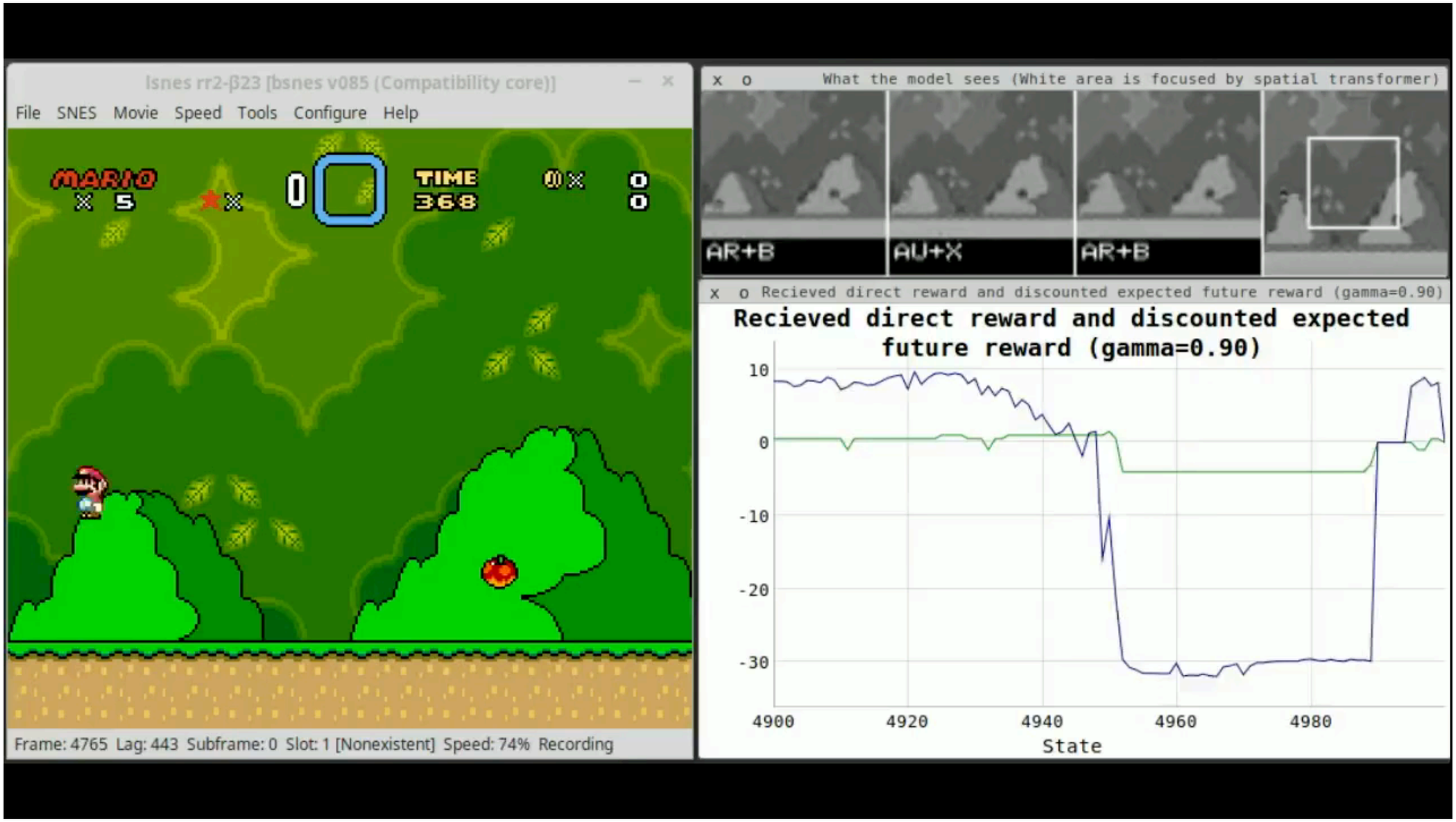
Reinforcement Learning

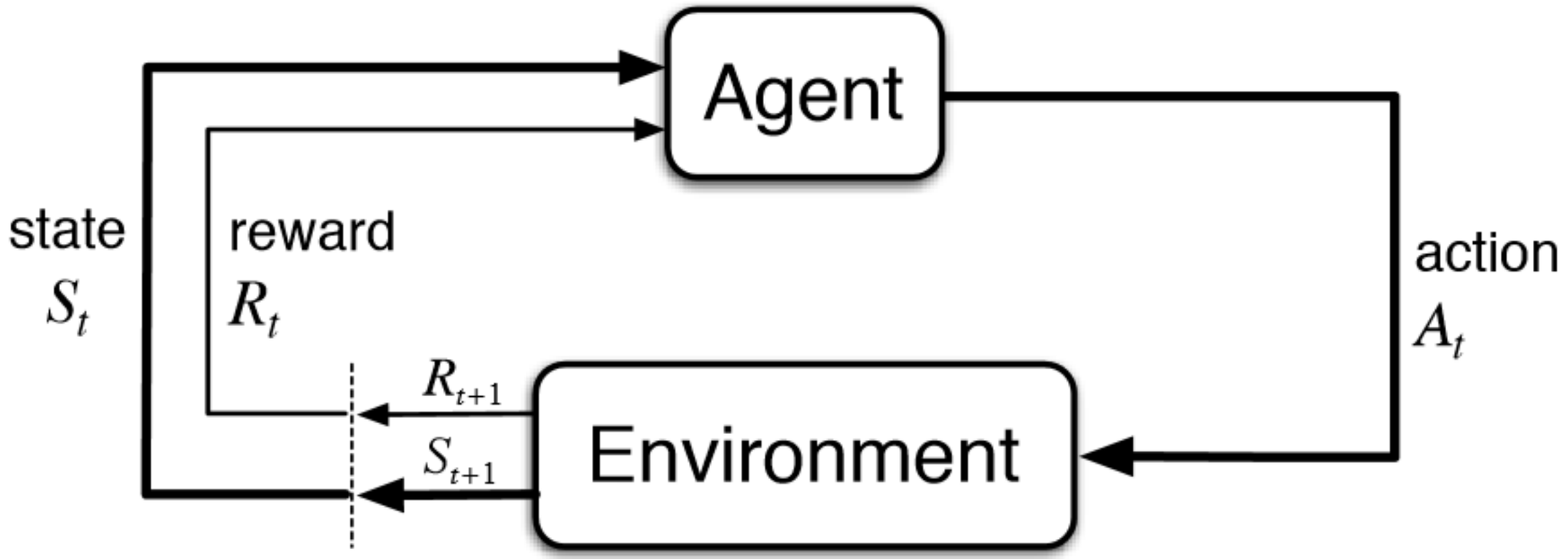
UCSD CSE 257

Sicun Gao

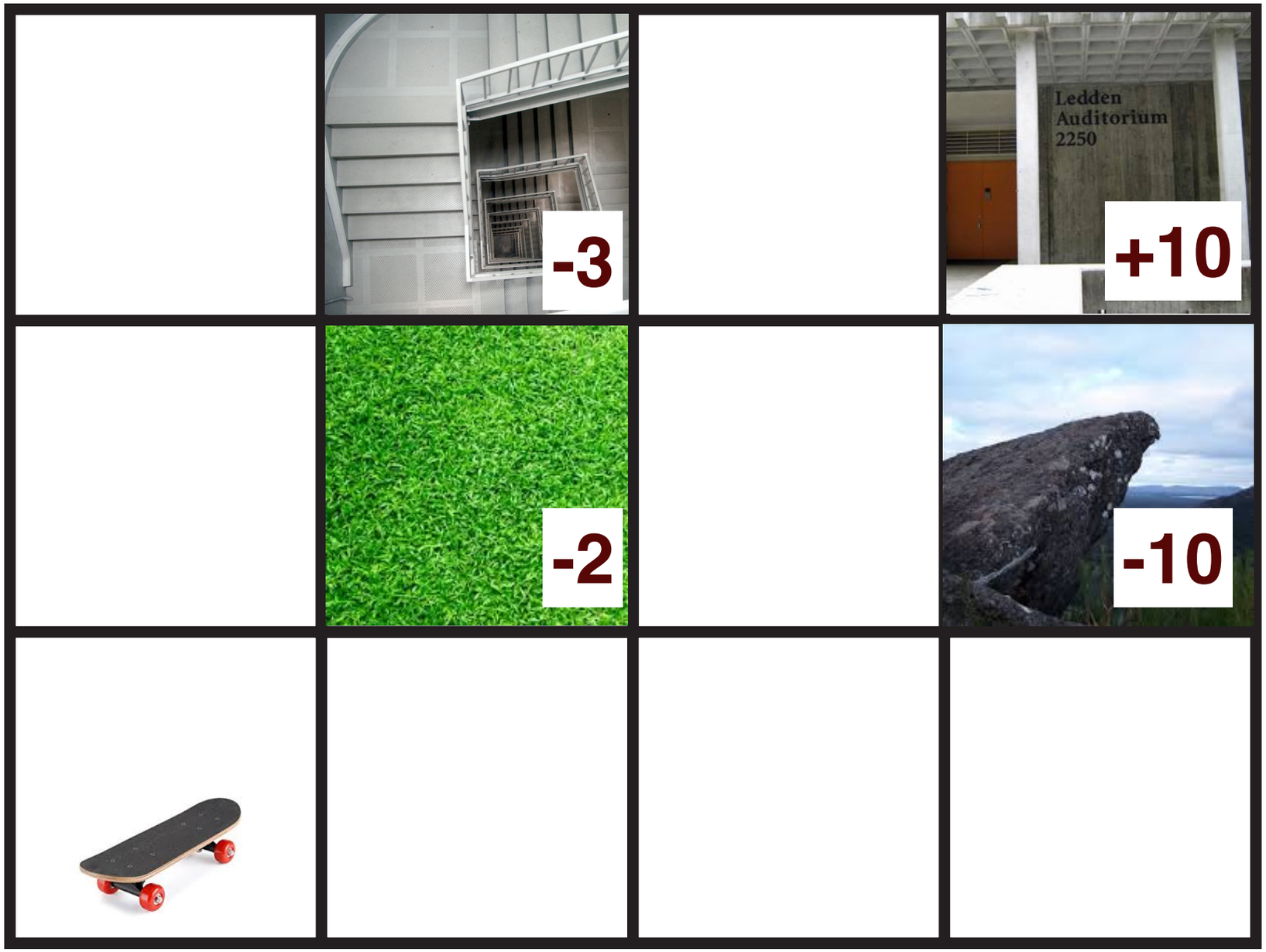

Isnes rr2-β23 [bsnes v085 (Compatibility core)]
File SNES Movie Speed Tools Configure Help
MARIO
X 5
TIME
368
Frame: 4765 Lag: 443 Subframe: 0 Slot: 1 [Nonexistent] Speed: 74% Recording
What the model sees (White area is focused by spatial transformer)
AR+B
AU+X
AR+B
Recieved direct reward and discounted expected future reward (gamma=0.90)
Recieved direct reward and discounted expected future reward (gamma=0.90)
10
0
-10
-20
-30
4900
4920
4940
4960
4980
State


Humanoid:
27 DoFs, 21 Actuators.

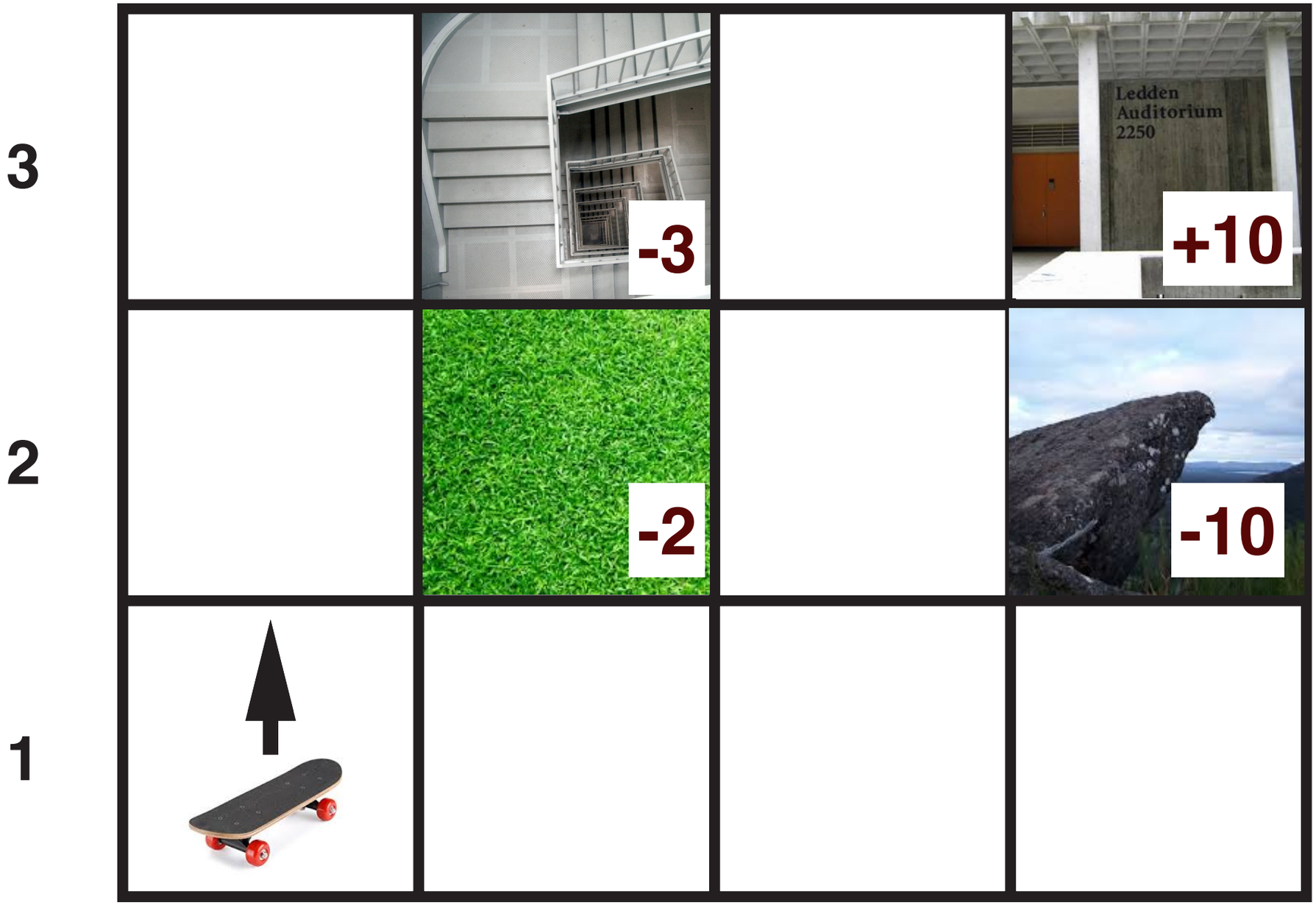
Markov Decision Processes



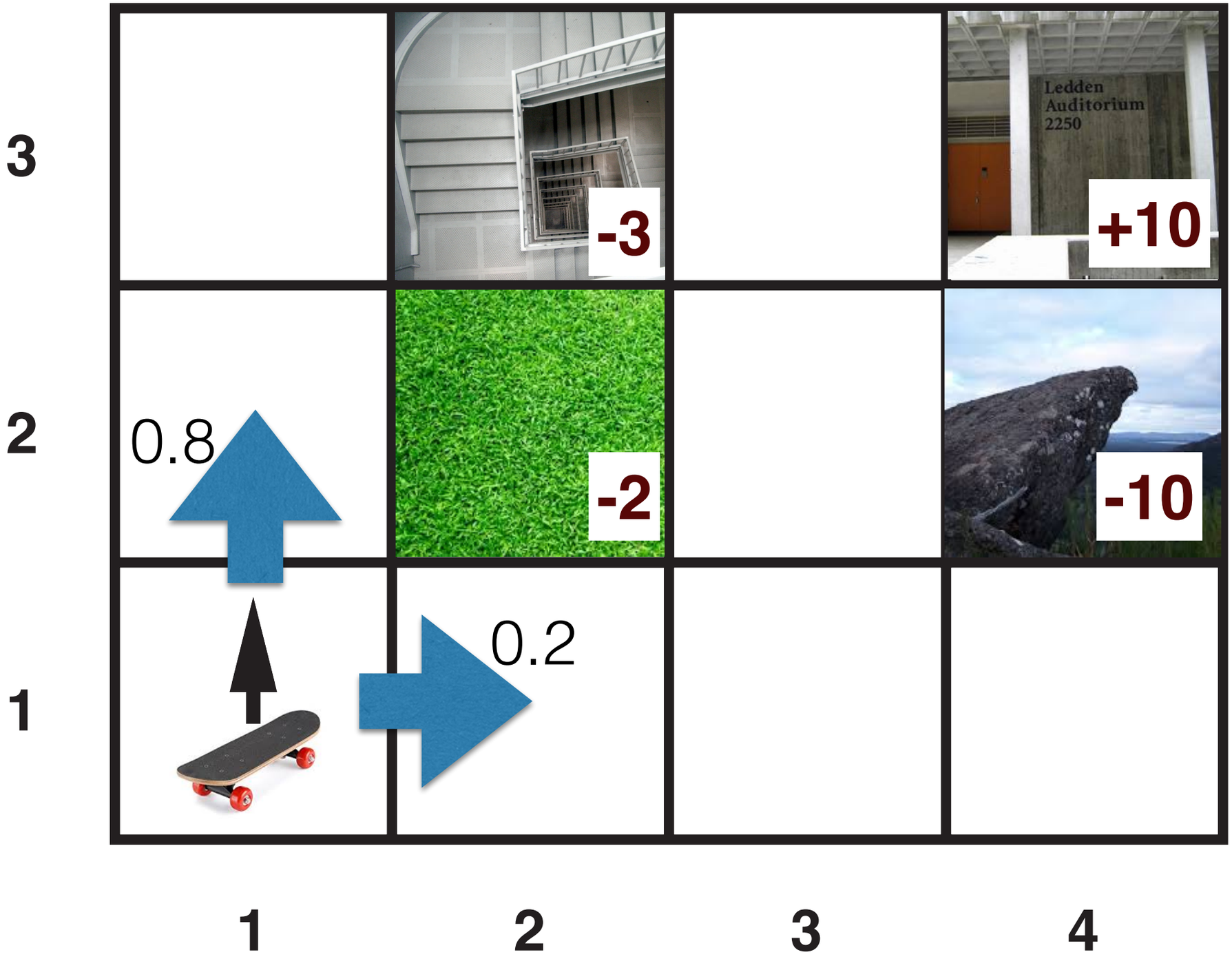
Skateboarding to Class



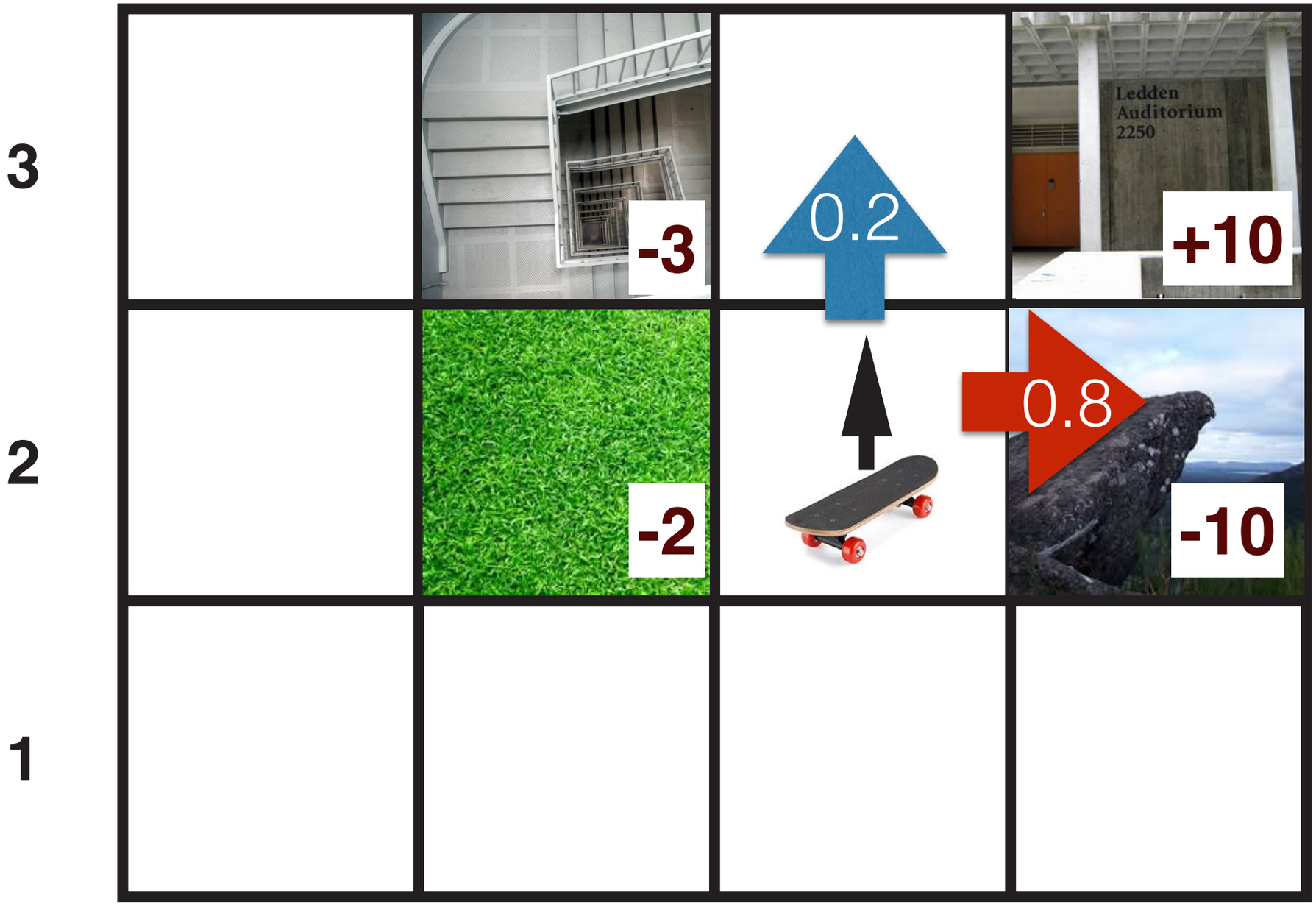
Skateboarding to Class



Skateboarding to Class



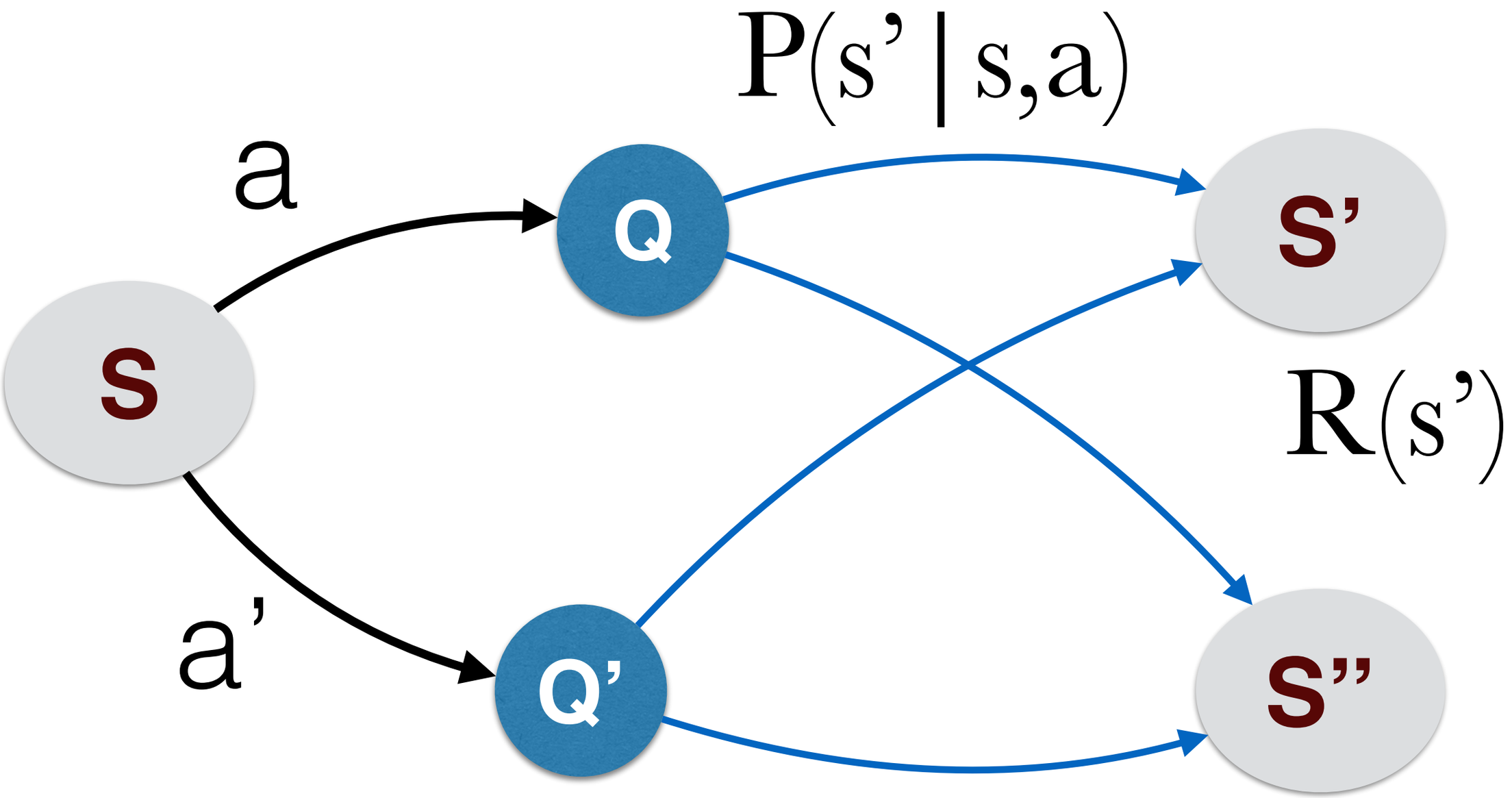
Skateboarding to Class



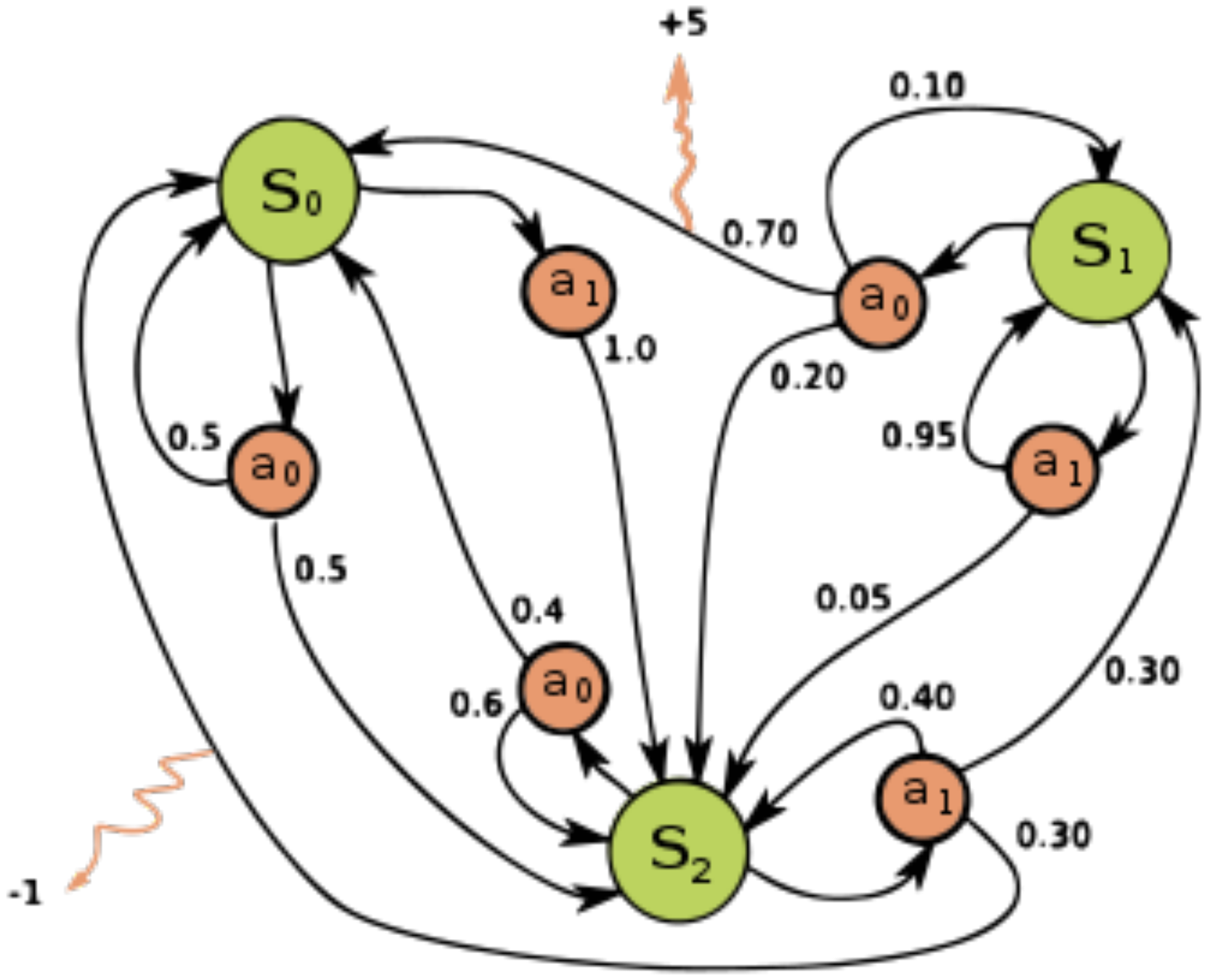
Markov Decision Processes

- States: {s, …}
- Actions: {a, …}
- Markovian Transition model: P(s’|s,a)
- Rewards: R(s)

Markov Decision Processes



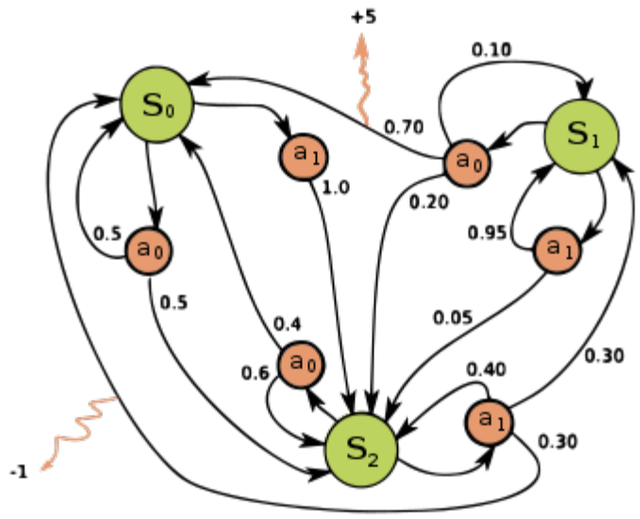
Markov Decision Processes

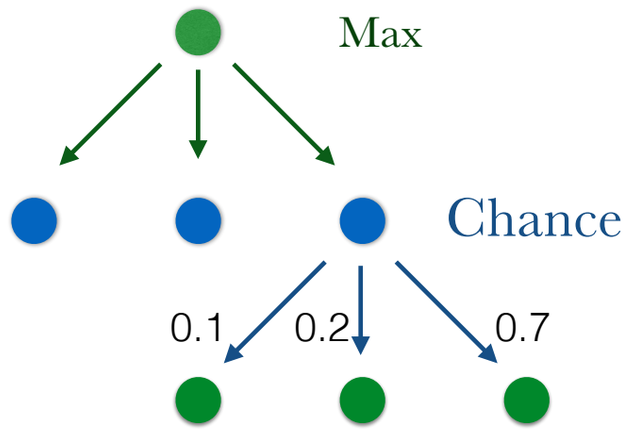


Policies and Values

- **Policy:** choosing actions from states
- **Value:** long-term performance

Think of the environment as a chance player





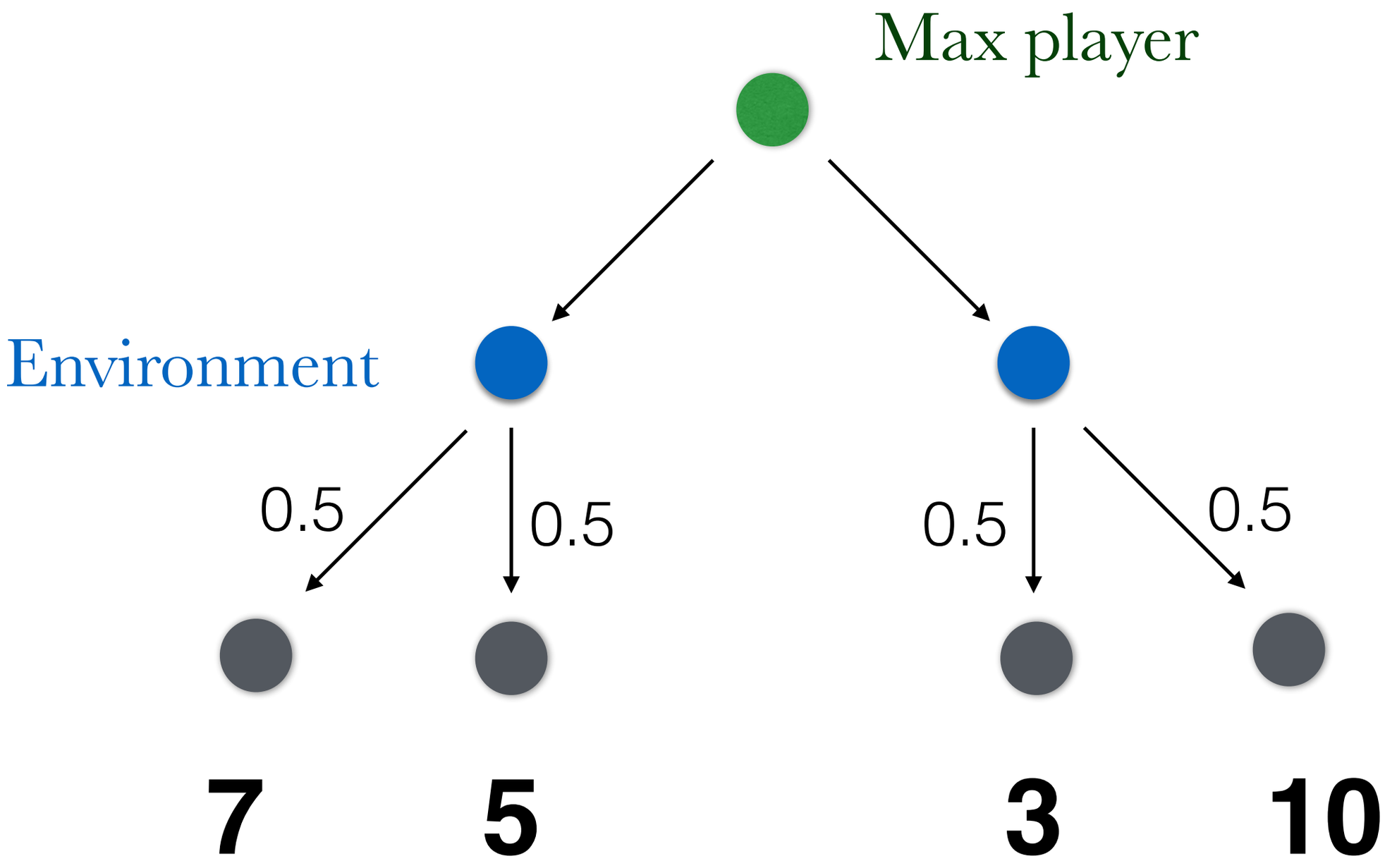
Policies and Values

- Policy (function of states):

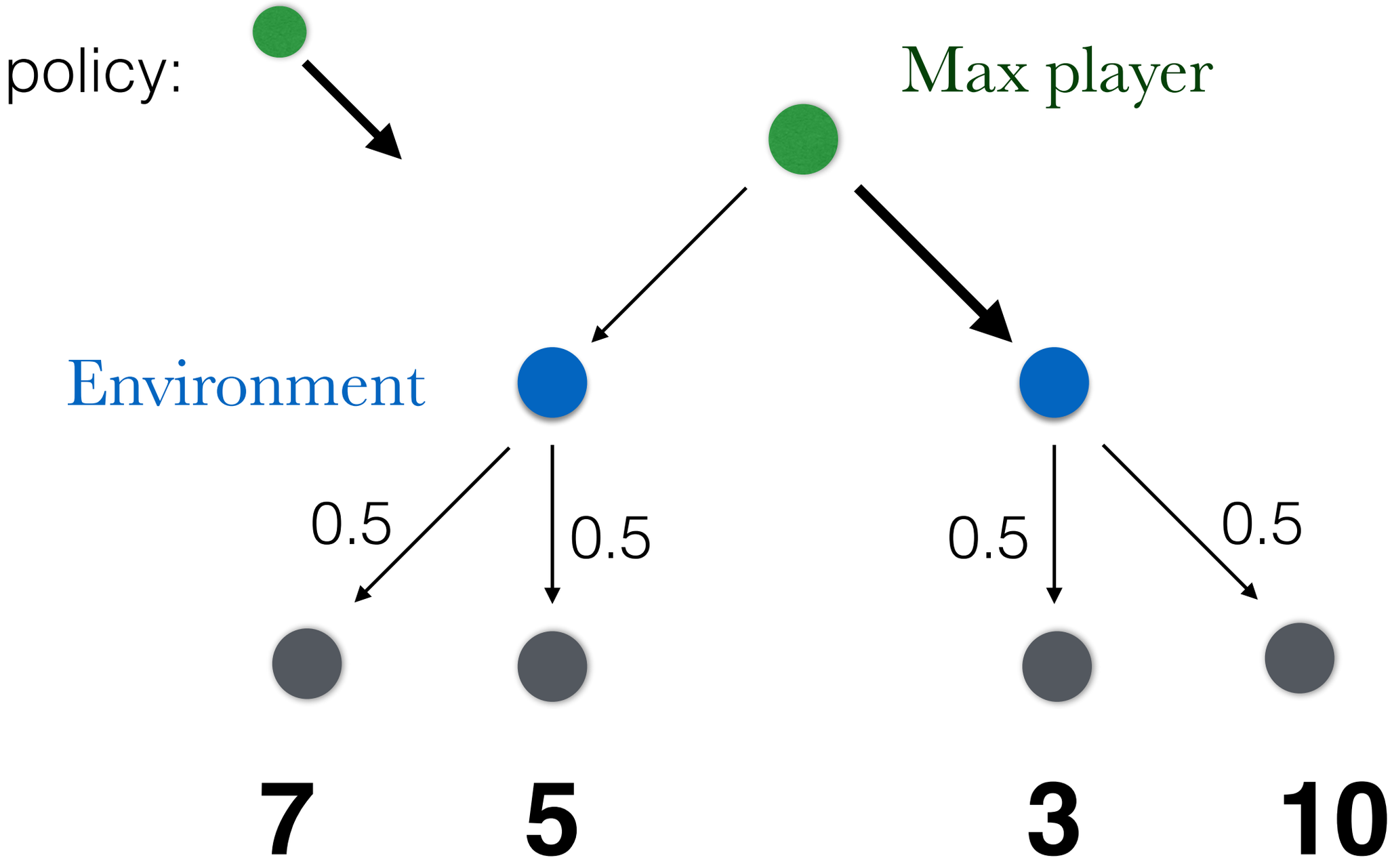
$\pi : S \to A$ (deterministic)

$\pi : S \times A \to [0,1]$ (probabilistic)

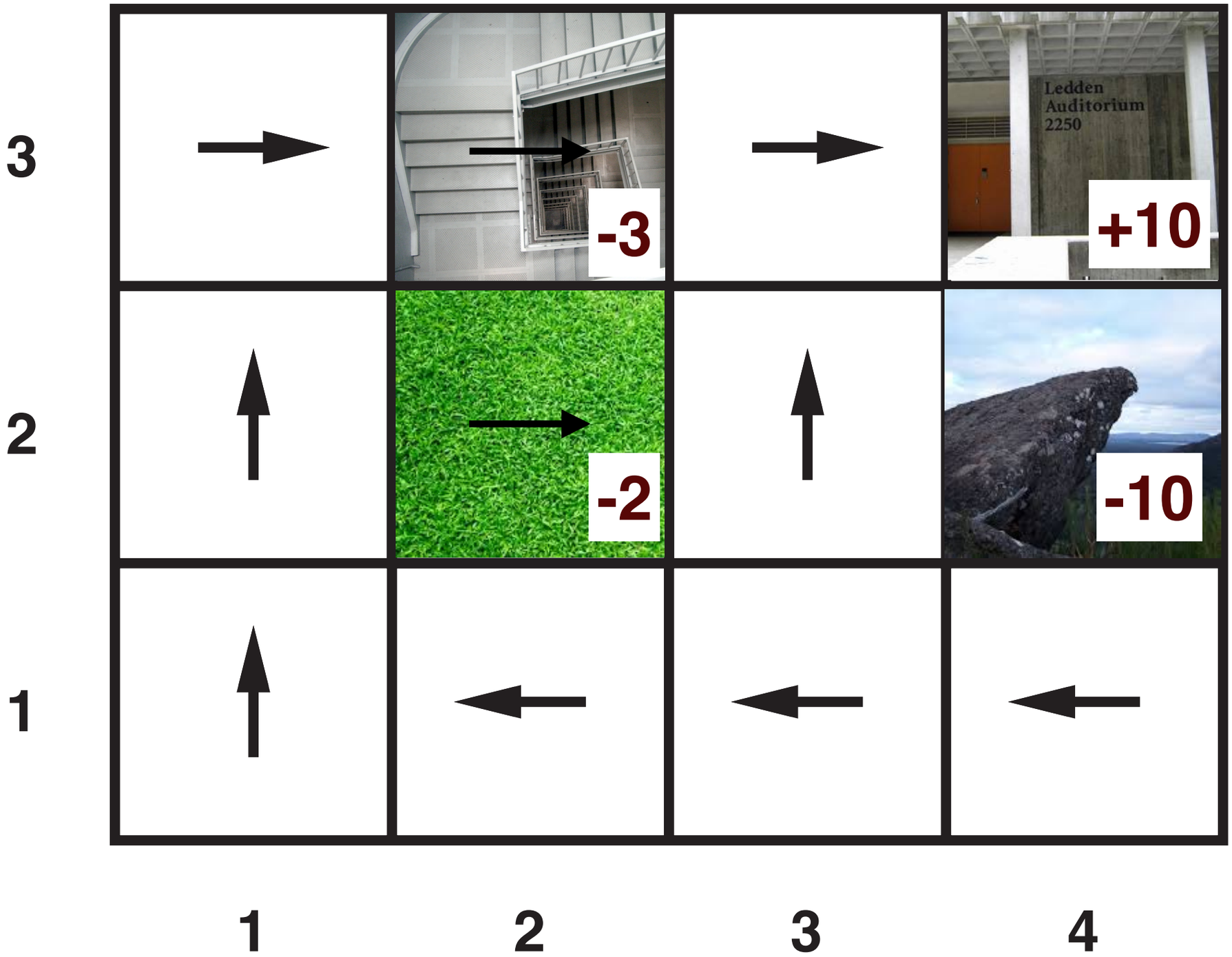
Policies and Values



Policies and Values



A Policy



Policies and Values

- State Value under a policy

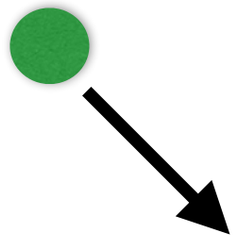
  Each policy can generate a lot of random trajectories

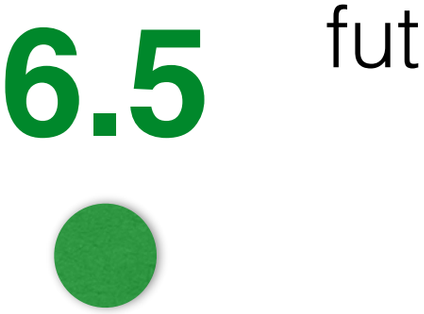
  $s_0, \pi(s_0), s_1, \pi(s_1), s_2, \pi(s_2), \cdots$

Policies and Values

state value: expected future reward at state

policy:

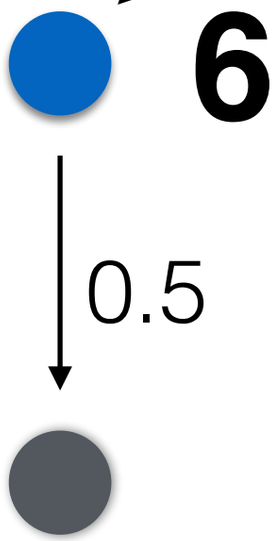




6.5

(Q value, more on this later)

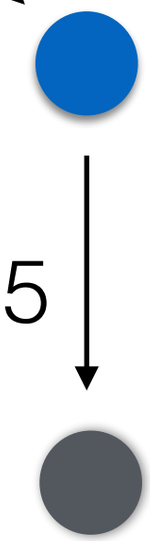
Environment



6

0.5

0.5



6.5

0.5

0.5

7

5

3

10

Policies and Values

- State Value under a policy

  Each policy can generate a lot of random trajectories

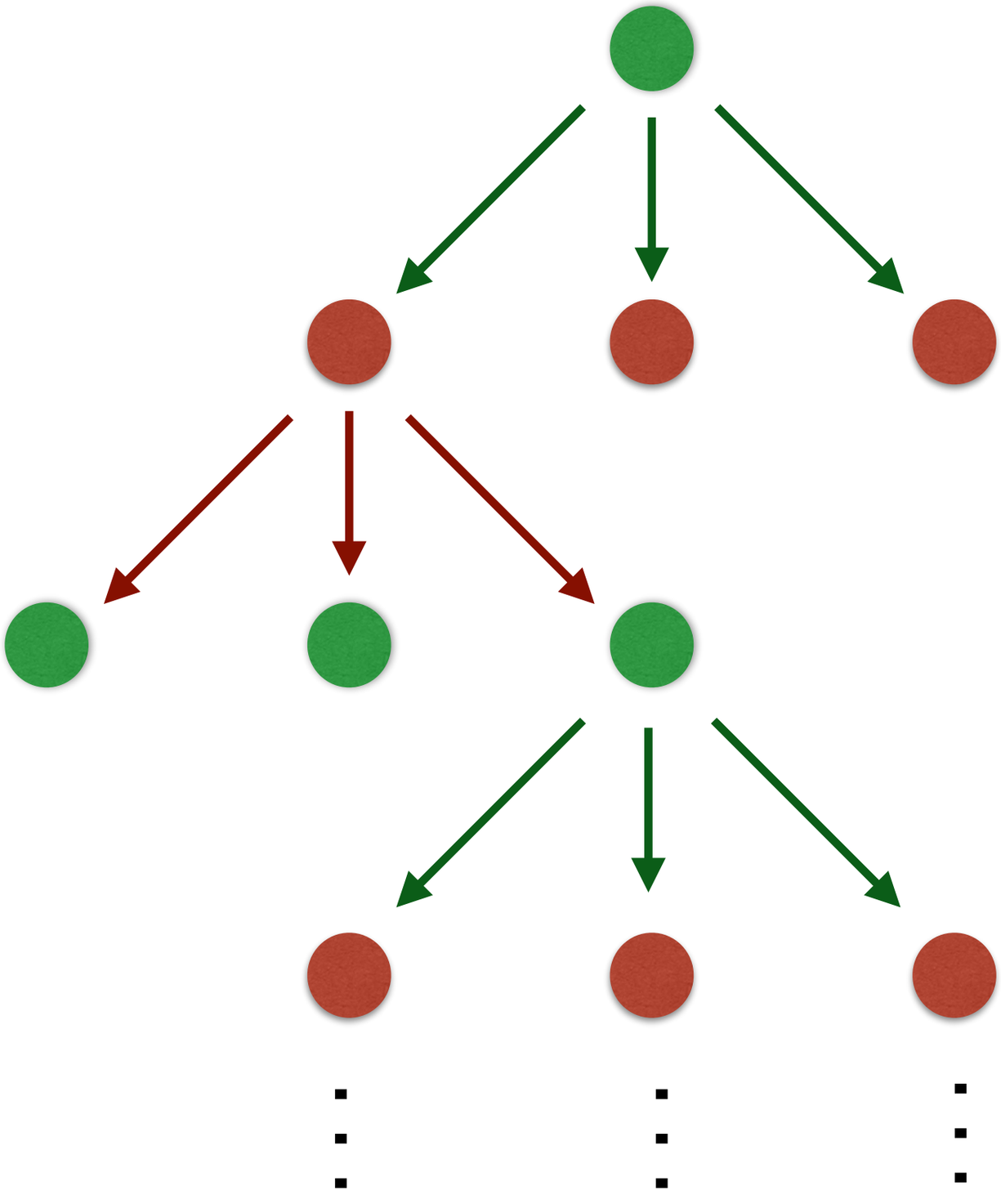
$$s_0, \pi(s_0), s_1, \pi(s_1), s_2, \pi(s_2), \cdots$$

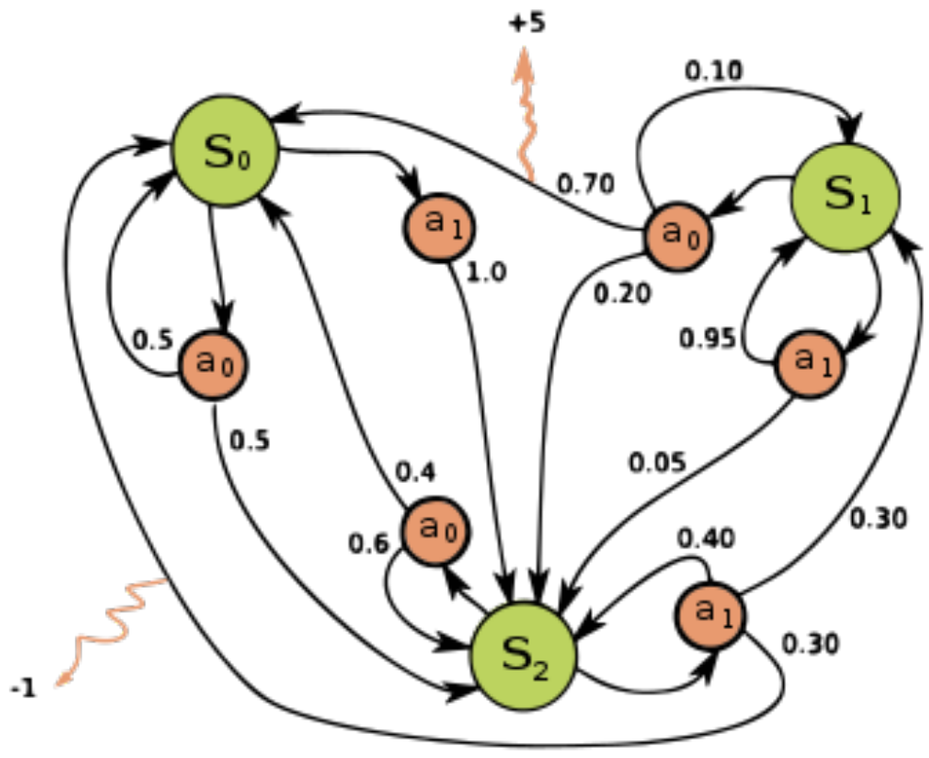
Tentative definition: expected total reward on all trajectories generated by a policy

$$V^{\pi}(s) = \mathbb{E}_{s_1, s_2, \ldots \sim \pi}[\sum_{i=0}^{\infty} R(s_i) \,|\, s_0 = s] \; ?$$

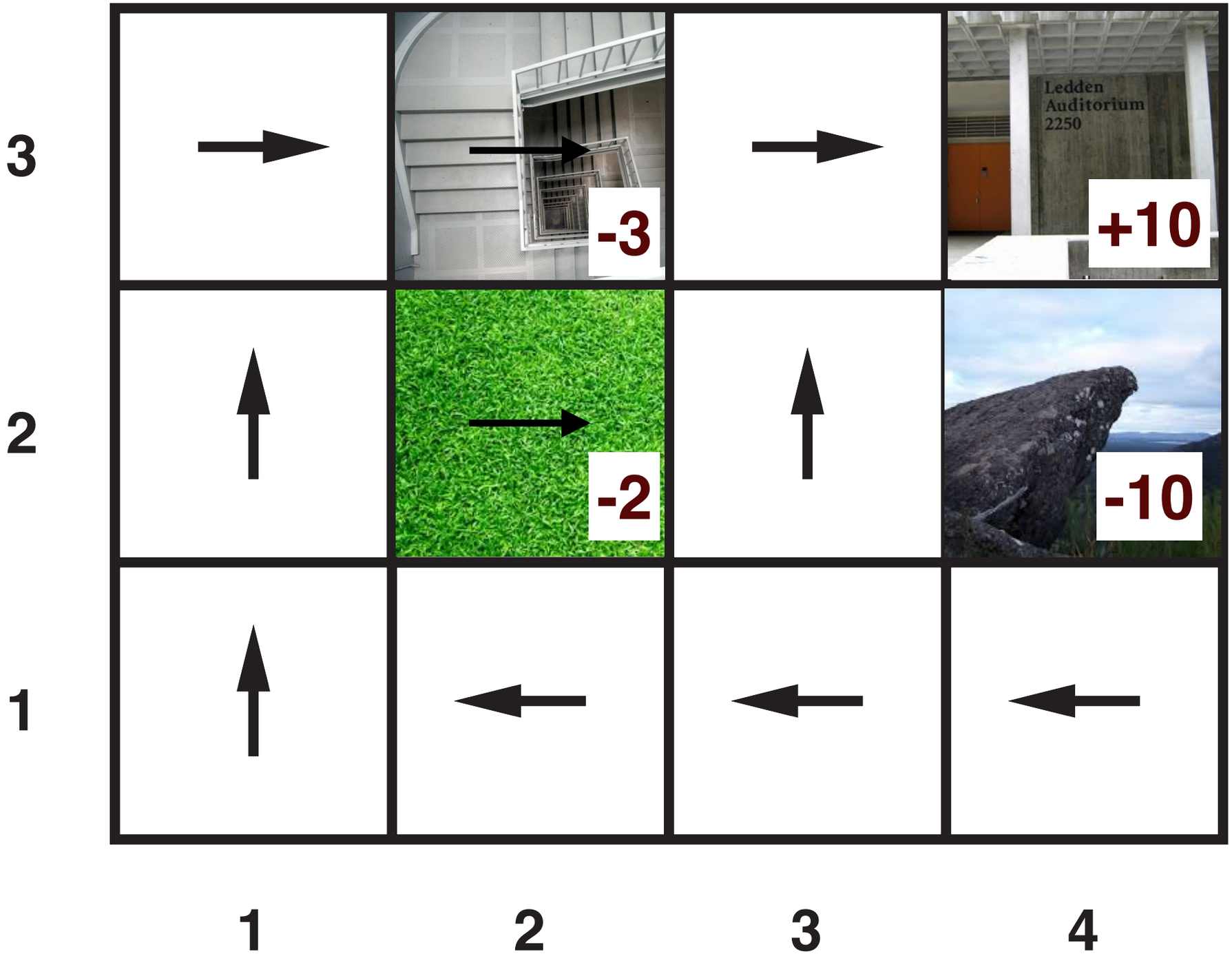
Problem?

Dealing with Infinite Paths



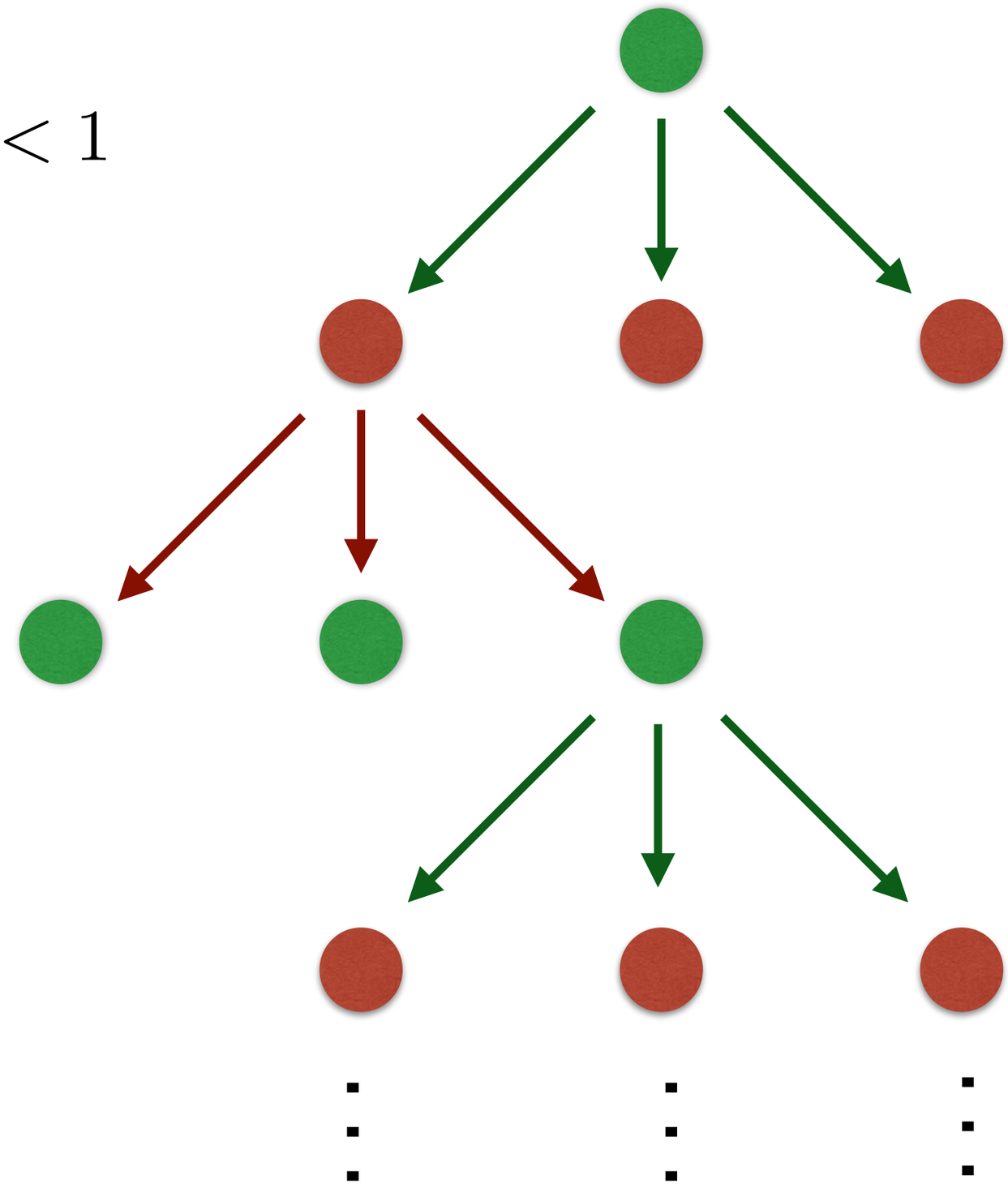


Dealing with Infinite Paths



Discount Factors

$$0 \leq \gamma < 1$$



γ^0

γ

γ^2

γ^3

Finite Total Reward in Infinite Paths

With discount factor: $0 \leq \gamma < 1$

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

$$= \sum_{i=0}^{\infty} \gamma^i R(s_i) \leq \frac{R_{\max}}{1-\gamma}$$

Now bounded and we can evaluate infinite paths!

Discount Factors

- Conceptual justification: Rewards that are in distant future may have higher uncertainty and do not provide immediate benefits.

- Mathematical benefits: Without it, the core algorithms will not converge. With it, all kinds of good properties follow.

Policies and Values

- Policy:

$$\pi : S \rightarrow A$$

- Value of a state under a policy:

$$V^{\pi}(s) = \mathbb{E}_{s_1, s_2, \ldots \sim \pi}[\sum_{i=0}^{\infty} \gamma^i R(s_i) \,|\, s_0 = s]$$

Blackjack

- If you play with “Hit whenever<20”, is it good or bad? How good? How bad?

$$V^{\pi}(s)$$

- How can we come up with an optimal way of playing?

$$\max_{\pi} V^{\pi}(s)$$

Optimal Policy

- Goal: find the policy that maximizes the value on all states

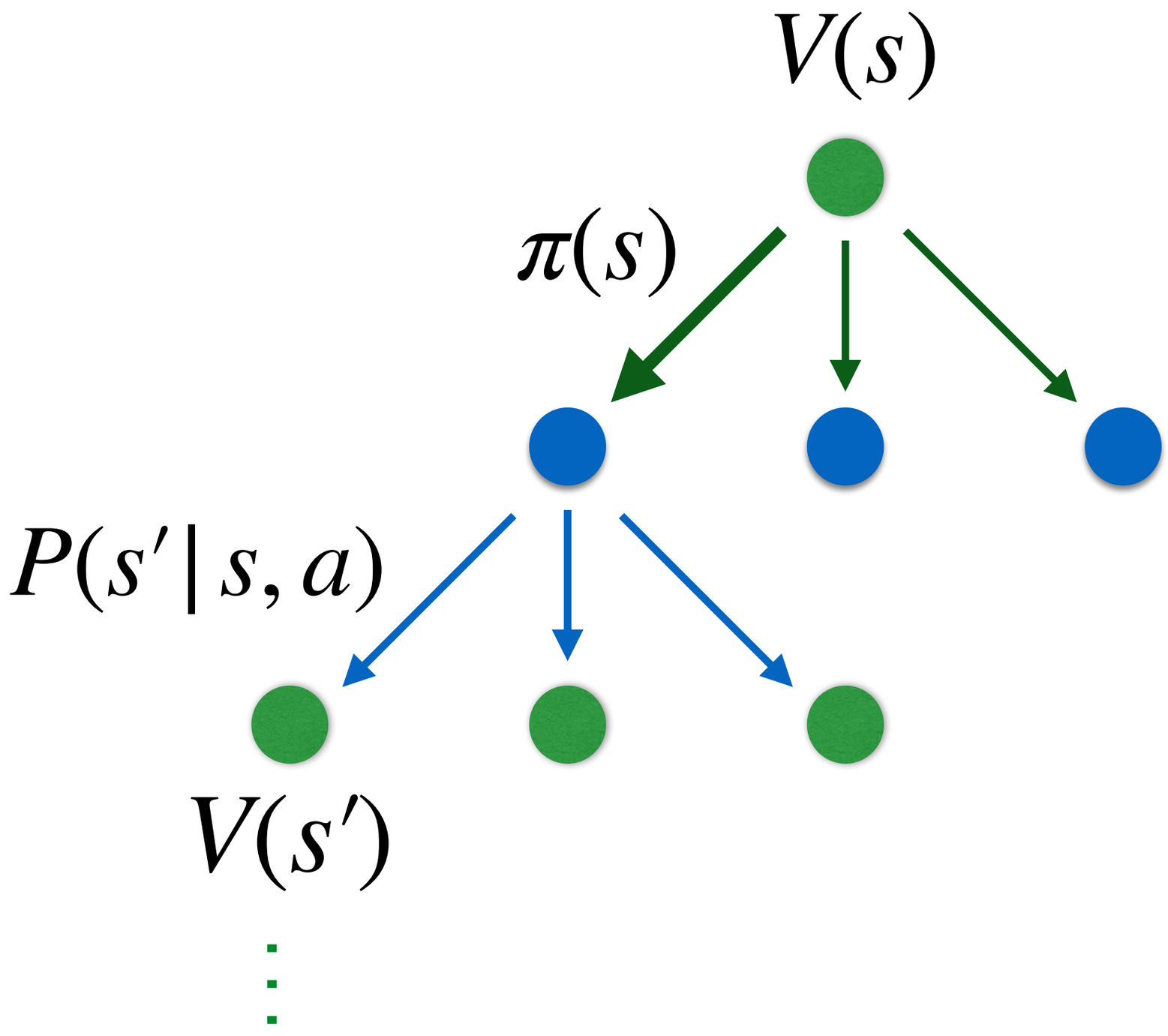
$$V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{i=0}^{\infty} \gamma^i R(s_i) \,|\, s_0 = s]$$

$$V^{*}(s) = \max_{\pi}(V^{\pi}(s))$$

Deriving Bellman Equations

- MDP seems complex but the value functions have interesting properties

Recall Expectimax



Deriving Bellman Equations

- MDP seems complex but the value functions have interesting properties

$$V^{\pi}(s) = R(s) + \gamma \sum_{i} P(s'_i \mid s, \pi(s)) V^{\pi}(s'_i)$$

Bellman Expectation Equation

$$V^{\pi}(s_0) = \mathbb{E}_{\pi}[\sum_{i=0}^{\infty} \gamma^i R(s_i)]$$

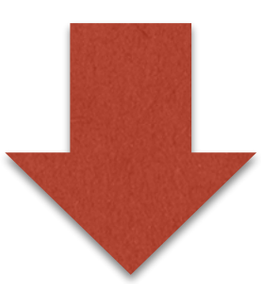
$$= R(s_0) + \mathbb{E}_{\pi}[\sum_{i=1}^{\infty} \gamma^i R(s_i)]$$

$$= R(s_0) + \mathbb{E}_{s_1 \sim \pi}\left(\gamma R(s_1) + \mathbb{E}_{s_2,.. \sim \pi}[\sum_{i=2}^{\infty} \gamma^i R(s_i)]\right)$$

$$\gamma V(s_1)$$

Bellman Expectation Equation

$$V^{\pi}(s_0) = \mathbb{E}_{\pi}[\sum_{i=0}^{\infty} \gamma^i R(s_i)]$$



Turning global estimates into local recurrence

$$V^{\pi}(s_0) = R(s_0) + \gamma \sum_{j} P(s_1^j) \cdot V^{\pi}(s_1^j)$$

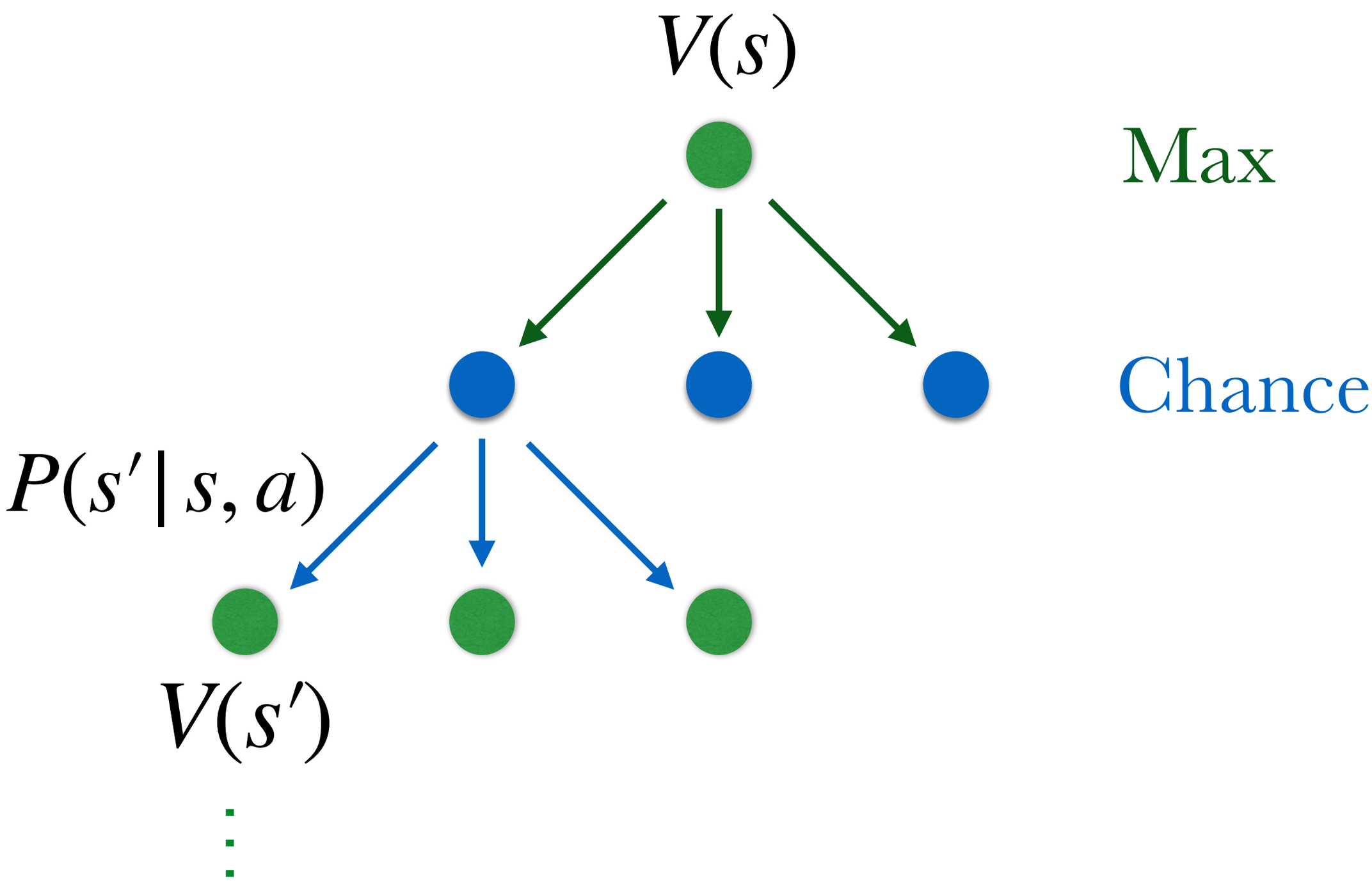
Bellman Expectation Equation

$$V^{\pi}(s) = R(s) + \gamma \sum_{i} P(s_i' \,|\, s, \pi(s)) \cdot V^{\pi}(s_i')$$

On both sides, apply $\max_{\pi}$

$$\max_{\pi} V^{\pi}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{i} P(s_i' \,|\, s, a) \cdot \max_{\pi}(V^{\pi}(s_i'))$$

Expectimax tree



Bellman Optimality Equation

Max-discounted-utility of next state

Immediate reward

$$V^*(s) = R(s) + \gamma \max_{a \in A(s)} \sum_i P(s'_i \mid s, a) \cdot V^*(s'_i)$$

This is exactly generalized Expectimax.

Algorithms for Computing π^*

- Value Iteration
  - Essentially Expectimax (Bellman Equation)
  - Important: Fixed Points

- Policy Iteration
  - Cheaper alternative if you don't need to all the values

Solving Bellman Equations

- For each MDP we have these equations

$$V(s_0) = R(s_0) + \gamma \max_{a \in A(s_0)} \sum_{s'} P(s'|s,a)V(s')$$

$$V(s_1) = R(s_1) + \gamma \max_{a \in A(s_1)} \sum_{s'} P(s'|s,a)V(s')$$

$$\cdots$$

$$V(s_n) = R(s_n) + \gamma \max_{a \in A(s_n)} \sum_{s'} P(s'|s,a)V(s')$$

Solving Bellman Equations

- How many unknowns?
- How many equations?

$$V(s_0) = R(s_0) + \gamma \max_{a \in A(s_0)} \sum_{s'} P(s'|s,a)V(s')$$

$$V(s_1) = R(s_1) + \gamma \max_{a \in A(s_1)} \sum_{s'} P(s'|s,a)V(s')$$

$$\cdots$$

$$V(s_n) = R(s_n) + \gamma \max_{a \in A(s_n)} \sum_{s'} P(s'|s,a)V(s')$$

Solving Bellman Equations

- Can we solve these equations?

$$V(s_0) = R(s_0) + \gamma \max_{a \in A(s_0)} \sum_{s'} P(s'|s,a)V(s')$$

$$V(s_1) = R(s_1) + \gamma \max_{a \in A(s_1)} \sum_{s'} P(s'|s,a)V(s')$$

$$\cdots$$

$$V(s_n) = R(s_n) + \gamma \max_{a \in A(s_n)} \sum_{s'} P(s'|s,a)V(s')$$

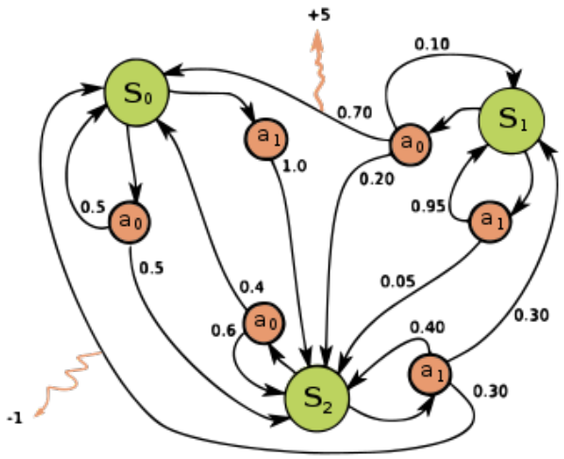
Solving Bellman Equations

The Most Naive Algorithm:

- Start with arbitrary values on each V(s)
- "Bellman Update"

$$V(s_i) \leftarrow R(s_i) + \gamma \max_{a \in A(s_i)} \sum_{s'} P(s'|s,a)V(s')$$

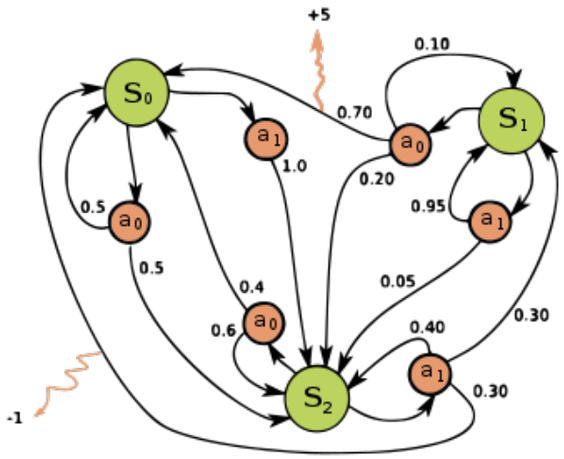
- Loop till convergence, if ever



$$\overrightarrow{V} = (V(s_1), \ldots, V(s_n))$$

V-Space

$\overrightarrow{V}_0$

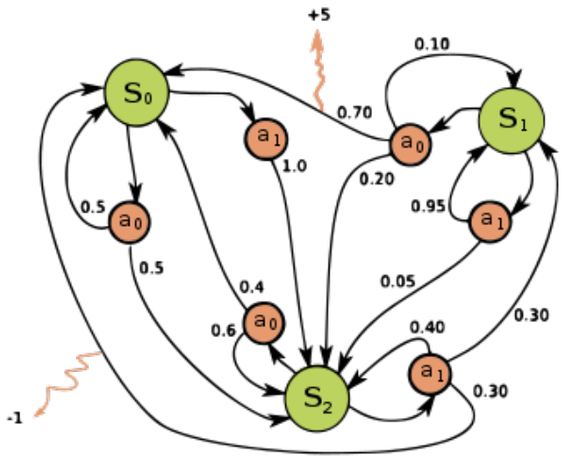


$$\overrightarrow{V} = (V(s_1), \ldots, V(s_n))$$

V-Space

$\overrightarrow{V}_1 = B(\overrightarrow{V}_0)$

$\overrightarrow{V}_0$



$$\overrightarrow{V} = (V(s_1), \ldots, V(s_n))$$

V-Space

$\overrightarrow{V}_2 = B(\overrightarrow{V}_1)$

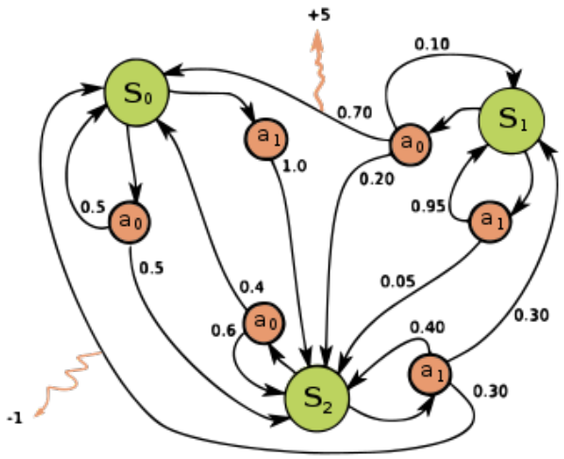
$\overrightarrow{V}_1 = B(\overrightarrow{V}_0)$

$\overrightarrow{V}_0$

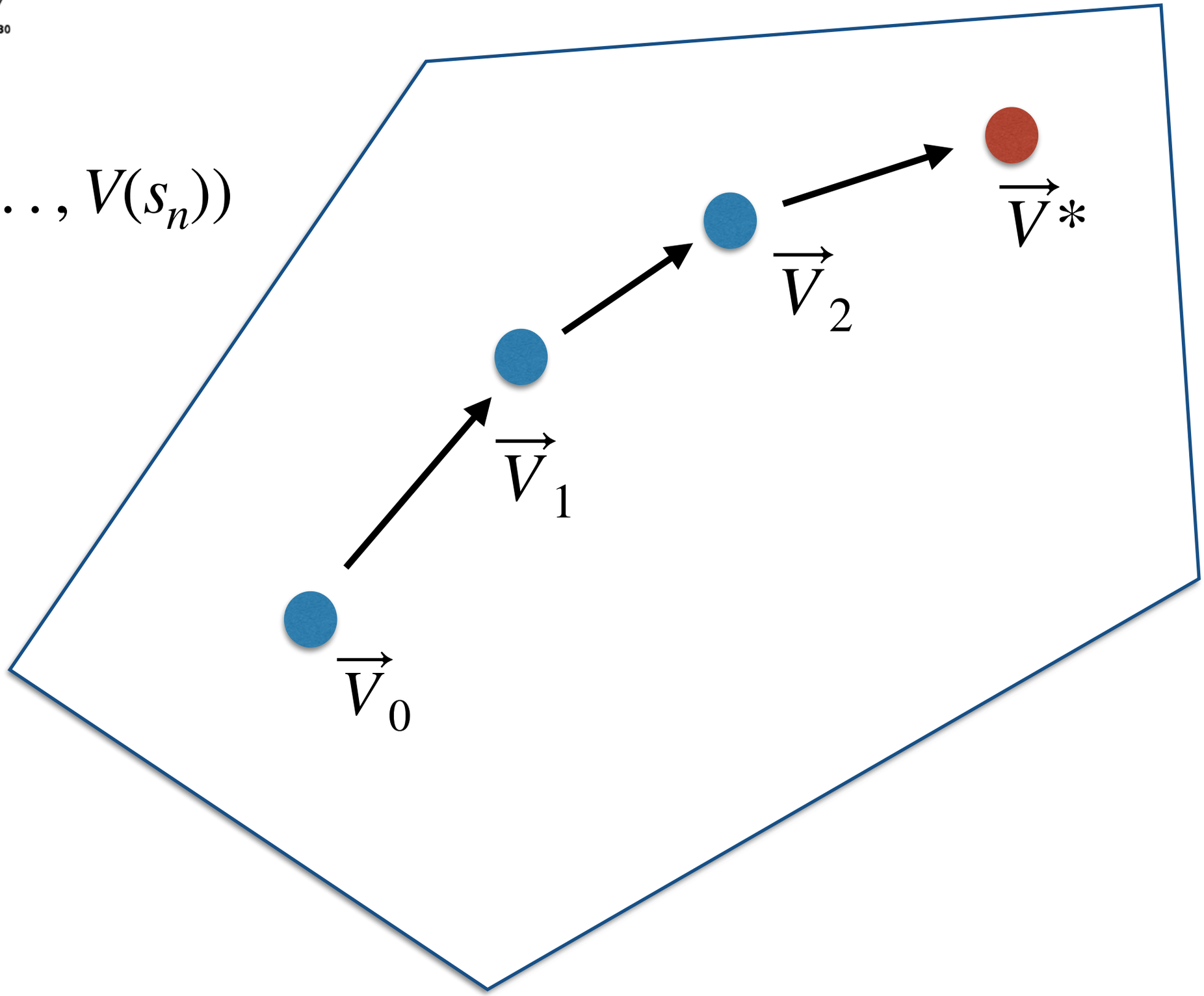
Solving Bellman Equations

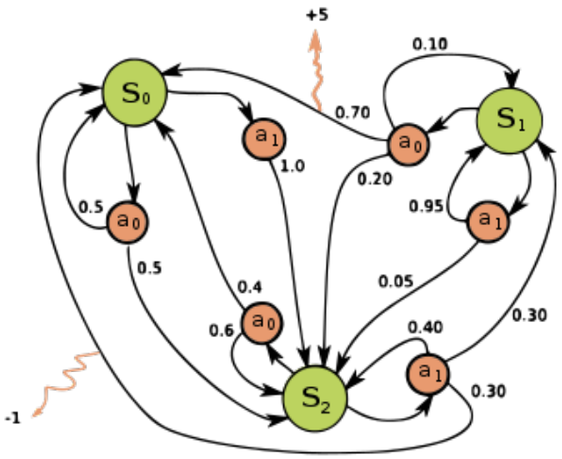
Theorem:
The Most Naive Algorithm works.

repeat
$U \leftarrow U'$; $\delta \leftarrow 0$
for each state s **in** S **do**
$$U'[s] \leftarrow R(s) \; + \; \gamma \; \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \; U[s']$$



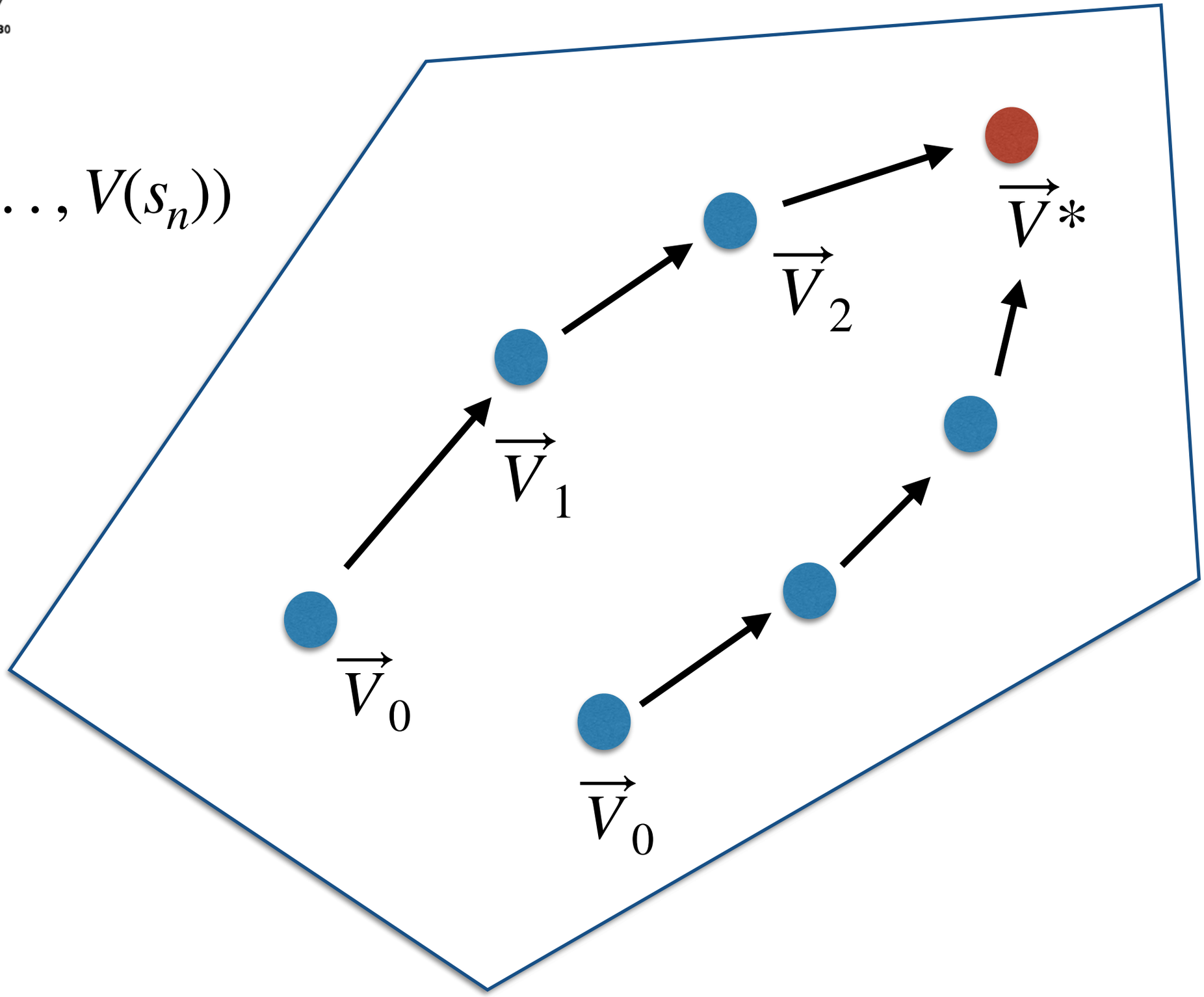
$$\vec{V} = (V(s_1), \ldots, V(s_n))$$





$$\vec{V} = (V(s_1), \ldots, V(s_n))$$

V-Space

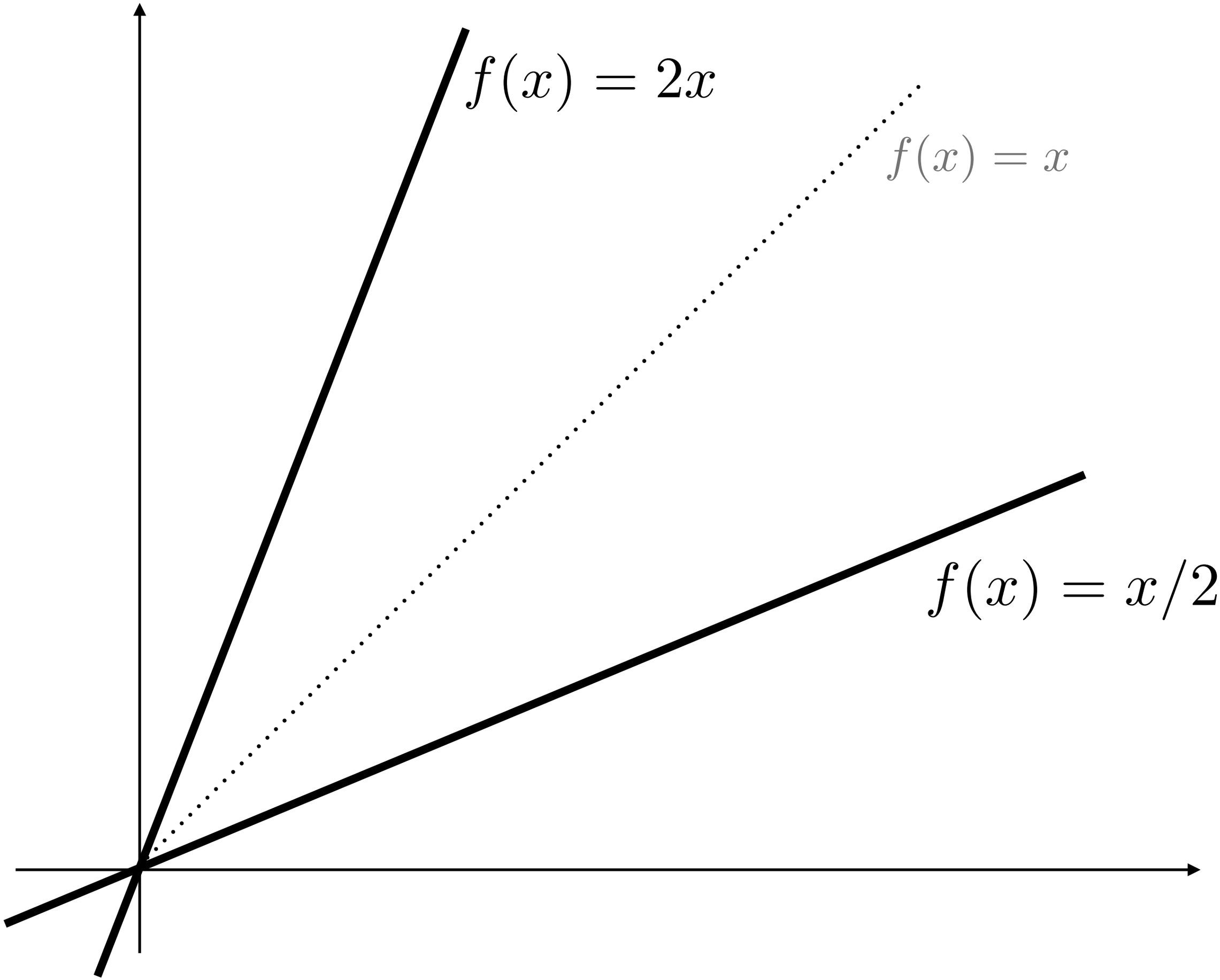


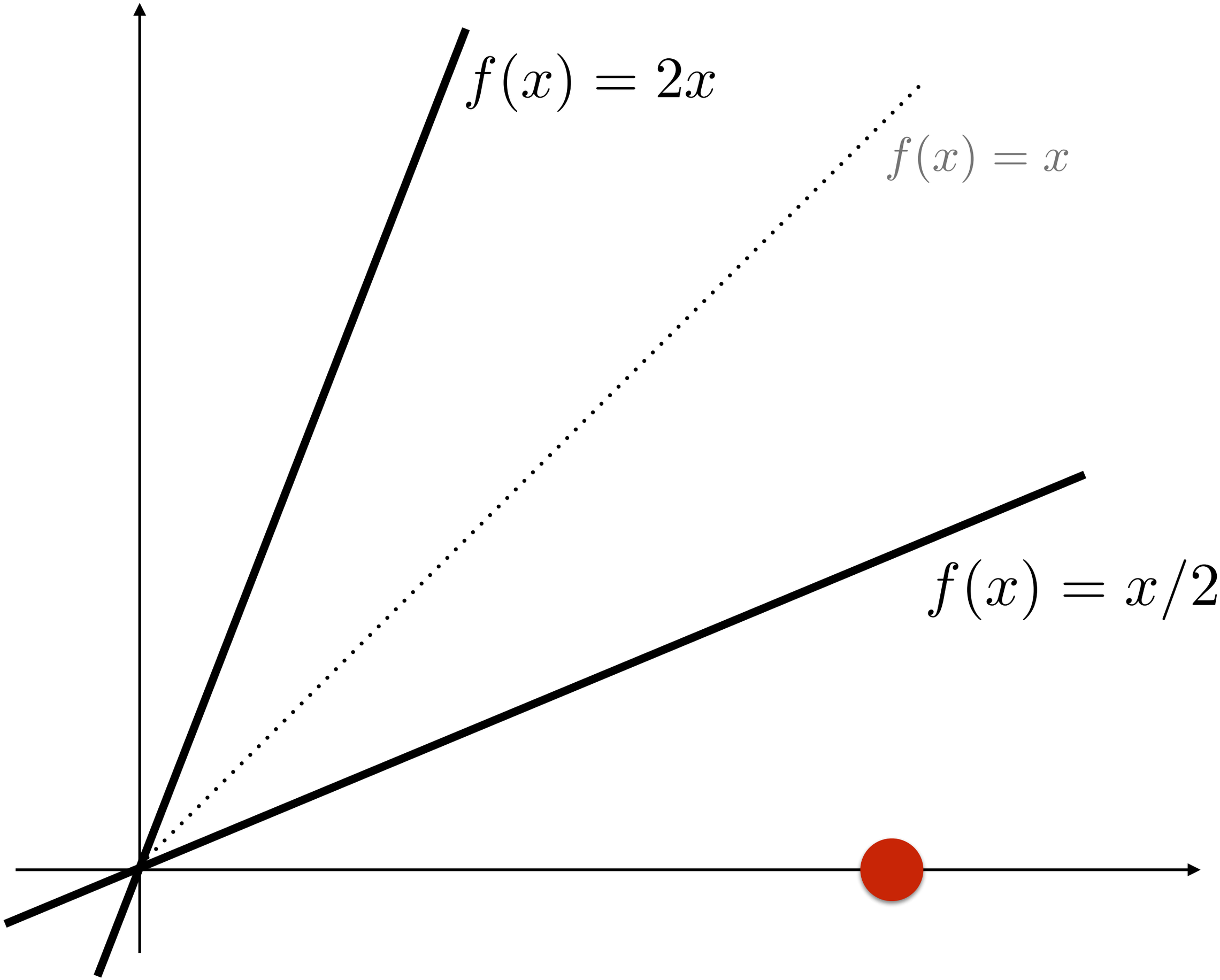
Contraction Mapping

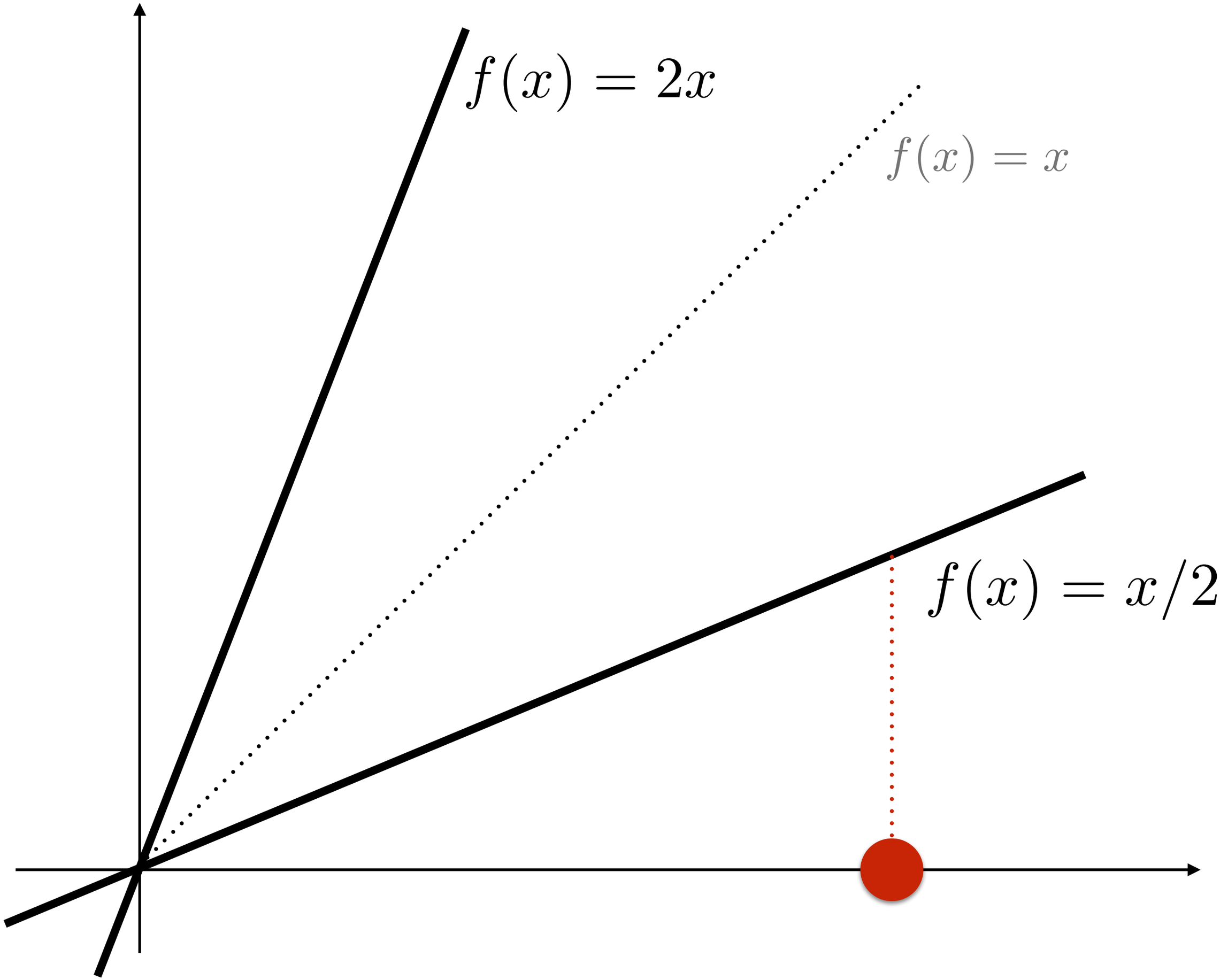
We say a function $f: \mathbb{R}^n \to \mathbb{R}^n$ is a contraction mapping for some metric $\|\cdot\|$, if there exists a contraction rate $k \in [0,1)$, such that for any $x_1, x_2 \in \mathbb{R}^n$,

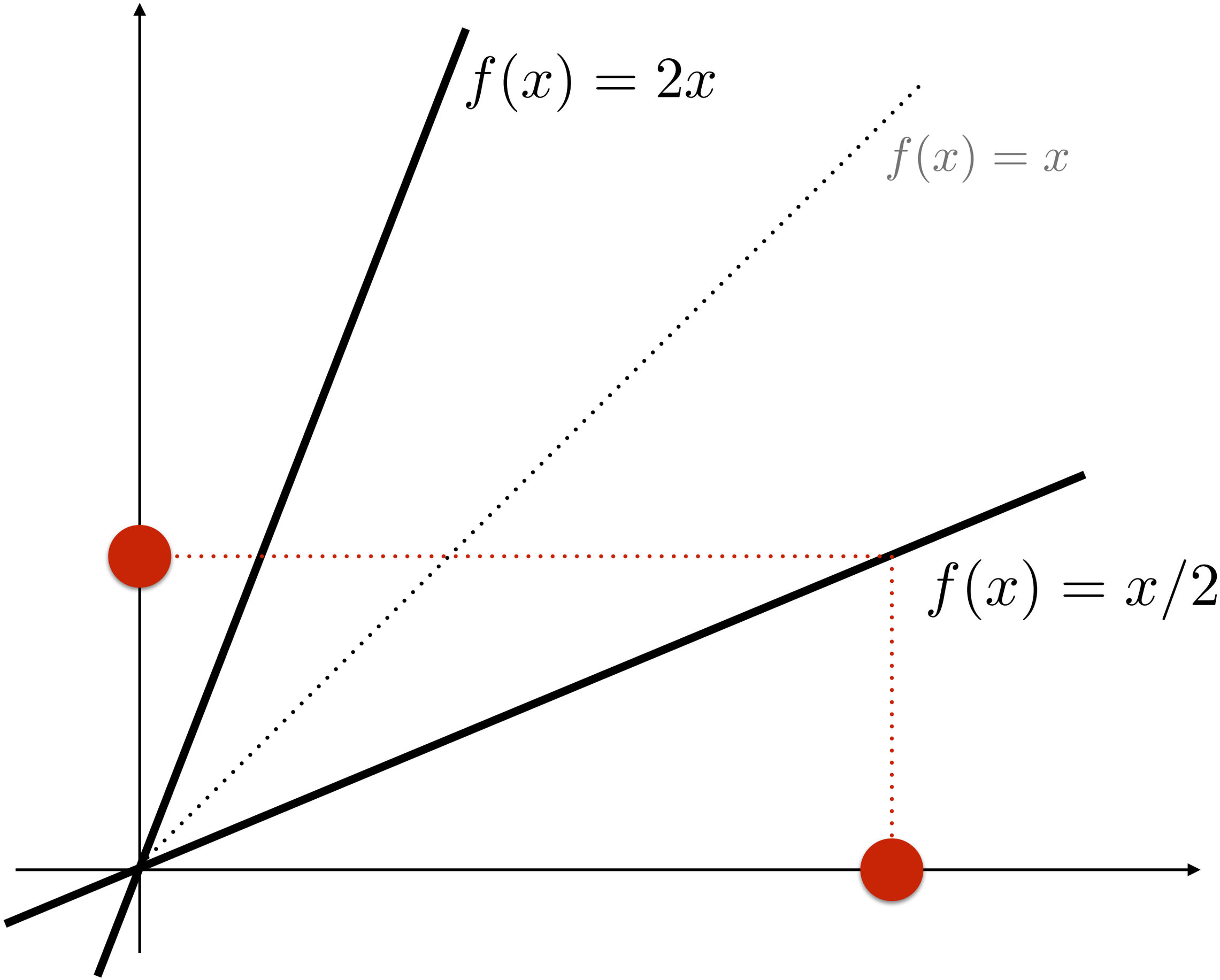
$$\|f(x_1) - f(x_2)\| \leq k\|x_1 - x_2\|$$

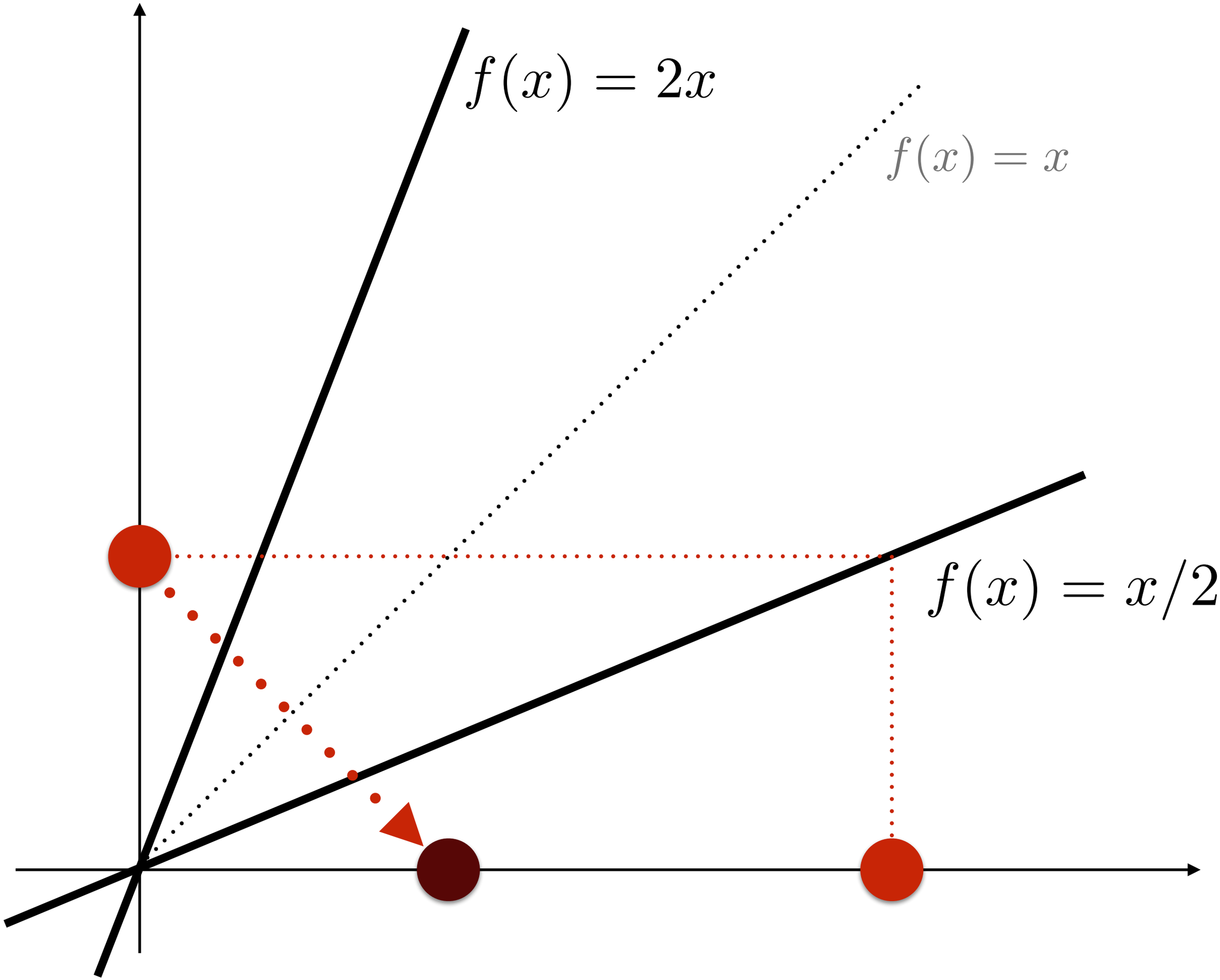
Example: $f(x) = x/2$


$f(x) = 2x$
$f(x) = x$
$f(x) = x/2$


$f(x) = 2x$
$f(x) = x$
$f(x) = x/2$


$f(x) = 2x$
$f(x) = x$
$f(x) = x/2$

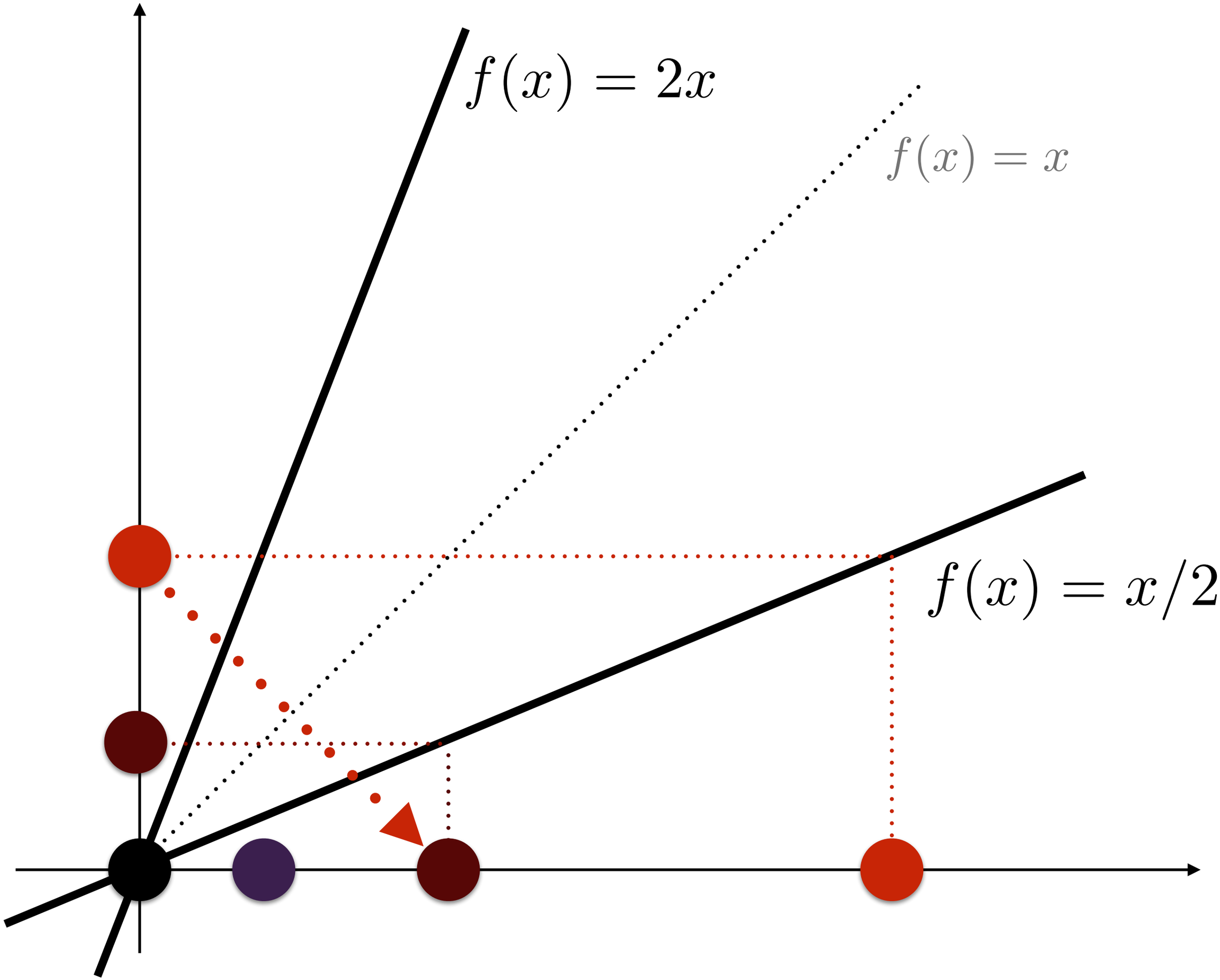

$f(x) = 2x$
$f(x) = x$
$f(x) = x/2$

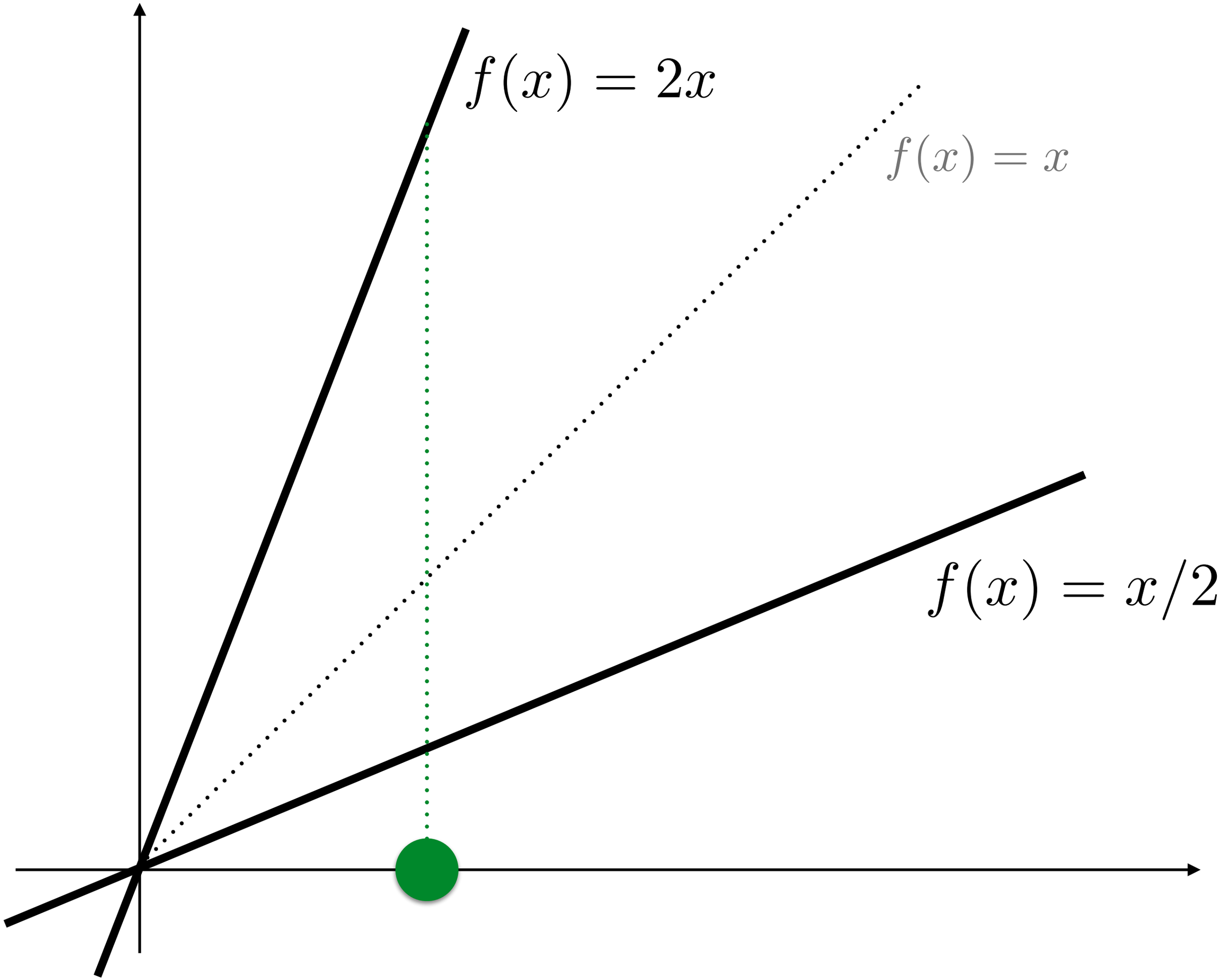

$f(x) = 2x$
$f(x) = x$
$f(x) = x/2$

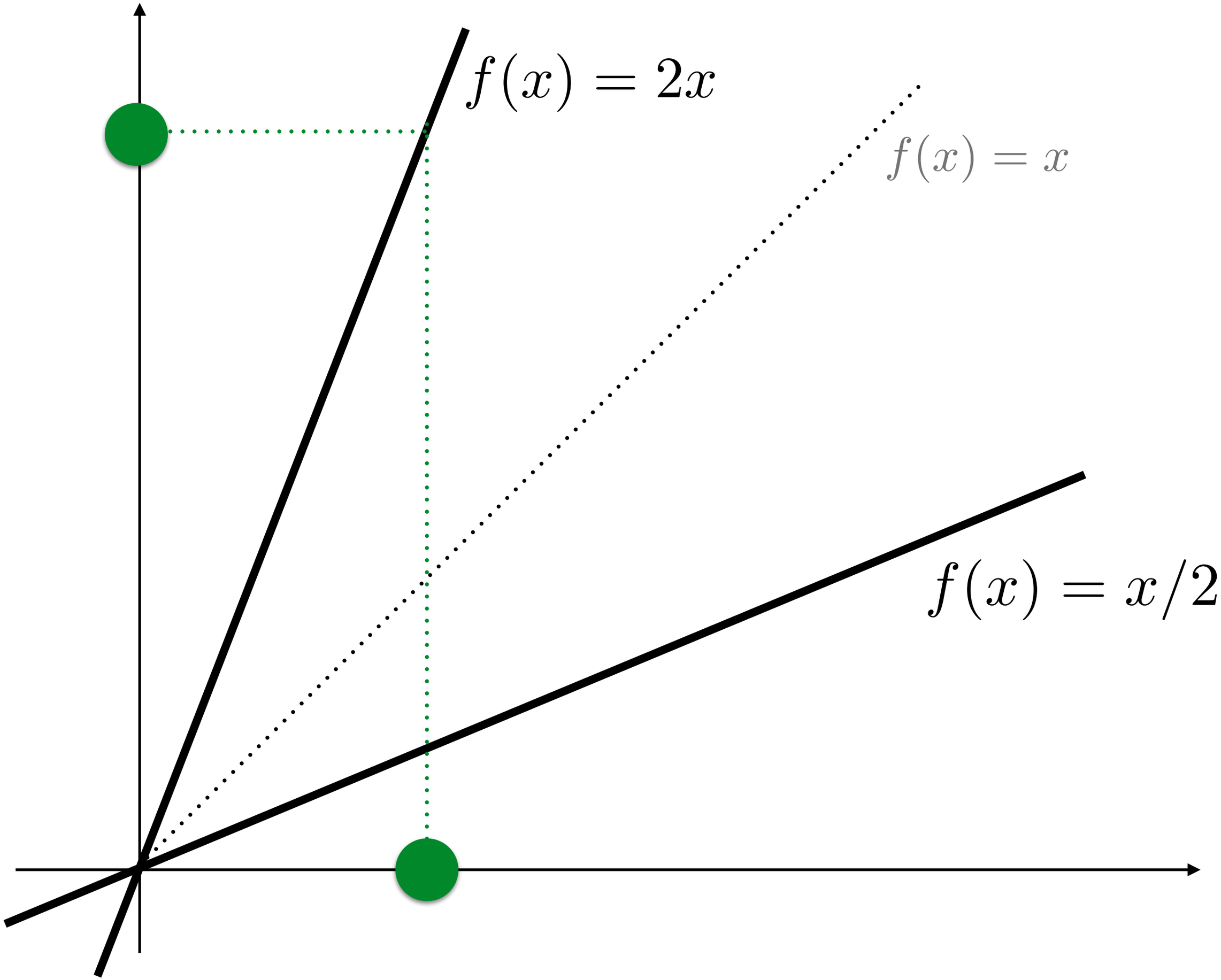
$f(x) = 2x$

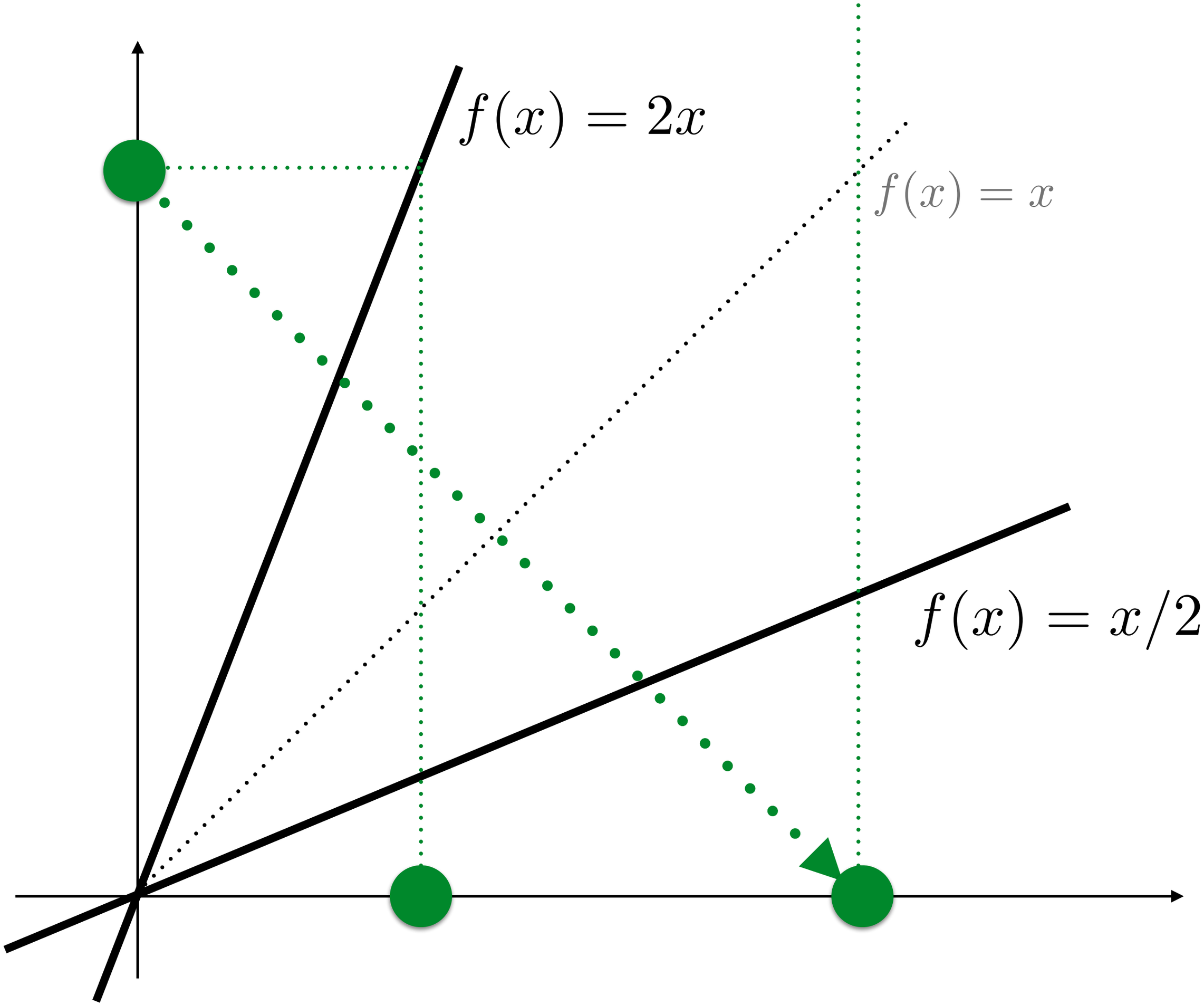
$f(x) = x$

$f(x) = x/2$


$f(x) = 2x$
$f(x) = x$
$f(x) = x/2$


$f(x) = 2x$
$f(x) = x$
$f(x) = x/2$


$f(x) = 2x$
$f(x) = x$
$f(x) = x/2$


$f(x) = 2x$
$f(x) = x$
$f(x) = x/2$

Why naive iteration works

Theorem: A contraction mapping has a unique **fixed point** that satisfies

$$x = f(x)$$

Why naive iteration works

$$x \leftarrow f(x)$$

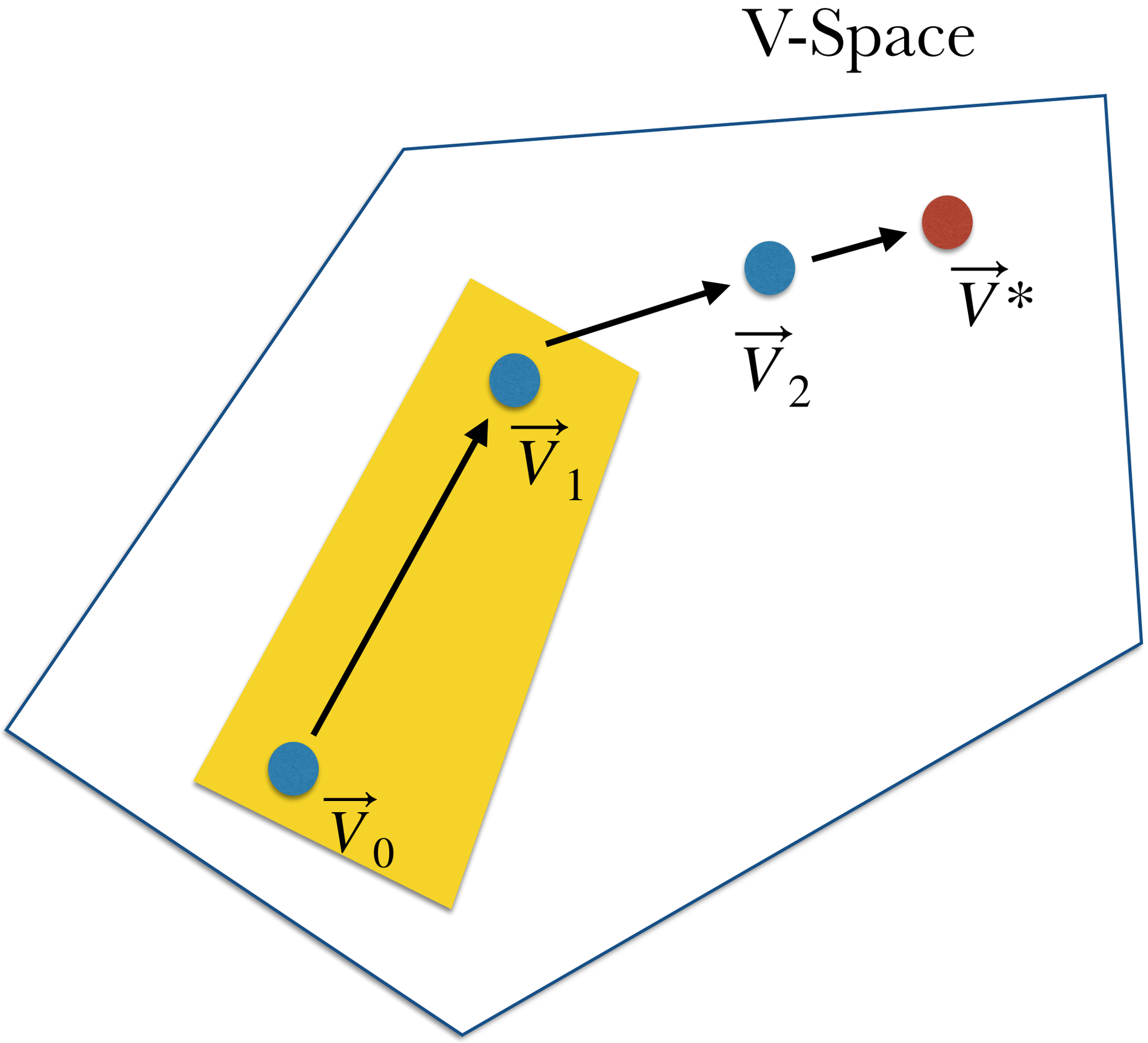
- Existence. Consider the sequence of movement under f from an arbitrary initial point. Contraction ensures a Cauchy sequence.
- Uniqueness. Suppose there are two fixed points, apply the mapping, they have to converge.

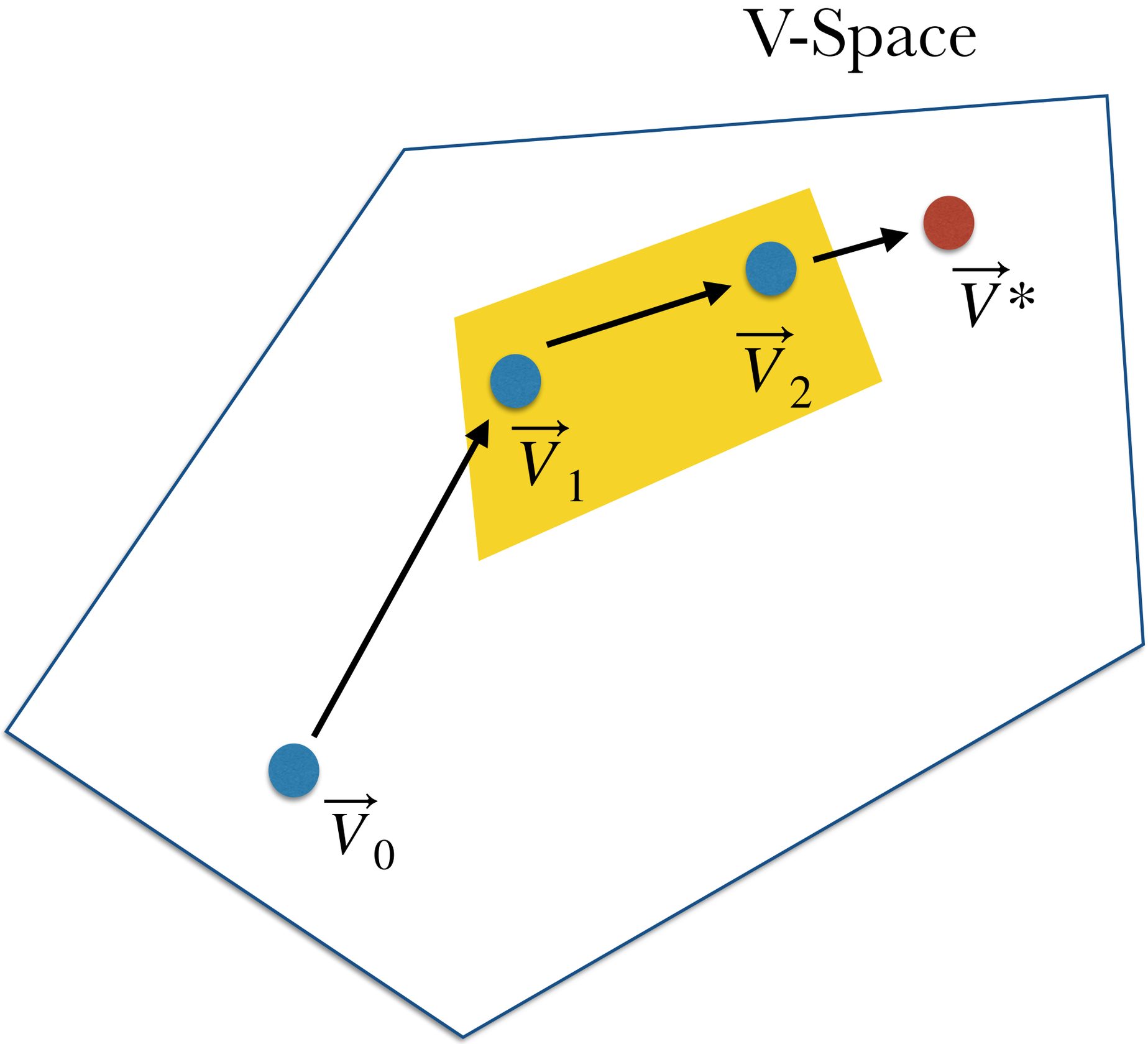
Why naive iteration works

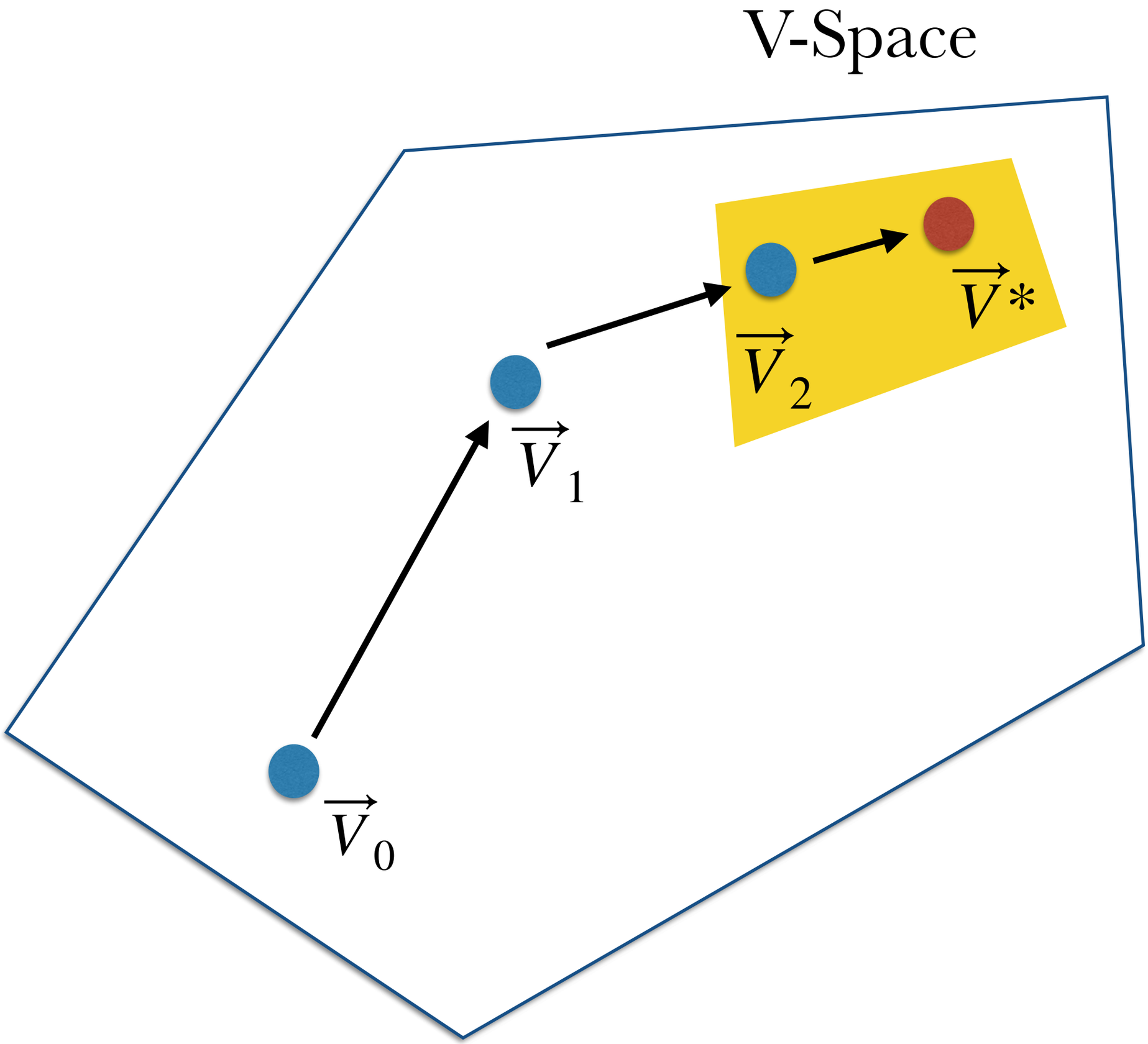
Key Lemma:

The Bellman Equations define a **contraction mapping** of utilities.

$$V(s_i) \leftarrow R(s_i) + \gamma \max_{a \in A(s_i)} \sum_{s'} P(s'|s,a)V(s')$$


V-Space
$\vec{V}_1$
$\vec{V}_2$
$\vec{V}*$
$\vec{V}_0$


V-Space
$\overrightarrow{V}_0$
$\overrightarrow{V}_1$
$\overrightarrow{V}_2$
$\overrightarrow{V}*$


V-Space
$\overrightarrow{V}_0$
$\overrightarrow{V}_1$
$\overrightarrow{V}_2$
$\overrightarrow{V}*$

Contraction of Bellman Expectation

Let's first look at the simpler case:

$$V^{\pi}(s_0) \leftarrow R(s_0) + \gamma \sum_{j} P(s_1^j) V^{\pi}(s_1^j)$$

We write this as:

$$V^{\pi} \leftarrow B(V^{\pi})$$

Contraction of Bellman Expectation

Let's first look at the simpler case:

$$\boxed{V^{\pi}(s_0)} \leftarrow R(s_0) + \gamma \sum_{j} P(s_1^j) \boxed{V^{\pi}(s_1^j)}$$

new

old

We write this as:

$$V^{\pi} \leftarrow B(V^{\pi})$$

Contraction of Bellman Expectation

- Consider the max norm

$$\|V^{\pi} - \hat{V}^{\pi}\|_{\infty} = \max_{i} |\hat{V}^{\pi}(s_i) - \hat{V}^{\pi}(s_i)|$$

- We want to show that

$$\|B(V^{\pi}) - B(\hat{V}^{\pi})\|_{\infty} \leq \gamma \|V^{\pi} - \hat{V}^{\pi}\|_{\infty}$$

Contraction of Bellman Expectation

$$\|B^{\pi}(V^{\pi}) - B^{\pi}(\hat{V}^{\pi})\|_{\infty}$$

$$= \|\left(R(s_i) + \gamma \sum_{s'} P(s')V^{\pi}(s')\right) - \left(R(s_i) + \gamma \sum_{s'} P(s')\hat{V}^{\pi}(s')\right)\|_{\infty}$$

Contraction of Bellman Expectation

$$\|B^{\pi}(V^{\pi}) - B^{\pi}(\hat{V}^{\pi})\|_{\infty}$$

$$= \|\left(R(s) + \gamma \sum_{s'} P(s')V^{\pi}(s')\right) - \left(R(s) + \gamma \sum_{s'} P(s')\hat{V}^{\pi}(s')\right)\|_{\infty}$$

$$= \gamma\| \sum_{s'} P(s')\left(V^{\pi}(s') - \hat{V}^{\pi}(s')\right)\|_{\infty}$$

$$\leq \gamma \sum_{s'} P(s')\left(\|V^{\pi}(s') - \hat{V}^{\pi}(s')\|_{\infty}\right)$$

$$\leq \gamma \max_{s'}(|V^{\pi}(s') - \hat{V}^{\pi}(s')|)$$

$$= \gamma\|V^{\pi}(s') - \hat{V}^{\pi}(s')\|_{\infty}$$

Expectation is no greater than Max

Contraction of Bellman Expectation

- So we used the max norm

$$\|V^{\pi} - \hat{V}^{\pi}\|_{\infty} = \max_{i} |\, V^{\pi}(s_i) - \hat{V}^{\pi}(s_i) \,|$$

- and showed that

$$\|B(V^{\pi}) - B(\hat{V}^{\pi})\|_{\infty} \leq \gamma \|V^{\pi} - \hat{V}^{\pi}\|_{\infty}$$

Contraction with Optimality Equation

Next, we can show that under the same max norm, the optimal Bellman updates also define contraction

$$\|B(V) - B(\hat{V})\|_\infty \leq \gamma \|V - \hat{V}\|_\infty$$

Optimality Equation Case: Deal with Max

$$\|B(V) - B(\hat{V})\|_\infty$$

$$= \|\left(R(s) + \gamma \max_{a \in A(s)} \sum_i P(s_i' \,|\, s, a) V(s_i')\right) - \left(R(s) + \gamma \max_{a' \in A(s)} \sum_i P(s_i' \,|\, s, a') \hat{V}(s_i')\right)\|$$

$$= \gamma \| \max_{a \in A(s)} \sum_i P(s_i' \,|\, s, a) V(s_i') - \max_{a' \in A(s)} \sum_i P(s_i' \,|\, s, a') \hat{V}(s_i') \|$$

Optimality Equation Case: Deal with Max

$$\|B(V) - B(\hat{V})\|_\infty$$

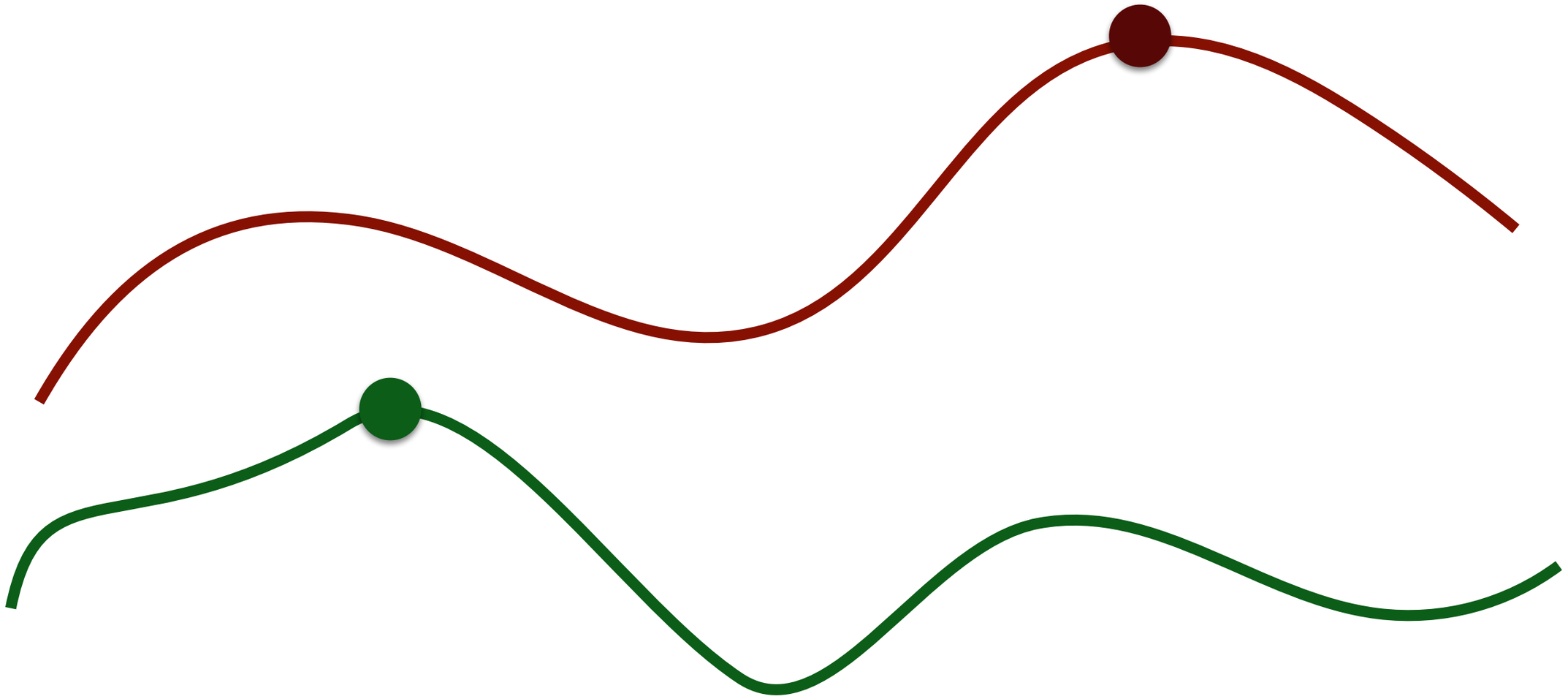
$$= \|\left(R(s) + \gamma \max_{a \in A(s)} \sum_i P(s_i' \mid s, a) V(s_i')\right) - \left(R(s) + \gamma \max_{a' \in A(s)} \sum_i P(s_i' \mid s, a') \hat{V}(s_i')\right)\|$$

$$= \gamma \| \max_{a \in A(s)} \sum_i P(s_i' \mid s, a) V(s_i') - \max_{a' \in A(s)} \sum_i P(s_i' \mid s, a') \hat{V}(s_i') \|$$

What to do with these two different max?

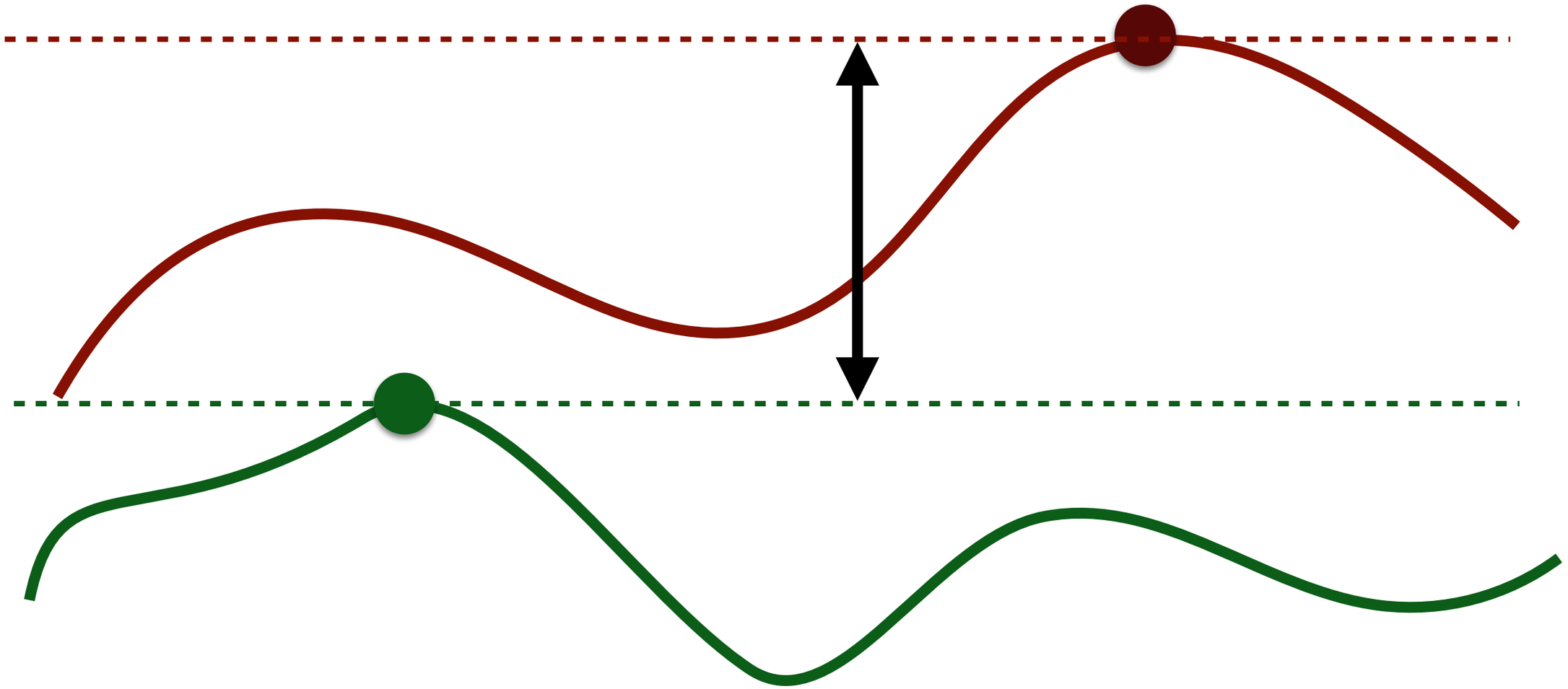
Important Fact

$$|\max_x f(x) - \max_y g(y)| \leq \max_x |f(x) - g(x)|$$



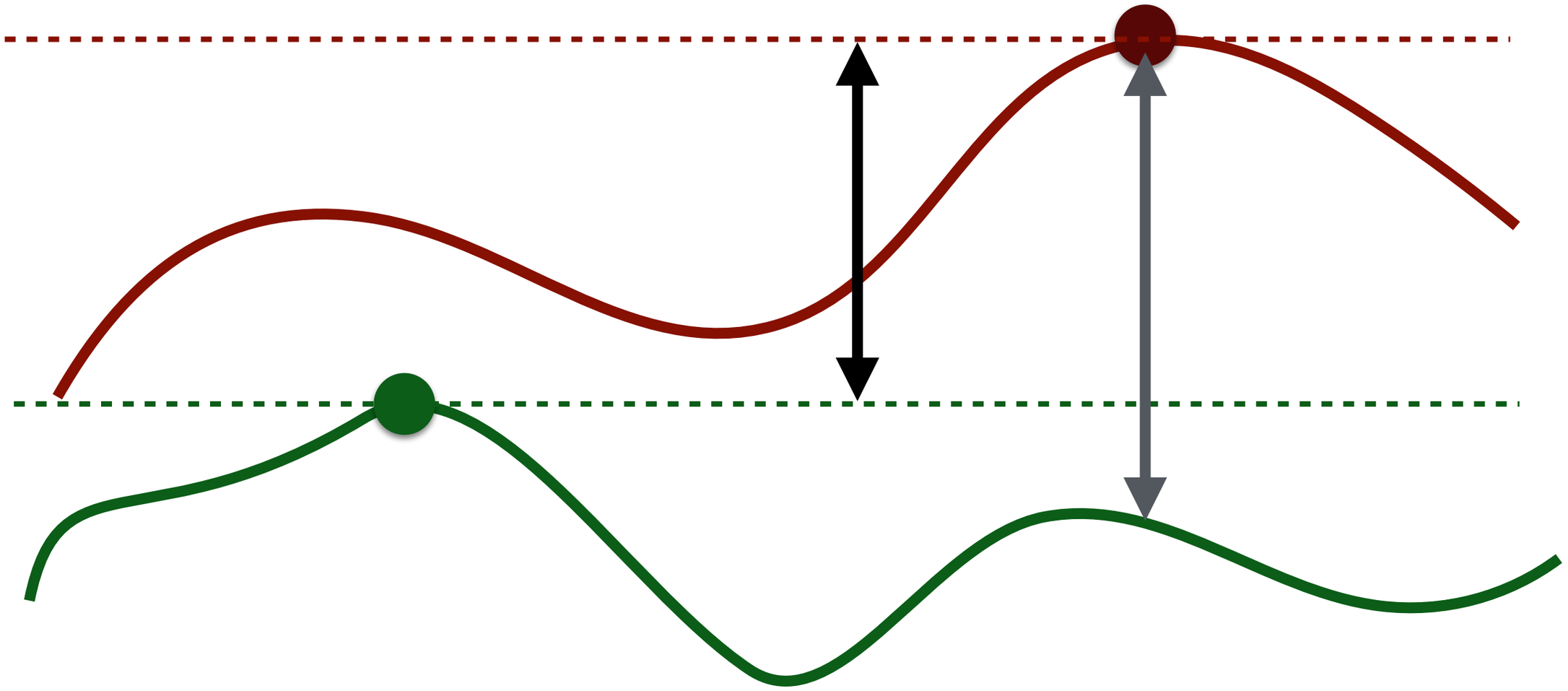
Important Fact

$$|\max_x f(x) - \max_y g(y)| \leq \max_x |f(x) - g(x)|$$



Important Fact

$$|\max_x f(x) - \max_y g(y)| \leq \max_x |f(x) - g(x)|$$



Optimality Equation Case: Deal with Max

$$\|B(V) - B(\hat{V})\|_\infty$$

$$= \|\left(R(s) + \gamma \max_{a \in A(s)} \sum_i P(s_i' \,|\, s, a) V(s_i')\right) - \left(R(s) + \gamma \max_{a' \in A(s)} \sum_i P(s_i' \,|\, s, a') \hat{V}(s_i')\right)\|$$

$$= \gamma \| \max_{a \in A(s)} \sum_i P(s_i' \,|\, s, a) V(s_i') - \max_{a' \in A(s)} \sum_i P(s_i' \,|\, s, a') \hat{V}(s_i') \|$$

$$\leq \gamma \max_a \| \sum_i P(s_i' \,|\, s, a) V(s_i') - \sum_i P(s_i' \,|\, s, a) \hat{V}(s_i') \|$$

$$= \gamma \max_a \| \sum_i P(s_i' \,|\, s, a) \left(V(s_i') - \hat{V}(s_i') \right) \|$$

Optimality Equation Case: Deal with Max

$$\|B(V) - B(\hat{V})\|_\infty$$

$$= \|\left(R(s) + \gamma \max_{a \in A(s)} \sum_i P(s_i' \,|\, s, a) V(s_i')\right) - \left(R(s) + \gamma \max_{a' \in A(s)} \sum_i P(s_i' \,|\, s, a') \hat{V}(s_i')\right)\|$$

$$= \gamma \| \max_{a \in A(s)} \sum_i P(s_i' \,|\, s, a) V(s_i') - \max_{a' \in A(s)} \sum_i P(s_i' \,|\, s, a') \hat{V}(s_i') \|$$

$$\leq \gamma \max_a \| \sum_i P(s_i' \,|\, s, a) V(s_i') - \sum_i P(s_i' \,|\, s, a) \hat{V}(s_i') \|$$

$$= \gamma \max_a \| \sum_i P(s_i' \,|\, s, a) \left(V(s_i') - \hat{V}(s_i') \right) \| \leq \gamma \| V(s_i') - \hat{V}(s_i') \|$$

Consequently:

Theorem:

The Most Naive Algorithm works.

Proof: Bellman Equations define contraction mappings.

$$||B(\mathbf{V}) - B(\mathbf{V}')||_\infty \leq \gamma ||\mathbf{V} - \mathbf{V}'||_\infty$$

Note that $\gamma < 1$.

Value Iteration

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

repeat
  $U \leftarrow U'$; $\delta \leftarrow 0$
  for each state s **in** S **do**
    $U'[s] \leftarrow R(s) \; + \; \gamma \; \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \; U[s']$
    if $|U'[s] \; - \; U[s]| \; > \; \delta$ **then** $\delta \leftarrow |U'[s] \; - \; U[s]|$
until $\delta \; < \; \epsilon(1 - \gamma)/\gamma$
return U

Algorithm 17.4, AIMA Ch17

Value Iteration

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

repeat
  $U \leftarrow U'$; $\delta \leftarrow 0$
  for each state s **in** S **do**
    $U'[s] \leftarrow R(s) \; + \; \gamma \; \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \; U[s']$ **“Bellman Update”**
    if $|U'[s] \; - \; U[s]| \; > \; \delta$ **then** $\delta \leftarrow |U'[s] \; - \; U[s]|$
until $\delta \; < \; \epsilon(1-\gamma)/\gamma$
return U

Algorithm 17.4, AIMA Ch17

Value Iteration

repeat

$U \leftarrow U'$; $\delta \leftarrow 0$

for each state s **in** S **do**

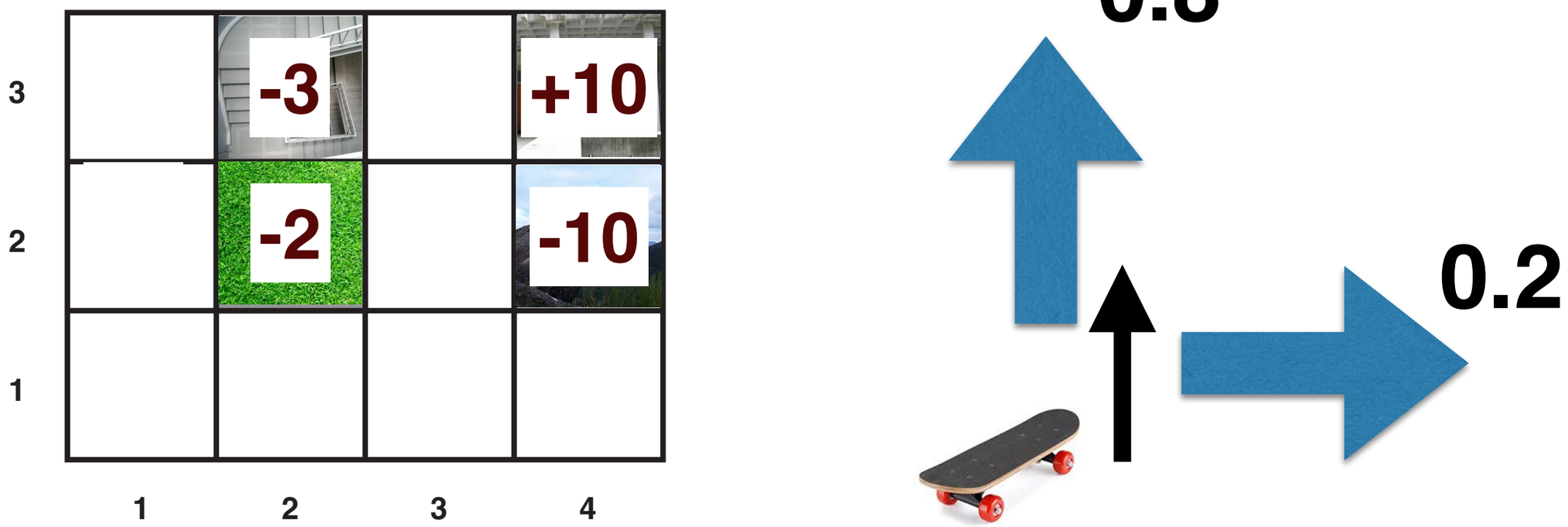
$$U'[s] \leftarrow R(s) \; + \; \gamma \; \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \; U[s']$$

if $|U'[s] \; - \; U[s]| \; > \; \delta$ **then** $\delta \leftarrow |U'[s] \; - \; U[s]|$

Remember:

There is only one optimal value function.

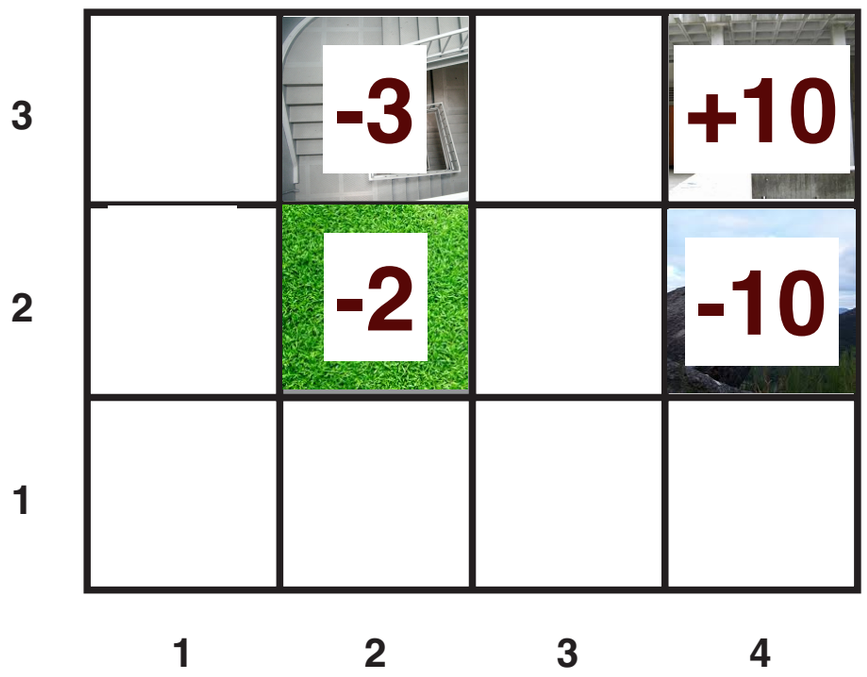
The optimal policy is defined by choosing actions that give the highest expected values.

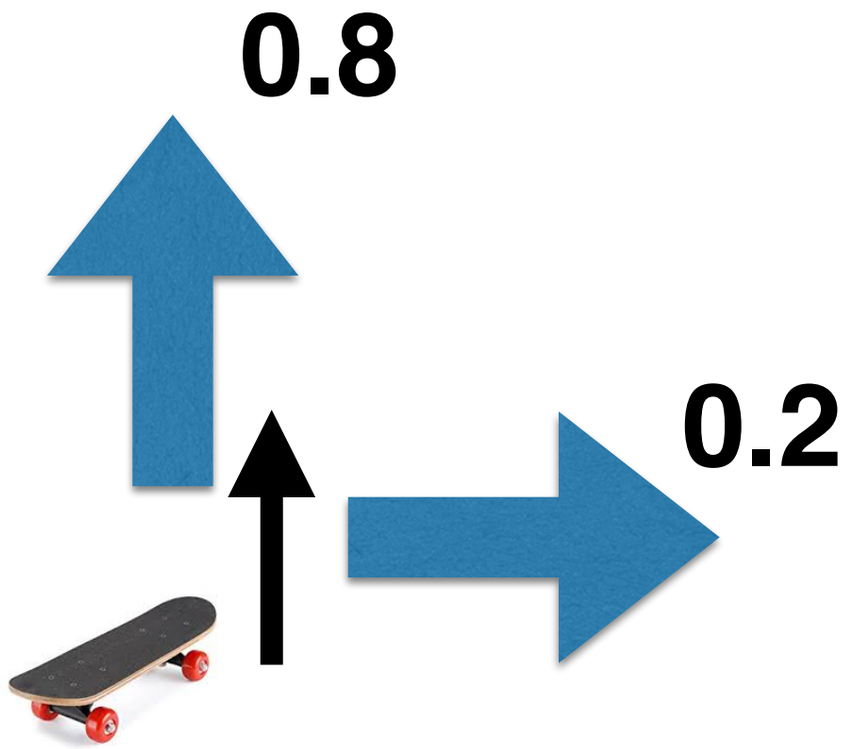


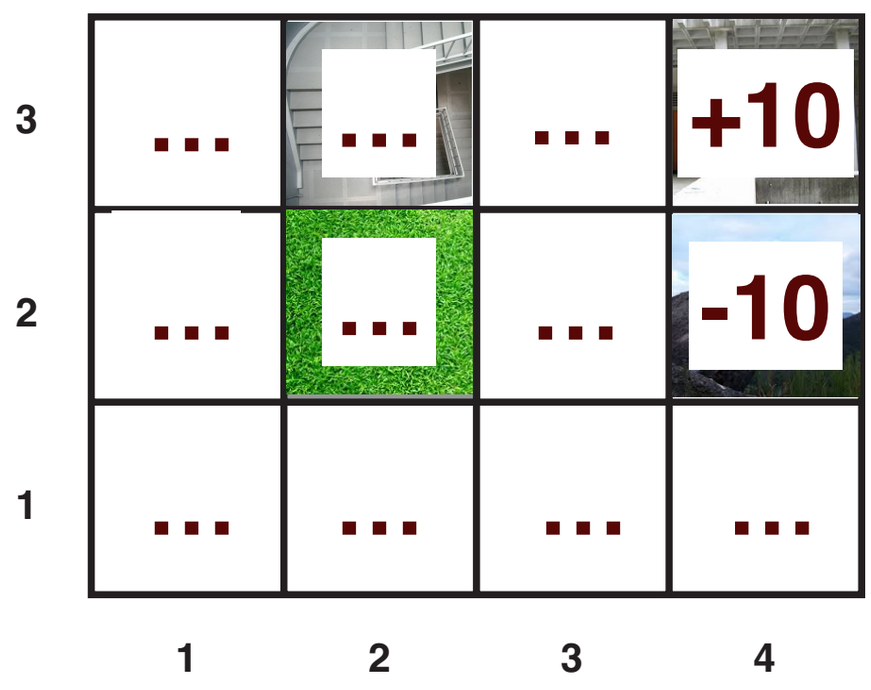
Suppose we define the transition model to be:

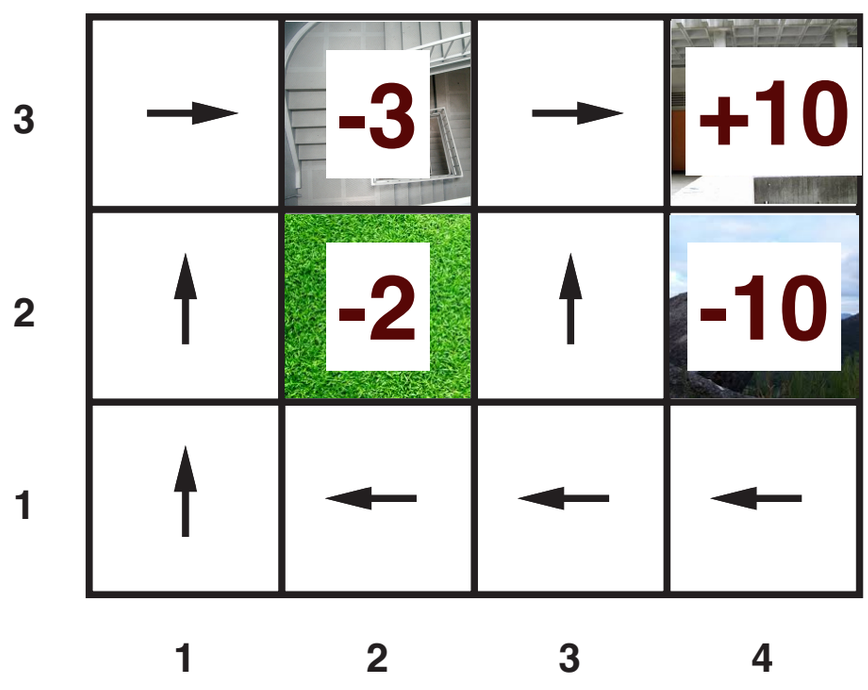
For each direction we intend to go, there is a 0.2 chance of slipping clockwise.

Try this!!!

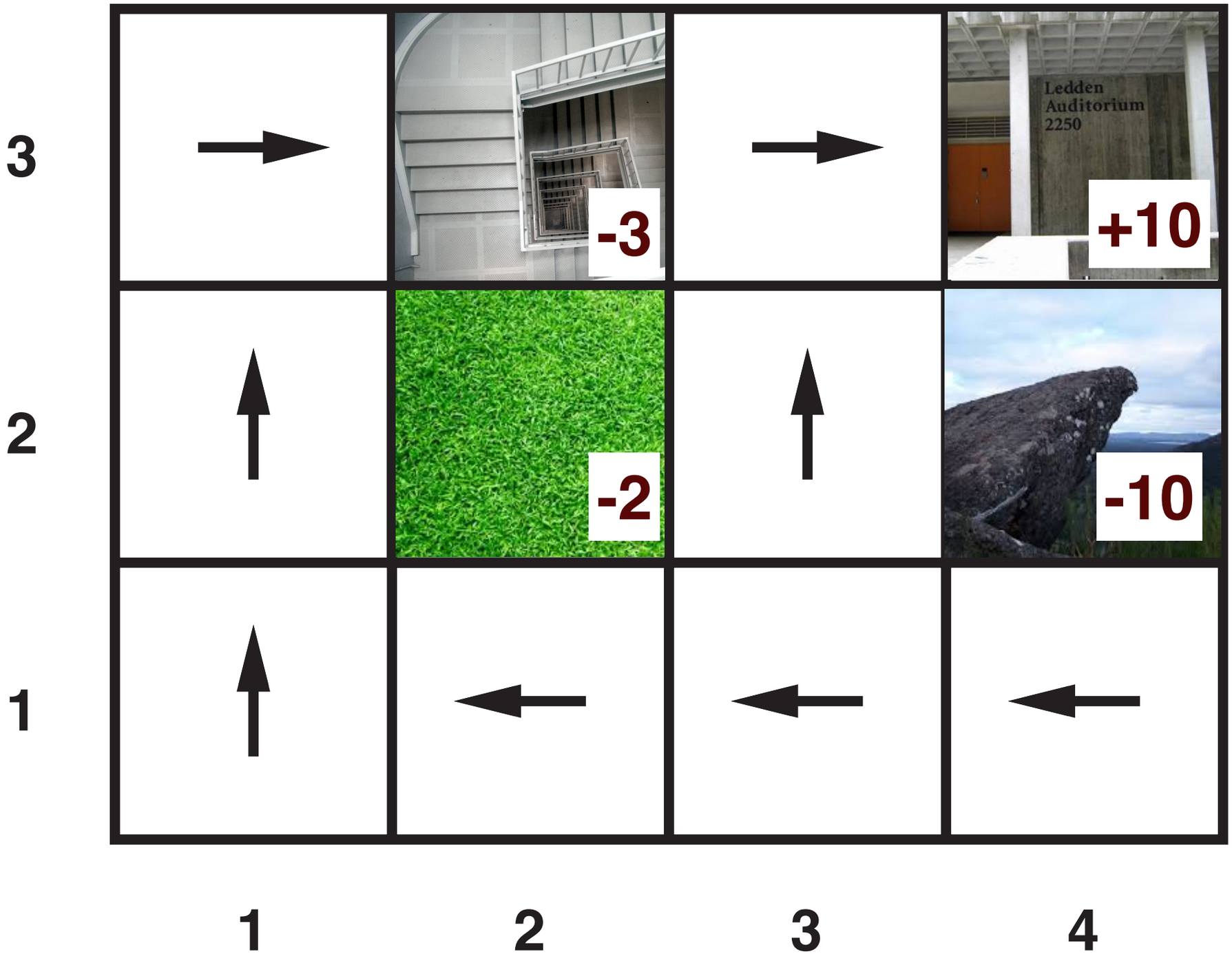

3
2
1
-3
+10
-2
-10
1
2
3
4


0.8
0.2


3
2
1
...
-10
+10
1
2
3
4


3
2
1
-3
-2
-10
+10
1
2
3
4

Started with some policy, want to improve



Policy Evaluation

The first thing we need to find out is the value function defined by a policy π

$$V(s_0) = R(s_0) + \gamma \max_{a \in A(s_0)} \sum_{s'} P(s'|s,a)V(s')$$

$$V(s_1) = R(s_1) + \gamma \max_{a \in A(s_1)} \sum_{s'} P(s'|s,a)V(s')$$

$$\dots$$

$$V(s_n) = R(s_n) + \gamma \max_{a \in A(s_n)} \sum_{s'} P(s'|s,a)V(s')$$

Policy Evaluation

The first thing we need to find out is the value function defined by a policy π

$$V(s_0) = R(s_0) + \gamma \sum_{s'} P(s'|s_0, \pi(s_0))V(s')$$

$$V(s_1) = R(s_1) + \gamma \sum_{s'} P(s'|s_1, \pi(s_1))V(s')$$

$$\bullet \bullet \bullet$$

$$V(s_n) = R(s_n) + \gamma \sum_{s'} P(s'|s_n, \pi(s_n))V(s')$$

Policy Evaluation: Simple!

All these equations are linear now!

$$V(s_0) = R(s_0) + \gamma \sum_{s'} P(s'|s_0, \pi(s_0))V(s')$$

$$V(s_1) = R(s_1) + \gamma \sum_{s'} P(s'|s_1, \pi(s_1))V(s')$$

$$\bullet \bullet \bullet$$

$$V(s_n) = R(s_n) + \gamma \sum_{s'} P(s'|s_n, \pi(s_n))V(s')$$

Policy Iteration:

First solve the equations to evaluate the values defined by the current policy,

and then check each state to see if you can improve the max value.

Policy Iteration:

function POLICY-ITERATION(mdp) **returns** a policy

  repeat

    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)

    $unchanged? \leftarrow$ true

    for each state s **in** S **do**

      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**

        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$

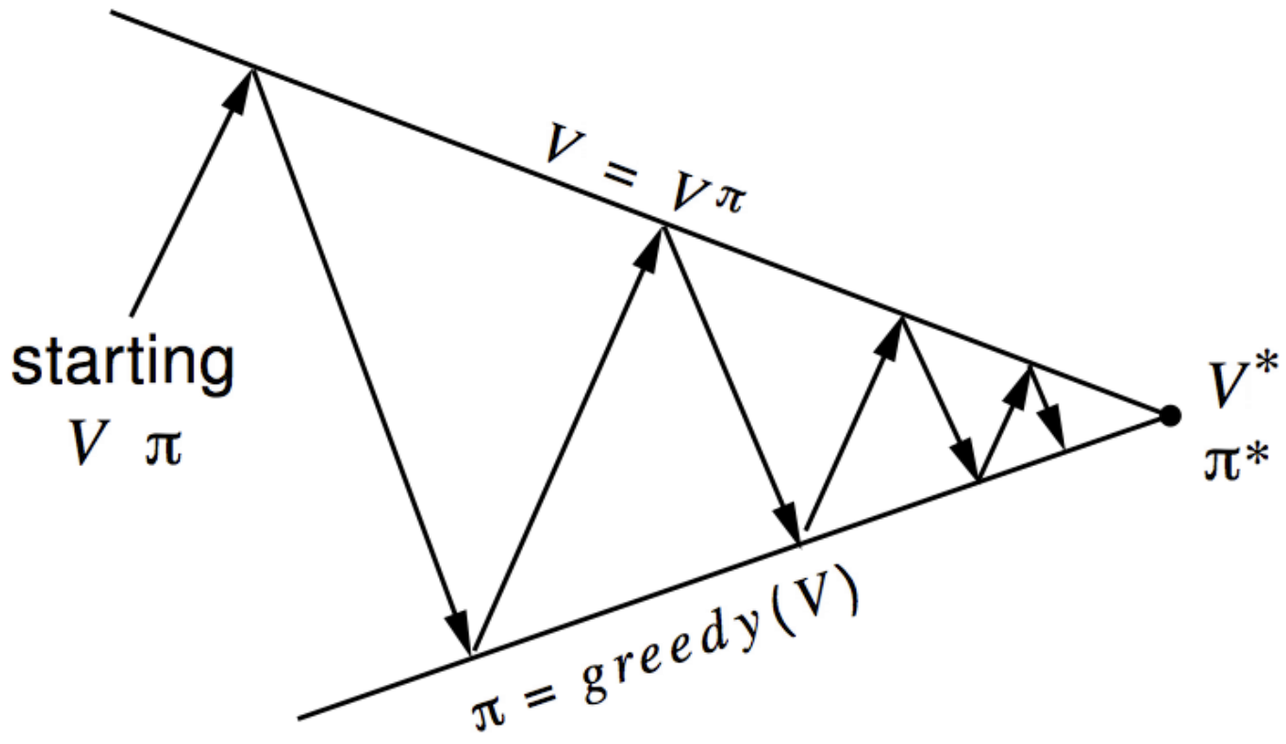
        $unchanged? \leftarrow$ false

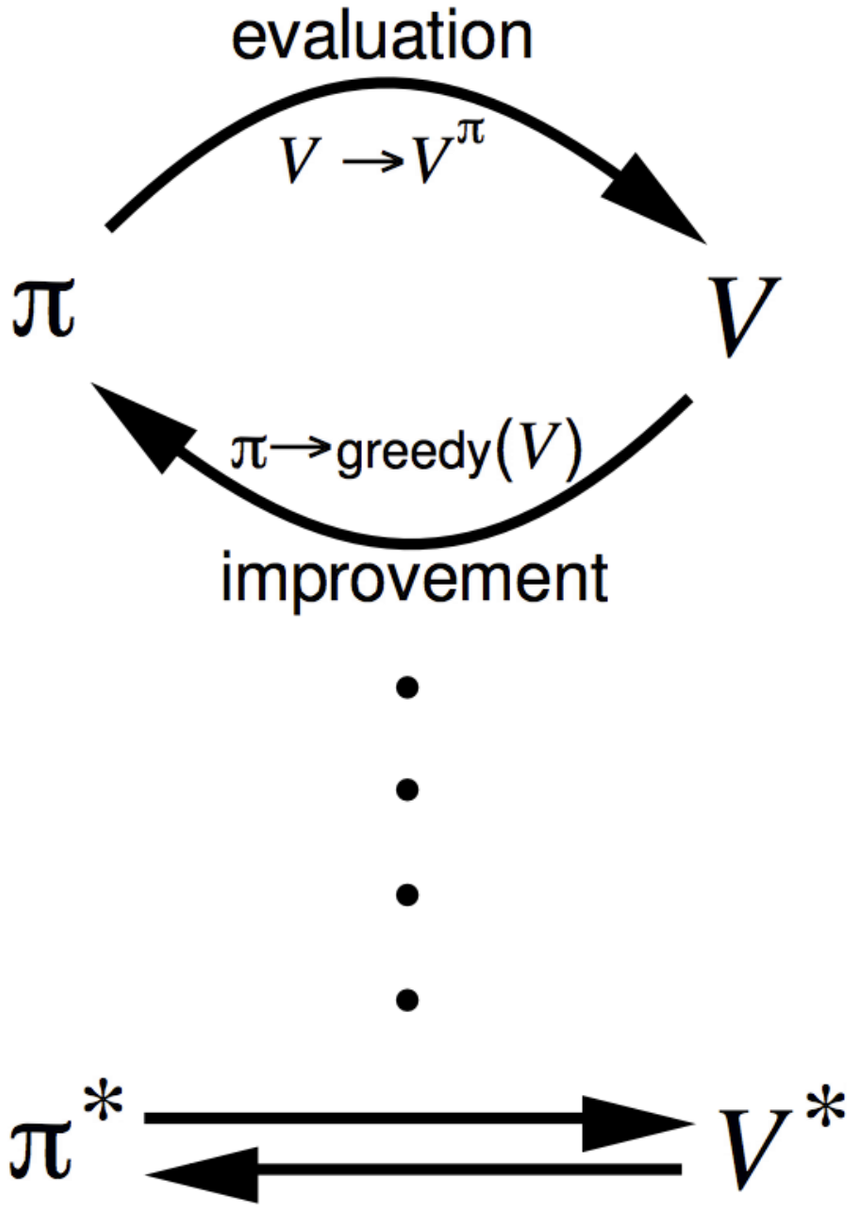
  until $unchanged?$

  return π

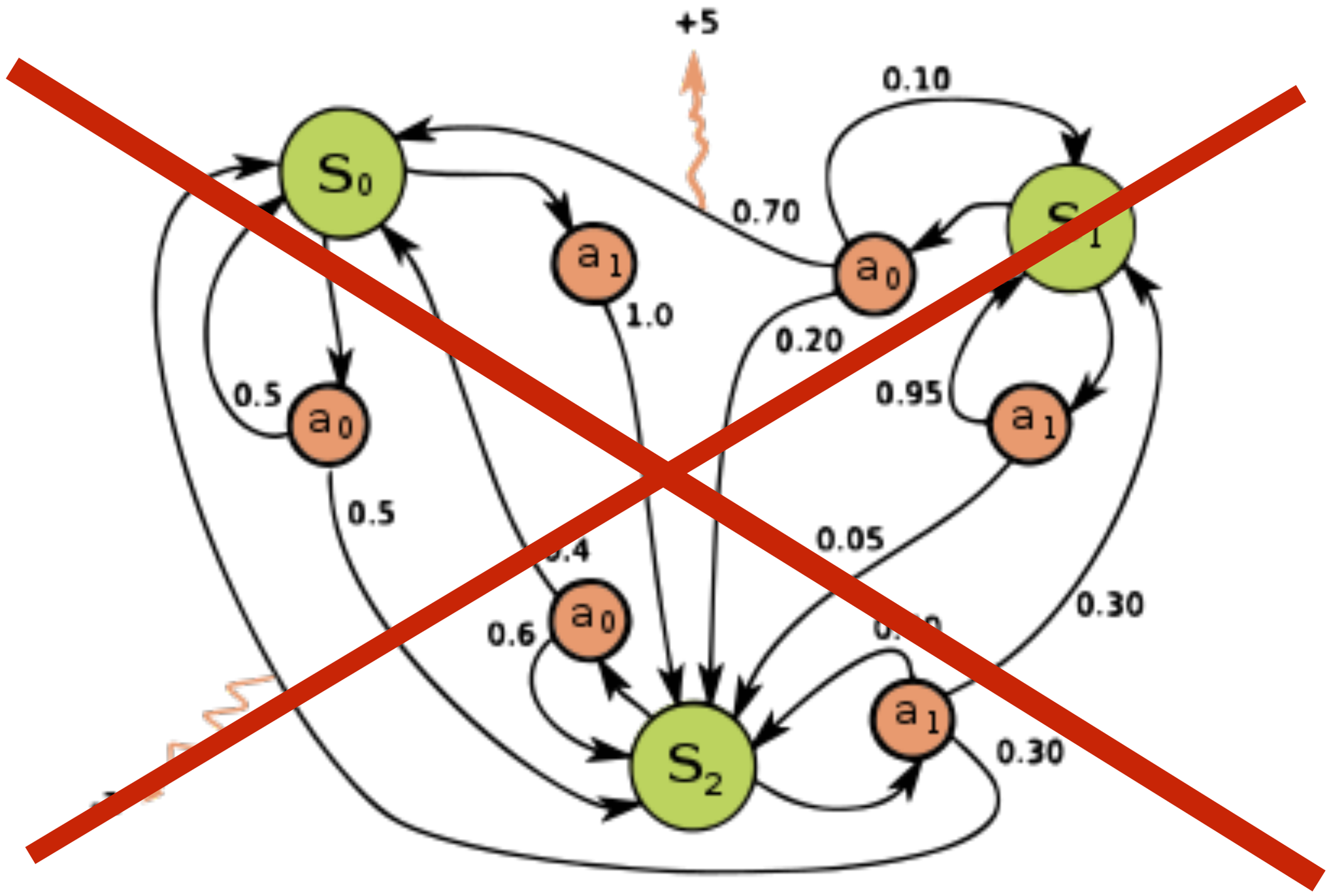
Algorithm 17.7, AIMA Ch17

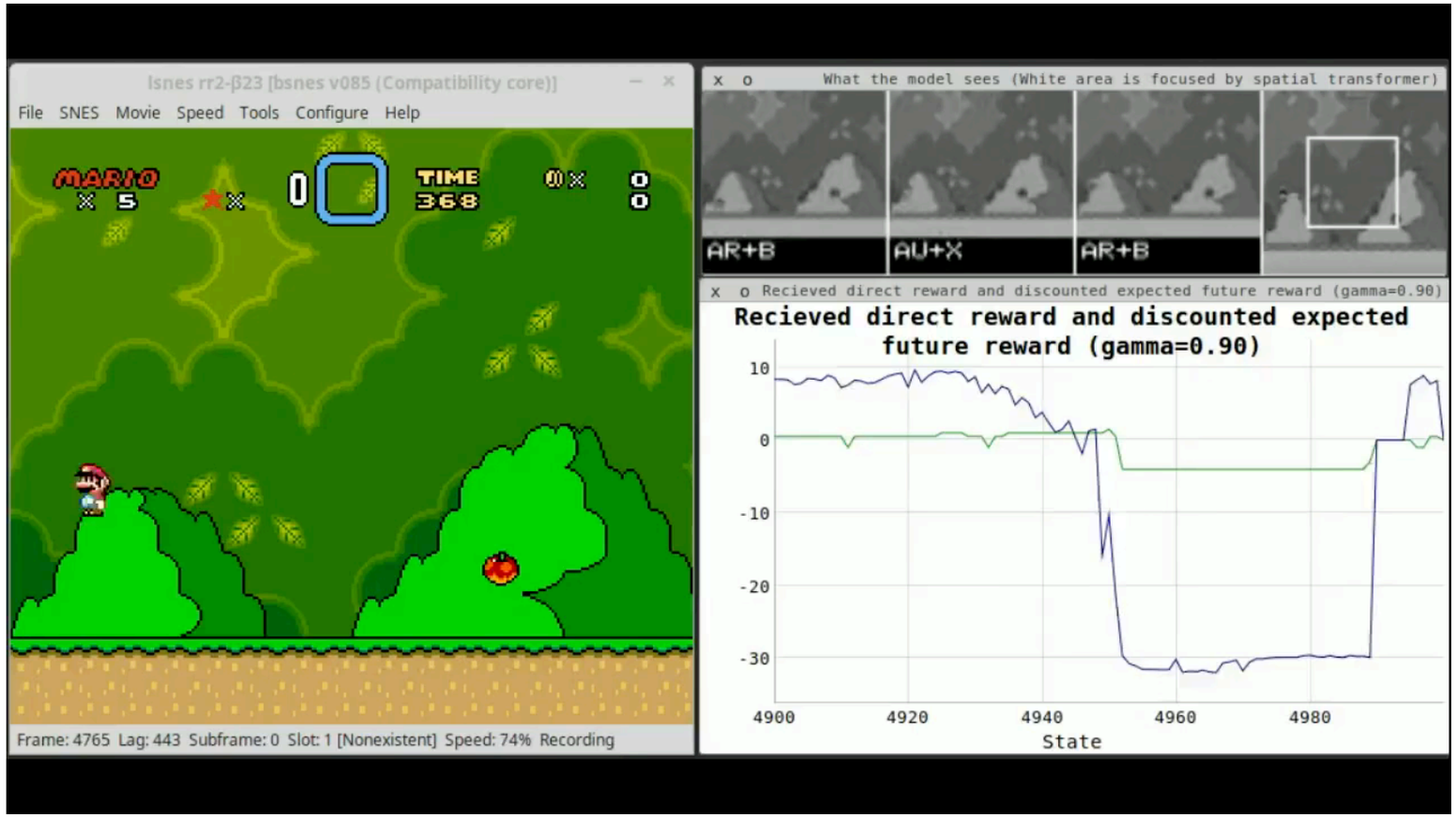
Values and Policies

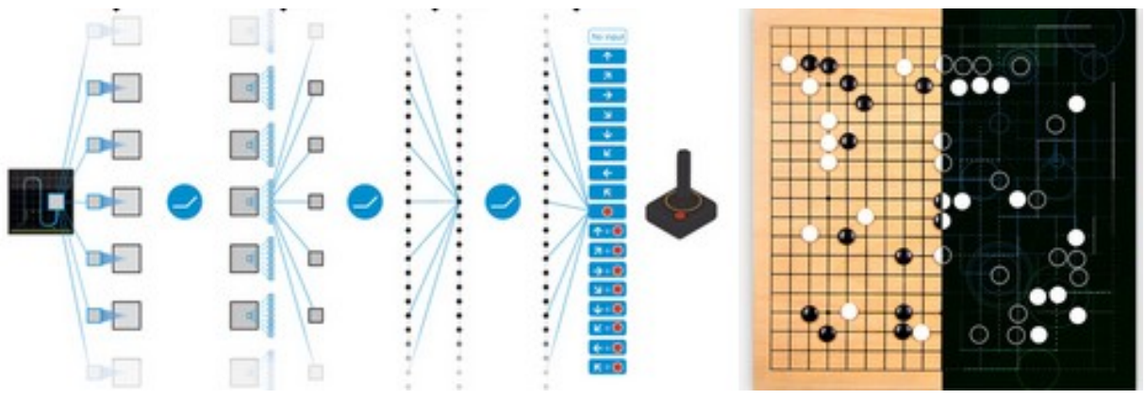


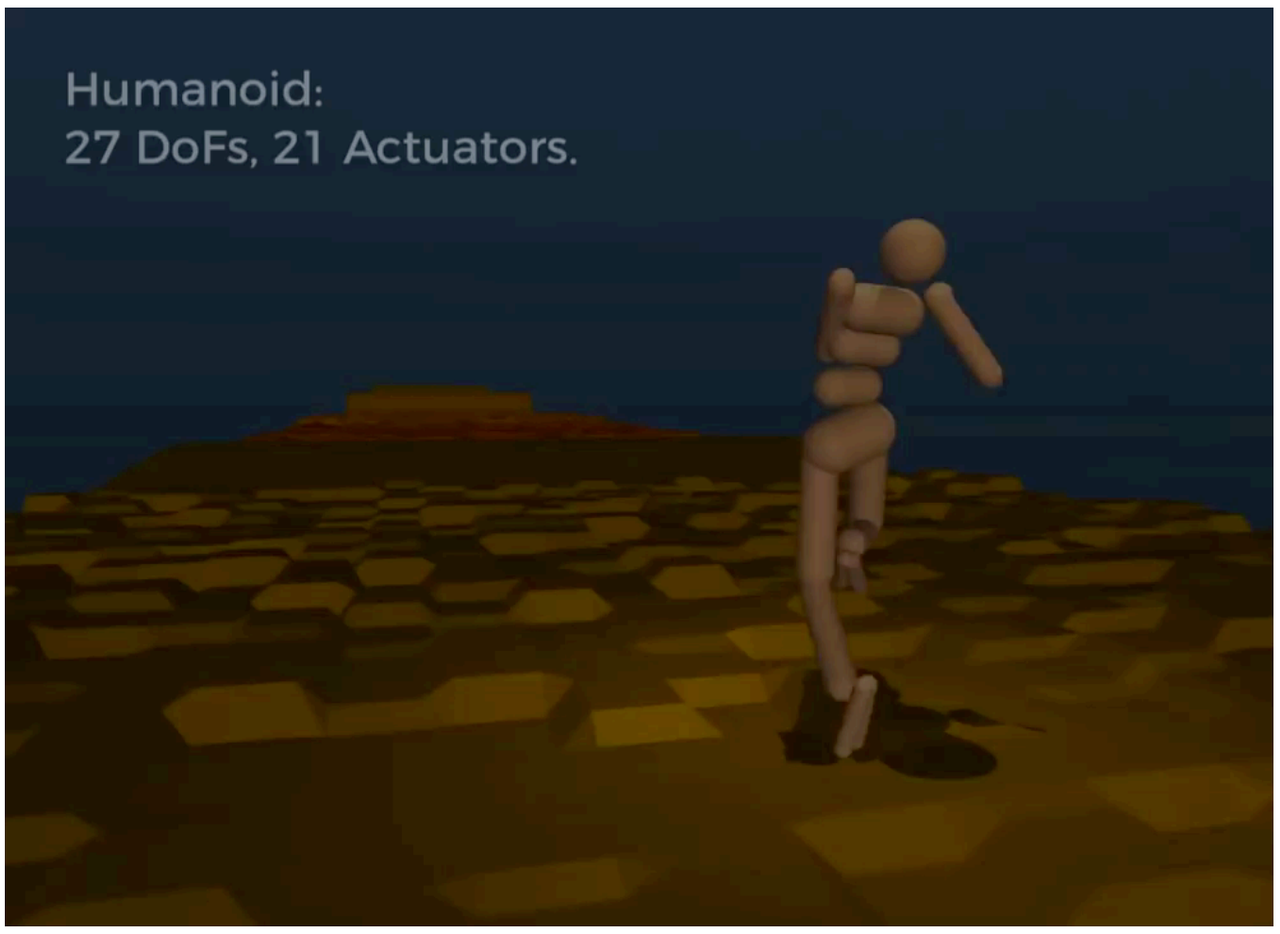


What if we don't know the MDP?




Isnes rr2-β23 [bsnes v085 (Compatibility core)]
File SNES Movie Speed Tools Configure Help
MARIO
x 5
TIME
368
Frame: 4765 Lag: 443 Subframe: 0 Slot: 1 [Nonexistent] Speed: 74% Recording
What the model sees (White area is focused by spatial transformer)
AR+B
AU+X
AR+B
Recieved direct reward and discounted expected future reward (gamma=0.90)
Recieved direct reward and discounted expected future reward (gamma=0.90)
10
0
-10
-20
-30
4900
4920
4940
4960
4980
State




Humanoid:
27 DoFs, 21 Actuators.

Learning Evaluations and Policies

- Evaluation/Prediction:

Can we estimate utility/value of a given policy without knowing the transition model of the environment?

- Optimization/Control:

Can we improve or optimize a given policy without the transition model?

Two Main Approaches

- Monte Carlo: Simulate a lot of sequences, and use the average

- Temporal Difference: Utilize the Bellman Equations and update on the way, without waiting for full sequences

- On-Policy vs Off-Policy

Monte Carlo Methods: Sample and Average

- Policy: $\pi(s_i) = a_i$
- Trajectories: $s_0, s_1, \ldots, s_T$
- Rewards: $R(s_0) + \gamma R(s_1) + \cdots + \gamma^T R(s_T)$

- Follow policy, and get a lot of samples of future trajectories, and estimate the expectation

$$V^{\pi}(s_0) = \mathbb{E}_{\pi}[\sum_{i=0}^{T} \gamma^i R(s_i)]$$

Monte Carlo Policy Evaluation
(Direct Utility Estimation)

- How do we estimate expected utility under a fixed policy? Simply run a lot of samples, and use the average!

- The sample mean converges to the expectation

$$V^{\pi}(s_0) = \mathbb{E}_{\pi}[\sum_{i=0}^{T} \gamma^i R(s_i)]$$

Monte Carlo Policy Evaluation
(Direct Utility Estimation)

- Reward-to-go in a trajectory $s_0, s_1, \ldots, s_T$

$$G(s_k) = \sum_{i=k}^{T} \gamma^{i-k} R(s_i)$$

- Remember: value is just expected reward-to-go

$$V^{\pi}(s_0) = \mathbb{E}_{\pi}[\sum_{i=0}^{T} \gamma^{i} R(s_i)] = \mathbb{E}_{\pi}[G(s_0)]$$

Monte Carlo Policy Evaluation
(Direct Utility Estimation)

- For any simulation/episode (terminating at T):

$$s_0, s_1, \ldots, s_T$$

- Collect reward to go at each state you see:

$$G(s_k) = \sum_{i=k}^{T} \gamma^{i-k} R(s_i)$$

Monte Carlo Policy Evaluation
(Direct Utility Estimation)

- Start at some state.
- Take actions following policy.
- Follow until termination (an episode).
- Compute average reward-to-go of each state from the episode (total return from the first occurrence of a state to the termination of an episode).
- Repeat this until time is up or the values stabilize.

Monte Carlo Policy Evaluation
(Direct Utility Estimation)

```
def MC-Policy-Evaluation(policy, states, R, gamma):
    for s in states:
        #G is a dict from state to sequence of reward values
        #initialize it to be empty
        G[s] = []
        #U(s) is the expected reward we want to approximate
        U[s] = 0
    while within computation budget:
        s = arbitrary state
        #An episode is simply a sequence of states
        #Obtained by simulating from s0 to termination
        episode = simulation_sequence(s, policy)
        for s in episode:
            #G(s) is a list of reward-to-go for the state
            G[s].append(reward_to_go(s))
            U[s] = average(G[s])
```

MC Policy Evaluation: Drawbacks

- We **have to wait** till sequences of actions terminate to calculate the average rewards.

- Not taking advantage of the forms of the **Bellman equations**. Missing a lot of opportunities for updating the values on-the-fly.

Temporal-Difference (TD) Prediction

We want to approximate this expected value

$$V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{i=0}^{\infty} \gamma^{i} R(s_i) \,|\, s_0 = s]$$

It can be turned into the recurrence:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R(s) + \gamma V^{\pi}(s')]$$

How to use this fact?

Updating with Incremental Average

The average of a sequence of values can be computed incrementally:

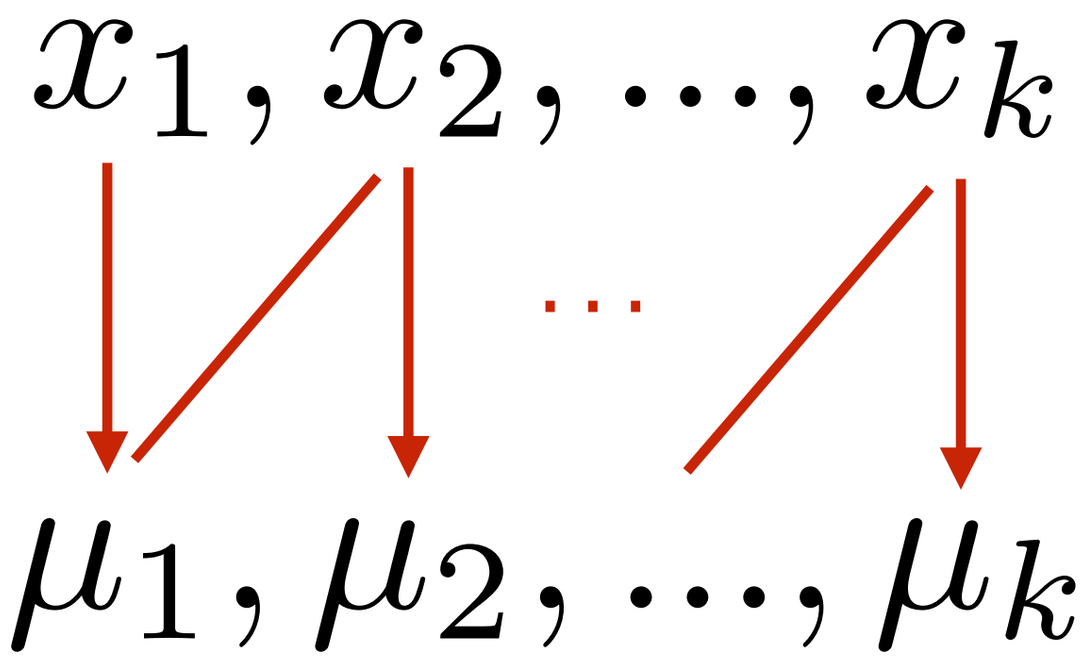
$$x_1, x_2, \ldots, x_k$$

$$\mu_1, \mu_2, \ldots, \mu_k$$

$$\mu_k = \frac{1}{k} \sum_{i=1}^{k} x_i$$

How do we update with new values?

$$\mu_k = \frac{1}{k} \sum_{i=1}^{k} x_i$$



Updating with Incremental Average

$$\mu_k = \frac{1}{k}\sum_{i=1}^{k} x_k$$

$x_1, x_2, ..., x_k$

$$= \frac{1}{k}(x_k + \sum_{i=1}^{k-1} x_i)$$

$\mu_1, \mu_2, ..., \mu_k$

$$= (1 - \frac{1}{k})\mu_{k-1} + \frac{1}{k}x_k$$

$$= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

Updating with Incremental Average

$$\mu_k = \frac{1}{k}\sum_{i=1}^{k} x_k$$

$$= \frac{1}{k}(x_k + \sum_{i=1}^{k-1} x_i)$$

$$= (1 - \frac{1}{k})\mu_{k-1} + \frac{1}{k}x_k$$

$$= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

$$\mu_k \leftarrow (1 - \alpha_k)\mu_{k-1} + \alpha_k x_k$$

"Interpolation"

Updating with Incremental Average

$$\mu_k = \frac{1}{k}\sum_{i=1}^{k} x_k$$

$$= \frac{1}{k}(x_k + \sum_{i=1}^{k-1} x_i)$$

$$= (1 - \frac{1}{k})\mu_{k-1} + \frac{1}{k}x_k$$

$$= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

$$\mu_k \leftarrow \mu_{k-1} + \alpha_k(x_k - \mu_{k-1})$$

"Gradient Descent"

Updating with Incremental Average

$$\mu_k = \frac{1}{k}\sum_{i=1}^{k} x_k$$

$$= \frac{1}{k}(x_k + \sum_{i=1}^{k-1} x_i)$$

$$= (1 - \frac{1}{k})\mu_{k-1} + \frac{1}{k}x_k$$

$$= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

Learning Rate:

Control how important the new values are

$$\mu_k \leftarrow (1 - \alpha_k)\mu_{k-1} + \alpha_k x_k$$

"Interpolation"

$$\mu_k \leftarrow \mu_{k-1} + \alpha_k(x_k - \mu_{k-1})$$

"Gradient Descent"

Temporal-Difference (TD) Prediction

$$\mu_k \leftarrow \mu_{k-1} + \alpha_k(x_k - \mu_{k-1})$$

- A key idea in reinforcement learning.

- In every step, update estimated values based on next sampled state without waiting for a final outcome.

Temporal-Difference (TD) Prediction

$$\mu_k \leftarrow \mu_{k-1} + \alpha_k(x_k - \mu_{k-1})$$

- Monte Carlo prediction can be done in this way as well:

$$V^{\pi}(s) = \mathbb{E}_{s\sim\pi}[G(s)]$$

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha\Big(G(s) - V^{\pi}(s)\Big)$$

Temporal-Difference (TD) Prediction

$$V^{\pi}(s) = \mathbb{E}_{s\sim\pi}[G(s)]$$

$$= \mathbb{E}_{s\sim\pi}[R(s) + \gamma \cdot G(s')]$$

$$= \mathbb{E}_{s\sim\pi}[R(s) + \gamma \cdot V(s')]$$

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(? - V^{\pi}(s))$$

Temporal-Difference (TD) Prediction

Thus, to compute this value

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R(s) + \gamma V^{\pi}(s')]$$

We only need to compute the average of:

$$R(s) + \gamma V^{\pi}(s')$$

By updating with just one sample each time:

$$\mu_k \leftarrow \mu_{k-1} + \alpha_k(x_k - \mu_{k-1})$$

Temporal-Difference (TD) Prediction

Thus, to compute this value

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R(s) + \gamma V^{\pi}(s')]$$

We only need to compute the average of:

$$R(s) + \gamma V^{\pi}(s')$$

By updating with just one sample each time:

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(R(s) + \gamma V^{\pi}(s') - V^{\pi}(s))$$

$$\mu_k \leftarrow \mu_{k-1} + \alpha_k(x_k - \mu_{k-1})$$

Temporal-Difference (TD) Prediction

Thus, to compute this value

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R(s) + \gamma V^{\pi}(s')]$$

We only need to compute the average of:

$$R(s) + \gamma V^{\pi}(s')$$

By updating with just one sample each time:

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(R(s) + \gamma V^{\pi}(s') - V^{\pi}(s))$$

$$\mu_k \leftarrow \mu_{k-1} + \alpha_k(x_k - \mu_{k-1})$$

Temporal-Difference (TD) Prediction

Thus, to compute this value

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R(s) + \gamma V^{\pi}(s')]$$

We only need to compute the average of:

$$R(s) + \gamma V^{\pi}(s')$$

By updating with just one sample each time:

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(\underline{R(s) + \gamma V^{\pi}(s')} - V^{\pi}(s))$$

New Sample

Temporal-Difference (TD) Prediction

Thus, to compute this value

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R(s) + \gamma V^{\pi}(s')]$$

We only need to compute the average of:

$$R(s) + \gamma V^{\pi}(s')$$

By updating with just one sample each time:

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(R(s) + \gamma V^{\pi}(s') - V^{\pi}(s))$$

Old Value **New Sample** **Old Value**

Temporal-Difference (TD) Prediction

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(R(s) + \gamma V^{\pi}(s') - V^{\pi}(s))$$

Old Value **New Sample** **Old Value**

The learning rate alpha should decrease with the number of visits to a state. Formally, to ensure convergence they should satisfy Robbins-Monro conditions:

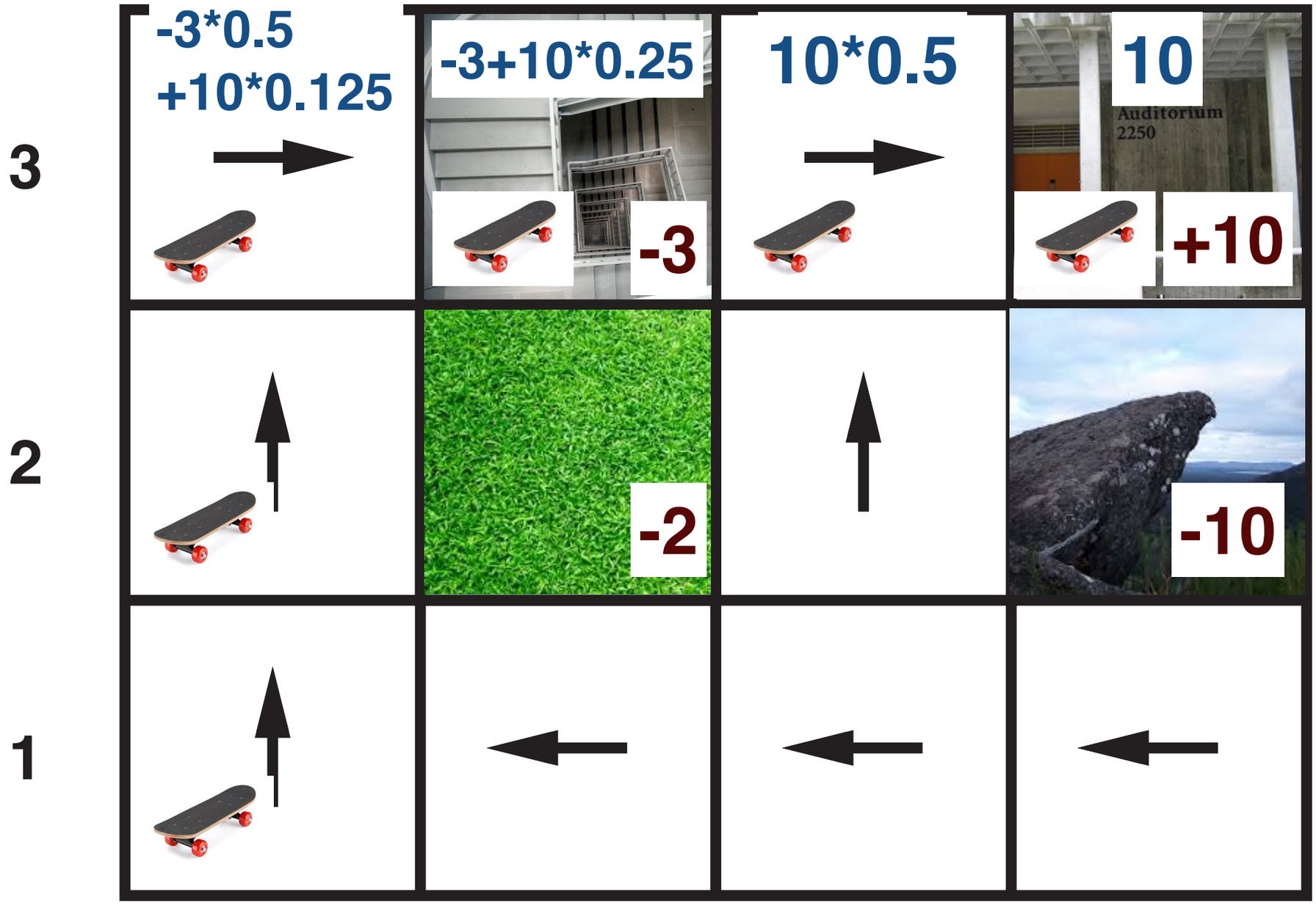
$$\sum_{N(s)=0}^{\infty} \alpha_{N(s)} \rightarrow \infty \quad \text{and} \quad \sum_{N(s)=0}^{\infty} \alpha_{N(s)}^2 < \infty$$

where $N(s)$ is the number of visits to state s.

Temporal-Difference (TD) Prediction

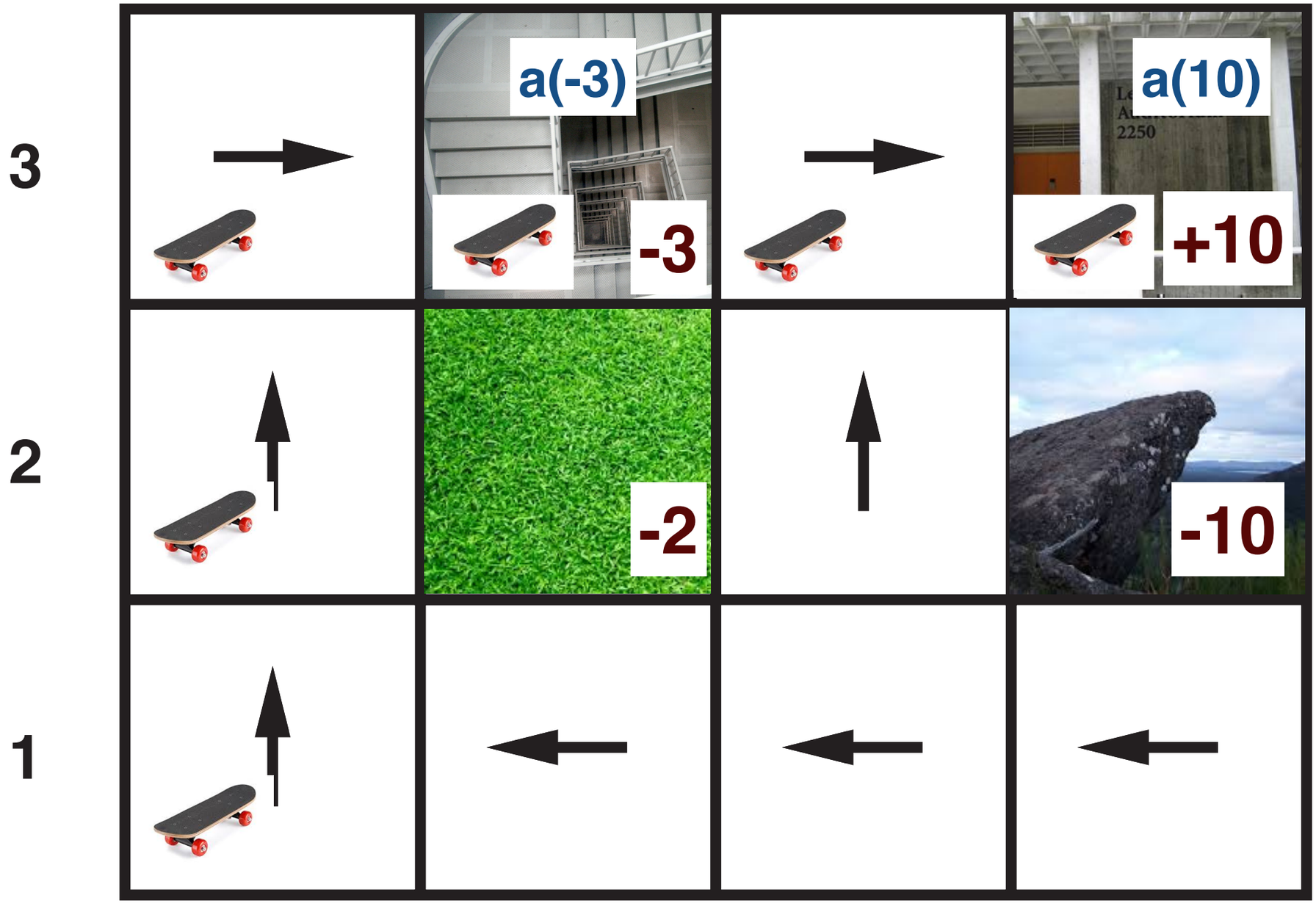
```
def TD-Policy-Evaluation(policy, states, R, gamma, alpha):
    for s in states:
        #U(s) is the expected reward we want to approximate
        U(s) = 0
    while within computation budget:
        s = arbitrary state
        #An episode is simply a sequence of states
        #Obtained by simulating from s0 to termination
        while s is not NULL:
            #when next_s is NULL, use zero for all values
            next_s = simulate_one_step(s, policy)
            U(s) = U(s) + alpha*(R(s)+gamma*U(next_s)-U(s))
            s = next_s
```

MC Policy Evaluation



TD Policy Evaluation

No immediate backup! Wait till next run!



Summary: Prediction/Evaluation

The key idea is (simply) about estimating expectations by averaging samples.

(MC) $$V^{\pi}(s_0) = \mathbb{E}_{\pi}[\sum_{i=0}^{T} \gamma^i R(s_i)]$$

(TD) $$V^{\pi}(s_0) = \mathbb{E}_{\pi}[R(s_0) + \gamma V^{\pi}(s_1)]$$

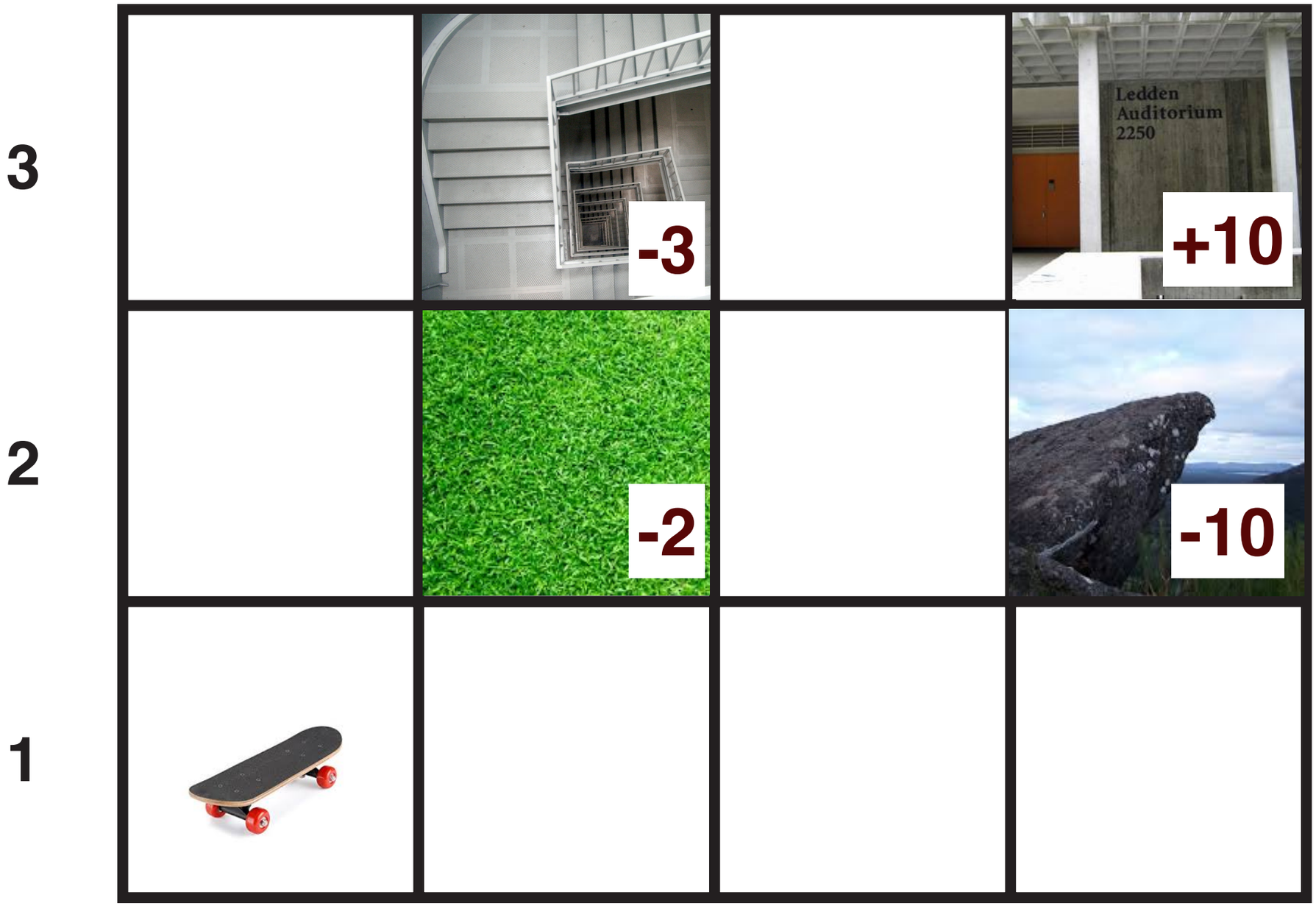
From Estimation to Control

- The previous algorithms are about evaluating the value of states based on a fixed policy

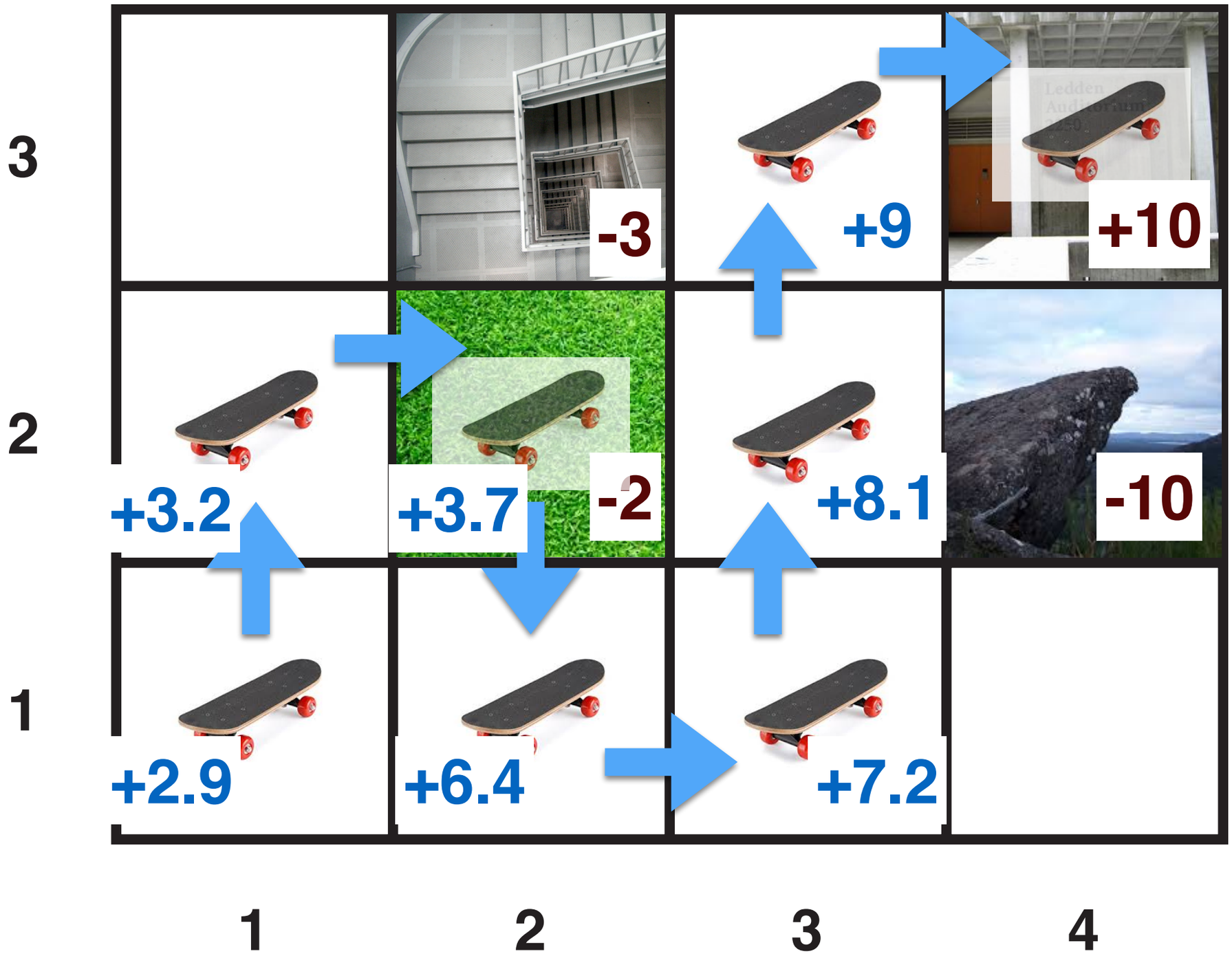
- The more important problem is how to learn about the environment and improve policy at the same time

- Q-learning: off-policy TD learning

Try some actions, and observe reward-to-go



Try some actions, and observe reward-to-go



Discount factor = 0.9

Try some actions, and observe reward-to-go

What do we do after collecting some rewards?

1. Should we start formulating a policy that always follows the best values?

2. How do we estimate values when the policy may keep changing?

Exploration vs. Exploitation

- If we just eagerly follow the seemingly best actions from the beginning, we will be stuck with some suboptimal moves.
- At the same time, we do want to follow the most promising actions eventually

Epsilon-Greedy Policy

- Simple idea: with a small probability *epsilon*, explore some other actions; with *(1-epsilon)*, follow the best action so far

Try some actions, and observe reward-to-go

- What do we do after collecting some rewards?

1. Should we start formulating a policy that always follows the best values?
   - Use the epsilon-greedy policy

2. How do we estimate values?

$$V(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$$

No model!

Q-Learning

- Can't use Bellman Equation to estimate the expectation directly, because of the max operator

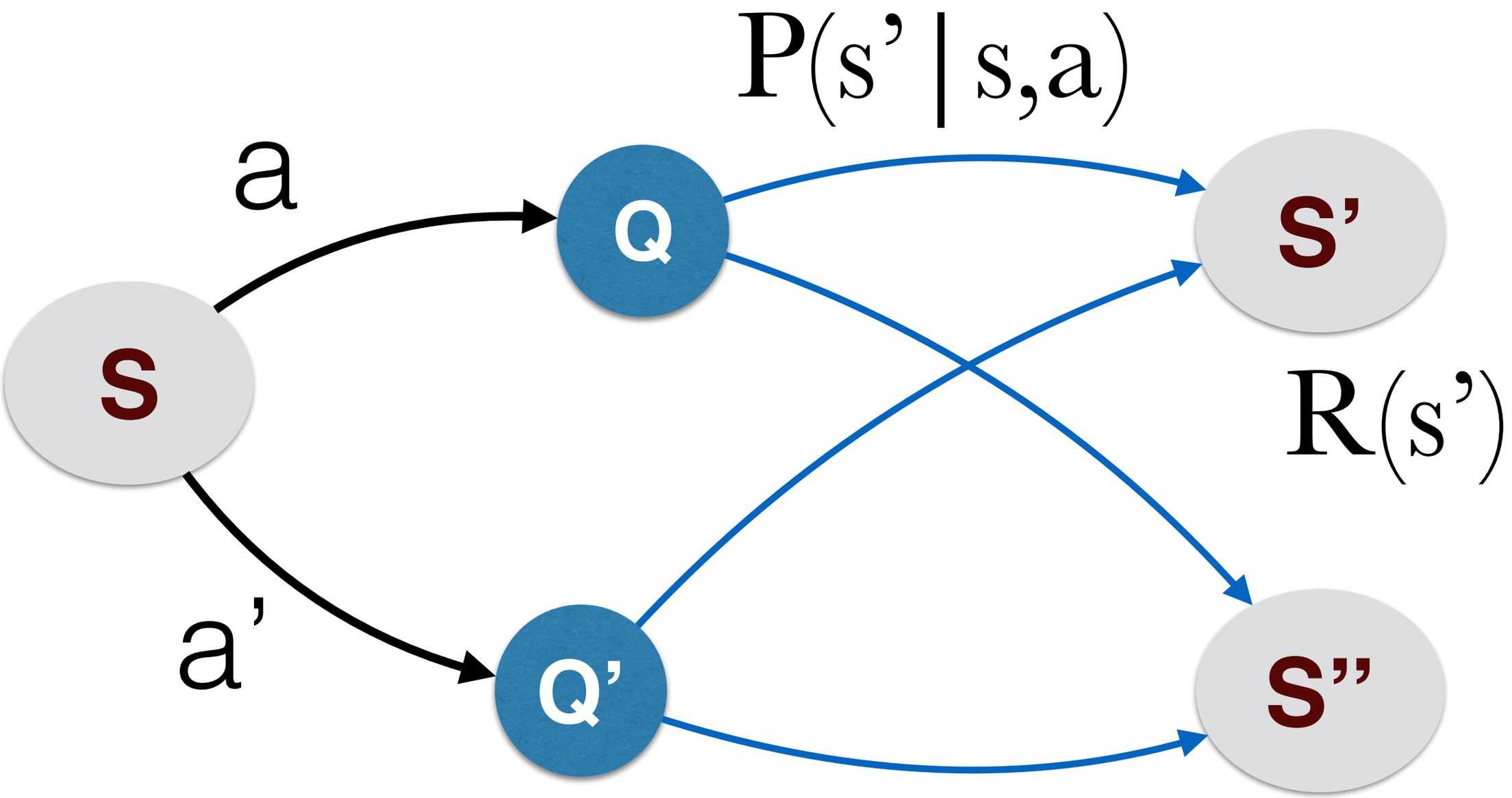
$$V(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$$

- Note the difference with TD evaluation

(TD) $$V^{\pi}(s) = \mathbb{E}_{\pi}[R(s) + \gamma V^{\pi}(s')]$$

(Now) $$V(s) = \max_{a} \left(\mathbb{E}[R(s) + \gamma V(s' \,|\, s, a)] \right)$$

Q-Learning



Q-Learning

- Q-states eliminates the problematic max

$$V(s) = \max_a Q(s, a)$$

$$V(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a) V(s')$$

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

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

Q-Learning

- Q-states eliminates the problematic max

$$V(s) = \max_a Q(s, a)$$

$$Q(s, a) = \sum_{s'} P(s' \mid s, a) \left(R(s) + \gamma \max_{a'} Q(s', a') \right)$$

- Then just estimate the expectation through samples

$$Q(s, a) = \mathbb{E}(R(s) + \gamma \max_{a'} Q(s', a'))$$

Q-Learning

- We can then just estimate the expectation.

$$Q(s,a) = \mathbb{E}(R(s) + \gamma \max_{a'} Q(s',a'))$$

- Use a TD-update rule in the Q values

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

$$\mu_k \leftarrow \mu_{k-1} + \alpha_k(x_k - \mu_{k-1})$$

Q-Learning

The algorithm is, again, simple:

- Start with knowing nothing
- Loop the following steps
  - Pick an action based on epsilon-greedy policy
  - Update Q values on the fly

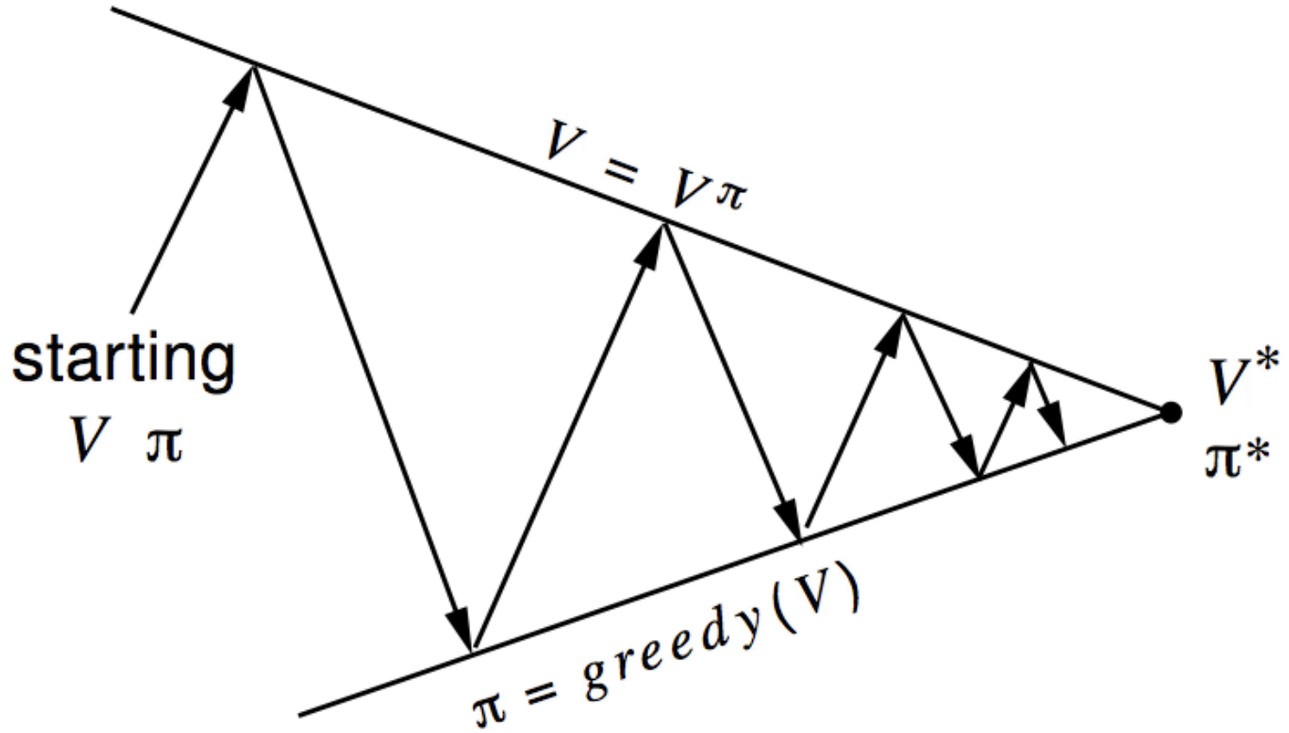
$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

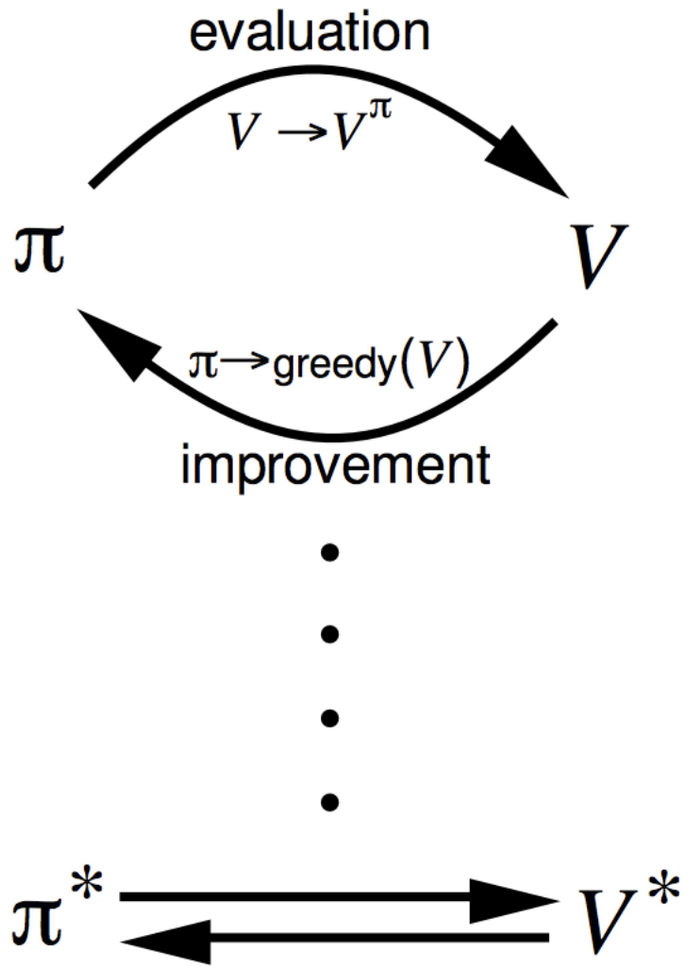
The policy will provably converge to the optimal.

Q-Learning

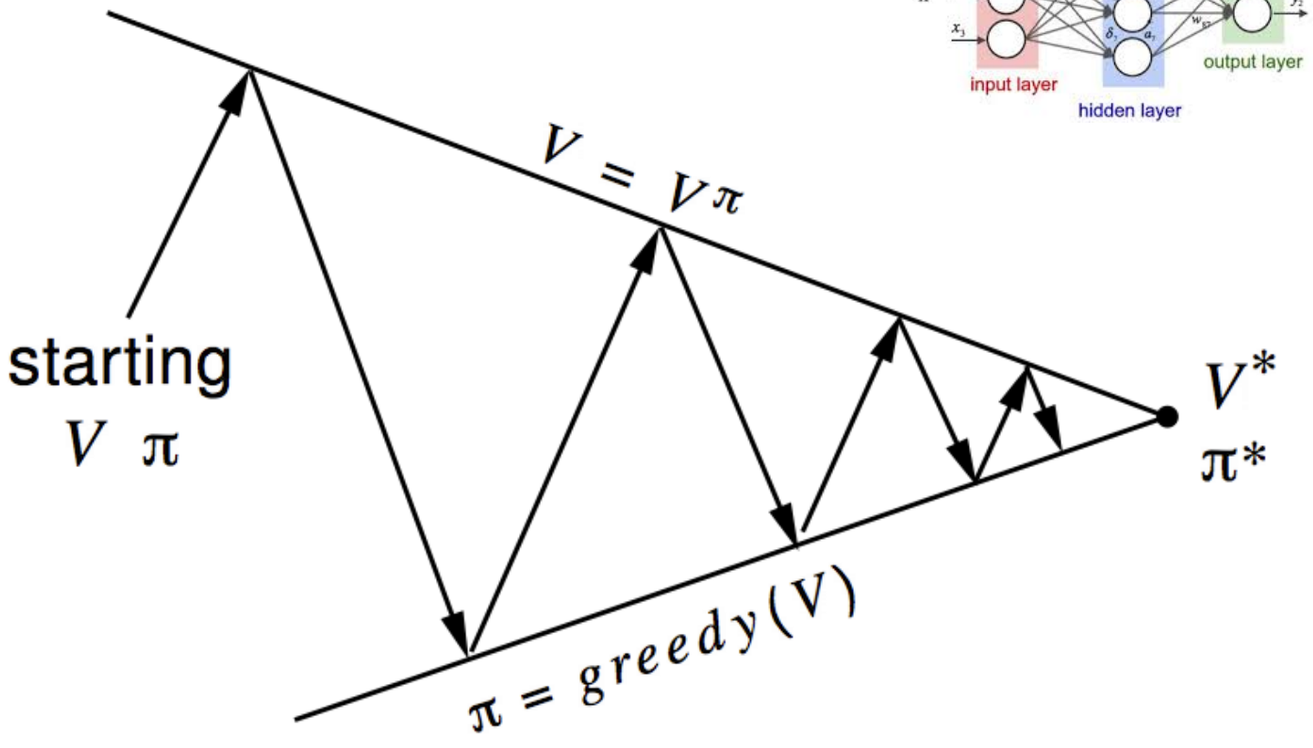
```
def pick_action(s,eps):
    #eps should be positive and very small
    if random([0,1]) < eps:
        return random(action(s))
    else:
        return argmax(Q(s,a))

def Q-Learning(states, actions, R, gamma, alpha):
    for (s, a) in states, actions:
        Q(s,a) = 0
    while within computation budget:
        s = arbitrary state
        eps = some small constant in range(0,0.5)
        while s is not NULL:
            #when next_s is NULL, use zero for all values
            a = pick_action(s, eps, Q)
            next_s = simulate_one_step(s, a)
            Q(s,a) = Q(s,a)+alpha*(R(s)+gamma*max(a,Q(next_s,a))-Q(s,a))
            s = next_s
```

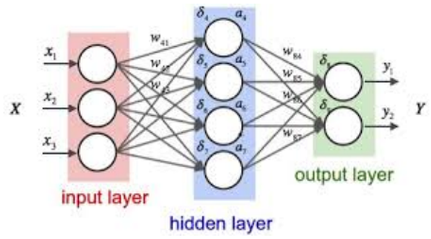
In MDP and RL, remember how simple algorithms can come from fairly complex (but nice) theory

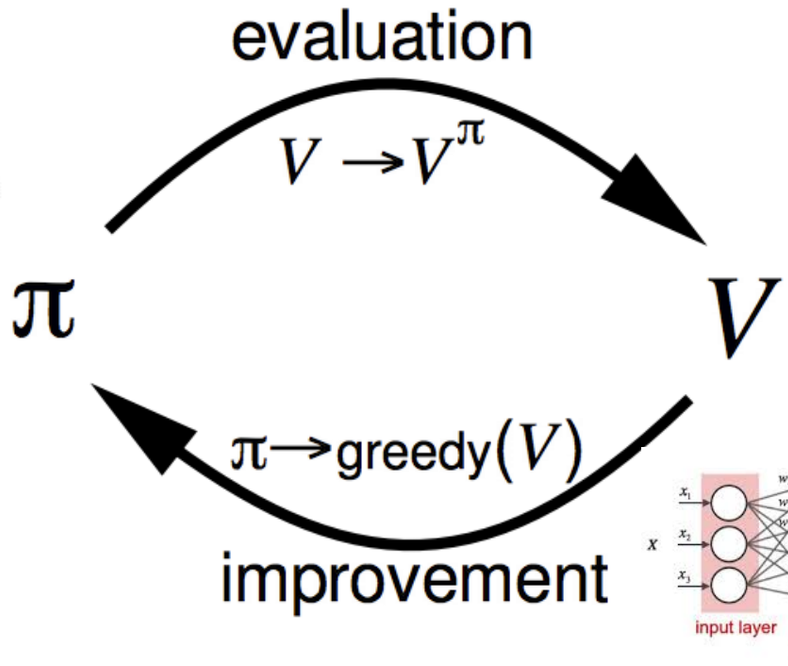


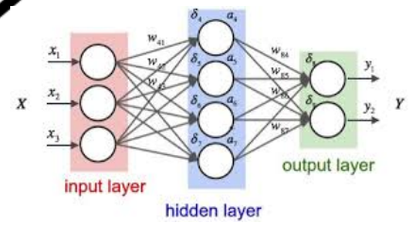


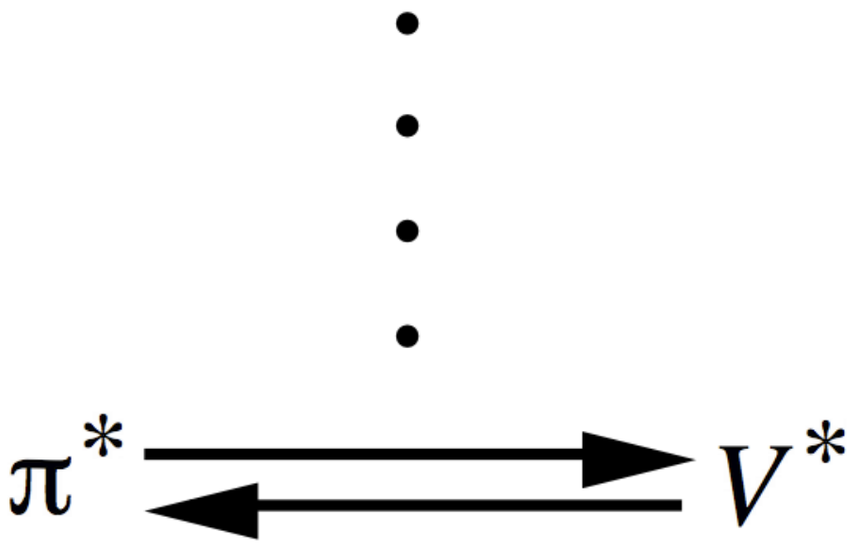
Deep RL: Use NN for Values and Policies







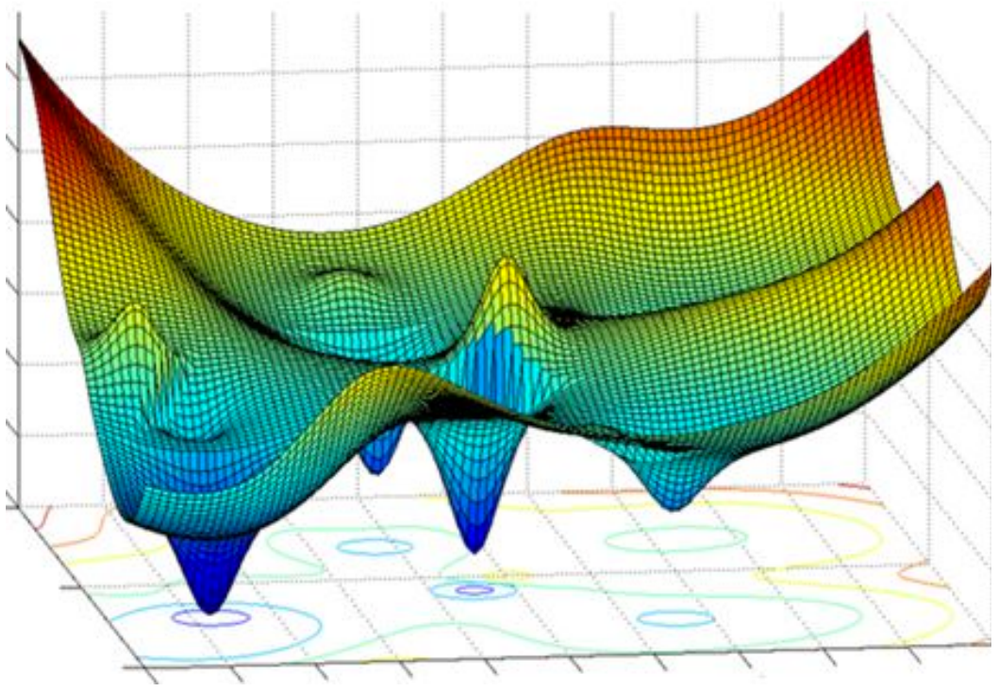


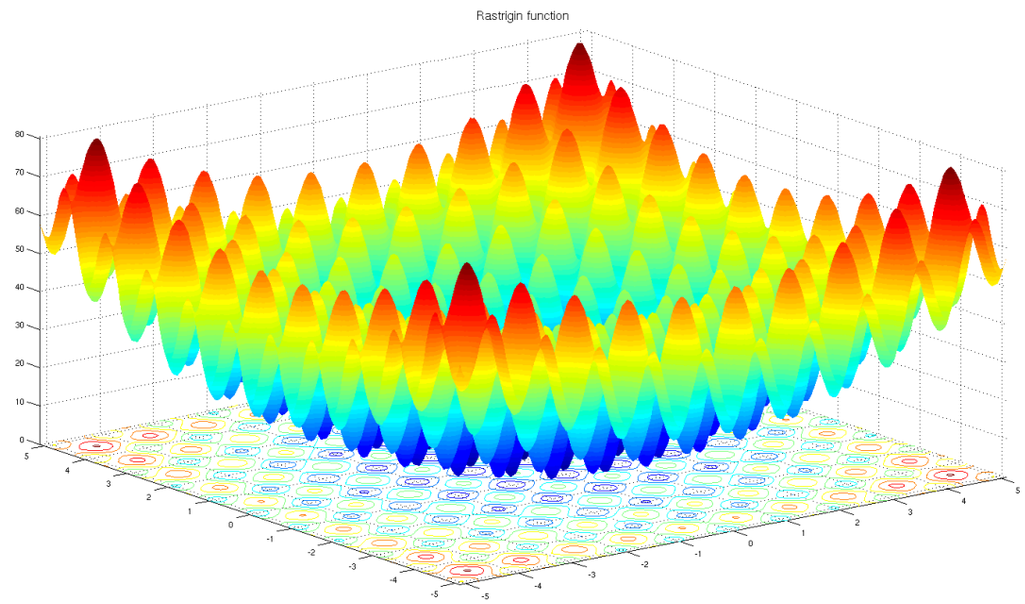


Deep Learning Recap

- Improve complicated objectives in the locally steepest directions and see what happens

$$f(\mathbf{x}+\mathbf{u}) = f(\mathbf{x}) + \mathbf{u}^T \nabla f(\mathbf{x}) + O(\|\mathbf{u}\|^2)$$





Deep Learning Recap

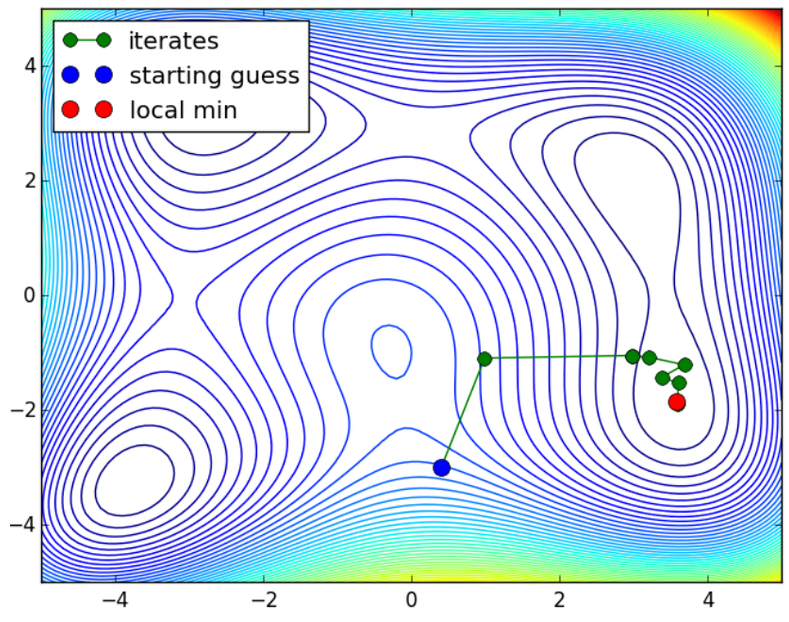
- MSE with dataset D

$$f(\theta) = \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in D} \left(f(\theta, \mathbf{x}_i) - \mathbf{y}_i \right)^2$$

- Minibatch size B

$$f(\theta) = \sum_{\{\mathbf{x}_i, \mathbf{y}_i\}_{i \in B} \sim D} \left(f(\theta, \mathbf{x}_i) - \mathbf{y}_i \right)^2$$

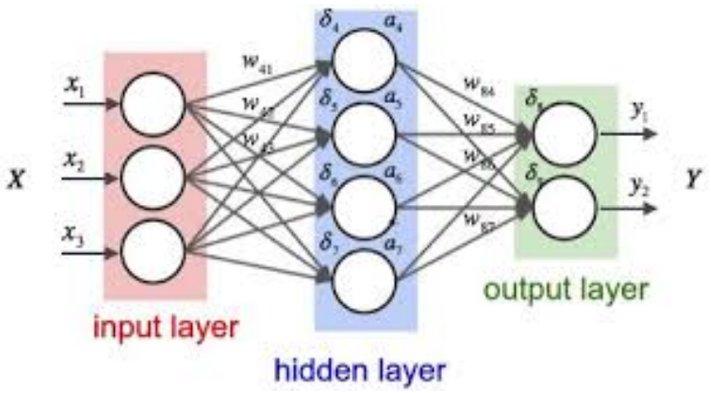
$$\theta \leftarrow \theta - \alpha \nabla f(\theta)$$

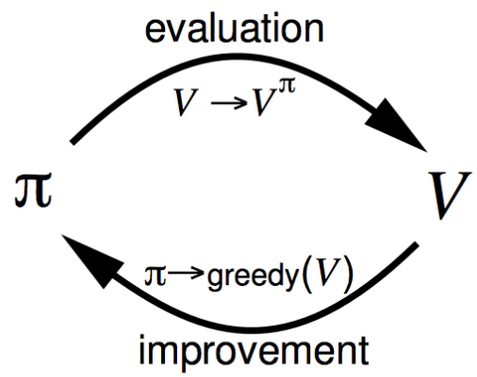


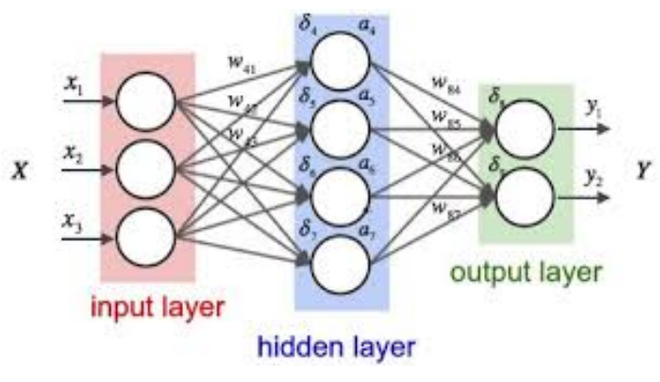
Deep RL

- Use expressive function approximators to store values or/and policies
- Benefits:
  - No longer limited by table size or discretization
  - Generalizability
  - Direct manipulation of policies

Deep RL

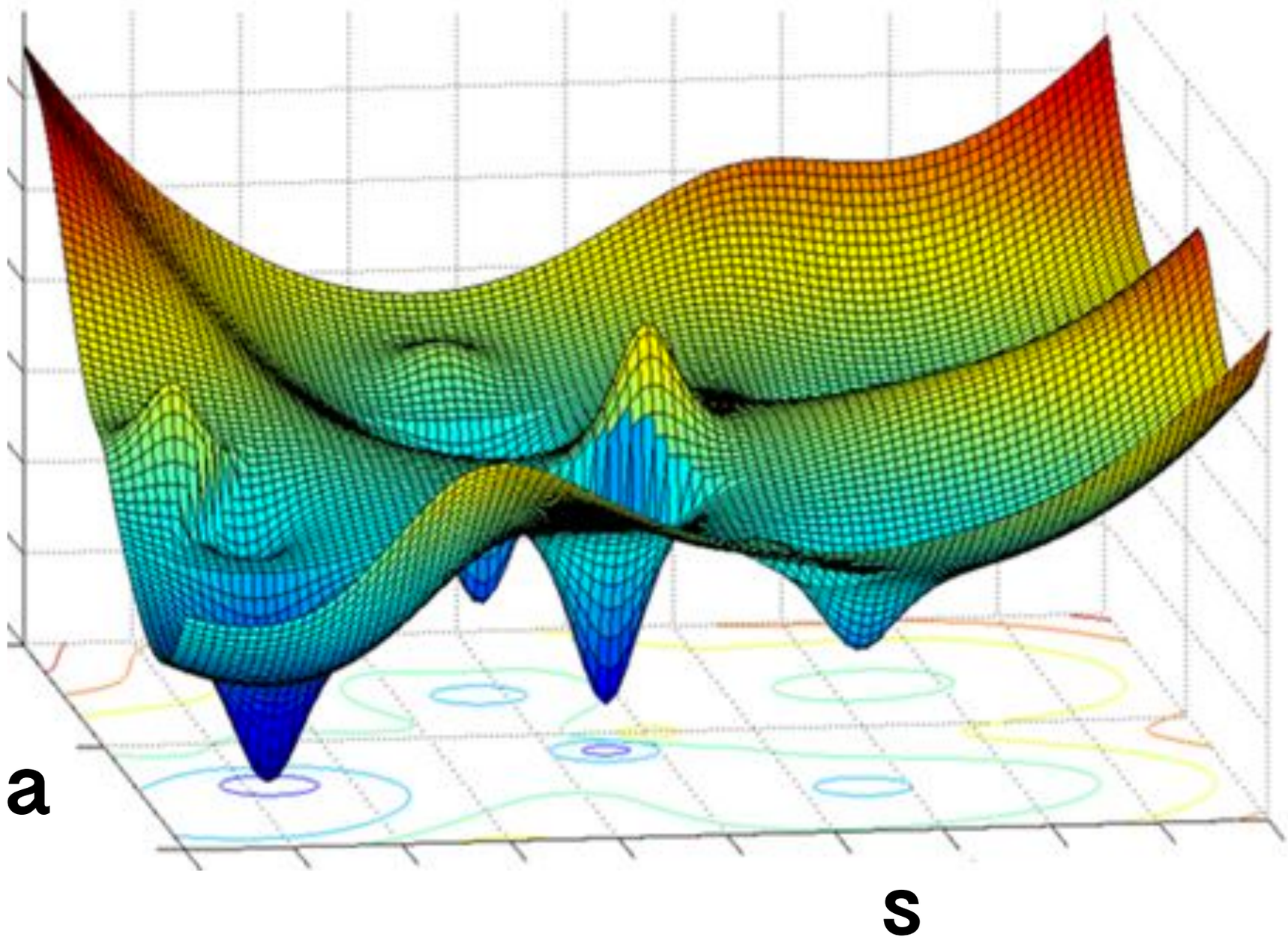






- Approximate value functions (DQN, DDPG)
- Approximate policy functions (REINFORCE, TRPO, PPO, …)
- Both (Actor-Critic, ACER, …)

Values and Policies



Tabular Q-Learning

- Repeat the following steps:
  - Take an action based on epsilon-greedy policy
  - TD-Update Q values

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Deep Q-Learning

- Repeat the following steps:
  - Take an action based on epsilon-greedy policy
  - Fit a Q-value function to the TD-updates (How?)

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Deep Q-Learning

- Repeat the following steps:
  - Take an action based on epsilon-greedy policy
  - Keep adding new experience (s, a, r, s') to a big pool of samples "Experience replay"
  - Take a batch from the big pool and update their values to Q_new
  - Fit a Q network to (s, a, Q_new)

Deep Q-Learning

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$ typo: ~~max~~ argmax
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

"Playing Atari with Deep Reinforcement Learning" Mnih et al. 2013

DDPG

- Extending DQN to continuous action spaces, but carry a deterministic "best-known" policy

- Avoid taking max Q value on the next state (hard to take max over continuous actions), use the current best policy to take actions instead

- Update value and policy networks through iterative interpolation (polyak averaging)

DDPG

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

"Continuous control with deep reinforcement learning" Lillicrap et al. 2015

Policy Gradient

- From now on we will think about stochastic policies over continuous space in general
- Remember the goal of RL is to find policies that maximize expected return globally

$$J_\pi = \mathbb{E}_{\tau \sim \pi}(R(\tau)) \qquad R(\tau) = \sum_{i=0}^{T} \gamma^i s_i$$

- We'll use the episodic setting: finite length T+1

Policy Gradient

- Remember the goal of RL is to find policies that maximize expected return

$$J_\pi = \mathbb{E}_{\tau \sim \pi}(R(\tau))$$

- Key observation: policies do not change trajectories, only their probabilities.

Policy Gradient

- A very general trick in stochastic optimization

$$\nabla_\theta \mathbb{E}_{x\sim P_\theta}[f(x)] = \mathbb{E}_{x\sim P_\theta}[\nabla_\theta \log(p(x\,|\,\theta))f(x)]$$

- Turning the gradient of an expectation into the expectation of a gradient, so that we can sample things.
  - Recall why Q-learning likes max inside the expectation.

Policy Gradient

$$J_\pi = \mathbb{E}_{\tau\sim\pi}(R(\tau)) = \int_{\mathcal{T}} R(\tau)p_\pi(\tau)\mathrm{d}\tau$$

space of all trajectories

- Any stochastic policy turns MDP into a Markov chain with a constant expected value.
- All policies can be ordered by this value.

Policy Gradient

$$J_\pi = \mathbb{E}_{\tau\sim\pi}(R(\tau)) = \int_{\mathcal{T}} R(\tau)p_\pi(\tau)\mathrm{d}\tau$$

- Goal: Change π directly to improve J_π .
- Parameterize π as $\pi(\theta)$, then gradient ascent.

Policy Gradient

- With parametrization:

$$J_{\pi(\theta)} = J(\theta) = \int_{\mathcal{T}} R(\tau) p(\tau \mid \theta) \mathrm{d}\tau$$

- The gradient of this objective is:

$$\nabla_\theta J(\theta) = \nabla \left(\int_{\mathcal{T}} R(\tau) p(\tau \mid \theta) \mathrm{d}\tau \right) = \int_{\mathcal{T}} R(\tau) \left(\nabla p(\tau \mid \theta) \right) d\tau$$

$$= \int_{\mathcal{T}} R(\tau) p(\tau \mid \theta) \nabla \log(p(\tau \mid \theta)) \mathrm{d}\tau$$

Policy Gradient

- The gradient of this objective is:

$$\nabla J(\theta) = \nabla \left(\int_{\mathcal{T}} R(\tau) p(\tau \mid \theta) \mathrm{d}\tau \right) = \int_{\mathcal{T}} R(\tau) \left(\nabla p(\tau \mid \theta) \right) d\tau$$

$$= \int_{\mathcal{T}} R(\tau) \nabla \log(p(\tau \mid \theta)) p(\tau \mid \theta) \mathrm{d}\tau$$

$$= \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau) \nabla \log(p_\theta(\tau))]$$

Policy Gradient

- The gradient of this objective is:

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau) \nabla \log(p_\theta(\tau))]$$

- Important: environment transitions do not matter

$$p_\theta(\tau) = p_\theta(s_0, s_1, \ldots, s_n)$$

$$= p(s_0) \prod_{i=0}^{n} \pi_\theta(a_i \mid s_i) p(s_{i+1} \mid s_i, a_i)$$

Policy Gradient

- The gradient of this objective is:

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau) \nabla \log(p_\theta(\tau))]$$

- Important: environment transitions do not matter

$$p_\theta(\tau) = p(s_0) \prod_{i=0}^{T-1} \pi_\theta(a_i \mid s_i) p(s_{i+1} \mid s_i, a_i)$$

$$\log(p_\theta(\tau)) = \log(p(s_0)) + \sum_{i=0}^{T-1} \log(\pi_\theta(a_i \mid s_i)) + \sum_{i=0}^{T-1} \log(p(s_{i+1} \mid s_i, a_i))$$

Policy Gradient

- The gradient of this objective is:

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)\nabla \log(p_\theta(\tau))]$$

- Important: environment model is eliminated

$$\log(P_\theta(\tau)) = \log(P(s_0)) + \sum_{i=0}^{T-1} \log(\pi_\theta(a_i \mid s_i)) + \sum_{i=0}^{T-1} \log(P(s_{i+1} \mid s_i, a_i))$$

$$\nabla \log(P_\theta(\tau)) = \sum_{i=0}^{T-1} \nabla \log(\pi_\theta(a_i \mid s_i))$$

Policy Gradient

- The gradient of this objective is:

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)\nabla \log(p_\theta(\tau))]$$

$$= \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)\left(\sum_{i=0}^{T-1} \nabla \log(\pi_\theta(a_i \mid s_i))\right)]$$

- We can now sample this.

Policy Gradient

- Naive policy gradient (high variance):

1. Initialize a policy network with random θ
2. Repeat:
   1. Generate a trajectory τ
   2. Calculate discounted total reward $R(\tau)$
   3. Update

$$\theta \leftarrow \theta + \alpha R(\tau) \sum_{i=0}^{|\tau|-1} \nabla \log(\pi_\theta(a_i \mid s_i))$$

This big sum leads to large variance

Policy Gradient

- The gradient can be expressed in many ways

$$\nabla_\theta J = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)\Big(\sum_{i=0}^{n-1}\nabla\log(\pi_\theta(a_i\,|\,s_i))\Big)]$$

The "REINFORCE" algorithm

$$\propto \mathbb{E}_{s,a\sim\pi_\theta}[G(s,a)\,\nabla\log(\pi_\theta(a\,|\,s))]$$

$$= \mathbb{E}_{s,a\sim\pi_\theta}[Q^{\pi_\theta}(s,a)\,\nabla\log(\pi_\theta(a\,|\,s))]$$

$$= \mathbb{E}_{s,a\sim\pi_\theta}[A^{\pi_\theta}(s,a)\,\nabla\log(\pi_\theta(a\,|\,s))]$$

Return: $G(s_k) = \sum_{i=k}^{T}\gamma^{i-k}R(s_i)$ Advantage: $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$

Policy Gradient

- Read for more details

Policy gradient methods maximize the expected total reward by repeatedly estimating the gradient $g := \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right]$. There are several different related expressions for the policy gradient, which have the form

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right], \tag{1}$$

where Ψ_t may be one of the following:

1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^\pi(s_t, a_t)$: state-action value function.
5. $A^\pi(s_t, a_t)$: advantage function.
6. $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$: TD residual.

The latter formulas use the definitions

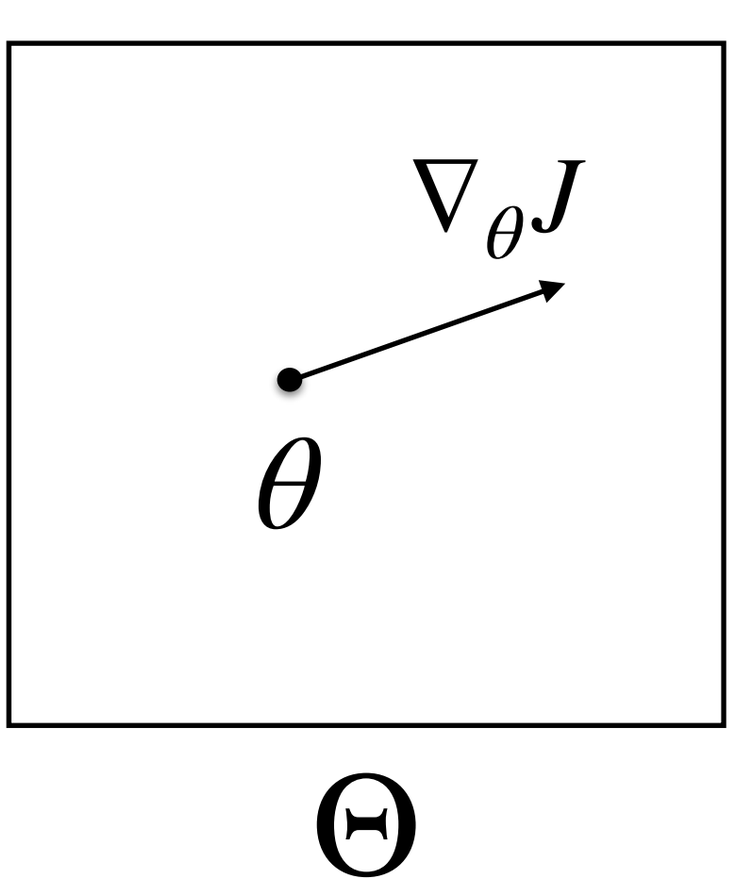
$$V^\pi(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \qquad Q^\pi(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t+1:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \tag{2}$$

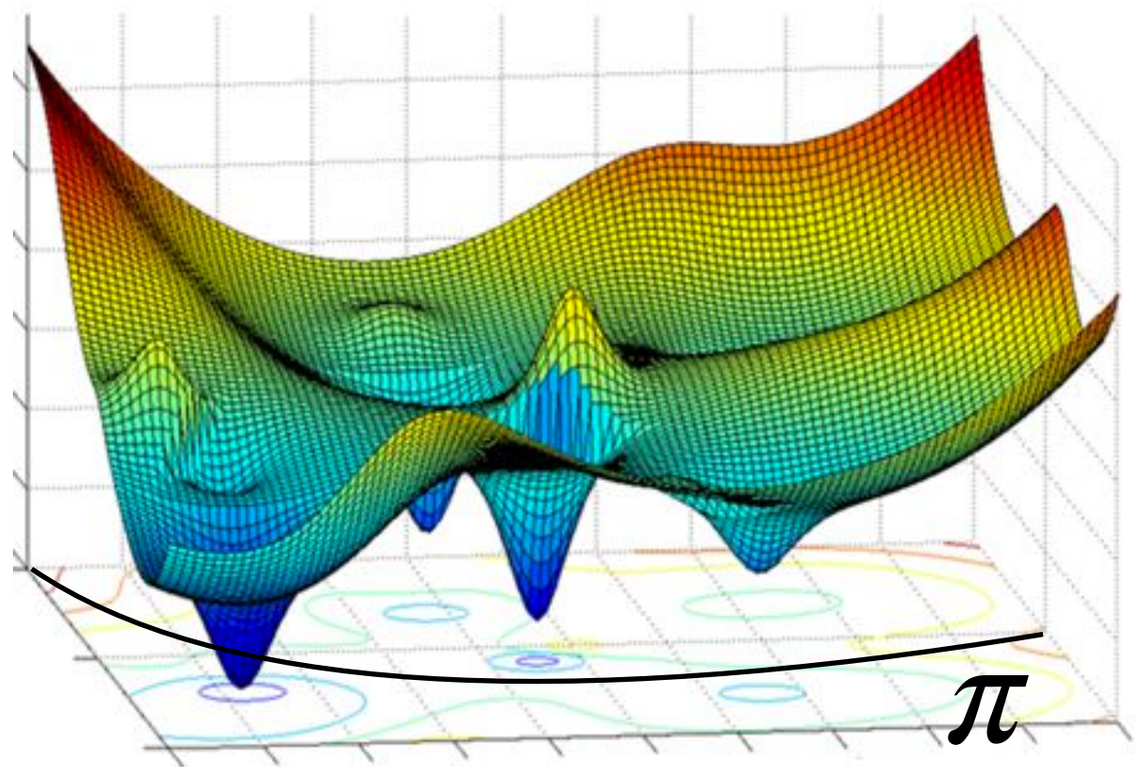
$$A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t), \quad \text{(Advantage function)}. \tag{3}$$

John Schulman, et al. "High-dimensional continuous control using generalized advantage estimation." ICLR 2016.

Trust-Region Policy Optimization

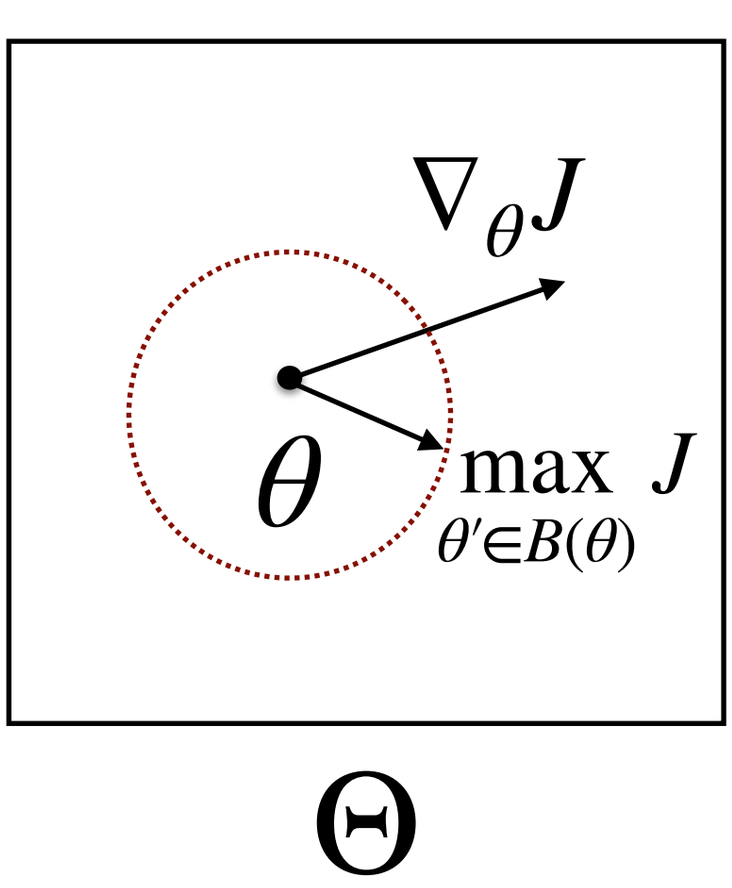
- Policy gradient only points at a local direction

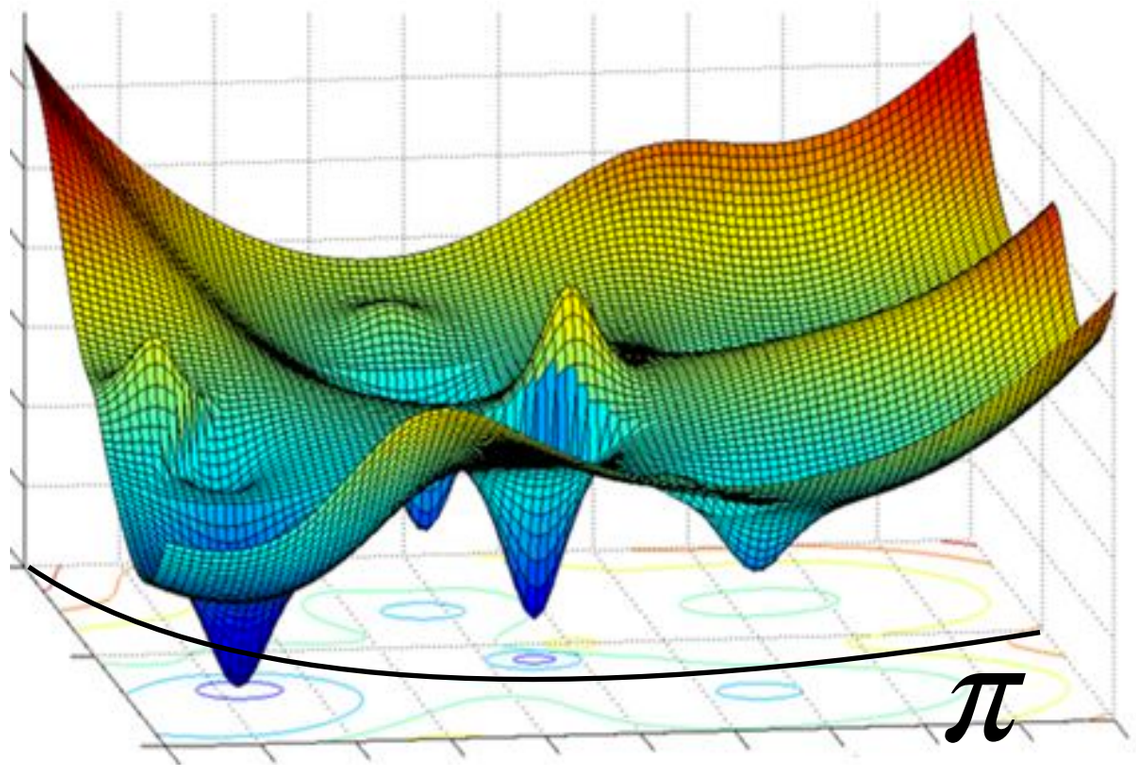




Trust-Region Policy Optimization

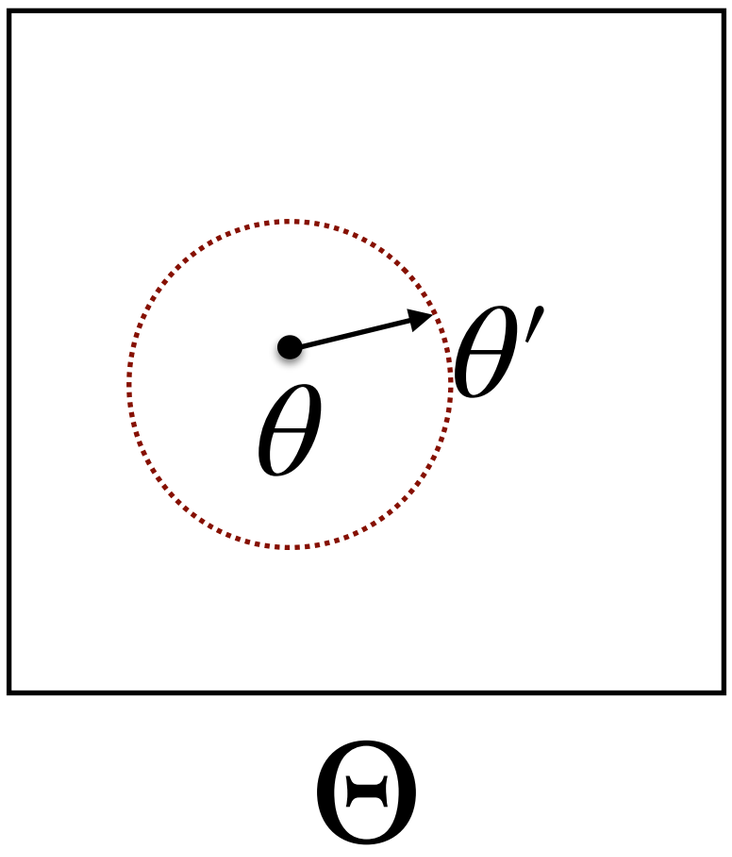
- TRPO aims to identify a neighborhood that guarantees improvement (at least in theory)





Trust-Region Policy Optimization

- Can we estimate how the performance changes?



$$J^{\pi'} = J^{\pi} + \mathbb{E}_{\pi'}[\sum_{i=0}^{\infty} \gamma^i A_{\pi}(s_i, a_i)]$$

$$A_{\pi}(s, a) = Q_{\pi}(s, a) - V_{\pi}(s)$$

Trust-Region Policy Optimization

- Can we estimate how the performance changes?

$$J^{\pi'} = J^{\pi} + \mathbb{E}_{\pi'}[\sum_{i=0}^{\infty} \gamma^i A_{\pi}(s_i, a_i)]$$

$$\mathbb{E}_{\pi'}[\sum_{i=0}^{\infty} \gamma^i A_{\pi}(s_i, a_i)] = \mathbb{E}_{\pi'}[\sum_{i=0}^{\infty} \gamma^i (Q_{\pi}(s_i, a_i) - V_{\pi}(s_i))]$$

$$= \mathbb{E}_{\pi'}[\sum_{i=0}^{\infty} \gamma^i (R(s_i) + \gamma V_{\pi}(s_{i+1}) - V_{\pi}(s_i))]$$

$$= -\mathbb{E}_{s_0}[V_{\pi}(s_0)] + \mathbb{E}_{\pi'}[\sum_{i=0}^{\infty} \gamma^i R(s_i)]$$

Trust-Region Policy Optimization

- Can we estimate how the performance changes?

$$J^{\pi'} = J^{\pi} + \mathbb{E}_{\pi'}[\sum_{i=0}^{\infty} \gamma^i A_{\pi}(s_i, a_i)]$$

$$\mathbb{E}_{\pi'}[\sum_{i=0}^{\infty} \gamma^i A_{\pi}(s_i, a_i)] = \mathbb{E}_{\pi'}[\sum_{i=0}^{\infty} \gamma^i (Q_{\pi}(s_i, a_i) - V_{\pi}(s_i))]$$

$$= -\mathbb{E}_{s_0}[V_{\pi}(s_0)] + \mathbb{E}_{\pi'}[\sum_{i=0}^{\infty} \gamma^i R(s_i)]$$

$$= -J^{\pi} + J^{\pi'}$$

Trust-Region Policy Optimization

- Can we estimate how the performance changes?

$$J^{\pi'} = J^{\pi} + \mathbb{E}_{\pi'}[\sum_{i=0}^{\infty} \gamma^i A_{\pi}(s_i, a_i)]$$

- It can be rewritten as:

$$J^{\pi'} = J^{\pi} + \sum_{i=0}^{\infty} \gamma^i \sum_{s} P(s_i = s \,|\, \pi') \Big(\sum_{a} \pi'(a \,|\, s) A_{\pi}(s, a) \Big)$$

$$= J^{\pi} + \sum_{s} \sum_{i=0}^{\infty} \gamma^i P(s_i = s \,|\, \pi') \Big(\sum_{a} \pi'(a \,|\, s) A_{\pi}(s, a) \Big)$$

Trust-Region Policy Optimization

- Can we estimate how the performance changes?

$$J^{\pi'} = J^{\pi} + \sum_{s} \sum_{i=0}^{\infty} \gamma^i P(s_i = s \mid \pi') \left(\sum_{a} \pi'(a \mid s) A_{\pi}(s, a) \right)$$

Define the discounted state visitation frequency:

$$\rho_{\pi}(s) = \sum_{i=0}^{\infty} \gamma^i P(s_i = s \mid \pi)$$

Trust-Region Policy Optimization

- Can we estimate how the performance changes?

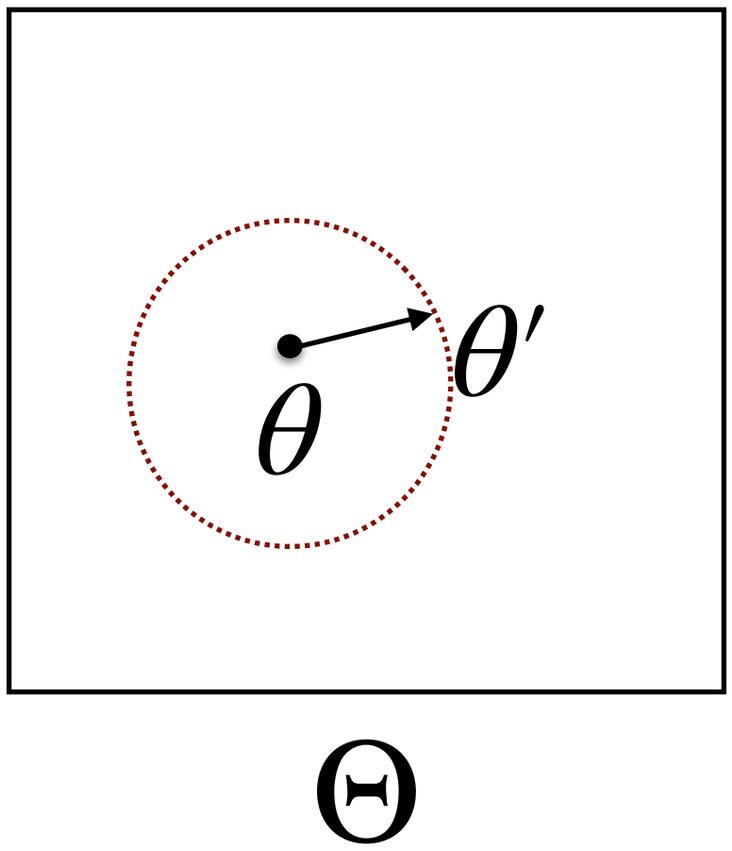
$$J^{\pi'} = J^{\pi} + \sum_{s} \sum_{i=0}^{\infty} \gamma^i P(s_i = s \mid \pi') \left(\sum_{a} \pi'(a \mid s) A_{\pi}(s, a) \right)$$

Define the discounted state visitation frequency: $\rho_{\pi}(s) = \sum_{i=0}^{\infty} \gamma^i P(s_i = s \mid \pi)$

$$J^{\pi'} = J^{\pi} + \sum_{s} \rho_{\pi'}(s) \left(\sum_{a} \pi'(a \mid s) A_{\pi}(s, a) \right)$$

Trust-Region Policy Optimization

- We can in principle estimate how policy values change



$$J^{\pi'} = J^{\pi} + \sum_{s} \rho_{\pi'}(s) \left(\sum_{a} \pi'(a \mid s) A_{\pi}(s, a) \right)$$

This is a very useful equality and shows a general form of policy iteration

However, we can only sample ρ_{π} and we don't know $\rho_{\pi'}(s)$!

Trust-Region Policy Optimization

- We can in principle estimate how policy values change

$$J^{\pi'} = J^{\pi} + \sum_{s} \rho_{\pi'}(s) \Big(\sum_{a} \pi'(a \mid s) A_{\pi}(s, a) \Big)$$

Locally, we work with the approximation

$$L_{\pi}(\pi') = J^{\pi} + \sum_{s} \rho_{\pi}(s) \Big(\sum_{a} \pi'(a \mid s) A_{\pi}(s, a) \Big)$$

When π and π' are close, $L_{\pi}(\pi')$ approximates $J^{\pi'}$ within clear bounds.

Trust-Region Policy Optimization

- In practice, when taking a step from θ to θ', choose

$$\arg\max_{\theta'} \mathbb{E}_{s,a\sim\pi_\theta}[\frac{\pi_{\theta'}(a\,|\,s)}{\pi_\theta(a\,|\,s)}A_{\pi_\theta}(s,a)]$$

  subject to KL-divergence

$$\mathbb{E}_{s\sim\pi_\theta}[D_{KL}(\pi_\theta(s)\|\pi_{\theta'}(s))] \le \delta$$

$$D_{KL}(P\|Q) = \int_X p(x)\log\frac{p(x)}{q(x)}dx$$

- The ratio in the objective comes from importance sampling

$$\mathbb{E}_p[f(x)] = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = \mathbb{E}_q[f(x)\frac{p(x)}{q(x)}]$$

Proximal Policy Optimization

- Instead of directly enforce bounds on the rational, instead of maximizing the TRPO objective

$$\mathbb{E}_{s,a\sim\pi_\theta}[\frac{\pi_{\theta'}(a\,|\,s)}{\pi_\theta(a\,|\,s)}A_{\pi_\theta}(s,a)]$$

subject to the KL-divergence constraint,
PPO simplifies the optimization to use a fixed cutoff rate

$$\frac{\pi_{\theta'}(a\,|\,s)}{\pi_\theta(a\,|\,s)} \in [1-\varepsilon, 1+\varepsilon]$$

Proximal Policy Optimization

- PPO objective (in the original paper):

$$\mathbb{E}_{\pi_\theta}[\min\left(r_t(\theta')A_{\pi_\theta}, \text{clip}\left(r_t(\theta'), 1-\varepsilon, 1+\varepsilon\right)A_{\pi_\theta}\right)]$$

$$r_{\theta'} = \frac{\pi_{\theta'}(a \mid s)}{\pi_\theta(a \mid s)}$$

$$\text{clip}(r_t(\theta'), 1-\varepsilon, 1+\varepsilon) = \begin{cases} r_t(\theta') & r_t(\theta') \in [1-\varepsilon, 1+\varepsilon] \\ 1 & \text{otherwise} \end{cases}$$

Recent development: Revisiting Design Choices in Proximal Policy Optimization (Hsu 2020)

Proximal Policy Optimization

- PPO objective (cleaner form):

$$\mathbb{E}_{\pi_\theta}[\min\left(r_t(\theta')A_{\pi_\theta}, g(\varepsilon, A_{\pi_\theta})\right)]$$

$$r_{\theta'} = \frac{\pi_{\theta'}(a\,|\,s)}{\pi_\theta(a\,|\,s)} \qquad g(\varepsilon, A_{\pi_\theta}) = \begin{cases} (1+\varepsilon)A_{\pi_\theta} & A_{\pi_\theta} \geq 0 \\ (1-\varepsilon)A_{\pi_\theta} & A_{\pi_\theta} < 0 \end{cases}$$

Recent development: Revisiting Design Choices in Proximal Policy Optimization (Hsu 2020)

Actor-Critic

- We can update both the policy networks and Q networks together, and use Q values instead of sampled return to update the policy each time

- Actor: policy network

- Critic: value network

Actor-Critic

1. Initialize s, θ, w at random; sample $a \sim \pi_\theta(a|s)$.
2. For $t = 1 \ldots T$:
    1. Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$;
    2. Then sample the next action $a' \sim \pi_\theta(a'|s')$;
    3. Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \ln \pi_\theta(a|s)$;
    4. Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
    and use it to update the parameters of action-value function:
    $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
    5. Update $a \leftarrow a'$ and $s \leftarrow s'$.

Model-Based RL

- Do not directly learn values or policies, but focus on learning the environment model

$$P(s'|s,a) \text{ and } r(s)$$

- Do this by simple supervised learning (still need replay buffer)

Model-Based RL

- At each state, when choosing an action, unroll the learned environment model for N steps, and find the best path in the tree.
- Can optionally train a policy with all the good (s,a).
- Pros: usually more sample-efficient, and the models can transfer.
- Cons: Computationally expensive, models are often much harder than policy, and can not deal with long-term effects.

Excitement about Deep RL