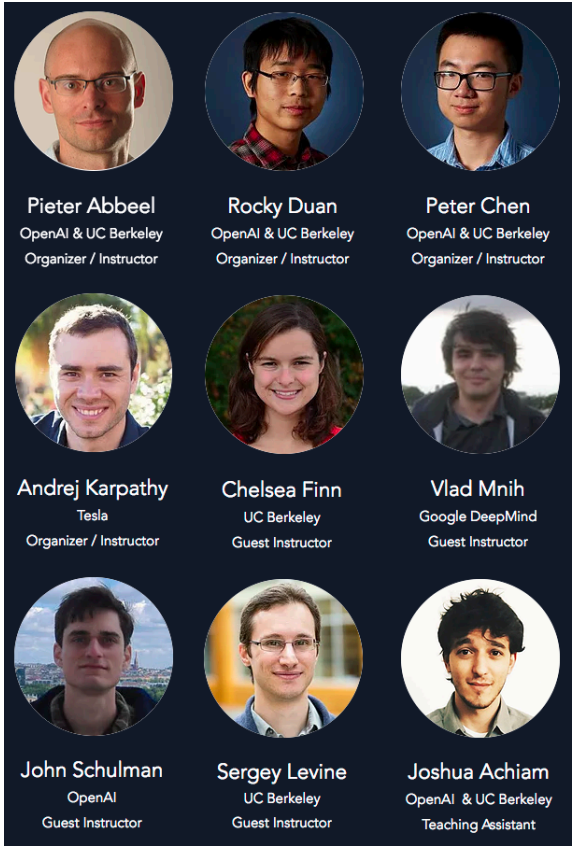
Deep Reinforcement Learning
Lecture 1: Motivation + Overview + MDPs + Exact Solution Methods

Pieter Abbeel

OpenAI / UC Berkeley / Gradescope

Many slides made with John Schulman, Yan (Rocky) Duan and Xi (Peter) Chen

Organizers / Instructors / TAs



Pieter Abbeel
OpenAI & UC Berkeley
Organizer / Instructor

Rocky Duan
OpenAI & UC Berkeley
Organizer / Instructor

Peter Chen
OpenAI & UC Berkeley
Organizer / Instructor

Andrej Karpathy
Tesla
Organizer / Instructor

Chelsea Finn
UC Berkeley
Guest Instructor

Vlad Mnih
Google DeepMind
Guest Instructor

John Schulman
OpenAI
Guest Instructor

Sergey Levine
UC Berkeley
Guest Instructor

Joshua Achiam
OpenAI & UC Berkeley
Teaching Assistant

Marcin Andrychowicz
OpenAI
Teaching Assistant

Richard Chen
OpenAI
Teaching Assistant

Ignasi Clavera
UC Berkeley
Teaching Assistant

Carlos Florensa
UC Berkeley
Teaching Assistant

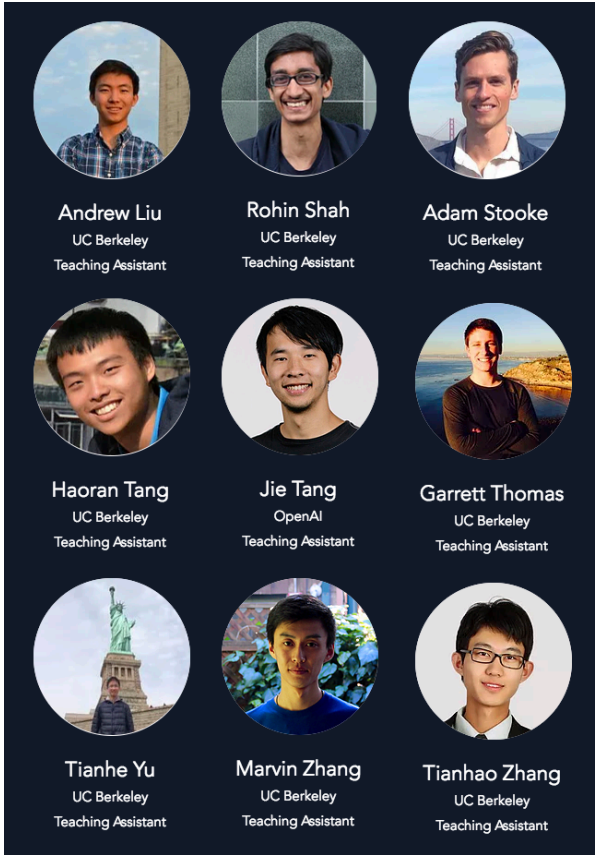
Tuomas Haarnoja
UC Berkeley
Teaching Assistant

Rein Houthooft
OpenAI
Teaching Assistant

Sandy Huang
UC Berkeley
Teaching Assistant

Thanard Kurutach
UC Berkeley
Teaching Assistant

Yang Liu
OpenAI & UIUC
Teaching Assistant



Andrew Liu
UC Berkeley
Teaching Assistant

Rohin Shah
UC Berkeley
Teaching Assistant

Adam Stooke
UC Berkeley
Teaching Assistant

Haoran Tang
UC Berkeley
Teaching Assistant

Jie Tang
OpenAI
Teaching Assistant

Garrett Thomas
UC Berkeley
Teaching Assistant

Tianhe Yu
UC Berkeley
Teaching Assistant

Marvin Zhang
UC Berkeley
Teaching Assistant

Tianhao Zhang
UC Berkeley
Teaching Assistant

Logistics

- Laptop setup for labs
  - Check your email!
- Communication with staff
  - In-person
  - Piazza
    - Help us out! ☺
- Laptops
  - Stay charged
- Tomorrow
  - Make sure to have your badges!

Bootcamp Goals

- Understand mathematical and algorithmic foundations of Deep RL
- Have implemented many of the core algorithms

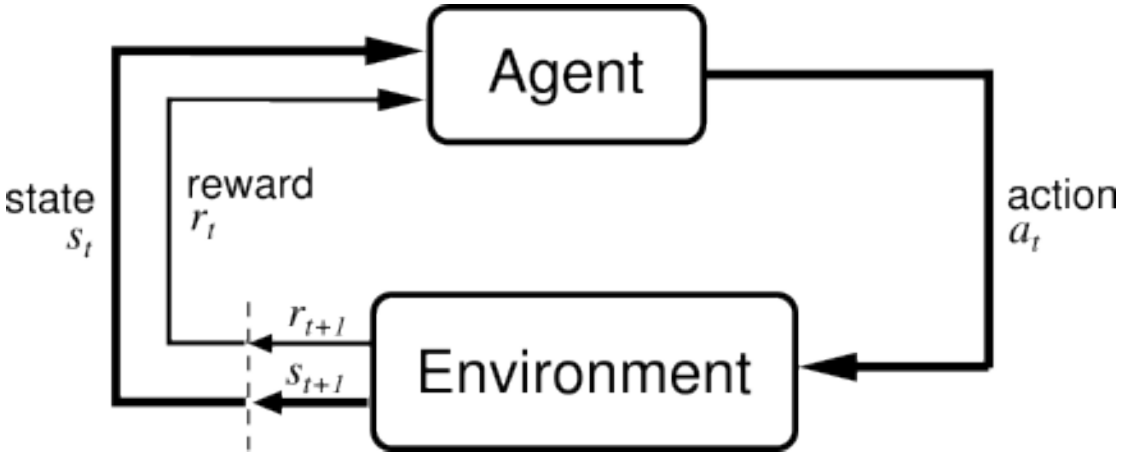
Schedule -- Saturday

8:30: Breakfast / Check-in
9-10: Core Lecture 1 Intro to MDPs and Exact Solution Methods (Pieter Abbeel)
10-10:10 Brief Sponsor Intros
10:10-11:10 Core Lecture 2 Sample-based Approximations and Fitted Learning (Rocky Duan)
11:10-11:30: Coffee Break
11:30-12:30 Core Lecture 3 DQN + variants (Vlad Mnih)
12:30-1:30 Lunch [catered]
1:30-2:15 Core Lecture 4a Policy Gradients and Actor Critic (Pieter Abbeel)
2:15-3:00 Core Lecture 4b Pong from Pixels (Andrej Karpathy)
3:00-3:30 Core Lecture 5 Natural Policy Gradients, TRPO, and PPO (John Schulman)
3:30-3:50 Coffee Break
3:50-4:30 Core Lecture 6 Nuts and Bolts of Deep RL Experimentation (John Schulman)
4:30-7 Labs 1-2-3
7-8 Frontiers Lecture I: Recent Advances, Frontiers and Future of Deep RL (Vlad Mnih)

Schedule -- Sunday

8:30: Breakfast
9-10: Core Lecture 7 SVG, DDPG, and Stochastic Computation Graphs (John Schulman)
10-11: Core Lecture 8 Derivative-free Methods (Peter Chen)
11-11:30: Coffee Break
11:30-12:30 Core Lecture 9 Model-based RL (Chelsea Finn)
12:30-1:30 lunch [catered]
1:30-2:30 Core Lecture 10 Utilities / Inverse RL (Pieter Abbeel / Chelsea Finn)
2:30-3:10 Two-minute Presentations by each TA
3:10-3:30 Coffee Break
3:30-6 Labs 4-5
6-7 Frontiers Lecture II: Recent Advances, Frontiers and Future of Deep RL (Sergey Levine)

Markov Decision Process



Assumption: agent gets to observe the state

[Drawing from Sutton and Barto, Reinforcement Learning: An Introduction, 1998]

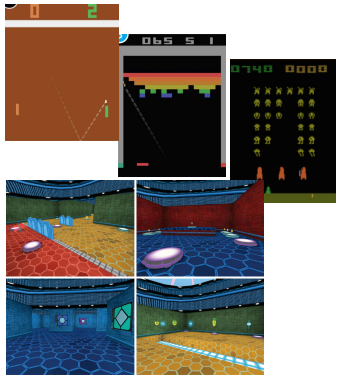
Some Reinforcement Learning Success Stories

Kohl and Stone, 2004

Ng et al, 2004

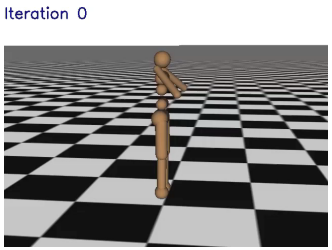
Tedrake et al, 2005

Kober and Peters, 2009

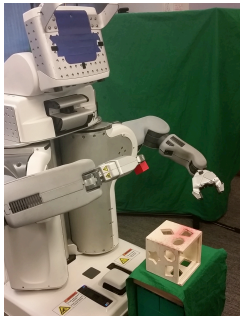


Mnih et al 2013 (DQN)
Mnih et al, 2015 (A3C)

Silver et al, 2014 (DPG)
Lillicrap et al, 2015 (DDPG)



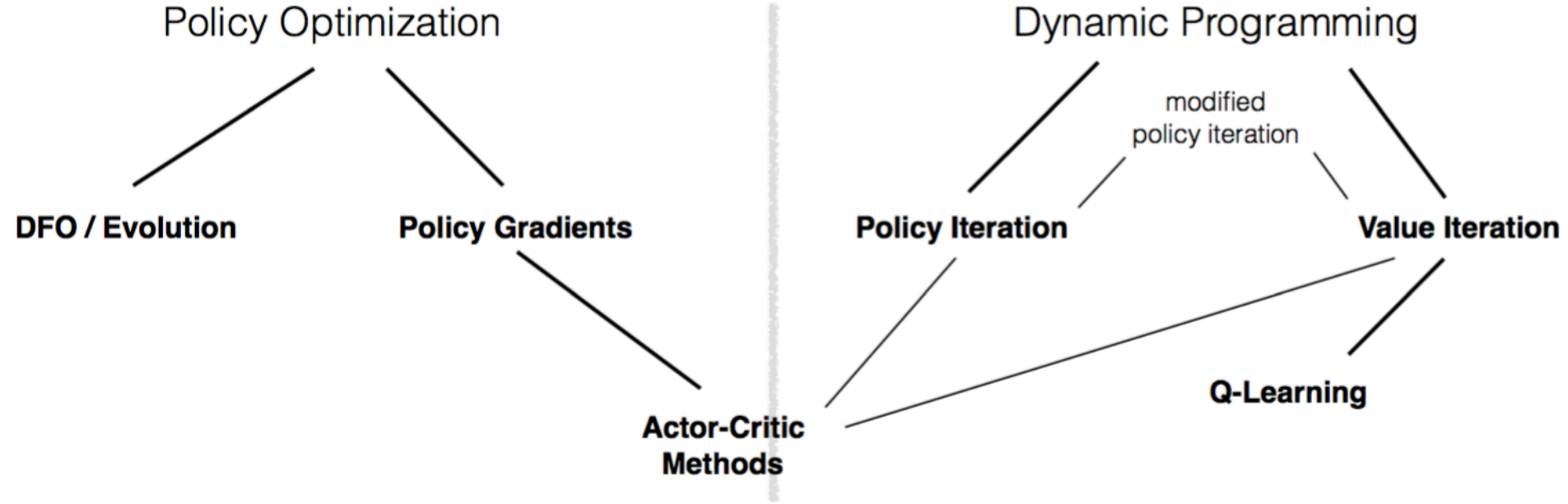
Schulman et al, 2016 (TRPO + GAE)



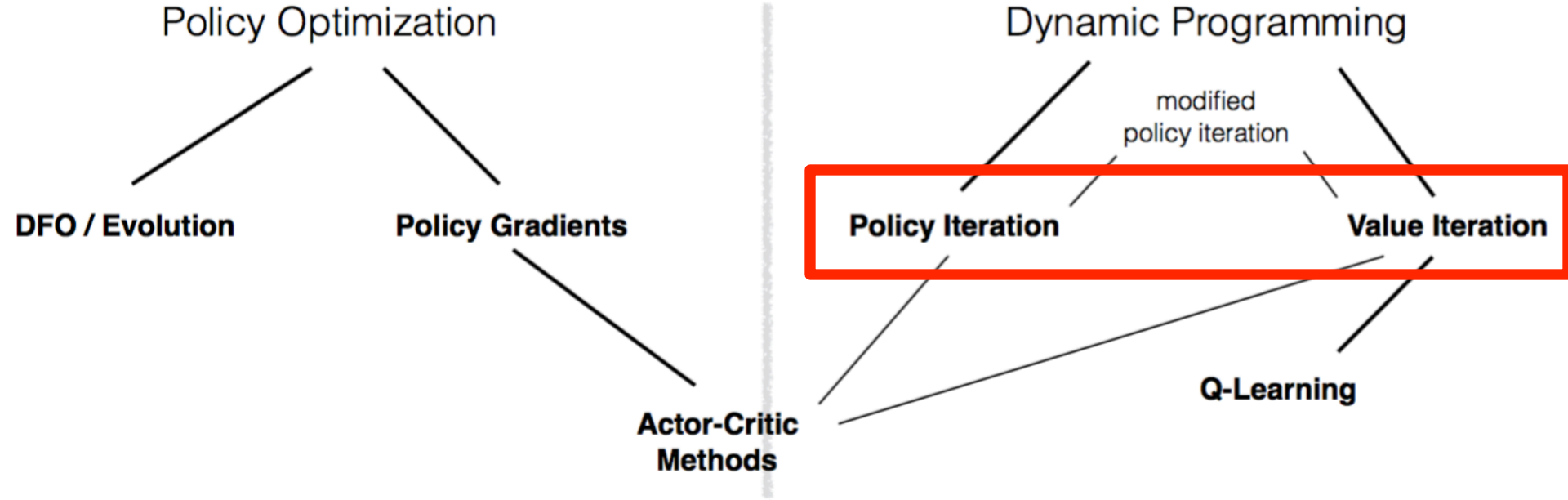
Levine*, Finn*, et al, 2016 (GPS)

Silver*, Huang*, et al, 2016 (AlphaGo)

RL Algorithms Landscape



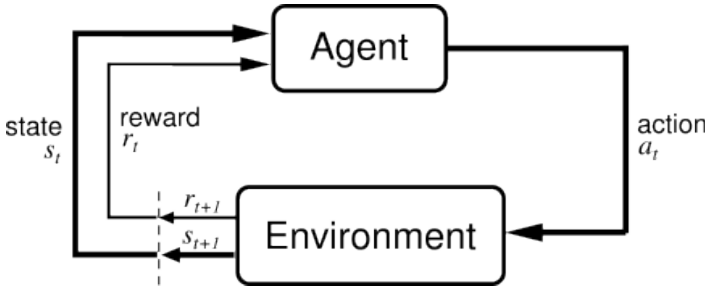
RL Algorithms Landscape



Markov Decision Process (MDP)

An MDP is defined by:

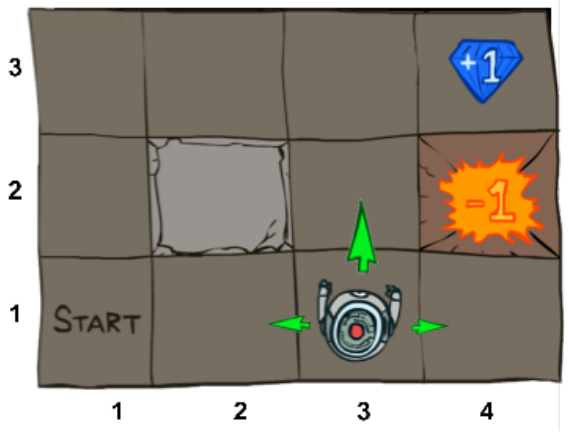
- Set of states S
- Set of actions A
- Transition function $P(s' \mid s, a)$
- Reward function $R(s, a, s')$
- Start state s_0
- Discount factor γ
- Horizon H



Example MDP: Gridworld

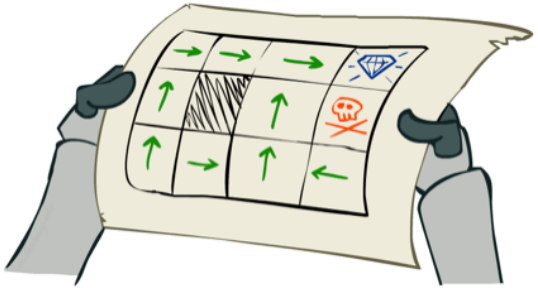
An MDP is defined by:

- Set of states S
- Set of actions A
- Transition function $P(s' \mid s, a)$
- Reward function $R(s, a, s')$
- Start state s_0
- Discount factor γ
- Horizon H



Goal: $max_\pi \mathrm{E}[\sum_{t=0}^{H} \gamma^t R(S_t, A_t, S_{t+1}) | \pi]$

π:



Outline

- Optimal Control

  =

  given an MDP (S, A, P, R, γ, H)

  find the optimal policy π^*

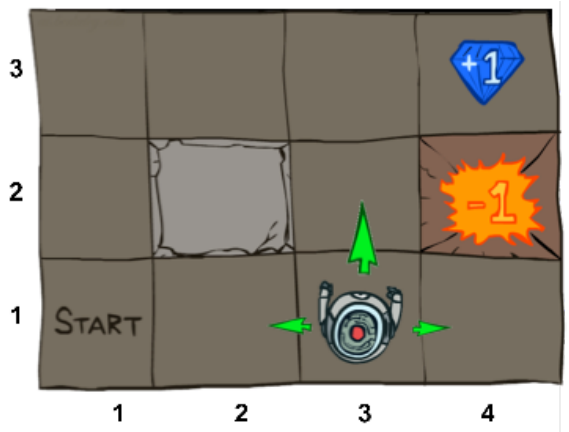
- Exact Methods:
  - ***Value Iteration***
  - Policy Iteration

For now: small discrete state-action spaces as they are simpler to get the main concepts across. We will consider large / continuous spaces later!

Optimal Value Function V*

$$V^*(s) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1}) \mid \pi, s_0 = s\right]$$

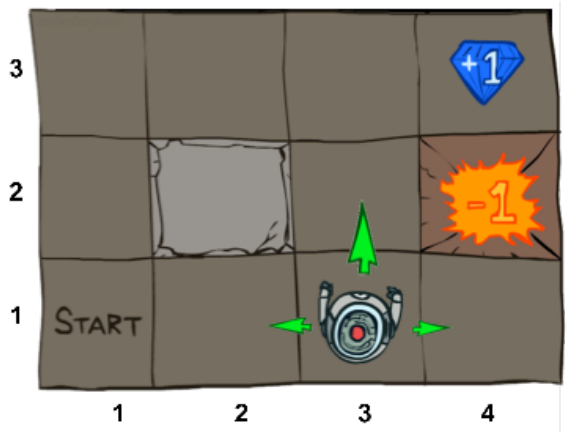
= sum of discounted rewards when starting from state s and acting optimally



Optimal Value Function V*

$$V^*(s) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1}) \mid \pi, s_0 = s\right]$$

= sum of discounted rewards when starting from state s and acting optimally



Let's assume:

actions deterministically successful, gamma = 1, H = 100

V*(4,3) =

V*(3,3) =

V*(2,3) =

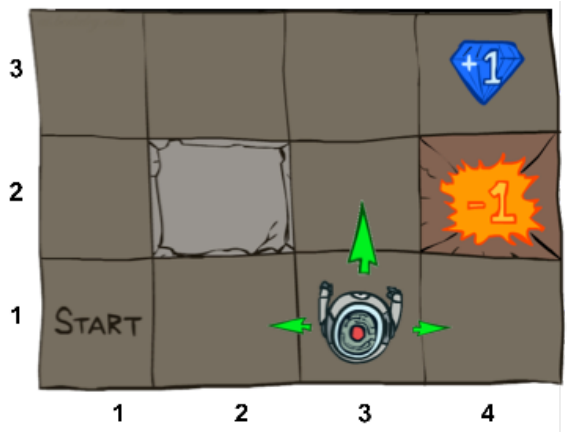
V*(1,1) =

V*(4,2) =

Optimal Value Function V*

$$V^*(s) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1}) \mid \pi, s_0 = s\right]$$

= sum of discounted rewards when starting from state s and acting optimally



Let's assume:

actions deterministically successful, gamma = 0.9, H = 100

V*(4,3) =

V*(3,3) =

V*(2,3) =

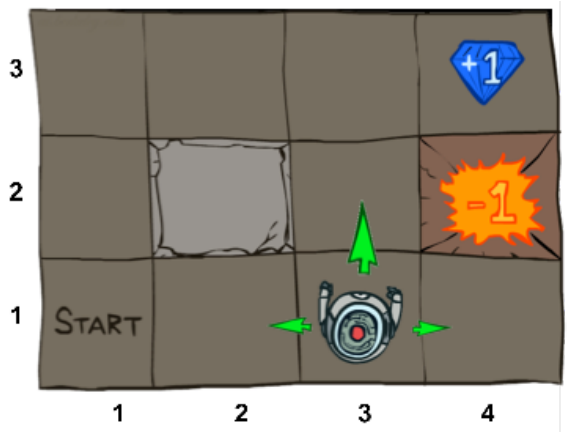
V*(1,1) =

V*(4,2) =

Optimal Value Function V*

$$V^*(s) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1}) \mid \pi, s_0 = s\right]$$

= sum of discounted rewards when starting from state s and acting optimally



Let's assume:

actions successful w/probability 0.8, gamma = 0.9, H = 100

V*(4,3) =

V*(3,3) =

V*(2,3) =

V*(1,1) =

V*(4,2) =

Value Iteration

- $V_0^*(s)$ = optimal value for state s when H=0
  - $V_0^*(s) = 0 \quad \forall s$
- $V_1^*(s)$ = optimal value for state s when H=1
  - $V_1^*(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_0^*(s'))$
- $V_2^*(s)$ = optimal value for state s when H=2
  - $V_2^*(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_1^*(s'))$
- $V_k^*(s)$ = optimal value for state s when H = k
  - $V_k^*(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_{k-1}^*(s'))$

Value Iteration

Algorithm:

Start with $V_0^*(s) = 0$ for all s.

For k = 1, … , H:

For all states s in S:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

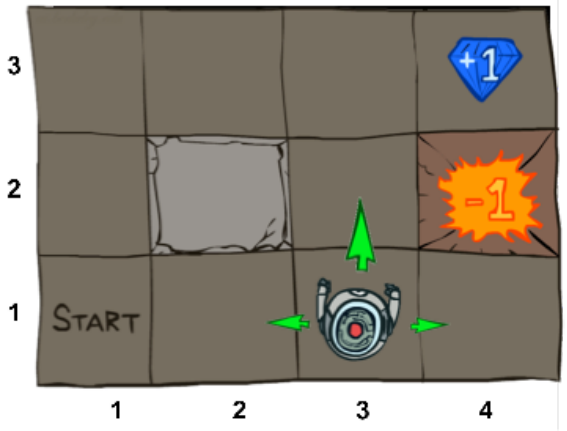
$$\pi_k^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

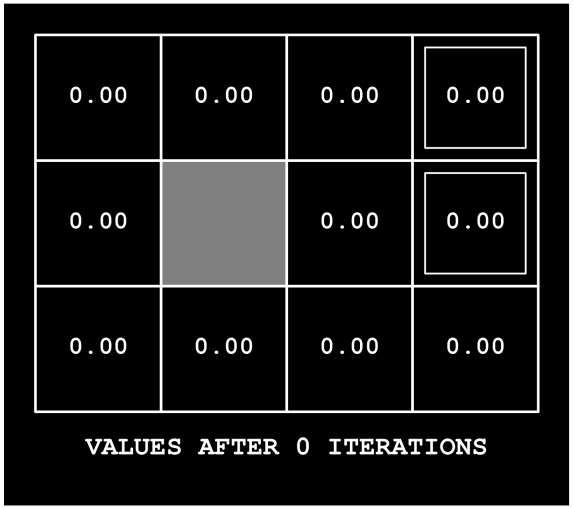
This is called a value update or Bellman update/back-up

Value Iteration

$$V_0(s) \leftarrow 0$$

k = 0

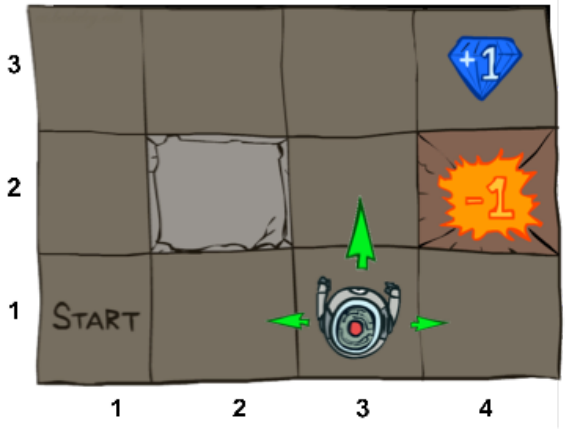




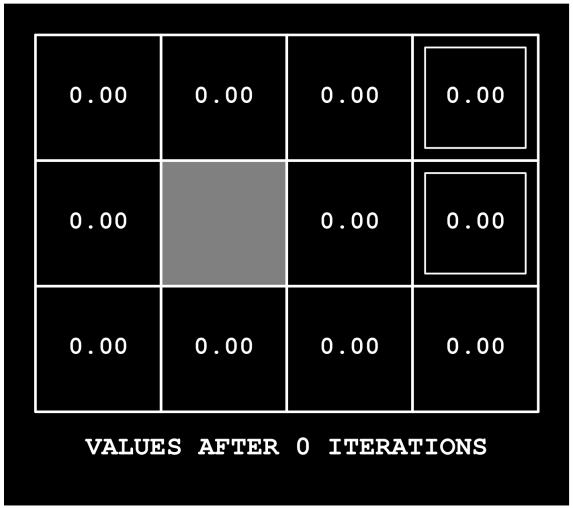
Noise = 0.2
Discount = 0.9

Value Iteration

$$V_1(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_0(s'))$$



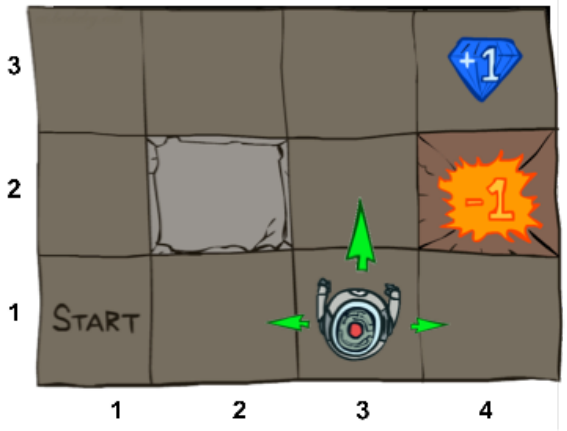
k = 0



Noise = 0.2
Discount = 0.9

Value Iteration

$$V_2(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_1(s'))$$



k = 1

Noise = 0.2
Discount = 0.9

Value Iteration

$$V_2(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_1(s'))$$

k = 2

Noise = 0.2
Discount = 0.9

Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 3

Noise = 0.2
Discount = 0.9

Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 4

Noise = 0.2
Discount = 0.9

Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 5

Noise = 0.2
Discount = 0.9

Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 6

Noise = 0.2
Discount = 0.9

Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 7

Noise = 0.2
Discount = 0.9

Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 8

Noise = 0.2
Discount = 0.9

Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 9

Noise = 0.2
Discount = 0.9

Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 10

Noise = 0.2
Discount = 0.9

Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 11

Noise = 0.2
Discount = 0.9

Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 12

Noise = 0.2
Discount = 0.9

Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 100

Noise = 0.2
Discount = 0.9

Value Iteration Convergence

Theorem. *Value iteration converges. At convergence, we have found the optimal value function V* for the discounted infinite horizon problem, which satisfies the Bellman equations*

$$\forall S \in S: \quad V^*(s) = \max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

- Now we know how to act for infinite horizon with discounted rewards!
  - Run value iteration till convergence.
  - This produces V*, which in turn tells us how to act, namely following:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

- Note: the infinite horizon optimal policy is stationary, i.e., the optimal action at a state s is the same action at all times. (Efficient to store!)

Convergence: Intuition

- $V^*(s)$ = expected sum of rewards accumulated starting from state s, acting optimally for ∞ steps
- $V_H^*(s)$ = expected sum of rewards accumulated starting from state s, acting optimally for H steps

- Additional reward collected over time steps H+1, H+2, …

$$\gamma^{H+1}R(s_{H+1}) + \gamma^{H+2}R(s_{H+2}) + \ldots \leq \gamma^{H+1}R_{max} + \gamma^{H+2}R_{max} + \ldots = \frac{\gamma^{H+1}}{1-\gamma}R_{max}$$

goes to zero as H goes to infinity

Hence $V_H^* \xrightarrow{H\to\infty} V*$

For simplicity of notation in the above it was assumed that rewards are always greater than or equal to zero. If rewards can be negative, a similar argument holds, using max |R| and bounding from both sides.

Convergence and Contractions

- Define the max-norm: $||U|| = \max_s |U(s)|$

- Theorem: For any two approximations U and V

$$||U_{i+1} - V_{i+1}|| \leq \gamma \, ||U_i - V_i||$$

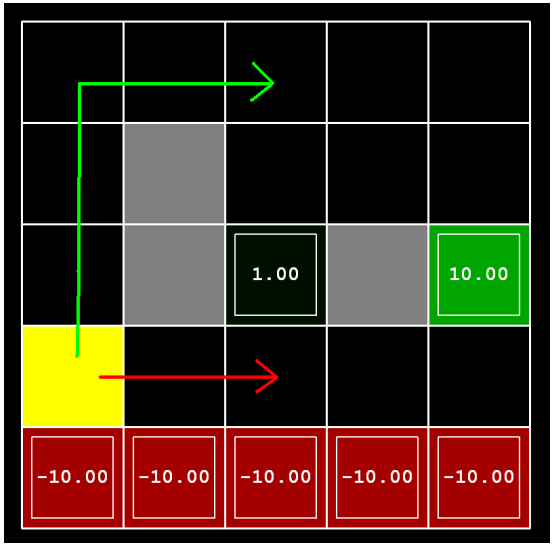
  - I.e., any distinct approximations must get closer to each other, so, in particular, any approximation must get closer to the true U and value iteration converges to a unique, stable, optimal solution

- Theorem:

$$||V_{i+1} - V_i|| < \epsilon, \Rightarrow ||V_{i+1} - V^*|| < 2\epsilon\gamma/(1-\gamma)$$

  - I.e. once the change in our approximation is small, it must also be close to correct

Exercise 1: Effect of Discount and Noise



(a) Prefer the close exit (+1), risking the cliff (-10)

(b) Prefer the close exit (+1), but avoiding the cliff (-10)

(c) Prefer the distant exit (+10), risking the cliff (-10)

(d) Prefer the distant exit (+10), avoiding the cliff (-10)

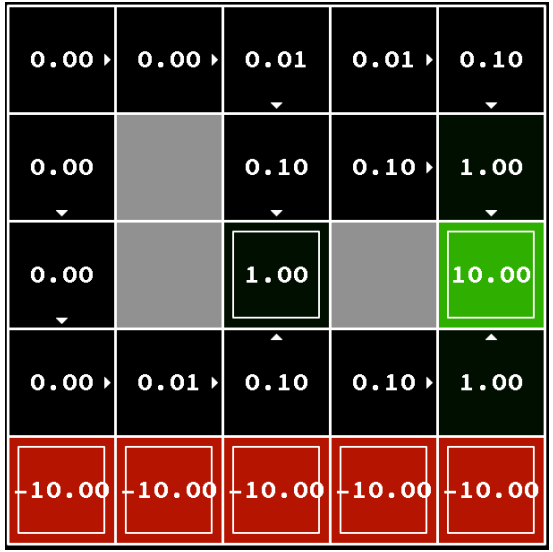
(1) $\gamma = 0.1$, noise = 0.5

(2) $\gamma = 0.99$, noise = 0

(3) $\gamma = 0.99$, noise = 0.5

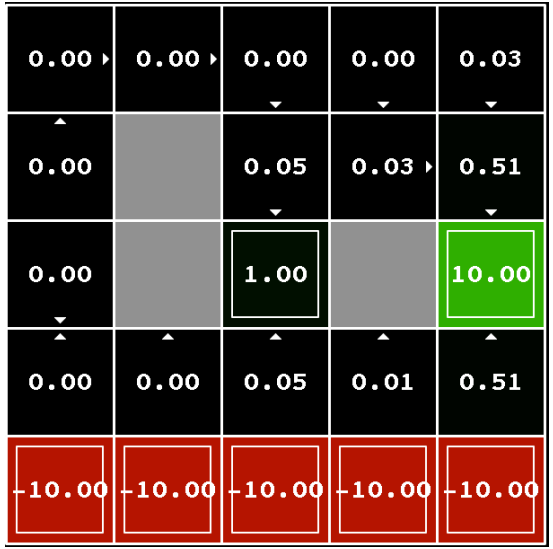
(4) $\gamma = 0.1$, noise = 0

Exercise 1 Solution



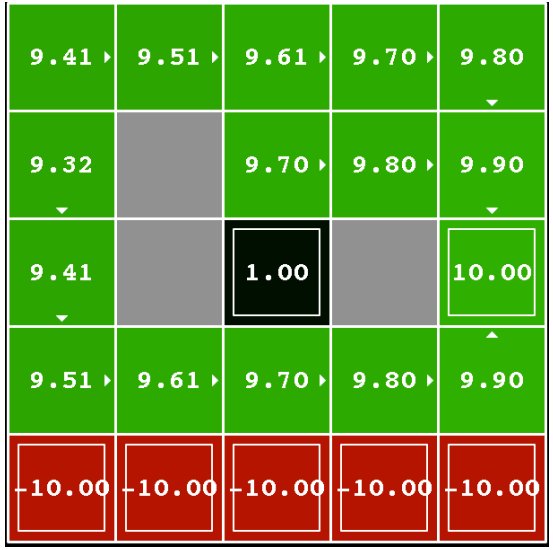
(a) Prefer close exit (+1), risking the cliff (-10) --- (4) γ = 0.1, noise = 0

Exercise 1 Solution



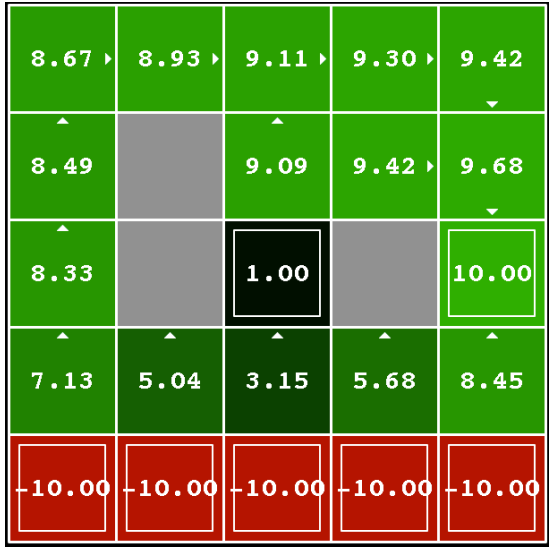
(b) Prefer close exit (+1), avoiding the cliff (-10) --- (1) $\gamma = 0.1$, noise = 0.5

Exercise 1 Solution



(c) Prefer distant exit (+1), risking the cliff (-10) --- (2) γ = 0.99, noise = 0

Exercise 1 Solution



(d) Prefer distant exit (+1), avoid the cliff (-10) --- (3) γ = 0.99, noise = 0.5

Q-Values

$Q^*(s, a)$ = expected utility starting in s, taking action a, and (thereafter) acting optimally

Bellman Equation:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

Q-Value Iteration:

$$Q^*_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*_k(s',a'))$$

Q-Value Iteration

$$Q^*_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*_k(s',a'))$$

k = 100

Noise = 0.2

Discount = 0.9

Outline

- Optimal Control

  =

  given an MDP (S, A, P, R, γ, H)

  find the optimal policy π^*

- Exact Methods:
  - ***Value Iteration***
  - Policy Iteration

For now: small discrete state-action spaces as they are simpler to get the main concepts across. We will consider large / continuous spaces later!

Policy Evaluation

- Recall value iteration:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

- Policy evaluation for a given $\pi(s)$:

$$V_k^\pi(s) \leftarrow \sum_{s'} P(s'|s,\pi(s))(R(s,\pi(s),s') + \gamma V_{k-1}^\pi(s))$$

At convergence:

$$\forall s \quad V^\pi(s) \leftarrow \sum_{s'} P(s'|s,\pi(s))(R(s,\pi(s),s') + \gamma V^\pi(s))$$

Exercise 2

Consider a *stochastic* policy $\pi(a|s)$, where $\pi(a|s)$ is the probability of taking action a when in state s. Which of the following is the correct update to perform policy evaluation for this stochastic policy?

1. $V^{\pi}_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V^{\pi}_{k}(s')\right)$
2. $V^{\pi}_{k+1}(s) \leftarrow \sum_{s'} \sum_{a} \pi(a|s) P(s'|s,a)\left(R(s,a,s') + \gamma V^{\pi}_{k}(s')\right)$
3. $V^{\pi}_{k+1}(s) \leftarrow \sum_{a} \pi(a|s) \max_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V^{\pi}_{k}(s')\right)$

Policy Iteration

One iteration of policy iteration:

- Policy evaluation for current policy π_k :
  - Iterate until convergence

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} P(s'|s, \pi_k(s)) \left[R(s, \pi(s), s') + \gamma V_i^{\pi_k}(s')\right]$$

- Policy improvement: find the best action according to one-step look-ahead

$$\pi_{k+1}(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s, a) \left[R(s, a, s') + \gamma V^{\pi_k}(s')\right]$$

- Repeat until policy converges
- At convergence: optimal policy; and converges faster than value iteration under some conditions

Policy Evaluation Revisited

- *Idea 1:* modify Bellman updates

$$V_0^\pi(s) = 0$$

$$V_{i+1}^\pi(s) \leftarrow \sum_{s'} P(s'|s,\pi(s)) \left[R(s,\pi(s),s') + \gamma V_i^\pi(s')\right]$$

- Idea 2: it is just a linear system, solve with Numpy (or whatever)

  variables: $V^\pi(s)$

  constants: P, R

$$\forall s \quad V^\pi(s) = \sum_{s'} P(s'|s,\pi(s)) \left[R(s,\pi(s),s') + \gamma V^\pi(s')\right]$$

Policy Iteration Guarantees

Policy Iteration iterates over:

- Policy evaluation for current policy π_k :
  - Iterate until convergence

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} P(s'|s, \pi_k(s)) \left[R(s, \pi(s), s') + \gamma V_i^{\pi_k}(s')\right]$$

- Policy improvement: find the best action according to one-step look-ahead

$$\pi_{k+1}(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s, a) \left[R(s, a, s') + \gamma V^{\pi_k}(s')\right]$$

Theorem. Policy iteration is guaranteed to converge and at convergence, the current policy and its value function are the optimal policy and the optimal value function!

Proof sketch:

(1) Guarantee to converge: In every step the policy improves. This means that a given policy can be encountered at most once. This means that after we have iterated as many times as there are different policies, i.e., (number actions)$^{(\text{number states})}$, we must be done and hence have converged.

(2) Optimal at convergence: by definition of convergence, at convergence $\pi_{k+1}(s) = \pi_k(s)$ for all states s. This means $\forall s \;\; V^{\pi_k}(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i^{\pi_k}(s')\right]$

Hence V^{π_k} satisfies the Bellman equation, which means V^{π_k} is equal to the optimal value function V*.

Outline

- Optimal Control

  =

  given an MDP (S, A, P, R, γ, H)

  find the optimal policy π^*

- Exact Methods:
  - ✓ ***Value Iteration***
  - ✓ ***Policy Iteration***

Limitations:

- Iteration over / storage for all states and actions: requires small, discrete state-action space
- Update equations require access to dynamics model