was creative and Unique. No human would've ever made

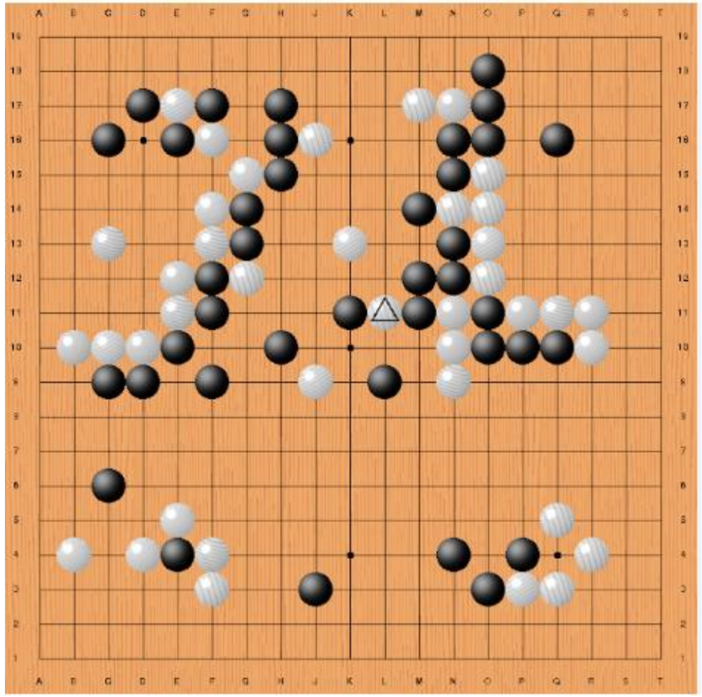


Black 37! This move proved so stunning that, when it appeared on the screen, many players thought the stone had been put down in the wrong place.

https://deepmind-media.storage.googleapis.com/alphago/pdf-files/english/ls-vs-ag2/LS%20vs%20AG%20-%20G2%20-%20English.pdf
https://www.wired.com/2016/03/two-moves-alphago-lee-sedol-redefined-future/

Lee Sedol against AlphaGo

Lee Sedol's Move 78 in fourth game was a strange move (it sacrificed stones to create a wedge move)



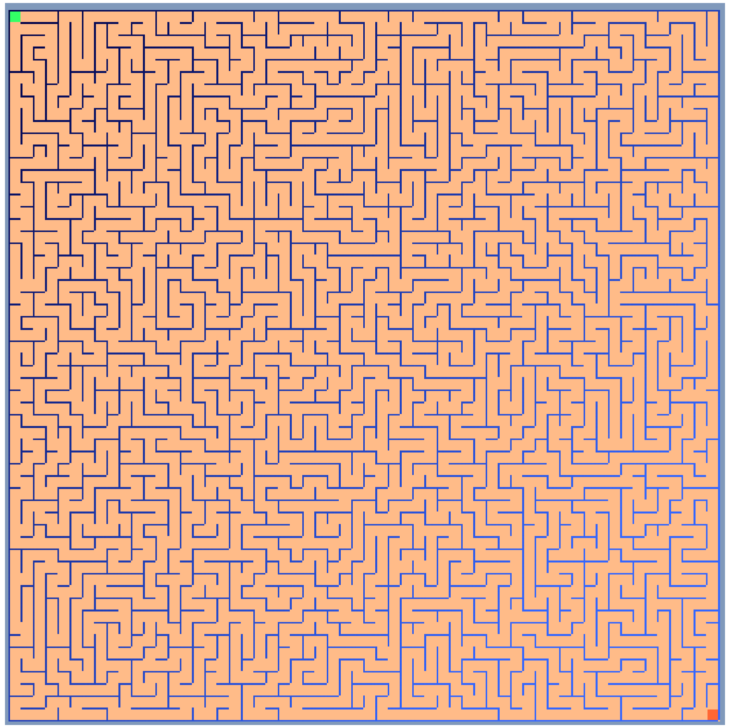
White 78! This move was so unexpected that made AlphaGO to collapse. It was an unlikely movement.

https://deeplearningskysthelimit.blogspot.com/2016/04/part-9-review-of-game-4-lee-sedols.html

Contents

Heuristics can work too

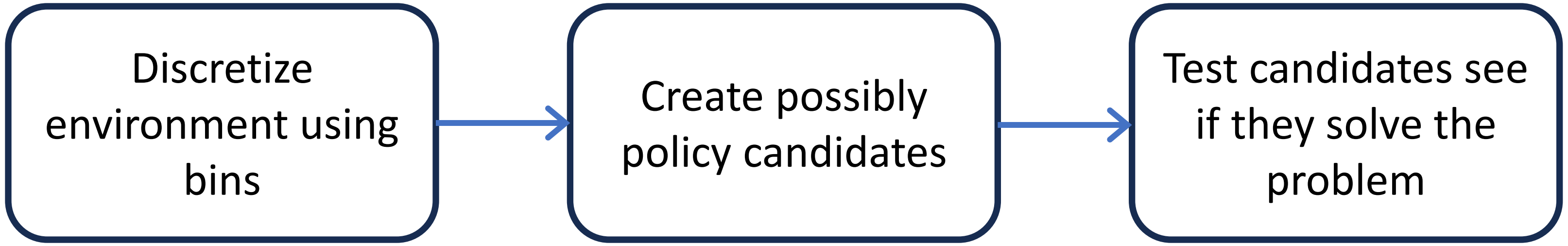
- Why if we devise a method that guides the agent in its universe?
- This method based on a rule that we apply to 'solve' the universe, rule that may be based on a sophisticated 'rule-of thumb'
- Is this an algorithm? Can we consider it Learning?



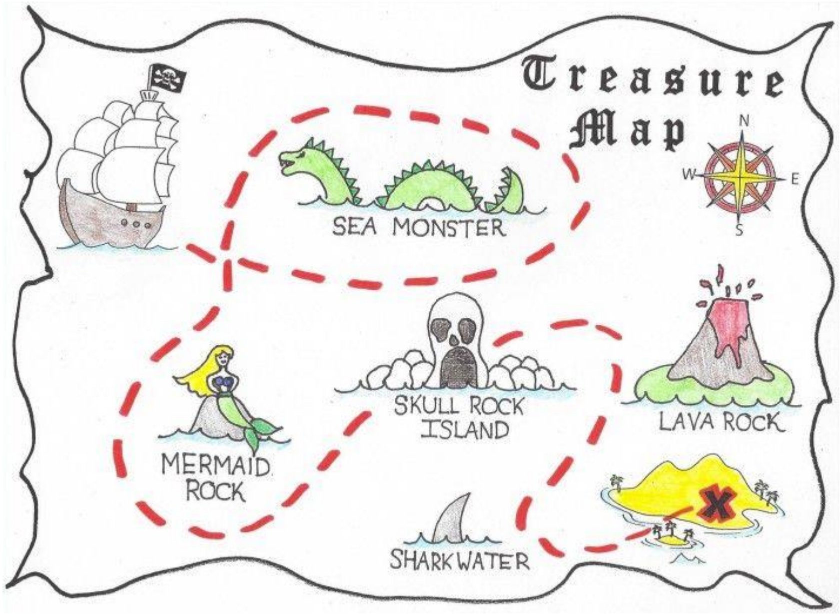
- Heuristic rule. Always turn left (or follow the left wall)
- Heuristic rules so clear and simple are difficult to find
- What happens in random environments? (example slippery ice, …)
- Impossible to find when the Action or Observation space is very large

Example of Heuristics

https://github.com/castorgit/RL_course/blob/main/050_CARTPOLE_heuristics.ipynb



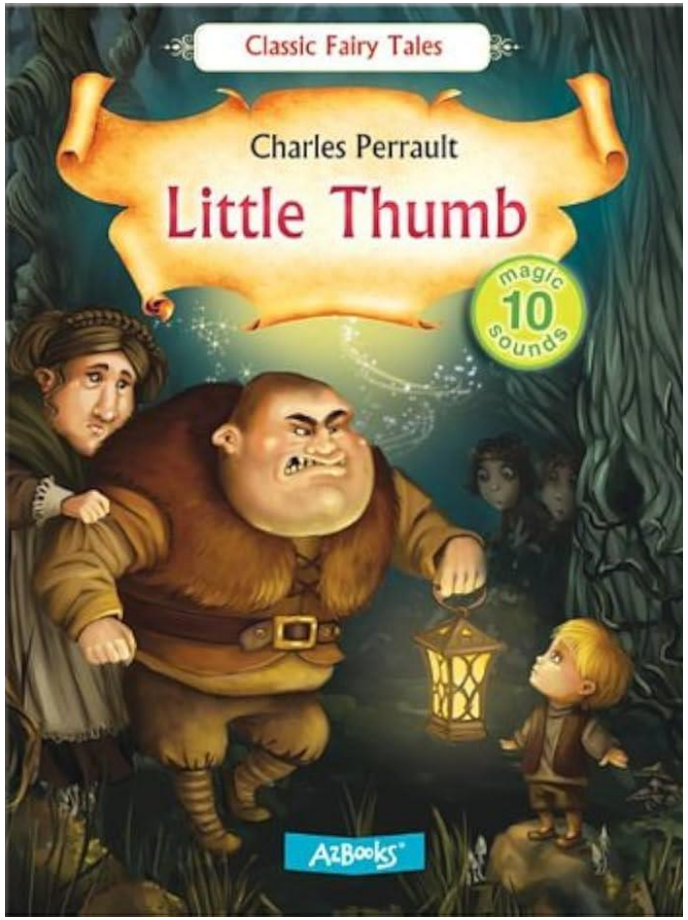
Policy and Value Function

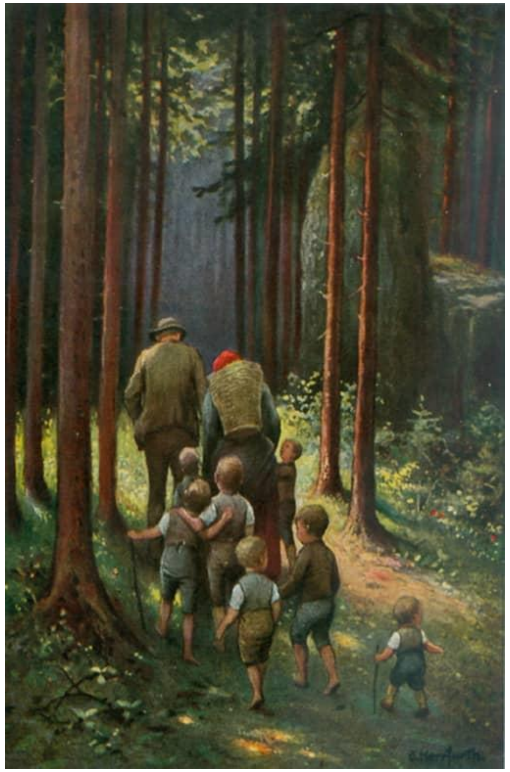


- **Policy** – Is a map

- **Value Function** – Is a compass that tells us the value of each action in a state

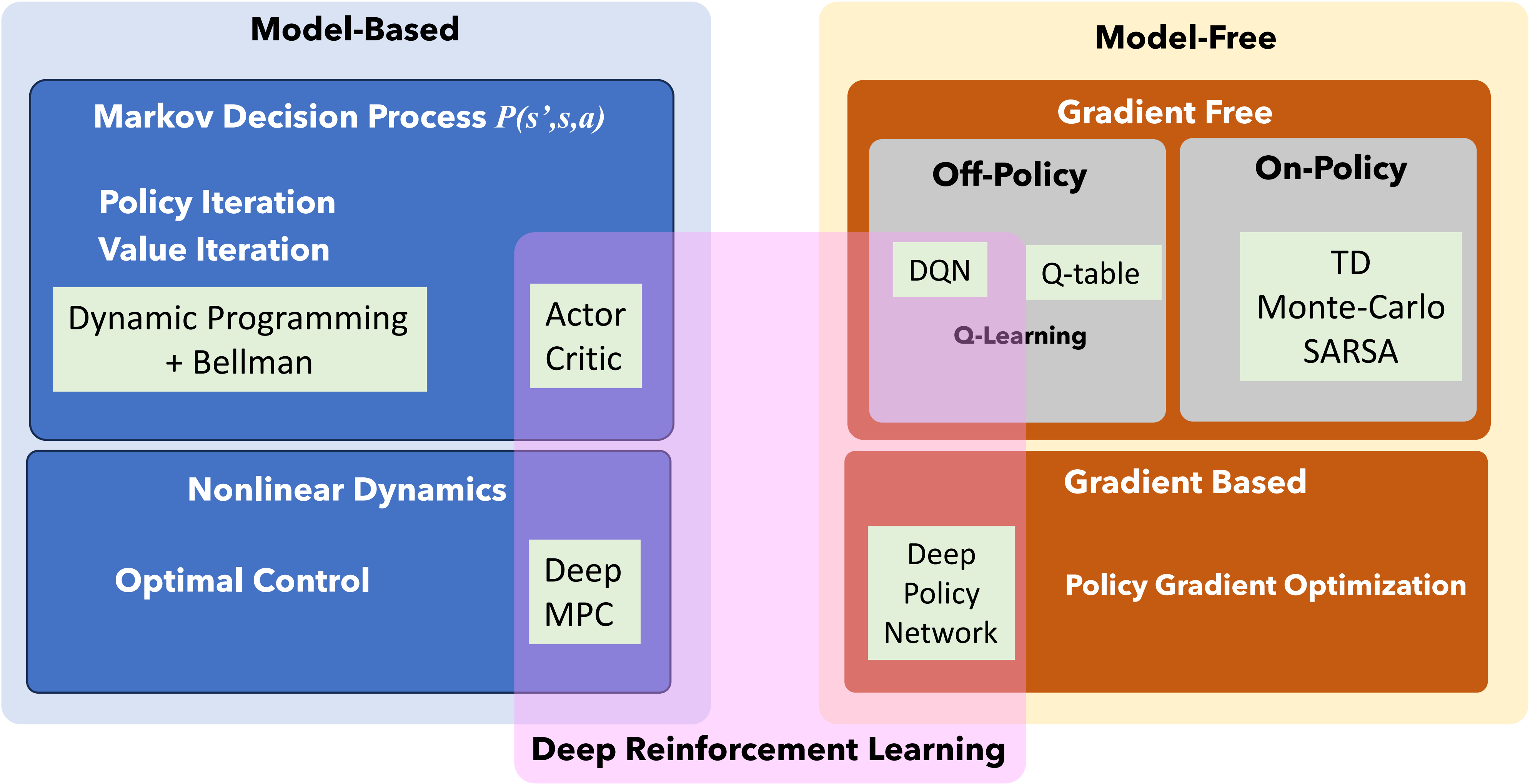
Policy and Value Function – Little thumb





Taxonomy review

Classical Silver classification



What is Model-based and Model-Free?

Model-based

- The agent tries to create a Map of the environment
- They use a policy function to guide the choices
- The policy helps to take the decision for the Best Next Action

- Learn Faster
- When the problem has clear outcomes and results is better
- Less risky in critical systems

Model-Free

- The agent learns by trial and error
- There is no policy function
- No knowledge of the environment

- Better when there are lots of data
- Computing intensive
- Better with uncertain situations

Markov Decision Process

What are Markov properties?

- An environment is Markovian if:

Markov Property

An environment is Markovian if and only if for each state S_t

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, \cdots, S_{t-1}, S_t)$$

- Another kind of definitions:
  - Future is independent of the past given the present
  - Knowing the state we are in, the past history is not relevant
  - The state is a sufficient information statistic of the future

Markovian decision process

- Future is independent of the past given the present
- Knowing the state we are in, the history is not relevant
- The state is a sufficient information statistic of the future

Careful, not all problems are Markovian

- A trader only sees the actual price of a stock

1 2 Are the points similar?

A definition

- A Reinforcement Learning can be formulated as a Markov Decision Process (MDP)
- A MDP is a tuple <S, A, P, R>
  - S: Finite set of states
  - A: Finite set of Actions
  - P: Transition Probabilities (follows Markov properties):

$$P_{ss'}^{a} = \Pr\{S_{t+1} = s' \mid s_t = s,\, a_t = a\} \forall s,\, s' \in S,\, a \in A(s).\}$$

  - R: Reward Probabilities :

$$\mathrm{R}_{ss'}^{\mathrm{a}} = \mathrm{E}\{\, \mathrm{r}_{\mathrm{t+1}} = s' \mid s_t = s,\, a_t = a\} \forall s,\, s' \in S,\, a \in A(s).\}$$

- We can tweak the model to represent continuous or infinite sets of actions and states (see later)

Recap with example

Two groups of RL Algorithms

Policy Learning

Find $\boldsymbol{\pi}(\boldsymbol{s})$

Sample $\boldsymbol{a} \sim \boldsymbol{\pi}(\boldsymbol{s})$

Value Learning

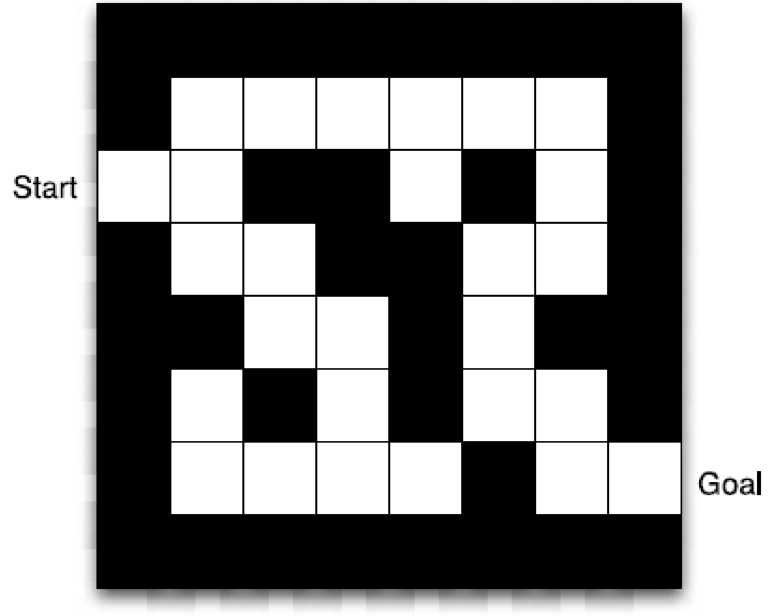
Find $\boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a})$

$\boldsymbol{a} = \underset{\boldsymbol{a}}{\boldsymbol{argmax}}\, \boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a})$

Policy Learning tries to optimize the policy function to maximize rewards

Value Learning obtains the value function for all states and applies it to navigate the universe

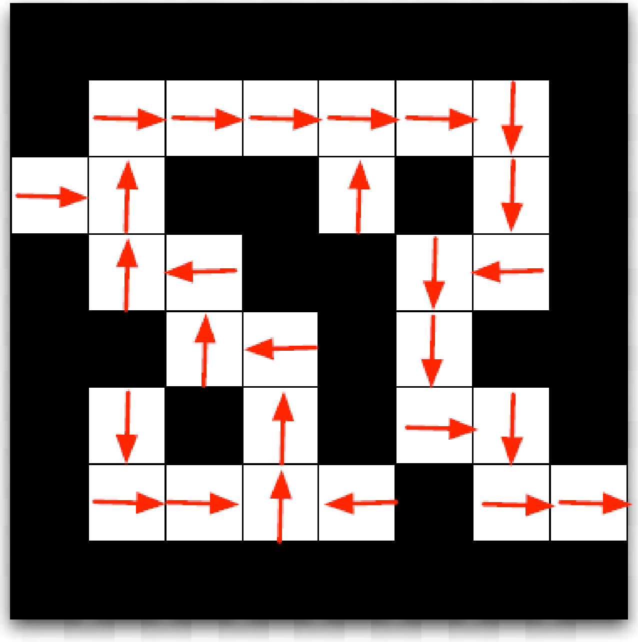
Recap with Example
Environment: MAZE



- Actions: N, S, W, E
- States: Each cell in the Maze
- Rewards: -1 per time-step

Recap with Example
Policy: MAZE

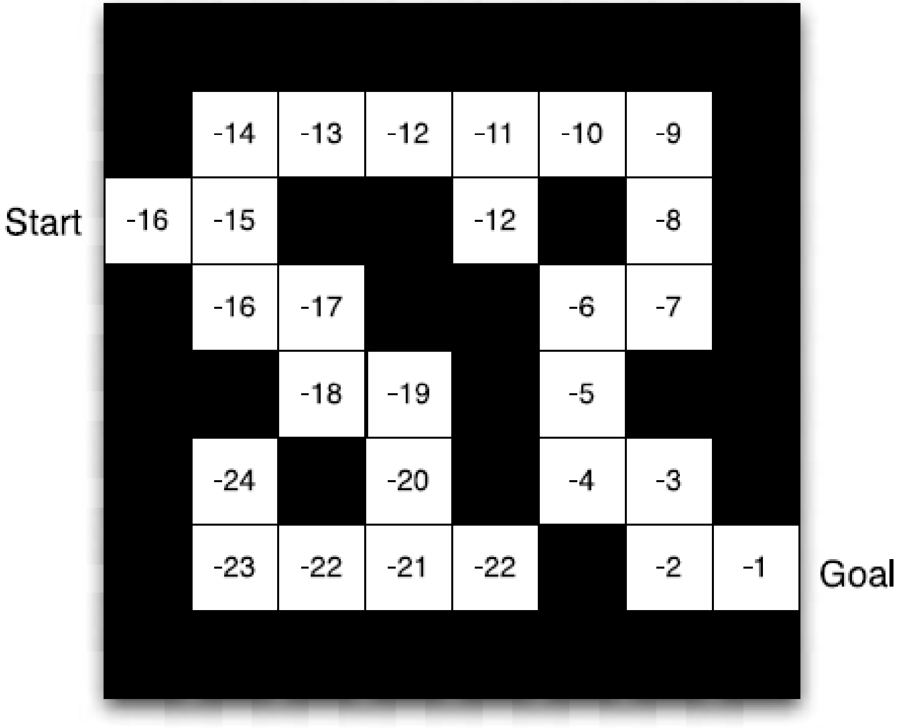
Start



Goal

- The arrow represents the policy $\boldsymbol{\pi}(\boldsymbol{s})$ for each state s

Value Function: MAZE



- The numbers represent $V^{\pi}(s)$ for each state s, for $\gamma = 1$
- How much is $Q^{\pi}(<2,1>,\downarrow)$?

Value Function: MAZE

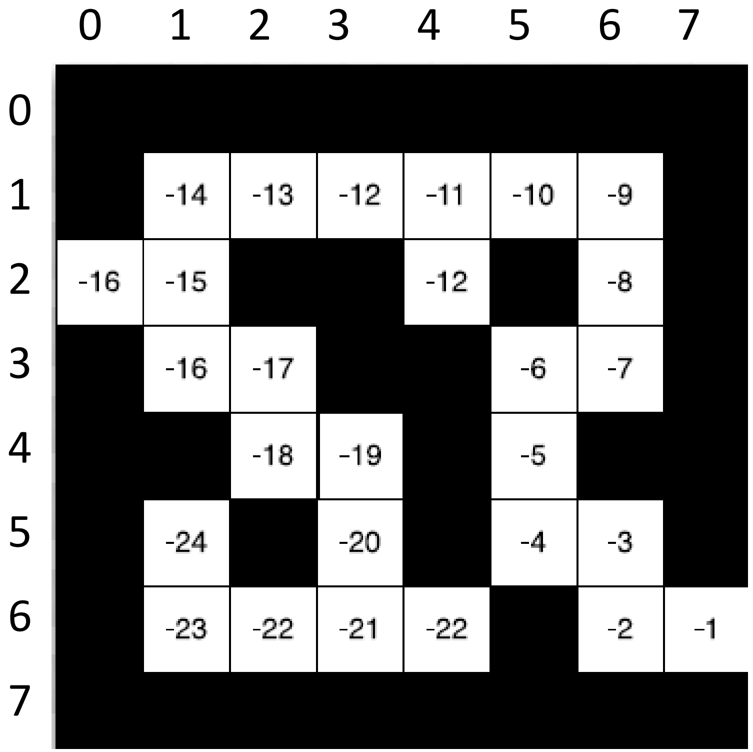
	0	1	2	3	4	5	6	7
0								
1		-14	-13	-12	-11	-10	-9	
2	-16	-15			-12		-8	
3		-16	-17			-6	-7	
4			-18	-19		-5		
5		-24		-20		-4	-3	
6		-23	-22	-21	-22		-2	-1
7								

- The numbers represent $V^{\pi}(s)$ for each state s, for $\gamma = 1$
- How much is $Q^{\pi}(<2,1>,\downarrow)$?

$$Q^{\pi}(s,a) = R(s,a) + \gamma V^{\pi}(s')$$

When writing code with these environments try to get a very clear view of the coordinates in the world you are using. It is a typical error to have issues with the starting point (0,1) what is a row and what is a column (and in 3D is even worse). Spend time understanding the coordinate system and how to map your agent into those coordinates

Value Function: MAZE



- The numbers represent $V^{\pi}(s)$ for each state s, for $\gamma = 1$
- How much is $Q^{\pi}(<2,1>,\ )$?

$\downarrow$

$$Q^{\pi}(s,a) = R(s,a) + \gamma\, V^{\pi}(s')$$

$$Q^{\pi}(s,a) = -1 + 1*(-16) = -17$$

Dynamic Programming

Definition

- Dynamic Programming is a mathematical optimization method developed by Richard Bellman in the 50s
- It is based on the use of value functions and policies, and finding the recursive relationship in the value function, in the whatsocalled Bellman Equation

$$V(s) = \max_{\pi} E\left(r_0 + \gamma V(s')\right)$$

Bellman's Equation

Value Function to Bellman's equation

For the optimum policy pi, being at state s this is the discounted value function it discounts future rewards

$$V_\pi(s) = E\left(\sum_k \gamma^k \, r_k \,\middle|\, s_0 = s\right)$$

$$V(s) = \max_\pi E\left(\sum_k^\infty \gamma^k \, r_k \,\middle|\, s_0 = s\right)$$

You don't know the best policy, but it exists

$$V(s) = \max_\pi E\left(r_0 + \sum_{k=1}^\infty \gamma^k \, r_k \,\middle|\, s_1 = s'\right)$$

s' is the next step

$$V(s) = \max_\pi E\left(r_0 + \gamma V(s')\right)$$

Bellman's Equation

Making it recursive

$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s)V(s')}_{\text{Discounted sum of future rewards}}$$

Policy vs Value learning

Policy Learning

- The policy is modeled and updated directly without consulting a value function.
- Policy gradient methods are commonly used to optimize the policy by estimating which direction improves returns

Value Learning

- In value-based RL, the focus is on learning the optimal value function, denoted Q* or V*. The value function estimates expected cumulative future rewards for being in a given state and following the current policy thereafter.
- Key notes on value-based methods:
- Finding the optimal value function allows deriving the optimal policy.
- Temporal-difference learning is commonly used to update value estimates.
- Algorithms Monte-Carlo, Q-Learning, SARSA, Actor-critic

Value Iteration and Policy Iteration

$$V(s) = \max_{a} \sum_{s'} P(s' \mid s, a)\big(R(s', s, a) + \gamma V(s')\big)$$ Value Iteration

$$\pi(s, a) = \text{argmax}_a P(s' \mid s, a)\big(R(s', s, a) + \gamma V(s')\big)$$ Policy Iteration

Policy Evaluation

How to obtain the Policy evaluation of a state

- Given $\boldsymbol{\pi}$, the policy evaluation methods obtain $\mathbf{V}^{\boldsymbol{\pi}}$
- We can have a Reward Function R(s,a)
- Remember we have a discount factor γ
- Calculation using iterative value policy evaluation
- The iteration finishes when it converges, it can be proven that using a Bellman operator the error reduces over time (see page 92 Sutton)
- The algorithm stops when the Δ Delta is below a small threshold θ

Pseudocode Algorithm Policy Evaluation Iterative method

Iterative Policy Evaluation Algorithm

Given:

- A Markov Decision Process (MDP) with states S, actions A, state transition probabilities $P(s'|s,a)$, and reward function $R(s,a)$.
- A policy $\pi(a|s)$ that specifies the probability of taking action a in state s.

Initialize the value function $V(s) = 0 \forall s \in S$.
repeat
$\quad \Delta \leftarrow 0$
$\quad$ **for each** $s \in S$
$\quad\quad v \leftarrow V(s)$
$\quad\quad V(s) \leftarrow \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P(s'|s,a) \left[R(s,a) + \gamma V(s')\right]$
$\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ {where θ is a small positive threshold}

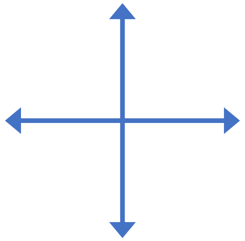
Output: The value function $V(s)$ for the policy π.

Policy Evaluation
What does this mean?

$$V(s) \leftarrow \overbrace{\sum_{a \in A} \pi(a|s) \underbrace{\sum_{s' \in S} P(s'|s,a)\left[R(s,a) + \gamma V(s')\right]}_{\text{Bellman Equation}}}^{\text{For all actions/states}}$$

Policy Evaluation
An example (I)

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	



R = -1 in all transitions

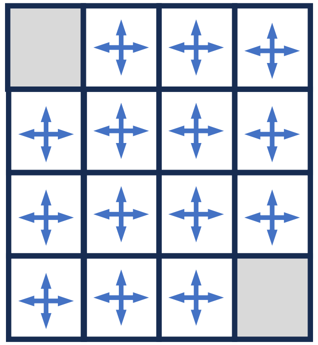
Probability to go to each position is equal (0.25)

Vk for the random policy

K=0

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Policy



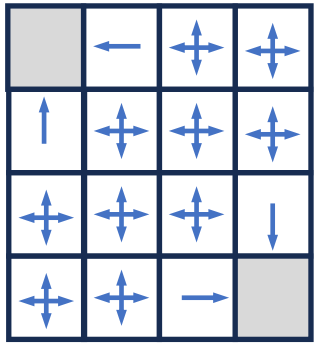
Policy Evaluation
An example (II)

V_k for the random policy

Policy

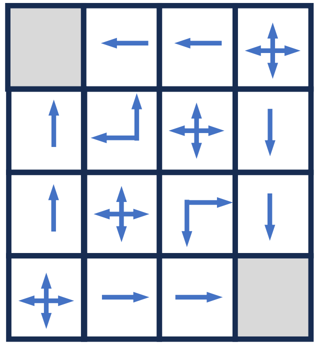
K=1

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



K=2

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



https://medium.com/mitb-for-all/sutton-and-barto-rl-textbook-key-takeaways-2-dp-and-gpi-f6763250698d

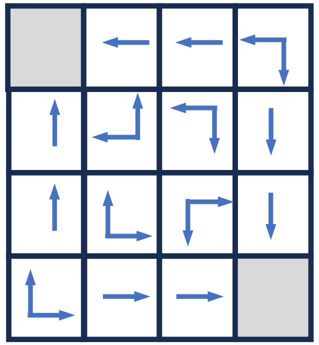
Policy Evaluation
An example (III)

Vk for the random policy

Policy

K=3

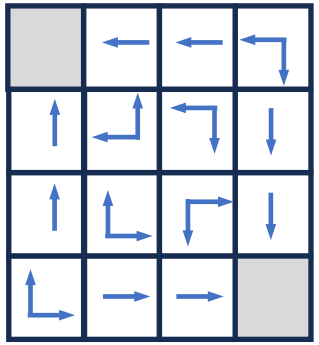
0.0	-2.4	--2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.0	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



Optimal Policy

K=10

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0



Optimal Policy

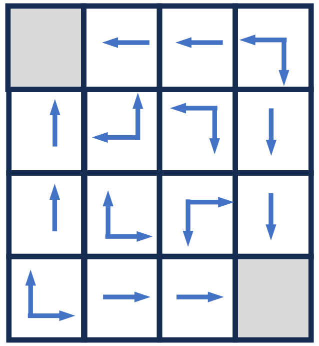
Policy Evaluation
An example (IV)

Vk for the random policy

K=infinite

0.0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0.0

Policy

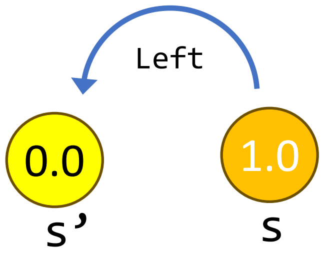


Optimal Policy

Policy Evaluation

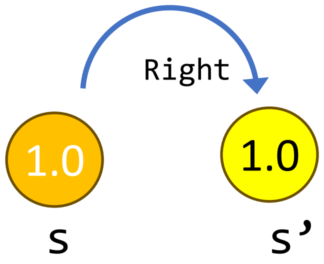
See Cell [0,1] V_k k=2

P(s|s’,a) = 0.25 (Equal probability to each action)
R is always -1



0.25*[-1+(0)] = -0.25

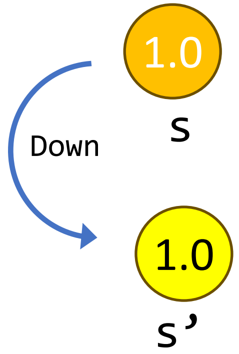
P(s,s’) = 1



0.25*[-1+(-1)] = -0.5

Out of bounds
Stays on place

0.25*[-1+(-1)] = -0.5



0.25*[-1+(-1)] = -0.5

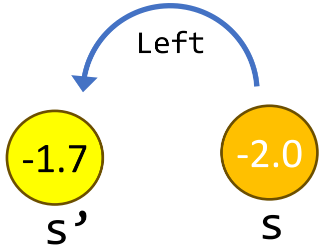
$$P(s' \mid s, a)[R(s', s, a) + \gamma V(s')] = 1.75$$

Policy Evaluation

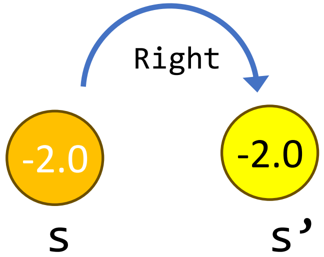
See Cell [1,1] V_k k=3

P(s|s’,a) = 0.25 (Equal probability to each action)

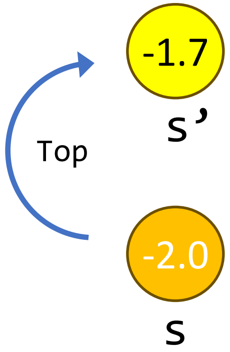
R is always -1



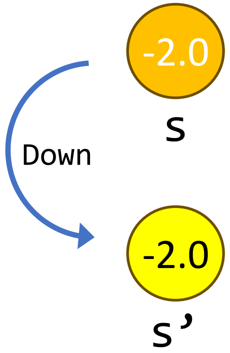
0.25*[-1+(-1.7)] = -0.675



0.25*[-1+(-2)] = -0.75



0.25*[-1+(-1.7)] = -0.675



0.25*[-1+(-2)] = -0. 75

$P(s' \mid s, a)[R(s', s, a) + \gamma V(s')]$ = -2.85

Policy Evaluation

How to calculate it

0	-1.7	-2
-1.7	-2	

	R	V(s')	R+V(s)	P	SUM
Up	-1	-1.7	-2.7	0.25	-0.675
Down	-1	-2	-3	0.25	-0.75
Left	-1	0	-1	0.25	-0.25
Right	-1	-2	-3	0.25	-0.75
			-9.7		-2.425

	R	V(s')	R+V(s)	P	SUM
Up	-1	-1.7	-2.7	0.25	-0.675
Down	-1	-2	-3	0.25	-0.75
Left	-1	0	-1	0.25	-0.25
Right	-1	-2	-3	0.25	-0.75
			-9.7		-2.425

	-1.7	
-1.7	-2	-2
	-2	

	R	V(s')	R+V(s)	P	SUM
Up	-1	-1.7	-2.7	0.25	-0.675
Down	-1	-2	-3	0.25	-0.75
Left	-1	-1.7	-2.7	0.25	-0.675
Right	-1	-2	-3	0.25	-0.75
			-11.4		-2.85

	-2.4	
-2.4	-2.9	-3
	-3	

	R	V(s')	R+V(s)	P	SUM
Up	-1	-2.4	-3.4	0.25	-0.85
Down	-1	-3	-4	0.25	-1
Left	-1	-2.4	-3.4	0.25	-0.85
Right	-1	-3	-4	0.25	-1
			-14.8		-3.7

	R	V(s')	R+V(s)	P	SUM
Up	-1	-1.7	-2.7	0.25	-0.675
Down	-1	-2	-3	0.25	-0.75
Left	-1	-1.7	-2.7	0.25	-0.675
Right	-1	-2	-3	0.25	-0.75
			-11.4		-2.85

Microsoft Excel
97-2003 Worksheet

Policy Improvement

Finding the right Policy

- A policy **π** can be improved if

$$\exists\, s \in S, a \in A \text{ such that } Q^{\pi}(s,a) > Q^{\pi}(s,\pi(s))$$

  - Obvious. In this case, π is not optimal and can be improved setting $\pi(s) = a$

- We can develop an algorithm to iterate to find the optimal policy
  1. Start from a random policy **π**
  2. Compute the $\mathbf{V^{\pi}}$ for this policy **π**
  3. Check for each state if the policy can be improved, if so , improve it
  4. When the policy does not improve any more, stop

The Policy improvement algorithm

Policy Improvement (Sutton)

Input: A policy π and value function V.
$policy\_stable \leftarrow True$
for each $s \in \mathcal{S}$
  $a \leftarrow \pi(s)$
  $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma V(s')]$
  if $a \neq \pi(s)$ **then**
    $policy\_stable \leftarrow False$
Output: Improved policy π

Iteration : Evaluation + Improvement

Complete Policy Iteration (Sutton)

1. Initialization

   $V(s) \in \mathrm{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

   Given:

   - A Markov Decision Process (MDP) with states S, actions A, state transition probabilities $P(s'|s,a)$, and reward function $R(s,a)$.
   - A policy $\pi(a|s)$ that specifies the probability of taking action a in state s.

   Initialize the value function $V(s) = 0$ for all $s \in S$.
   repeat
     for each $s \in S$
       $v \leftarrow V(s)$
       $V(s) \leftarrow \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P(s'|s,a) \left[R(s,a) + \gamma V(s')\right]$
       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ {where θ is a small positive threshold}
   Output: The value function $V(s)$ for the policy π.

3. Policy Improvement

   Input: A policy π and value function V.
   $policy\_stable \leftarrow True$
   for each $s \in \mathcal{S}$
     $a \leftarrow \pi(s)$
     $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma V(s')\right]$
     if $a \neq \pi(s)$ **then**
       $policy\_stable \leftarrow False$
   Output: Improved policy π

Issues

- Problem with Policy iteration: policy evaluation inside the main loop
- Policy evaluation takes a lot of time. Has to be done before improving the policy
- We can stop policy evaluation before complete convergence of policy
- Evaluation In the extreme case, we can even stop policy evaluation after a single sweep (one update of each state).
- This algorithm is called Value Iteration and can be proved to converge to the optimal policy
- It combines in one step improvement of the policy and computation of V

Value Iteration

The V(s)

- $V\pi(s)$ is the "state" value function of an MDP (Markov Decision Process). It's the expected return starting from state s following policy π:
- It is a vector with as many positions as states in the environment
- There are as many Value vectors as policies

The Algorithm

Value Iteration (Sutton)

Initialize the array V arbitrarilty (e.g., $V(s) = 0$ for all $s \in S$).
repeat
  $\Delta \leftarrow V(s)$
  for each $s \in S$
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ {where θ is a small positive threshold}
Output: A deterministic policy, π, such that
$\pi(s) = \arg\max_a \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma V(s')\right]$

Dynamic Programming

Pros-cons

- Number of iteration in policy iteration before convergence is polynomial, and usually needs less iterations to stop than Value iteration
- Value iteration needs a lot of iterations to converge to small errors, however, value iteration converges to optimal policy long before it converges to correct value in this MDP
- Policy iteration requires fewer iterations that value iteration, but each iteration requires solving a linear system instead of just applying Bellman operator
- In practice, policy iteration is often faster, especially if the transition probabilities are structured (e.g., sparse) to make solution of linear system efficient

Dynamic Programming

What is the P - transition probability function

- It tells us the probability and return of each state transitioning in the environment
- In finite state models it is usually represented as a dictionary with the following components
- State – State to Transition,

State | Action | Proability | State to | Reward | Is it a terminal state?

```
{0: {0: [(0.8666666666666667, 0, -0.04, False), (0.06666666666666667, 1, -0.04, False), (0.06666666666666667, 4, -0.04, False)],
     1: [(0.8, 1, -0.04, False),   (0.13333333333333333, 0, -0.04, False),   (0.06666666666666667, 4, -0.04, False)],
     2: [(0.8, 4, -0.04, False),   (0.13333333333333333, 0, -0.04, False),   (0.06666666666666667, 1, -0.04, False)],
     3: [(0.8666666666666667, 0, -0.04, False),   (0.06666666666666667, 1, -0.04, False),   (0.06666666666666667, 4, -0.04, False)]},

 1: {0: [(0.8666666666666667, 1, -0.04, False),   (0.06666666666666667, 2, -0.04, False),   (0.06666666666666667, 0, -0.04, False)],
```

A visual explanation

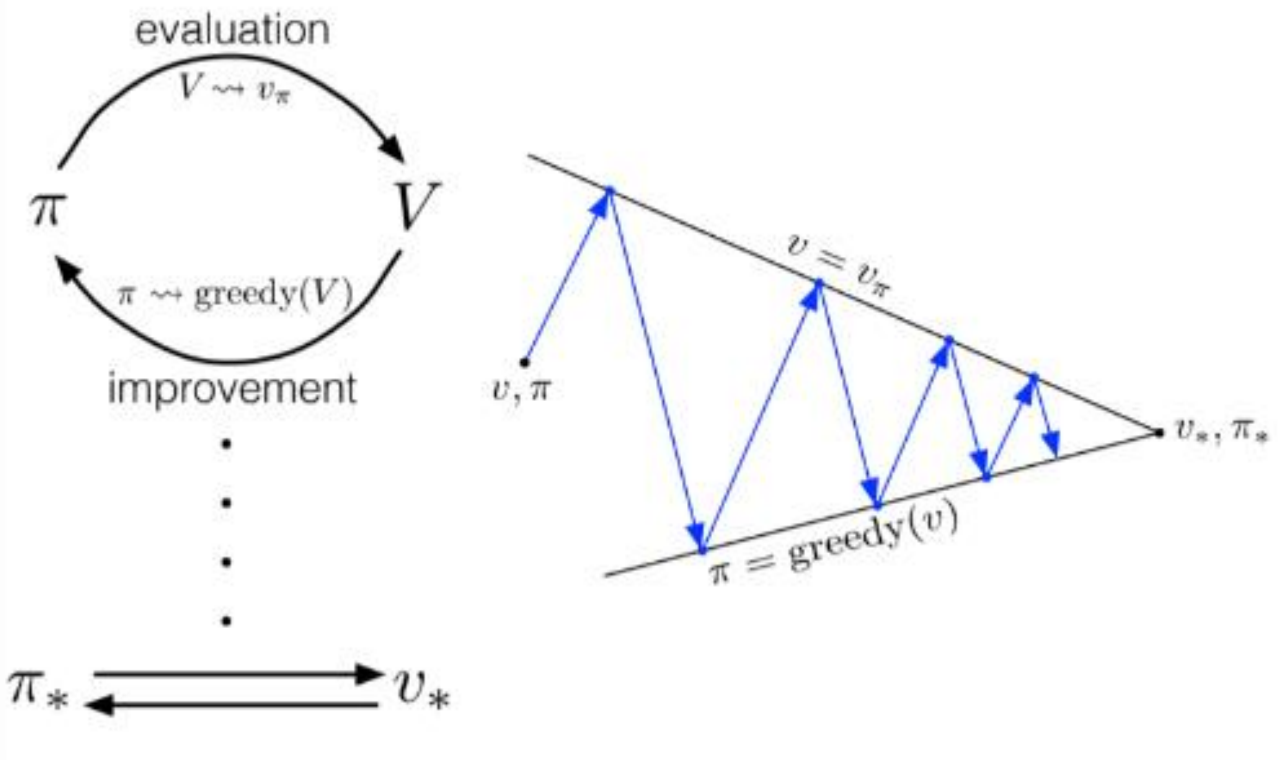


Fig 4.4. *GPI Interaction*

Wrap-up

Lecture 4

- Policy evaluation refers to the (typically) iterative computation of the value function given a policy
- Policy improvement refers to the computation of an improved policy given the value function for current policy
- Putting these two together gives policy iteration and value iteration
- DP methods sweep through the state space, performing an expected update operation for each state
- Expected updates are Bellman equations turned into assignments to update the value of a state based on the values of all possible successor states, weighted by their probability of occurring
- GPI refers to the interleaving of policy evaluation and policy improvement to reach convergence
- Asynchronous DP frees us for the complete state-space sweep
- We update estimates of the values of states based on estimates of the values of successor states. Using other estimates to update the value of one estimate is called bootstrapping

DP requires a complete model of the environment and does bootstrapping (i.e. creating estimates out of other estimates). In the next chapter (Monte Carlo methods) we don't require any model, and we don't bootstrap. In the chapter after (TD-learning) we do not require a model either, but we do bootstrap.

Lecture 4

- **Dynamic Programming** are a set of algorithms to solve the RL problem using Value and Policy iterations
- The **Bellman equation** is the central function to create valid Dynamic Programming methods
- We have two approaches, **policy iteration** or **Value iteration**. Depending on the function we optimize
- Dynamic Programming is useful in simple environments with discrete observation and action spaces

END
Session 4