REINFORCEMENT LEARNING

Gergely Neu
upf.
Universitat Pompeu Fabra
Barcelona

Lecture 7:
Basic RL algorithms

Basic RL algorithms

1. building blocks of RL methods
2. policy evaluation
   - Monte Carlo
   - temporal difference learning
3. policy evaluation and learning
   - SARSA
   - Q-learning

Basic RL algorithms

1. building blocks of RL methods
2. policy evaluation
   - Monte Carlo
   - temporal difference learning
3. policy evaluation and learning
   - SARSA
   - Q-learning

EPILOGUE

from
Dynamic Programming
to
Reinforcement Learning

FROM DYNAMIC PROGRAMMING TO VALUE-BASED REINFORCEMENT LEARNING

Policy iteration:

$$V_k$$

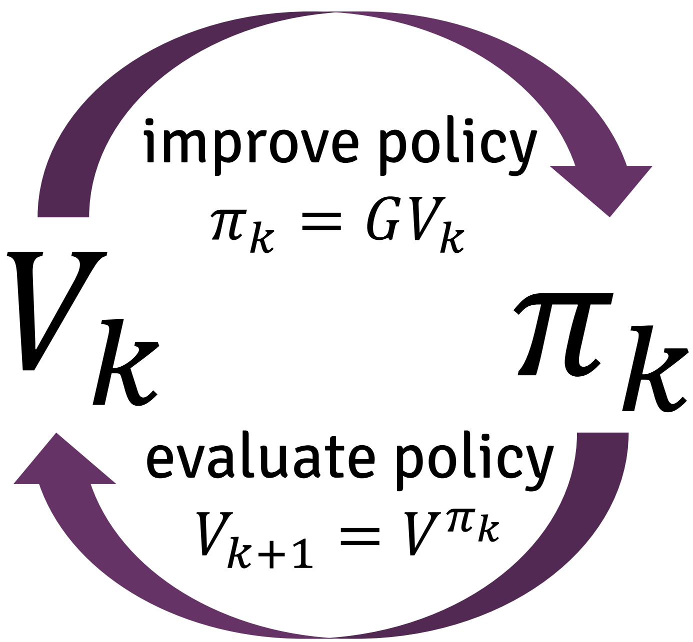
FROM DYNAMIC PROGRAMMING TO VALUE-BASED REINFORCEMENT LEARNING

Policy iteration:

$$V_k \xrightarrow[\pi_k = GV_k]{\text{improve policy}} \pi_k$$

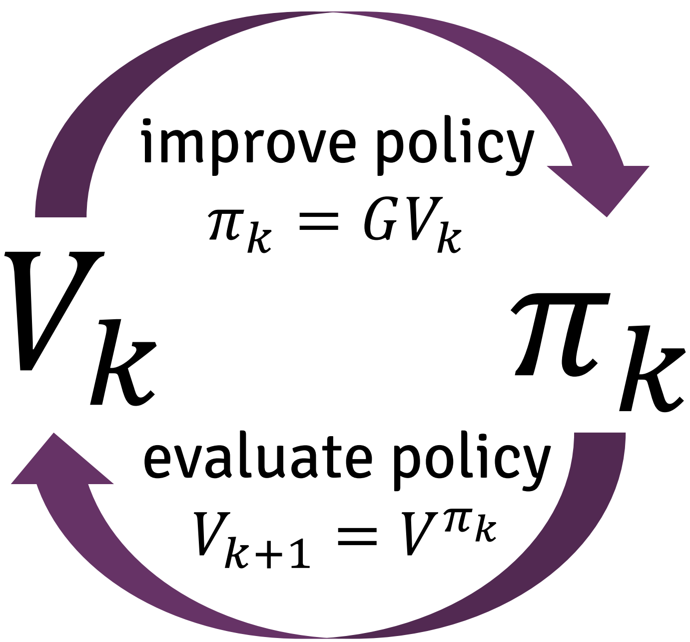
FROM DYNAMIC PROGRAMMING TO VALUE-BASED REINFORCEMENT LEARNING

Policy iteration:

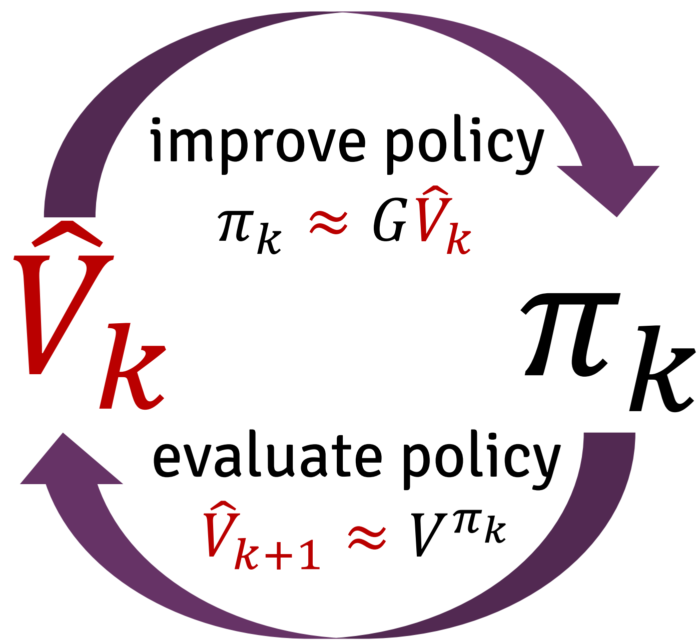


FROM DYNAMIC PROGRAMMING TO VALUE-BASED REINFORCEMENT LEARNING

Policy iteration:



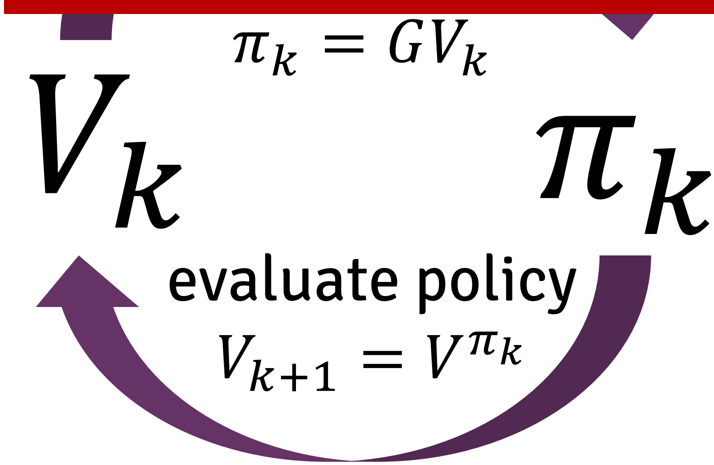
Approximate policy iteration:



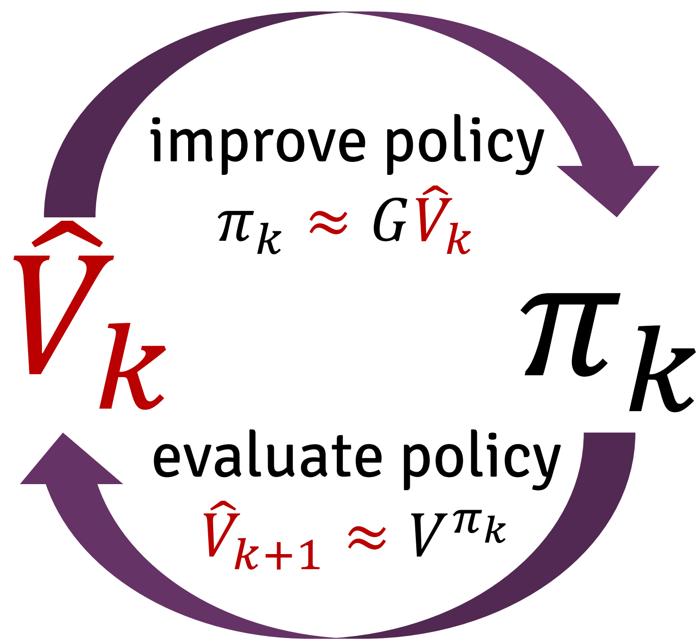
FROM DYNAMIC PROGRAMMING TO VALUE-BASED REINFORCEMENT LEARNING

Fundamental RL tasks:

- Policy evaluation
- Policy improvement



Approximate policy iteration:



FROM DYNAMIC PROGRAMMING TO VALUE-BASED REINFORCEMENT LEARNING

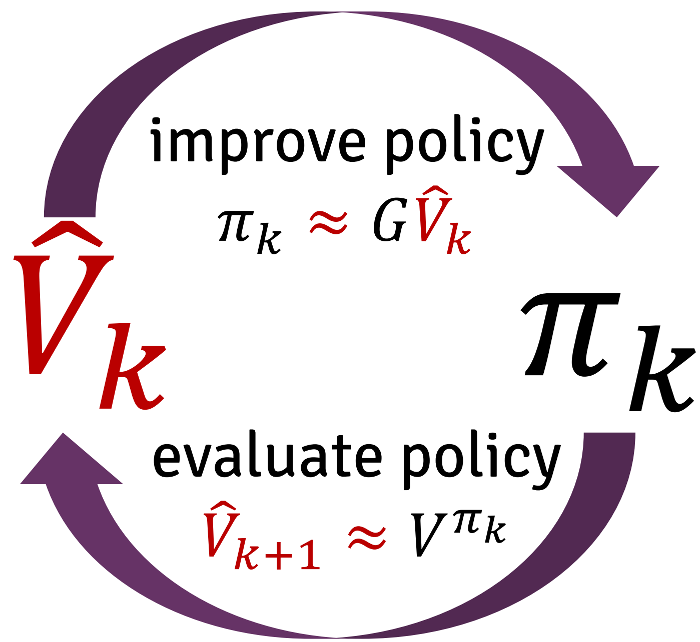
Fundamental RL tasks:

- Policy evaluation
- Policy improvement

Challenges in RL:

- Unknown transition and reward functions ⇒ have to learn from sample access only
- State/action space can be large ⇒ V^* and π^* cannot be stored in memory

Approximate policy iteration:



FROM DYNAMIC PROGRAMMING TO VALUE-BASED REINFORCEMENT LEARNING

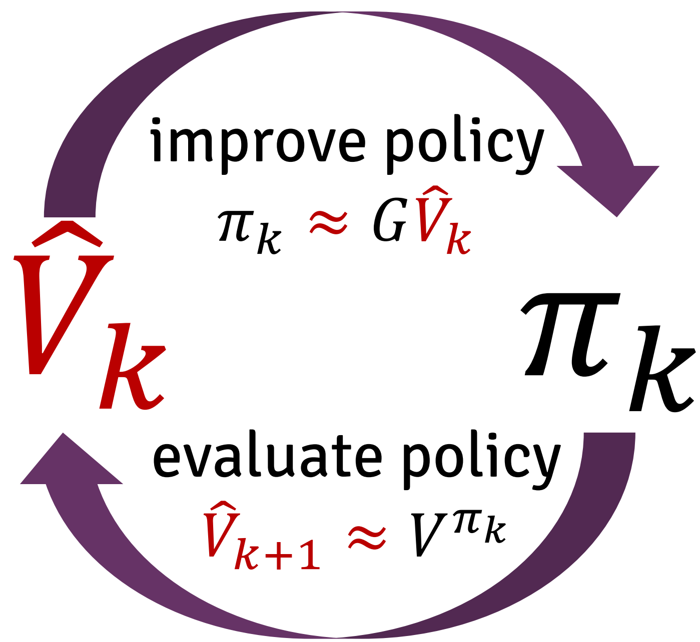
Fundamental RL tasks:

- Policy evaluation
- Policy improvement

Challenges in RL:

- Unknown transition and reward functions ⇒ have to learn from sample access only
- State/action space can be large ⇒ V^* and π^* cannot be stored in memory

Approximate policy iteration:



Unknown transition and reward functions
⇒ have to learn from sample access only

LEVELS OF SAMPLE ACCESS

Unknown transition and reward functions
⇒ have to learn from sample access only

LEVELS OF SAMPLE ACCESS

Full knowledge of P
⇒ Planning (not RL)

Unknown transition and reward functions
⇒ have to learn from sample access only

LEVELS OF SAMPLE ACCESS

Generative model:
Full sample access to $P(\cdot \mid x, a)$ for any (x, a)

Full knowledge of P
⇒ Planning (not RL)

Unknown transition and reward functions
$\Rightarrow$ have to learn from sample access only

LEVELS OF SAMPLE ACCESS

Samples from full trajectories
+ reset action or save states

Generative model:
Full sample access to $P(\cdot\,|x,a)$ for any (x,a)

Full knowledge of P
$\Rightarrow$ Planning (not RL)

Unknown transition and reward functions
⇒ have to learn from sample access only

LEVELS OF SAMPLE ACCESS

Samples from a single trajectory
⇒ online RL

Samples from full trajectories
+ reset action or save states

Generative model:
Full sample access to $P(\cdot\,|x,a)$ for any (x,a)

Full knowledge of P
⇒ Planning (not RL)

Unknown transition and reward functions
⇒ have to learn from sample access only

LEVELS OF SAMPLE ACCESS

Samples from a single trajectory
⇒ online RL

Samples from full trajectories
+ reset action or save states

Generative model:
Full sample access to $P(\cdot\,|x, a)$ for any (x, a)

Full knowledge of P
⇒ Planning (not RL)

State/action space can be large
$\Rightarrow V^*$ and π^* cannot be stored in memory

DEALING WITH LARGE STATE SPACES

Idea: approximate V^* and/or π^* in a computationally tractable way!

State/action space can be large
$\Rightarrow V^*$ and π^* cannot be stored in memory

DEALING WITH LARGE STATE SPACES

Idea: approximate V^* and/or π^* in a computationally tractable way!

Approximating V^*: linear function approximation

- Define a set of d features:
$$\phi_i : X \to \mathbf{R}$$
- Parametrize value functions as
$$V_\theta(x) = \theta^\top \phi(x)$$
- Learning $V^* \Leftrightarrow$ Learning a good θ_*
$$V_{\theta^*} \approx V^*$$

State/action space can be large
$\Rightarrow V^*$ and π^* cannot be stored in memory

DEALING WITH LARGE STATE SPACES

Idea: approximate V^* and/or π^* in a computationally tractable way!

Approximating V^*: linear function approximation

- Define a set of d features:
  $$\phi_i : X \to \mathbf{R}$$
- Parametrize value functions as
  $$V_\theta(x) = \theta^\top \phi(x)$$
- Learning $V^* \Leftrightarrow$ Learning a good θ_*
  $$V_{\theta^*} \approx V^*$$

Approximating π^*: parametrized policies

- Define a set of d features:
  $$\phi_i : X \times A \to \mathbf{R}$$
- Parametrize (stochastic) policies as
  $$\pi_\theta(a|x) \propto \exp\left(\theta^\top \phi(x)\right)$$
- Learning $\pi^* \Leftrightarrow$ Learning a good θ_*
  $$\pi_{\theta^*} \approx \pi^*$$

State/action space can be large
$\Rightarrow V^*$ and π^* cannot be stored in memory

DEALING WITH LARGE STATE SPACES

Idea: approximate V^* and/or π^* in a computationally tractable way!

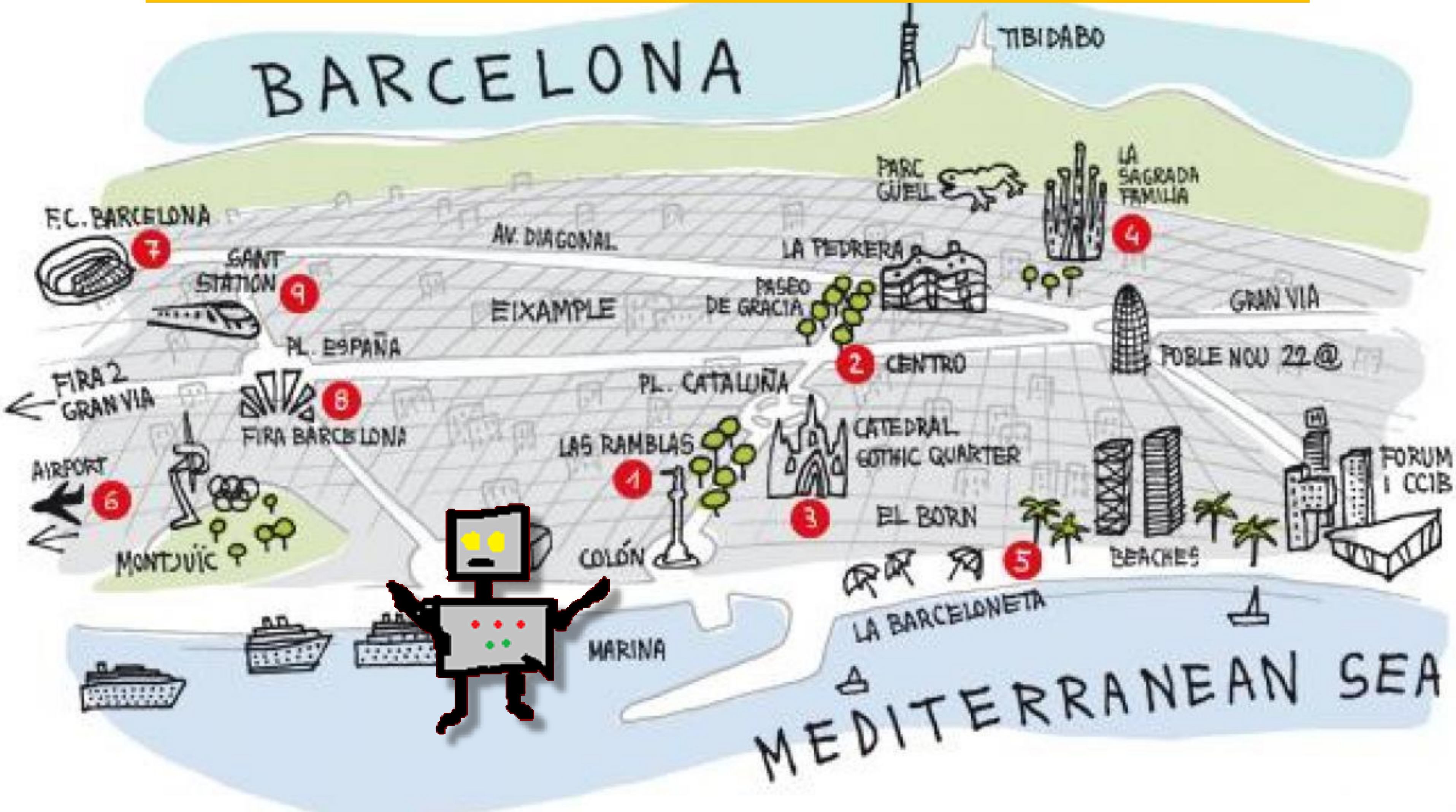
Approximating V^*: linear function approximation

- Define a set of d features:
  $$\phi_i : X \to \mathbf{R}$$
- Parametrize value functions as
  $$V_\theta(x) = \theta^\top \phi(x)$$
- Learning $V^* \Leftrightarrow$ Learning a good θ_*
  $$V_{\theta^*} \approx V^*$$

Approximating π^*: parametrized policies

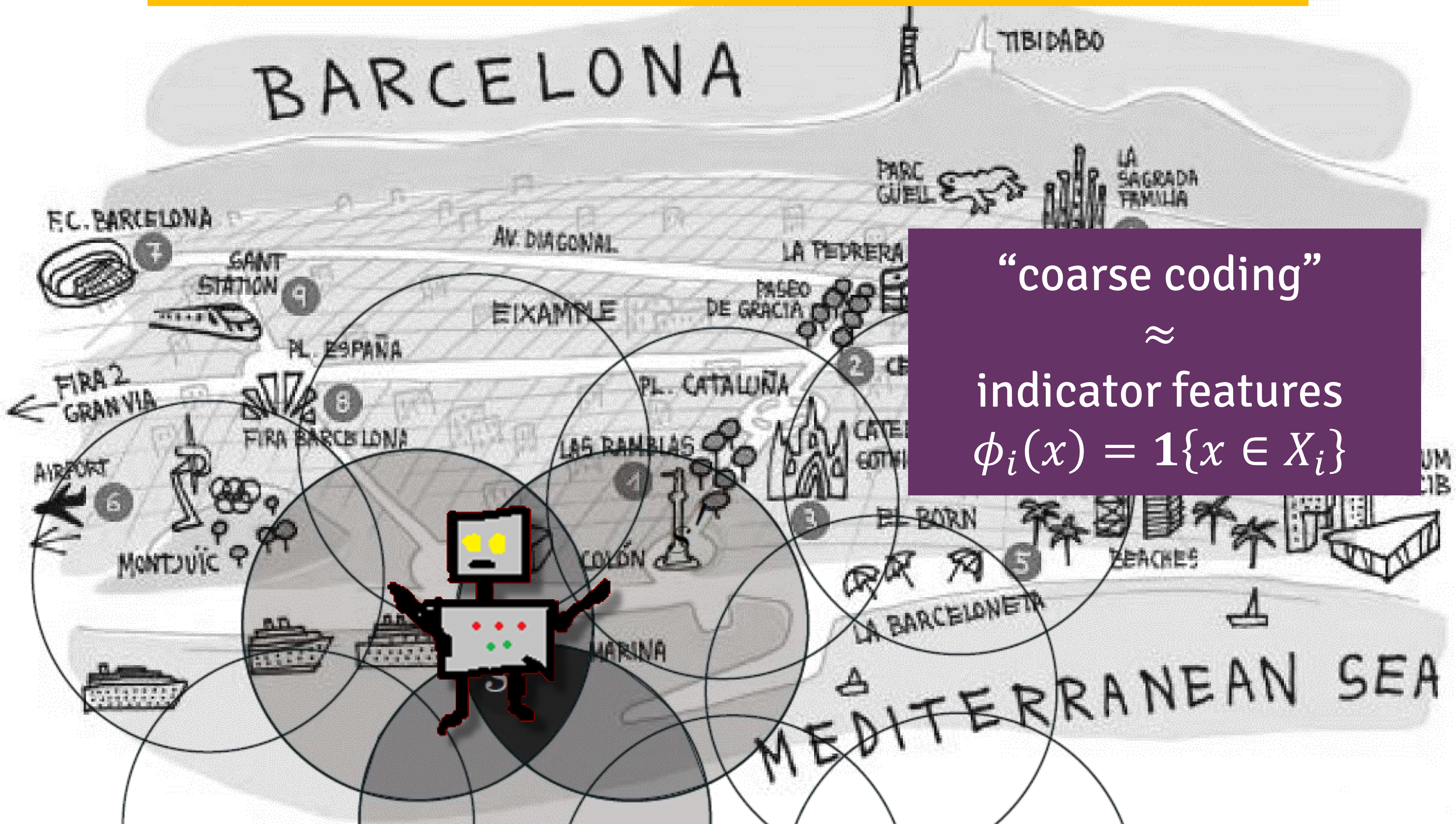
- Define a set of d features:
  $$\phi_i : X \times A \to \mathbf{R}$$
- Parametrize (stochastic) policies as
  $$\pi_\theta(a|x) \propto \exp\left(\theta^\top \phi(x)\right)$$
- Learning $\pi^* \Leftrightarrow$ Learning a good θ_*
  $$\pi_{\theta^*} \approx \pi^*$$

FEATURE MAP EXAMPLE



FEATURE MAP EXAMPLE

FEATURE MAP EXAMPLE



“coarse coding”

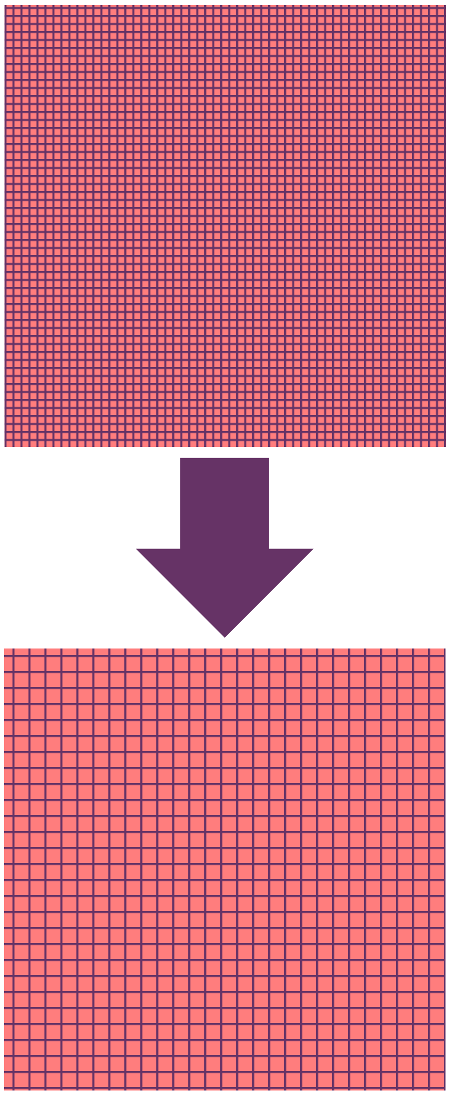
$\approx$

indicator features

$\phi_i(x) = \mathbf{1}\{x \in X_i\}$

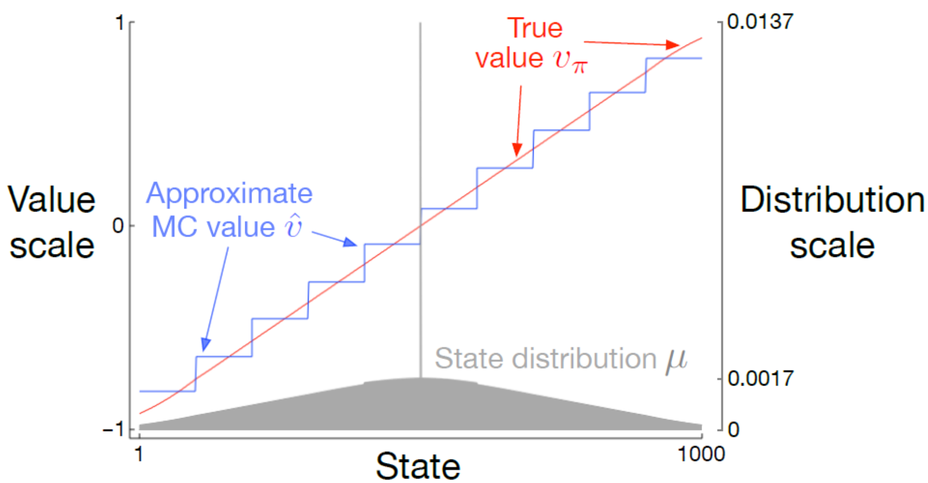
STATE AGGREGATION

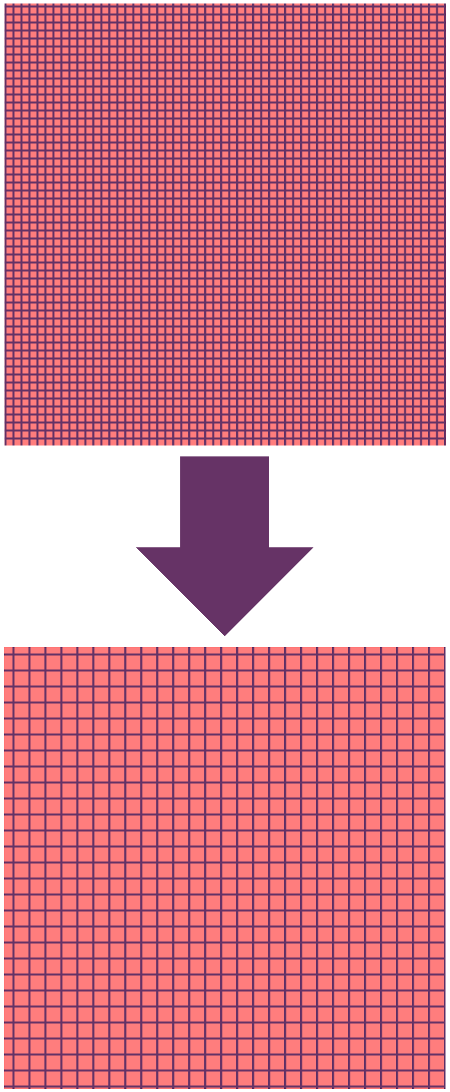
- Partition the state space X into d sets:
$$X = C_1 \cup C_2 \cup \cdots \cup C_d$$
with $C_i \cap C_j = \emptyset$ when $i \neq j$
- Define feature i as $\phi_i(x) = \mathbf{1}\{x \in C_i\}$



STATE AGGREGATION

- Partition the state space X into d sets:
$$X = C_1 \cup C_2 \cup \cdots \cup C_d$$
with $C_i \cap C_j = \emptyset$ when $i \neq j$
- Define feature i as $\phi_i(x) = \mathbf{1}\{x \in C_i\}$
- Parametrizes piecewise constant functions

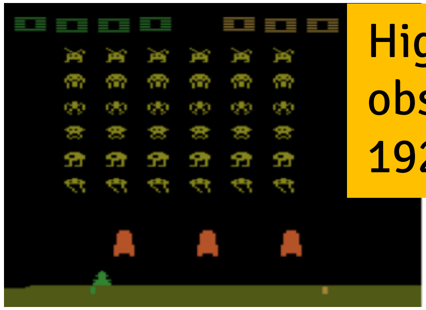




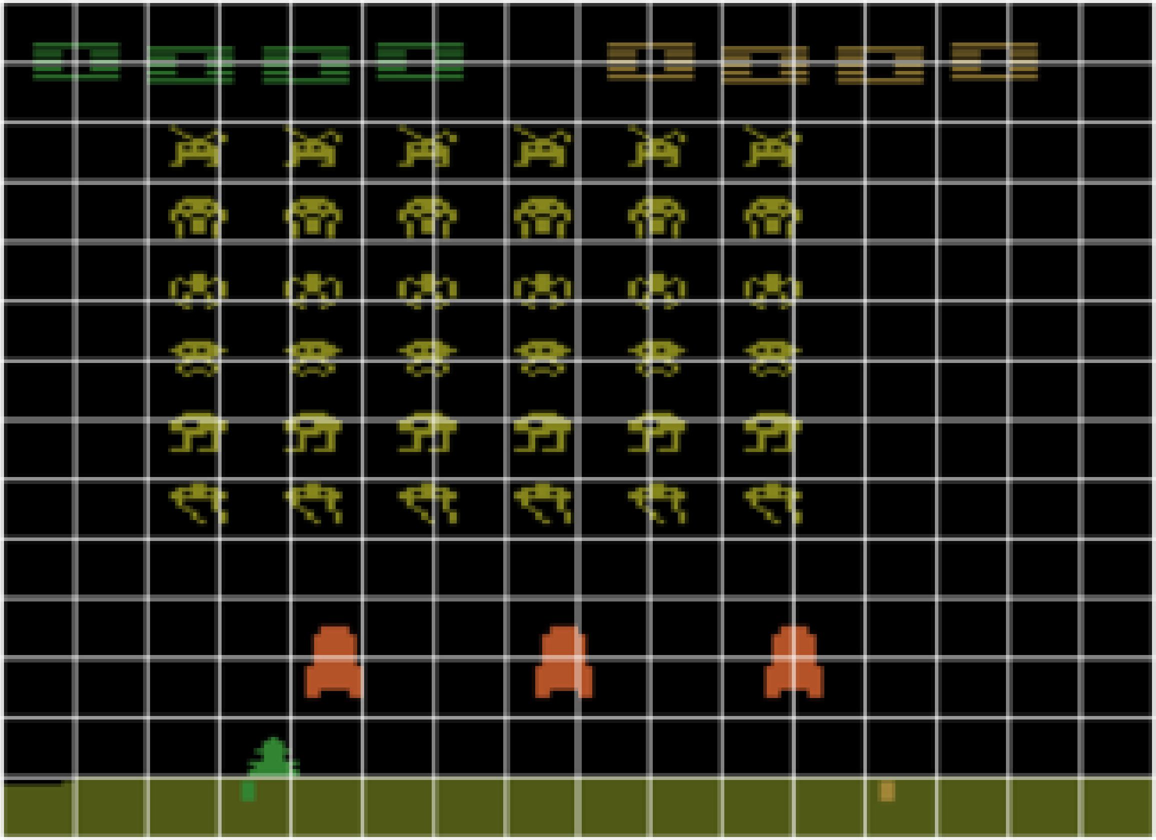
STATE AGGREGATION EXAMPLE: “PROST” FEATURES FOR ATARI GAMES

High-dimensional observations: 192×160 pixels

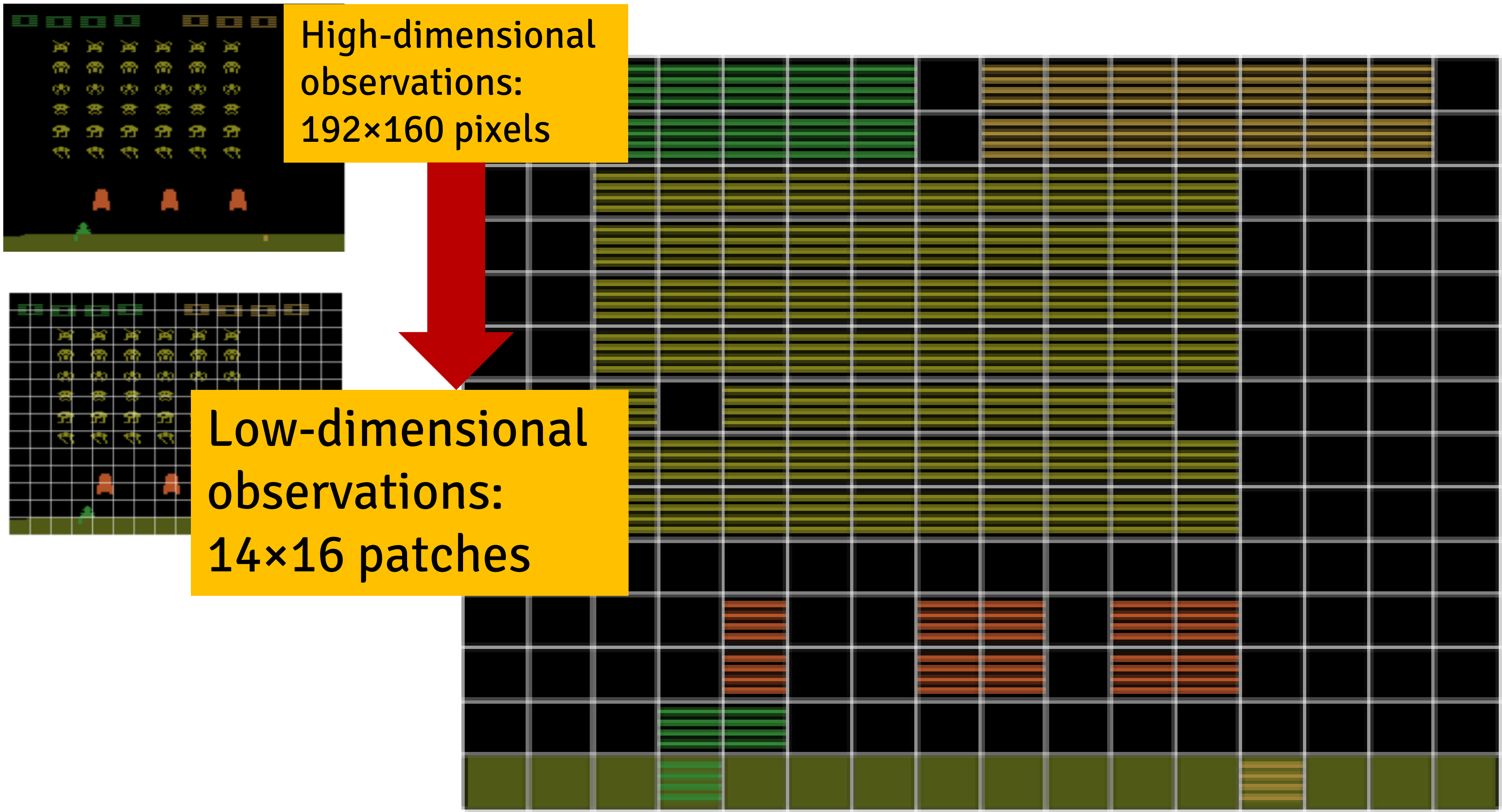
STATE AGGREGATION EXAMPLE: “PROST” FEATURES FOR ATARI GAMES



High-dimensional observations: 192×160 pixels



STATE AGGREGATION EXAMPLE: "PROST" FEATURES FOR ATARI GAMES



TRIVIAL STATE AGGREGATION: THE TABULAR CASE

Strictly generalizes the basic setting:
for each state $x \in X$, introduce a feature ϕ_x as

$$\phi_x(x') = \mathbf{1}\{x = x'\}$$

TRIVIAL STATE AGGREGATION: THE TABULAR CASE

Strictly generalizes the basic setting:
for each state $x \in X$, introduce a feature ϕ_x as

$$\phi_x(x') = \mathbf{1}\{x = x'\}$$

Every function can be represented by these features: $V(x) = \sum_{x'} V(x')\phi_{x'}(x)$

TRIVIAL STATE AGGREGATION: THE TABULAR CASE

Strictly generalizes the basic setting:
for each state $x \in X$, introduce a feature ϕ_x as

$$\phi_x(x') = \mathbf{1}\{x = x'\}$$

Every function can be represented by these features: $V(x) = \sum_{x'} V(x')\phi_{x'}(x)$

Only makes sense when X is finite and small

Value function $\approx$ lookup table

TRIVIAL STATE AGGREGATION: THE TABULAR CASE

Strictly generalizes the basic setting:
for each state $x \in X$, introduce a feature ϕ_x as

$$\phi_x(x') = \mathbf{1}\{x = x'\}$$

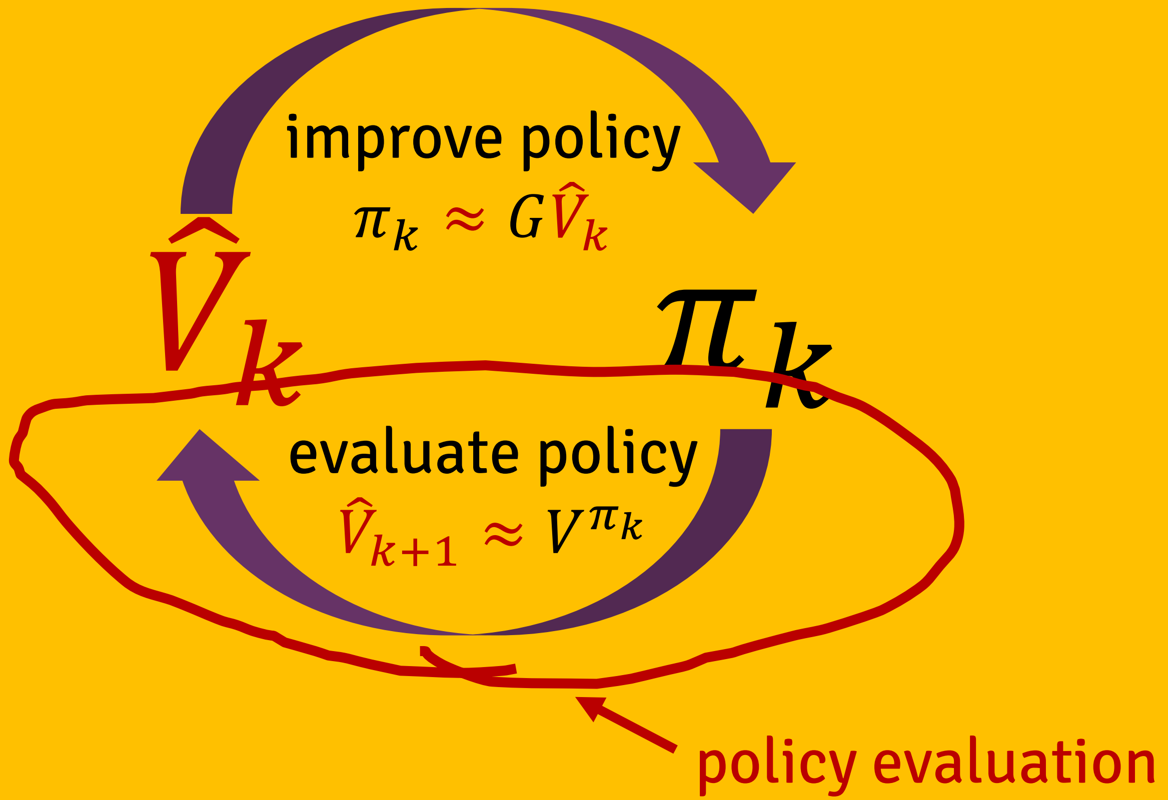
Every function can be represented by these features: $V(x) = \sum_{x'} V(x')\phi_{x'}(x)$

Only makes sense when X is finite and small

Value function $\approx$ lookup table

Basic RL algorithms

1. building blocks of RL methods
2. policy evaluation
   - Monte Carlo
   - temporal difference learning
3. policy evaluation and learning
   - SARSA
   - Q-learning



METHODS FOR POLICY EVALUATION

A GENTLE START: MONTE CARLO

Observe:

Policy evaluation = estimating V^{π}:

$$V^{\pi}(x) = \mathbf{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^{t} r(x_t, a_t) \,|x_0 = x]$$

A GENTLE START: MONTE CARLO

Observe:
Policy evaluation = estimating V^{π}:
$V^{\pi}(x) = \mathbf{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} r(x_t, a_t) \,|x_0 = x\right]$

Idea:
approximate $\mathbf{E}_{\pi}[\cdot]$ by sample averages!

- Simulate N trajectories using policy π
- For every state x that appears in the trajectories, let

$$\hat{V}_N(x) = \mathrm{avg}\big(R_{1:N}(x)\big)$$

A GENTLE START: MONTE CARLO

Idea:
approximate $\mathbf{E}_\pi[\cdot]$ by sample averages!

- Simulate N trajectories using policy π
- For every state x that appears in the trajectories, let
$$\hat{V}_N(x) = \mathrm{avg}\big(R_{1:N}(x)\big)$$

A GENTLE START: MONTE CARLO

Idea:
approximate $\mathbf{E}_{\pi}[\cdot]$ by sample averages!

- Simulate N trajectories using policy π
- For every state x that appears in the trajectories, let
$$\hat{V}_N(x) = \operatorname{avg}\big(R_{1:N}(x)\big)$$

Collection of discounted returns $\sum_{t=0}^{T'} \gamma^t r_t$ after first visit to x

A GENTLE START: MONTE CARLO

Idea:
approximate $\mathbf{E}_\pi[\cdot]$ by sample averages!

- Simulate N trajectories using policy π
- For every state x that appears in the trajectories, let

$$\hat{V}_N(x) = \mathrm{avg}\big(R_{1:N}(x)\big)$$

Average of i.i.d. random variables: $\lim_{N\to\infty} \hat{V}_N = V^\pi$

Collection of discounted returns $\sum_{t=0}^{T'} \gamma^t r_t$ after first visit to x

MONTE CARLO WITH FEATURES

Monte Carlo policy evaluation

Input:

N trajectories $\sim \pi$, feature map $\phi : X \to \mathbb{R}^d$

Output:

$$\hat{V}_N = \arg\min_{\theta \in \mathbb{R}^d} \mathbf{E}_x \left[\sum_{i=1}^{N} \left(\theta^\top \phi(x) - R_i(x) \right)^2 \right]$$

MONTE CARLO WITH FEATURES

Monte Carlo policy evaluation

Input:

N trajectories $\sim \pi$, feature map $\phi: X \to \mathbb{R}^d$

Output:

$$\hat{V}_N = \arg\min_{\theta \in \mathbb{R}^d} \mathbf{E}_x \left[\sum_{i=1}^{N} \left(\theta^\top \phi(x) - R_i(x) \right)^2 \right]$$

Least-squares fit of discounted returns

Recovers the previous definition:

$$\arg\min_v \sum_i \left(v(x) - R_i(x) \right)^2 = \operatorname{avg}\left(R_{1:N}(x) \right)$$

MONTE CARLO EXAMPLE: BLACKJACK

MONTE CARLO EXAMPLE: BLACKJACK

In each episode:

- Start with 2 cards
- Observe 1 showing card of dealer
- Decide to either hit (draw another card) or stick
- Episode ends when you stick or go bust (sum over 21)
- Outcome:
    - Dealer’s sum $<$ your sum $\Rightarrow$ win
    - Dealer’s sum $=$ your sum $\Rightarrow$ draw
    - Dealer’s sum $>$ your sum $\Rightarrow$ lose

MONTE CARLO EXAMPLE: BLACKJACK

MDP formulation:

- State description:
- current sum of player (12-21)
- dealer's showing card (1-10)
- presence of usable ace

- Actions:
- hit
- stick

- Rewards:
- lose: -1
- draw: 0
- win: 1

- Transition dynamics:
- random initial state
- dealer hits below 17
- infinite deck

In each episode:

- Start with 2 cards
- Observe 1 showing card of dealer
- Decide to either hit (draw another card) or stick
- Episode ends when you stick or go bust (sum over 21)
- Outcome:
    - Dealer's sum < your sum ⇒ win
    - Dealer's sum = your sum ⇒ draw
    - Dealer's sum > your sum ⇒ lose

MDP formulation:

- State description:
  - current sum of player (12-21)
  - dealer's showing card (1-10)
  - presence of usable ace
- Actions:
  - hit
  - stick
- Rewards:
  - lose: -1
  - draw: 0
  - win: 1
- Transition dynamics:
  - random initial state
  - dealer hits below 17
  - infinite deck

In each episode:

- Start with 2 cards
- Observe 1 showing card of dealer
- Decide to either hit (draw another card) or stick
- Episode ends when you stick or go bust (sum over 21)
- Outcome:
  - ⇒ win
  - ⇒ draw
  - ⇒ lose

Transition function P is complicated to represent ☹

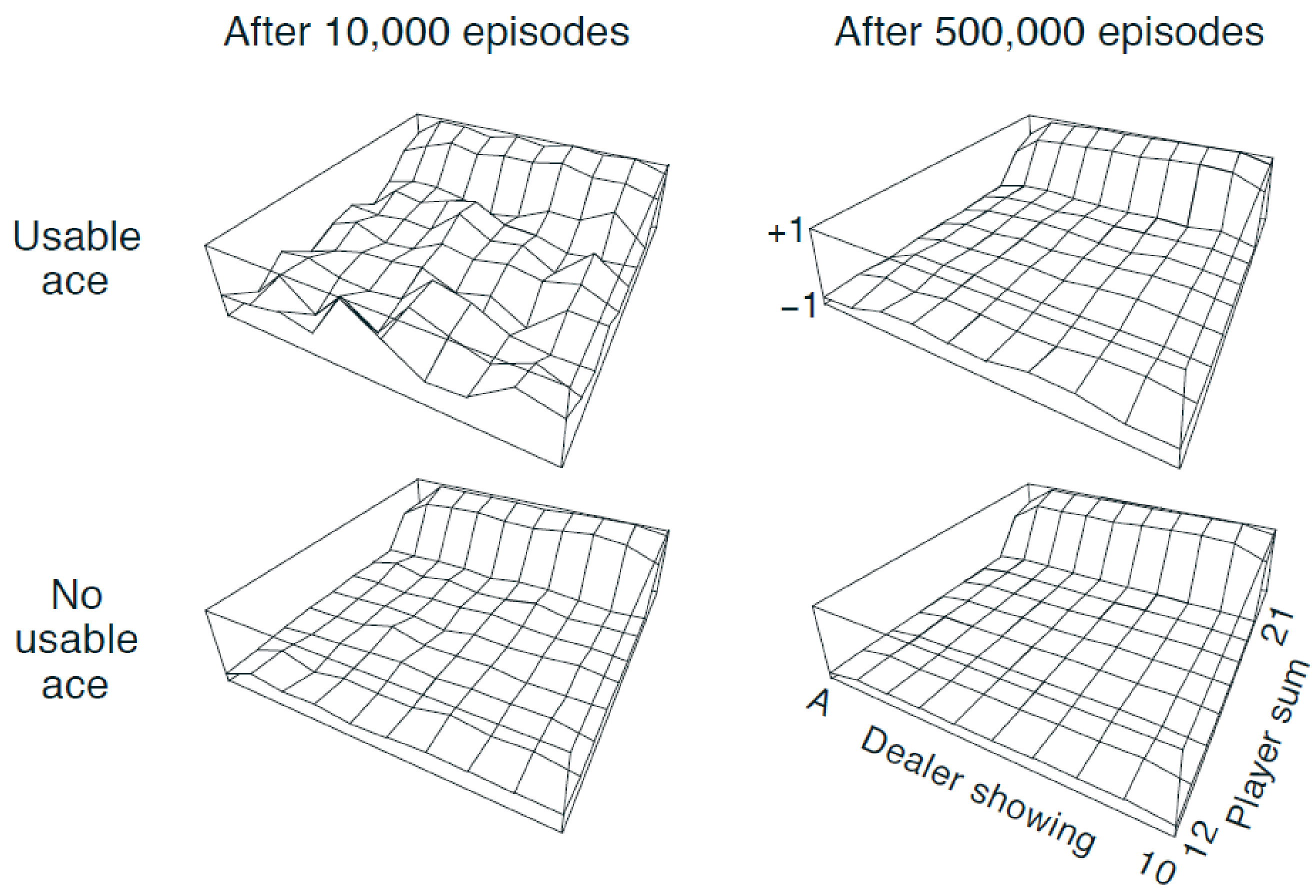
BUT

sample episodes are easy to generate! ☺

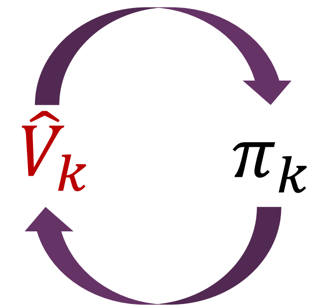
MONTE CARLO EXAMPLE: BLACKJACK

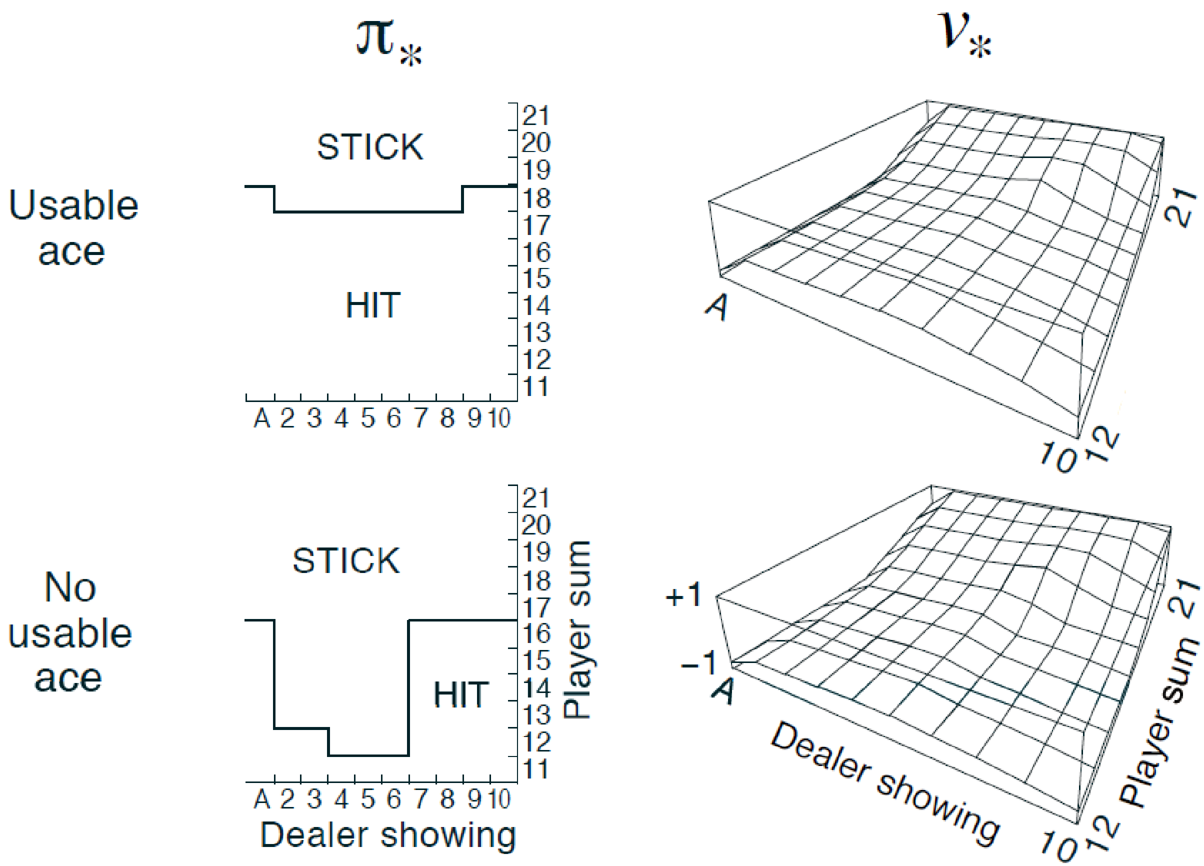
MONTE CARLO EXAMPLE: BLACKJACK

Value of policy that always hits below 20



MONTE CARLO POLICY ITERATION FOR BLACKJACK





PROPERTIES OF MONTE CARLO

☺ Value estimates converge to true values ☺

☺ Doesn't need prior knowledge of P or r ☺

PROPERTIES OF MONTE CARLO

☺ Value estimates converge to true values ☺

☺ Doesn't need prior knowledge of P or r ☺

☹ Doesn't make use of the Bellman equations ☹

Basic RL algorithms

1. building blocks of RL methods
2. policy evaluation
   - Monte Carlo
   - temporal difference learning
3. policy evaluation and learning
   - SARSA
   - Q-learning

A BETTER OBJECTIVE?

Idea: construct an objective that uses the Bellman equations

$$V^{\pi} \approx T^{\pi} V^{\pi}$$

A BETTER OBJECTIVE?

Idea: construct an objective that uses the Bellman equations

$$V^\pi \approx T^\pi V^\pi$$

The Bellman error

$$\Delta_V(x) = T^\pi V(x) - V(x)$$

A BETTER OBJECTIVE?

Idea: construct an objective that uses the Bellman equations

$$V^\pi \approx T^\pi V^\pi$$

The Bellman error

$$\Delta_V(x) = T^\pi V(x) - V(x)$$

Idea: use stochastic approximation to find $\hat{V}$ such that $\mathbf{E}_{x\sim\mu}[\Delta_{\hat{V}}(x)] \approx 0$

TEMPORAL DIFFERENCE LEARNING

TD(0)

Input: arbitrary function $\hat{V}_0 : X \to \mathbf{R}$

For $t = 0,1, \ldots,$

$$\delta_t = r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)$$
$$\hat{V}_{t+1}(x_t) = \hat{V}_t(x_t) + \alpha_t \delta_t$$

TEMPORAL DIFFERENCE LEARNING

TD(0)

Input: arbitrary function $\hat{V}_0 : X \to \mathbf{R}$

For $t = 0, 1, \ldots,$

$$\delta_t = r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)$$
$$\hat{V}_{t+1}(x_t) = \hat{V}_t(x_t) + \alpha_t \delta_t$$

$\delta_t =$ "what I saw" $-$ "what I expected to see"
"target" $-$ "current prediction"

"temporal difference"

THE CONVERGENCE OF TD(0)

Sequence of estimates $(\hat{V}_1, \hat{V}_2, \ldots)$ is a dynamical system

Converges to equilibrium if step-sizes satisfy

$$\sum_{t=0}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

Equilibrium:

$$\mathbf{E}[\delta_t] = \mathbf{E}\left[r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)\right] = 0$$

THE CONVERGENCE OF TD(0)

Sequence of estimates $(\hat{V}_1, \hat{V}_2, \dots)$ is a dynamical system

Converges to equilibrium if step-sizes satisfy

$$\sum_{t=0}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

Equilibrium:

$$\mathbf{E}[\delta_t] = \mathbf{E}\left[r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)\right] = 0$$

$$\boxed{\mathbf{E}[\Delta_{\hat{V}}(x)] = 0}$$

THE CONVERGENCE OF TD(0)

Sequence of estimates $(\hat{V}_1, \hat{V}_2, \dots)$ is a dynamical system

Converges to equilibrium if step-sizes satisfy

$$\sum_{t=0}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

Equilibrium:

$$\mathbf{E}[\delta_t] = \mathbf{E}\left[r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)\right] = 0$$

$$\boxed{\mathbf{E}[\Delta_{\hat{V}}(x)] = 0}$$

Intuition: for small stepsizes, $\hat{V}_t$ tracks the path of the ODE

$$\frac{\mathrm{d}\tilde{V}(t)}{\mathrm{d}t} = (T^\pi - I)\tilde{V}(t)$$

which is globally asymptotically stable with equilibrium $V^\pi = T^\pi V^\pi$

TD(0) WITH LINEAR FUNCTION APPROXIMATION

Let $\phi: X \to \mathbf{R}^d$ be a feature vector

TD(0) WITH LINEAR FUNCTION APPROXIMATION

Let $\phi: X \to \mathbf{R}^d$ be a feature vector

Approximating $V^\pi(x) \approx \theta^\top \phi(x)$ by TD(0):

TD(0) with LFA

Input: arbitrary param. vector $\theta_0 \in \mathbf{R}^d$

For $t = 0,1, \ldots,$

$$\delta_t = r_t + \gamma \theta_t^\top \phi(x_{t+1}) - \theta_t^\top \phi(x_t)$$

$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \phi(x_t)$$

TD(0) WITH LINEAR FUNCTION APPROXIMATION

Let $\phi\colon X \to \mathbf{R}^d$ be a feature vector

Approximating $V^\pi(x) \approx \theta^\top \phi(x)$ by TD(0):

TD(0) with LFA

Input: arbitrary param. vector $\theta_0 \in \mathbf{R}^d$

For $t = 0,1, \dots,$

$$\delta_t = r_t + \gamma \theta_t^\top \phi(x_{t+1}) - \theta_t^\top \phi(x_t)$$
$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \phi(x_t)$$

This still converges!!!

to the projection of V^π to the span of the features

TD(0) VS. SGD FOR LINEAR REGRESSION

Linear least-squares regression:

- Data set: $(\phi_t, y_t)_{t=1}^{n}$ drawn from distribution $\mathcal{D}$ with feature vectors $\phi_t \in \mathbb{R}^d$, and outputs y_t
- Goal: find θ^* minimizing
  $L(\theta) = \mathbb{E}_{\phi,y\sim\mathcal{D}}[\ell(\theta;\phi,y)] = \mathbb{E}_{\phi,y\sim\mathcal{D}}[(\phi^\top\theta - y)^2]$

TD(0) VS. SGD FOR LINEAR REGRESSION

Linear least-squares regression:

- Data set: $(\phi_t, y_t)_{t=1}^n$ drawn from distribution $\mathcal{D}$ with feature vectors $\phi_t \in \mathbb{R}^d$, and outputs y_t
- Goal: find θ^* minimizing $L(\theta) = \mathbb{E}_{\phi,y\sim\mathcal{D}}[\ell(\theta;\phi,y)] = \mathbb{E}_{\phi,y\sim\mathcal{D}}[(\phi^\top\theta - y)^2]$

SGD for linear regression

Input: arbitrary param. vector $\theta_0 \in \mathbf{R}^d$

For $t = 0,1,\dots,$

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha_t \nabla\ell(\theta_t;\phi_t,y_t)\\ &= \theta_t + \alpha_t(\phi_t^\top\theta_t - y_t)\phi_t\end{aligned}$$

TD(0) VS. SGD FOR LINEAR REGRESSION

Linear least-squares regression:

- Data set: $(\phi_t, y_t)_{t=1}^{n}$ drawn from distribution $\mathcal{D}$ with feature vectors $\phi_t \in \mathbb{R}^d$, and outputs y_t
- Goal: find θ^* minimizing $L(\theta) = \mathbb{E}_{\phi,y\sim\mathcal{D}}[\ell(\theta;\phi,y)] = \mathbb{E}_{\phi,y\sim\mathcal{D}}[(\phi^\top\theta - y)^2]$

SGD for linear regression

Input: arbitrary param. vector $\theta_0 \in \mathbf{R}^d$

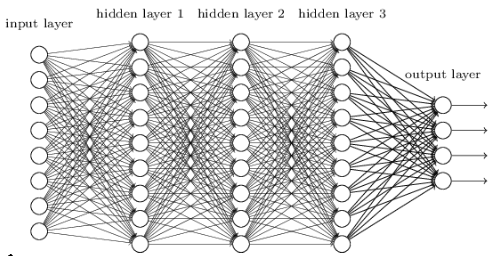
For $t = 0,1,\ldots,$

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha_t \nabla\ell(\theta_t;\phi_t,y_t)\\ &= \theta_t + \alpha_t(\phi_t^\top\theta_t - y_t)\phi_t\end{aligned}$$

$\delta_t = (\phi_t^\top\theta_t - y_t) =$ "prediction error"

TD(0) WITH NONLINEAR FUNCTION APPROXIMATION

Let $V_\theta: X \to R$ be a parametric class of functions (e.g., deep neural network)



Approximating $V^\pi(x) \approx V_\theta(x)$ by TD(0):

<u>TD(0) with general FA</u>

Input: arbitrary param. vector $\theta_0 \in \mathbf{R}^d$

For $t = 0, 1, \ldots,$

$$\delta_t = r_t + \gamma V_{\theta_t}(x_{t+1}) - V_{\theta_t}(x_t)$$

$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \nabla_\theta V_{\theta_t}(x_t)$$

TD(0) WITH NONLINEAR FUNCTION APPROXIMATION

Let $V_\theta : X \to R$ be a p
functions (e.g., deep

Not much is known about convergence ☹

Approximating $V^\pi(x) \approx V_\theta(x)$ by TD(0):

TD(0) with general FA

Input: arbitrary param. vector $\theta_0 \in \mathbf{R}^d$

For $t = 0,1, \dots,$

$$\delta_t = r_t + \gamma V_{\theta_t}(x_{t+1}) - V_{\theta_t}(x_t)$$
$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \nabla_\theta V_{\theta_t}(x_t)$$

PROPERTIES OF TD(0)

☺ Value estimates converge to true values ☺

☺ Doesn't need prior knowledge of P or r ☺

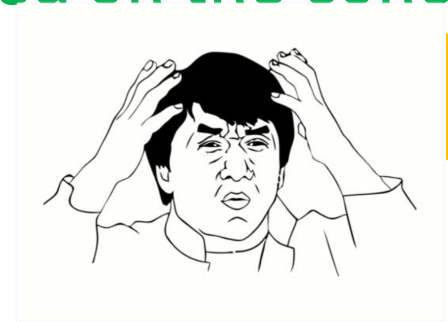
☺ Based on the concept of Bellman error ☺

PROPERTIES OF TD(0)

☺ Value estimates converge to true values ☺

☺ Doesn't need prior knowledge of P or r ☺

☺ Based on the concept of Bellman error ☺



= "bootstrapping"

PROPERTIES OF TD(0)

☺ Value estimates converge to true values ☺

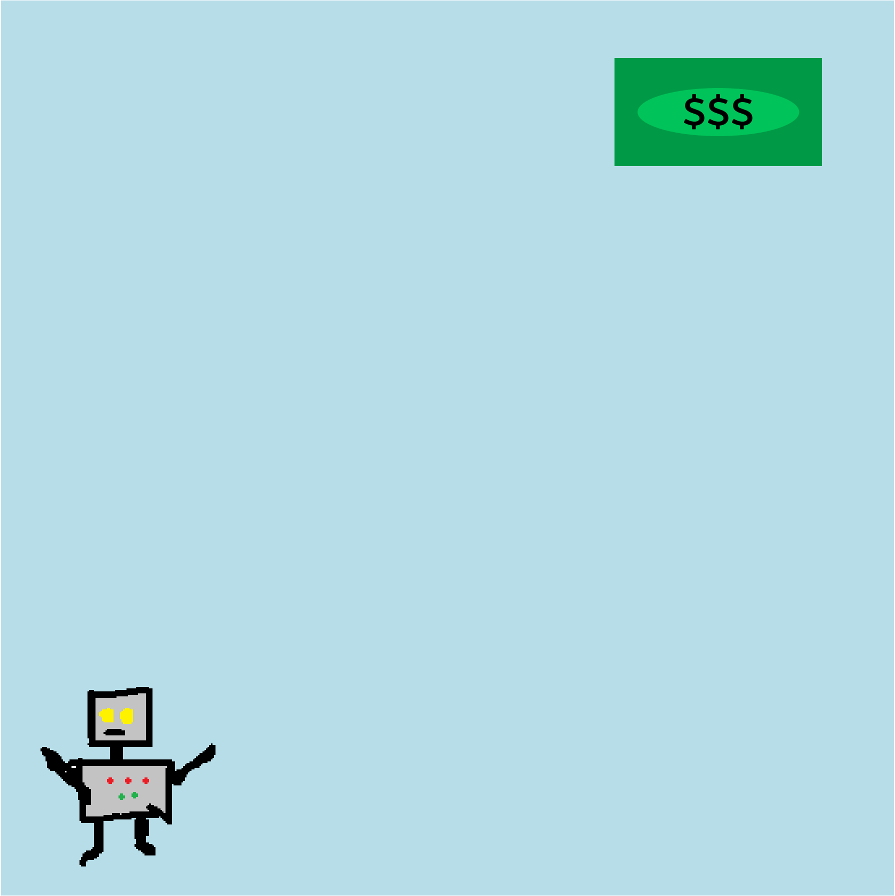
☺ … r ☺

+ new problem:
updates propagate very slowly!

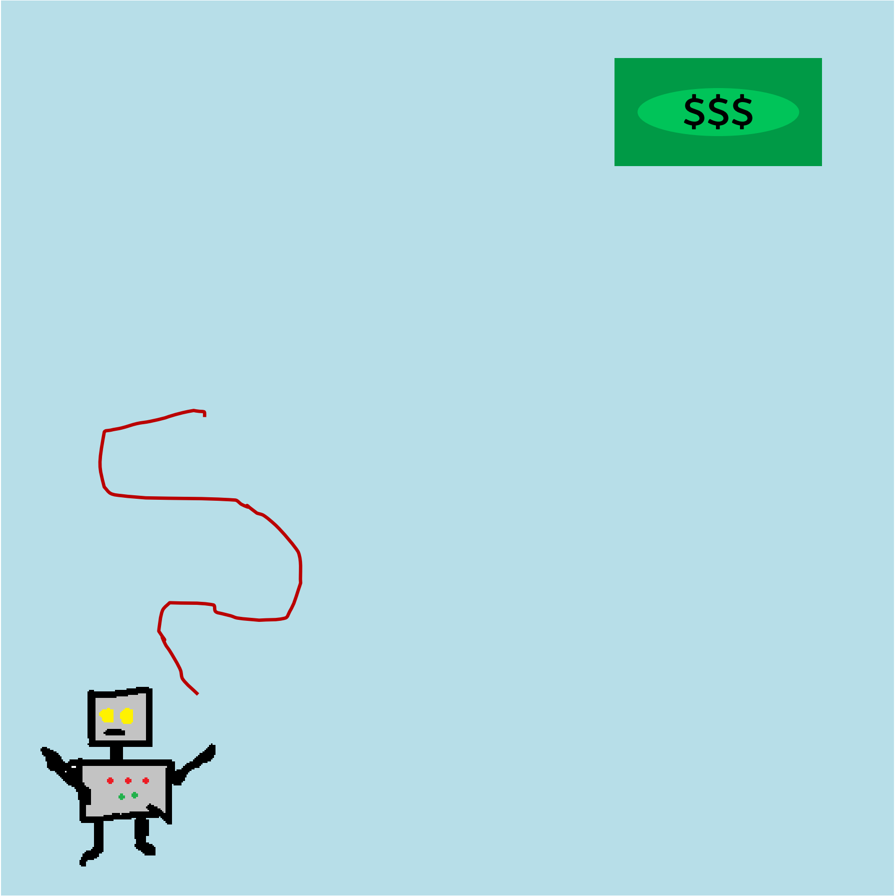
☺ Based on the concept of Bellman error ☺

= "bootstrapping"

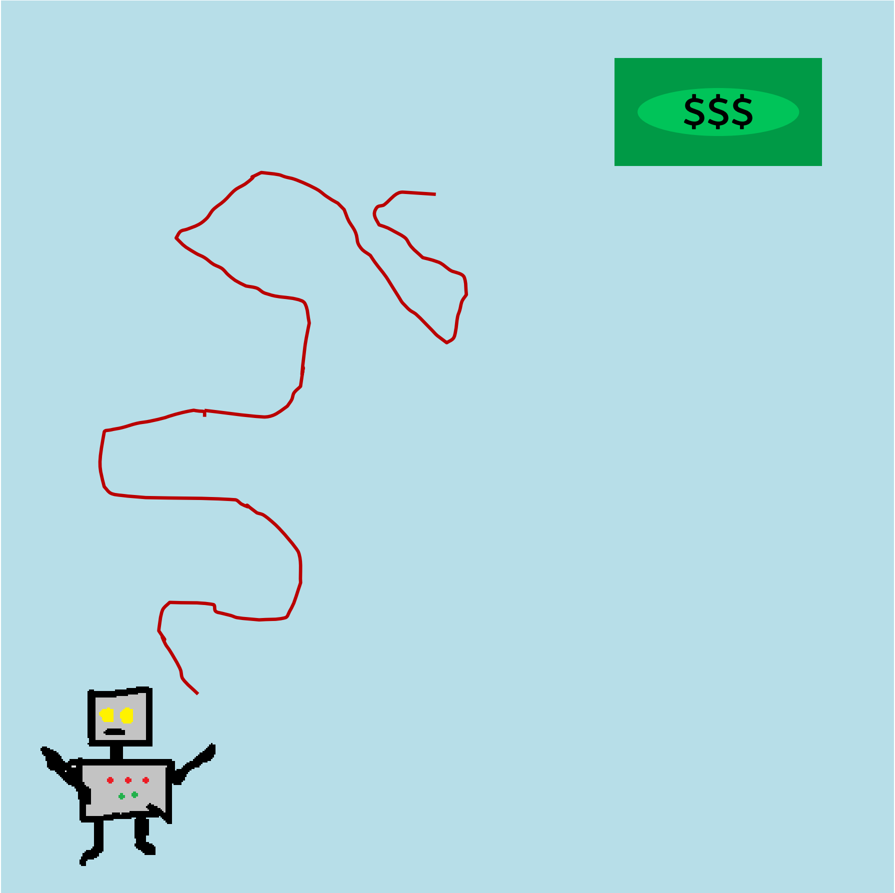
EXAMPLE



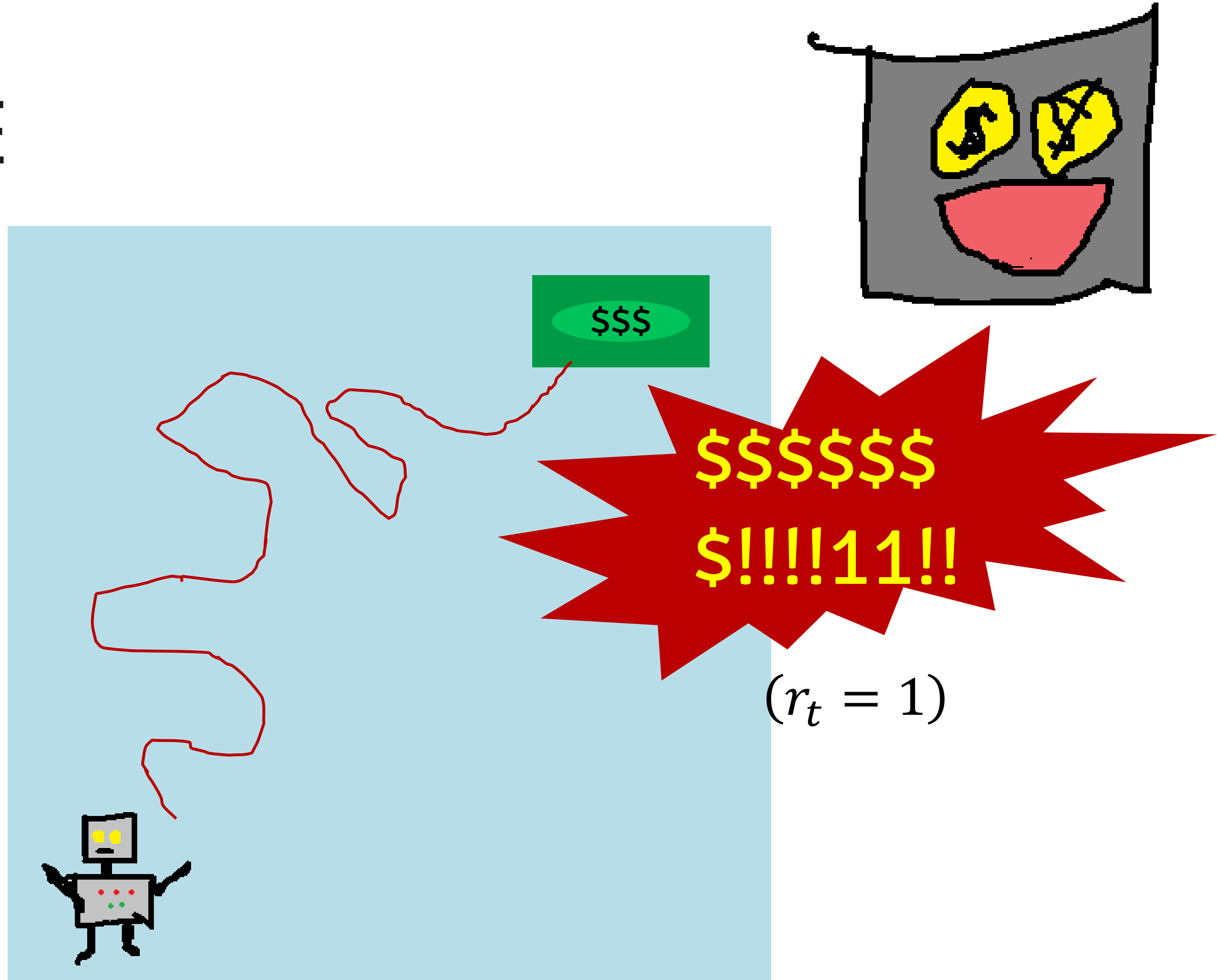
EXAMPLE



EXAMPLE

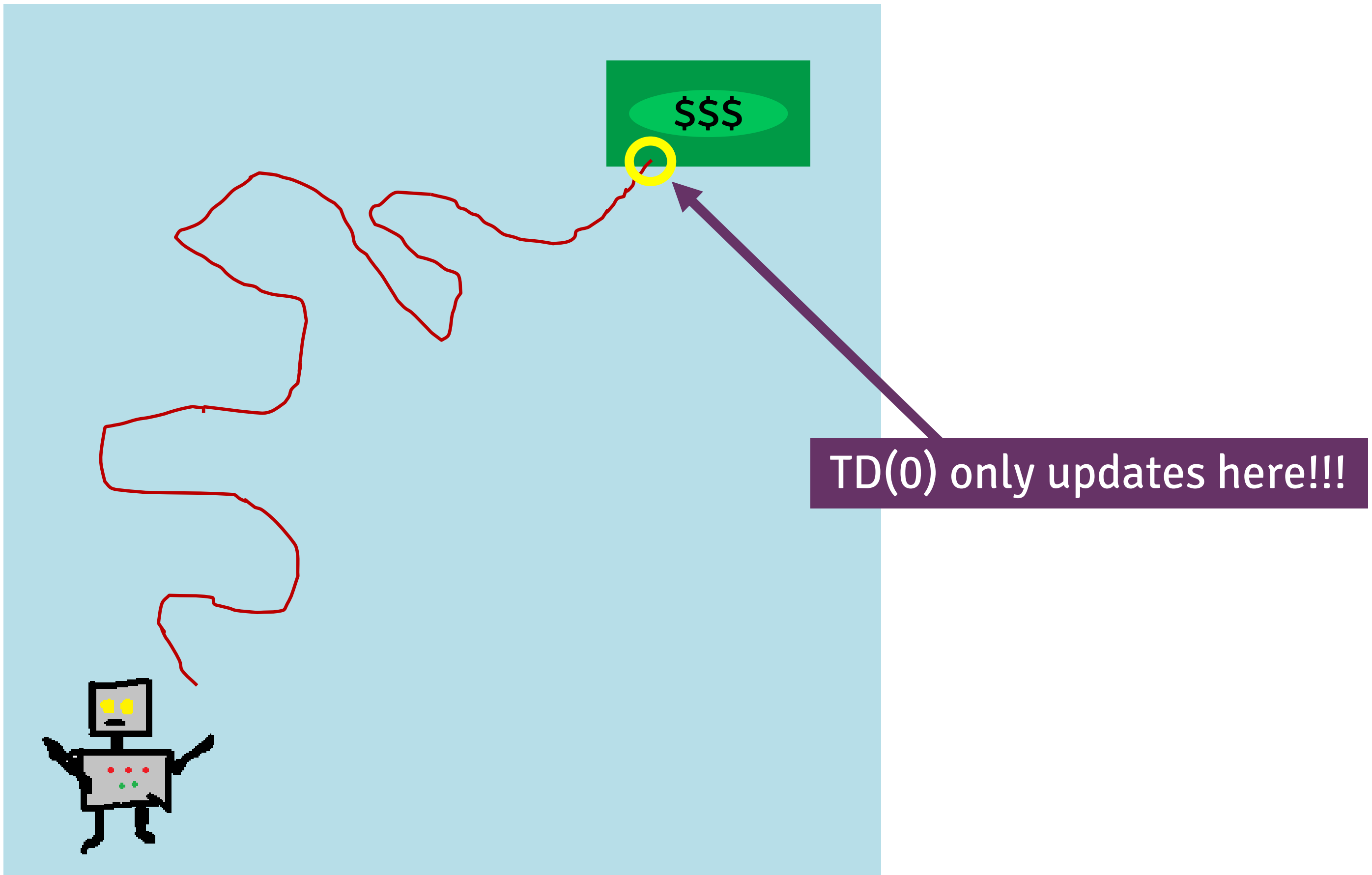


EXAMPLE

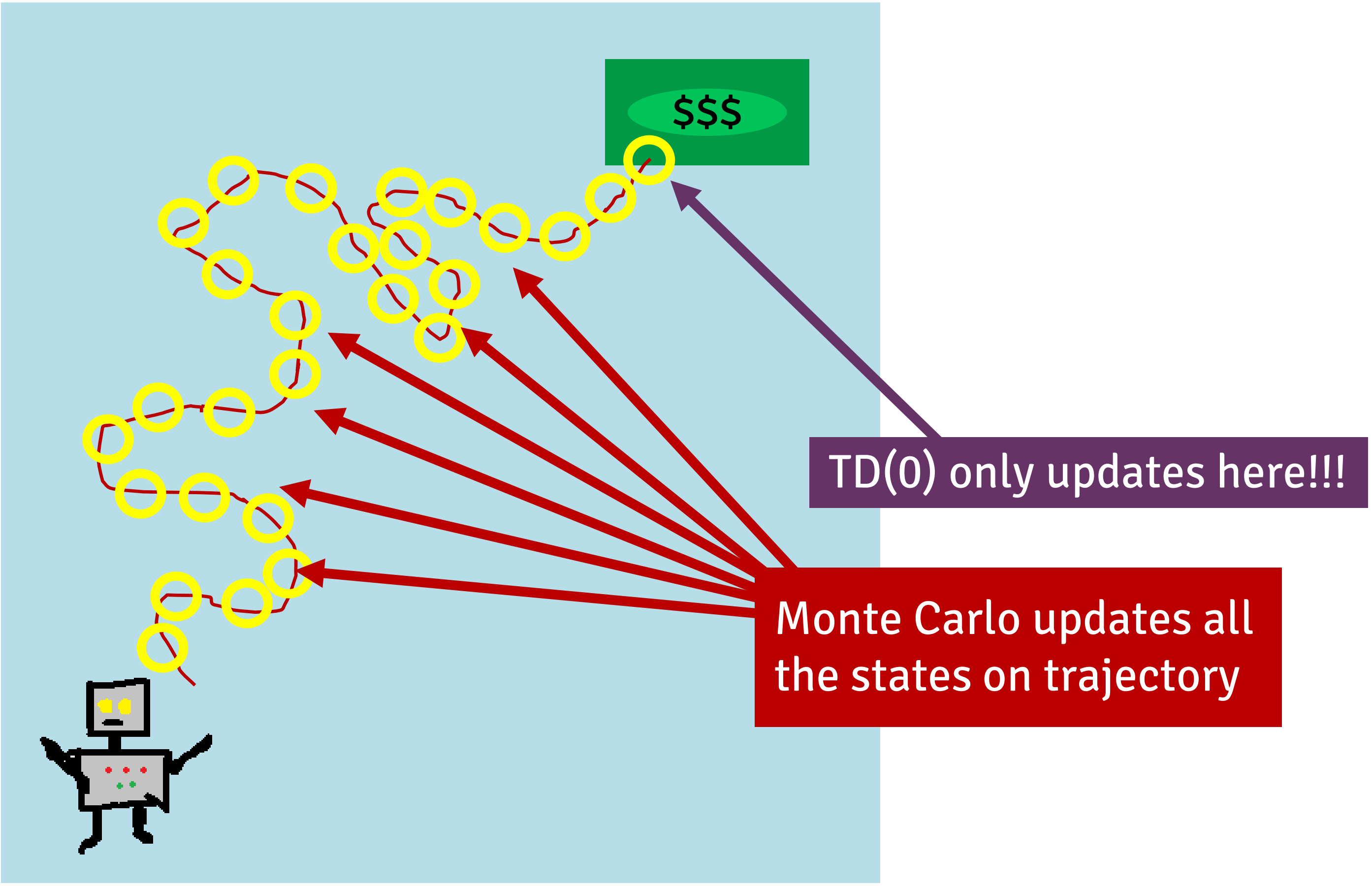


$(r_t = 1)$

EXAMPLE



EXAMPLE


$$$
TD(0) only updates here!!!
Monte Carlo updates all the states on trajectory

BETWEEN MONTE CARLO AND TD(0): TD(λ)

Idea: propagate the updates δ_t to all previous states on the trajectory

Between Monte Carlo and TD(0): TD(λ)

Idea: propagate the updates δ_t to all previous states on the trajectory

- Update in x_t: $\hat{V}_{t+1}(x_t) = \hat{V}_t(x_t) + \alpha_t \delta_t$

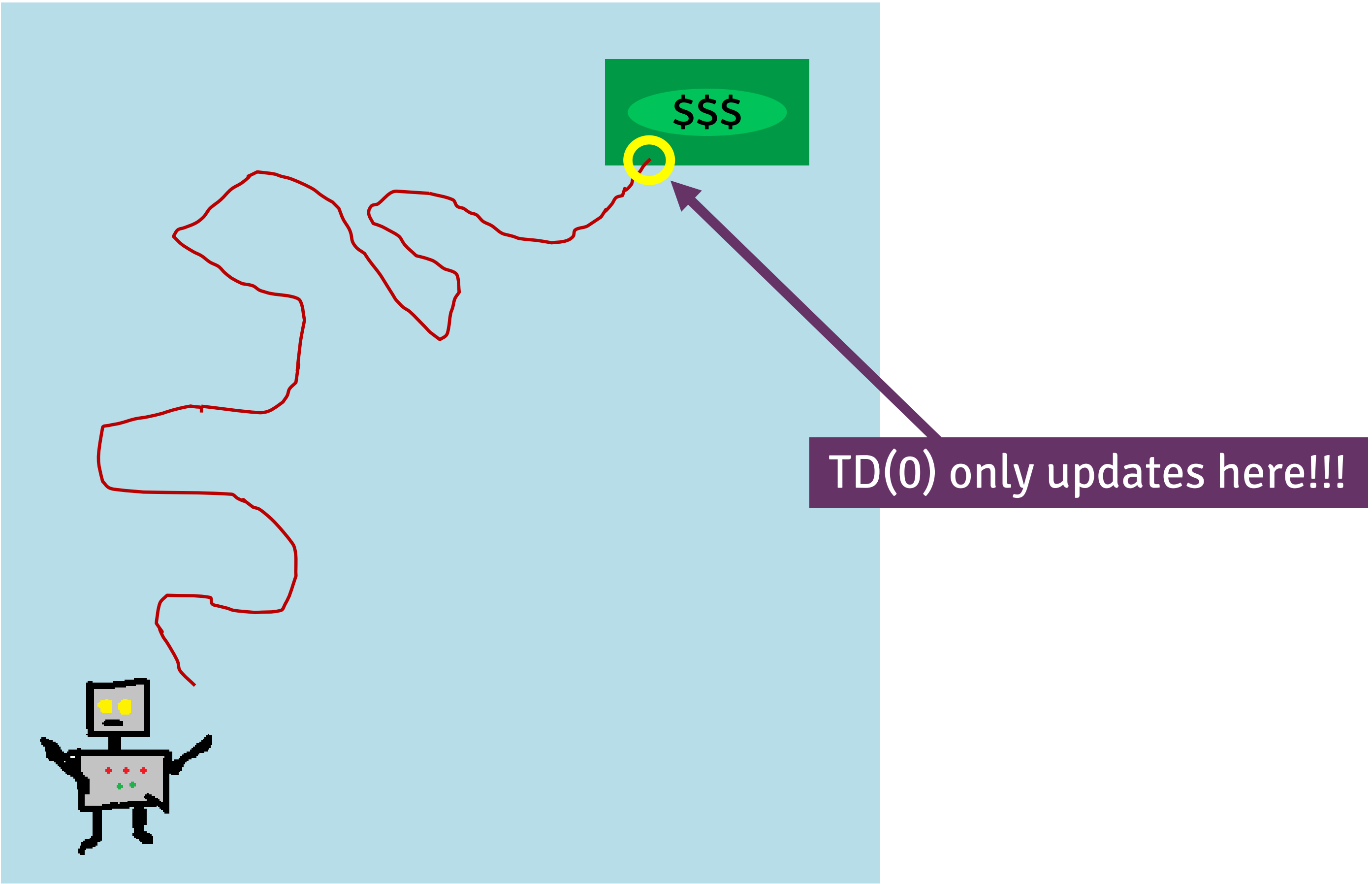
BETWEEN MONTE CARLO AND TD(0): TD(λ)

Idea: propagate the updates δ_t to all previous states on the trajectory

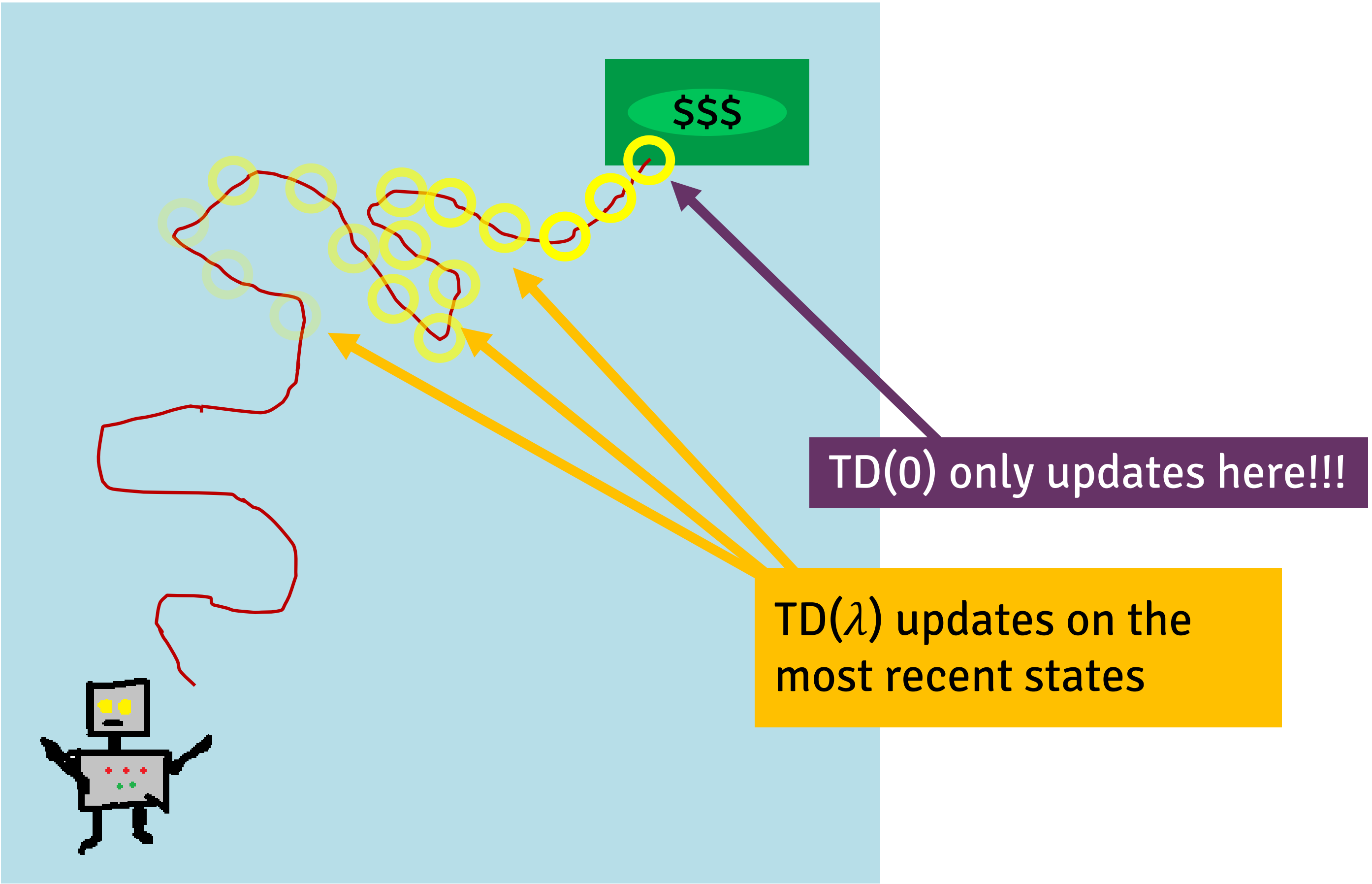
- Update in x_t: $\hat{V}_{t+1}(x_t) = \hat{V}_t(x_t) + \alpha_t \delta_t$
- Update in x_{t-1}: $\hat{V}_{t+1}(x_{t-1}) = \hat{V}_t(x_{t-1}) + \alpha_t \gamma\lambda \delta_t$
- Update in x_{t-2}: $\hat{V}_{t+1}(x_{t-2}) = \hat{V}_t(x_{t-2}) + \alpha_t (\gamma\lambda)^2 \delta_t$
- …
- Update in x_{t-k}: $\hat{V}_{t+1}(x_{t-k}) = \hat{V}_t(x_{t-k}) + \alpha_t (\gamma\lambda)^k \delta_t$

$$\lambda \in [0,1]$$

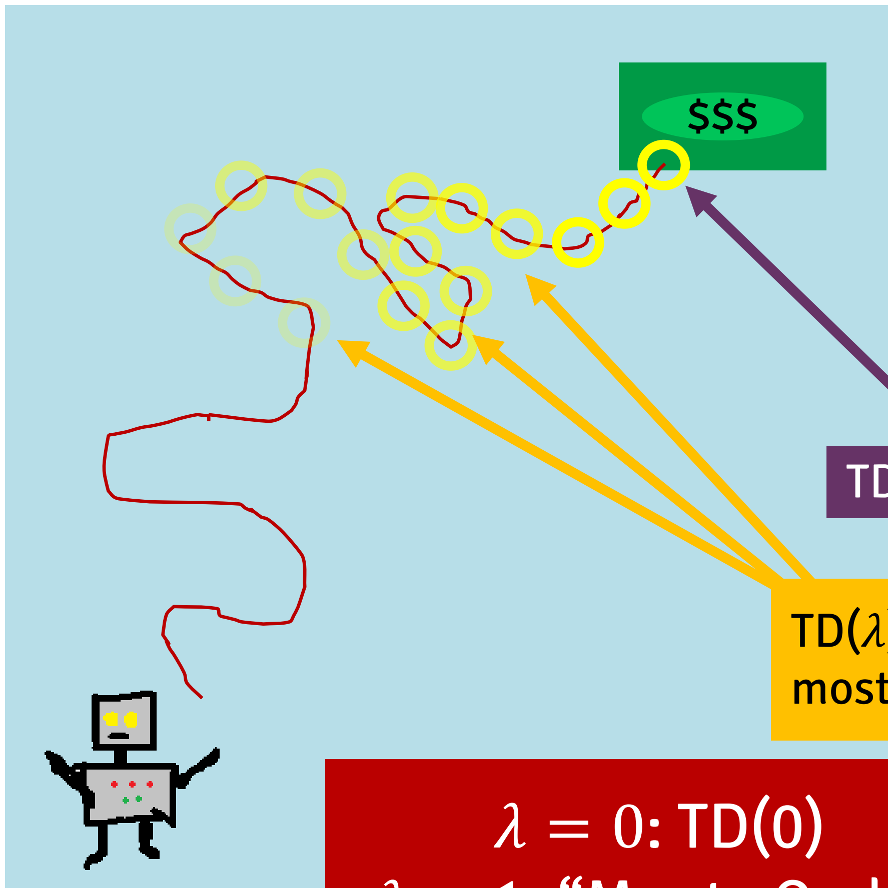
EXAMPLE



EXAMPLE


$$$
TD(0) only updates here!!!
TD(λ) updates on the most recent states

EXAMPLE



TD(0) only updates here!!!

TD(λ) updates on the most recent states

$\lambda = 0$: TD(0)
$\lambda = 1$: "Monte Carlo"

TD(λ) WITH FUNCTION APPROXIMATION

TD(0) with LFA

Input: arbitrary param. vector $\theta_0 \in \mathbf{R}^d$

For $t = 0,1, \dots,$

$$\delta_t = r_t + \gamma\theta_t^\top \phi(x_{t+1}) - \theta_t^\top \phi(x_t)$$
$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \phi(x_t)$$

TD(λ) WITH FUNCTION APPROXIMATION

TD(λ) with LFA

Input: arbitrary param. vector $\theta_0 \in \mathbf{R}^d$

For $t = 0, 1, \ldots,$

$$z_t = \gamma\lambda z_{t-1} + \phi(x_t)$$

$$\delta_t = r_t + \gamma\theta_t^\top\phi(x_{t+1}) - \theta_t^\top\phi(x_t)$$

$$\theta_{t+1} = \theta_t + \alpha_t\delta_t z_t$$

TD(λ) WITH FUNCTION APPROXIMATION

TD(λ) with LFA

Input: arbitrary param. vector $\theta_0 \in \mathbf{R}^d$

For $t = 0,1, \dots,$

$$z_t = \gamma \lambda z_{t-1} + \phi(x_t)$$

$$\delta_t = r_t + \gamma \theta_t^\top \phi(x_{t+1}) - \theta_t^\top \phi(x_t)$$

$$\theta_{t+1} = \theta_t + \alpha_t \delta_t z_t$$

z_t: "eligibility trace"

TD(λ) WITH FUNCTION APPROXIMATION

TD(λ) with LFA

Input: arbitrary param. vector $\theta_0 \in \mathbf{R}^d$

For $t = 0,1,\ldots,$

$$z_t = \gamma\lambda z_{t-1} + \phi(x_t)$$

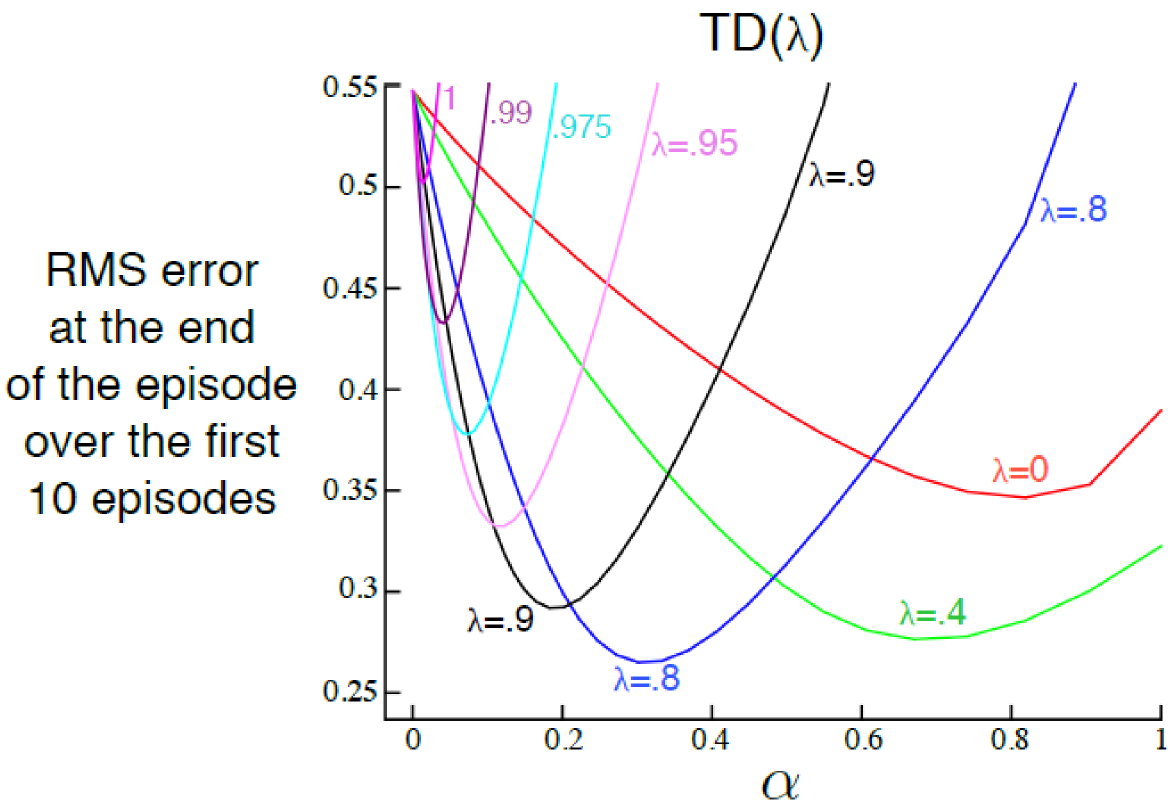
$$\delta_t = r_t + \gamma\theta_t^\top\phi(x_{t+1}) - \theta_t^\top\phi(x_t)$$

$$\theta_{t+1} = \theta_t + \alpha_t\delta_t z_t$$

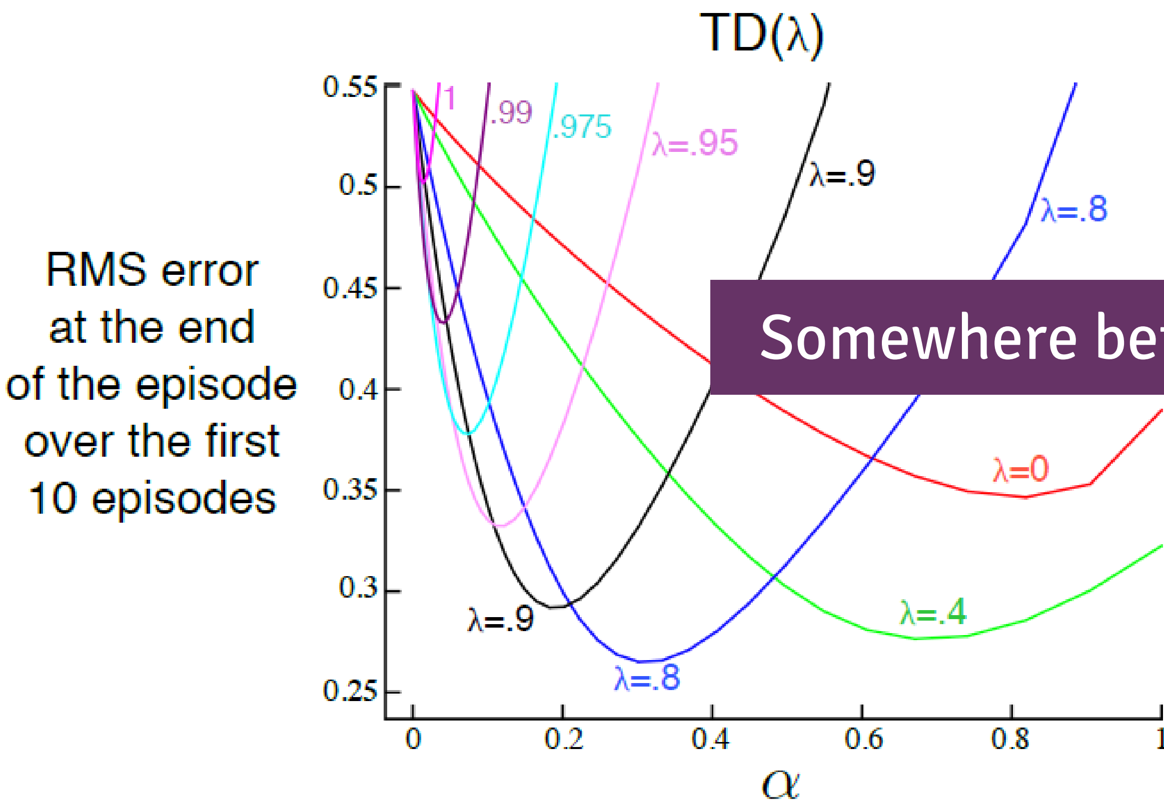
z_t: “eligibility trace”

Equilibrium:
$\mathbf{E}[\delta_t z_t] = 0$

WHAT IS THE RIGHT CHOICE OF λ?

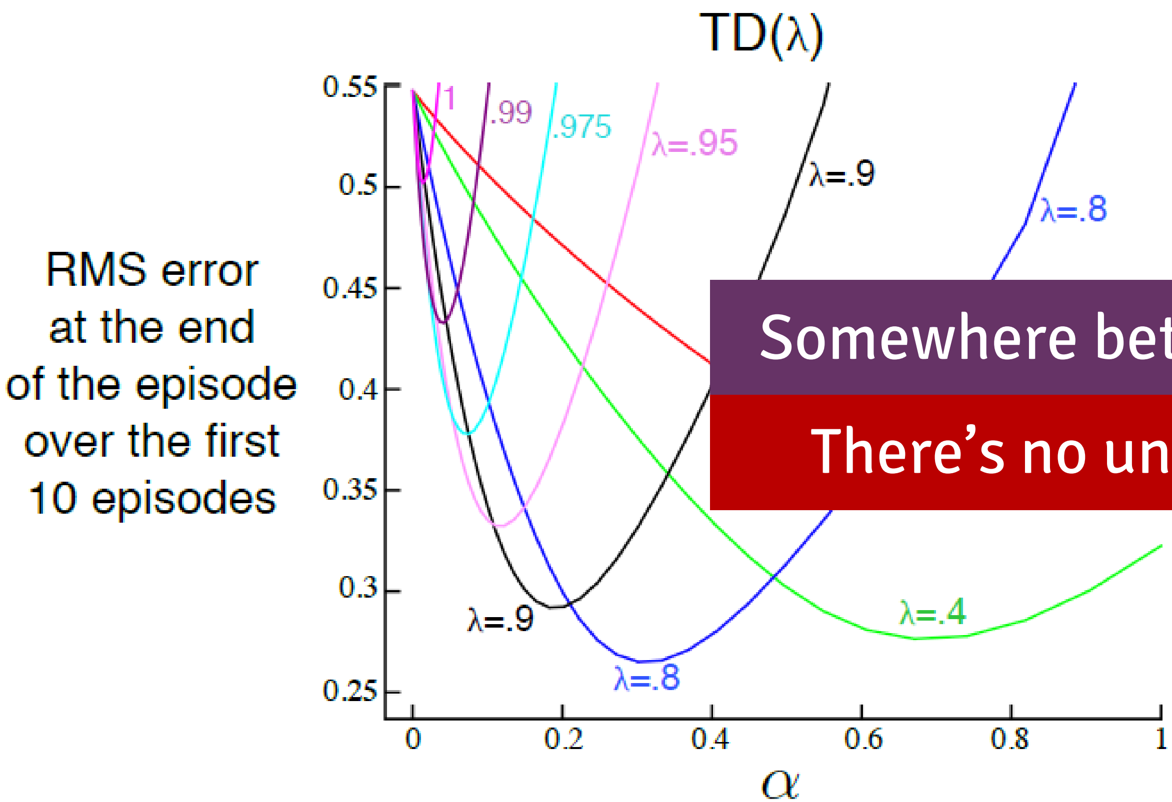


WHAT IS THE RIGHT CHOICE OF λ?



Somewhere between 0 and 1 ☺

WHAT IS THE RIGHT CHOICE OF λ?

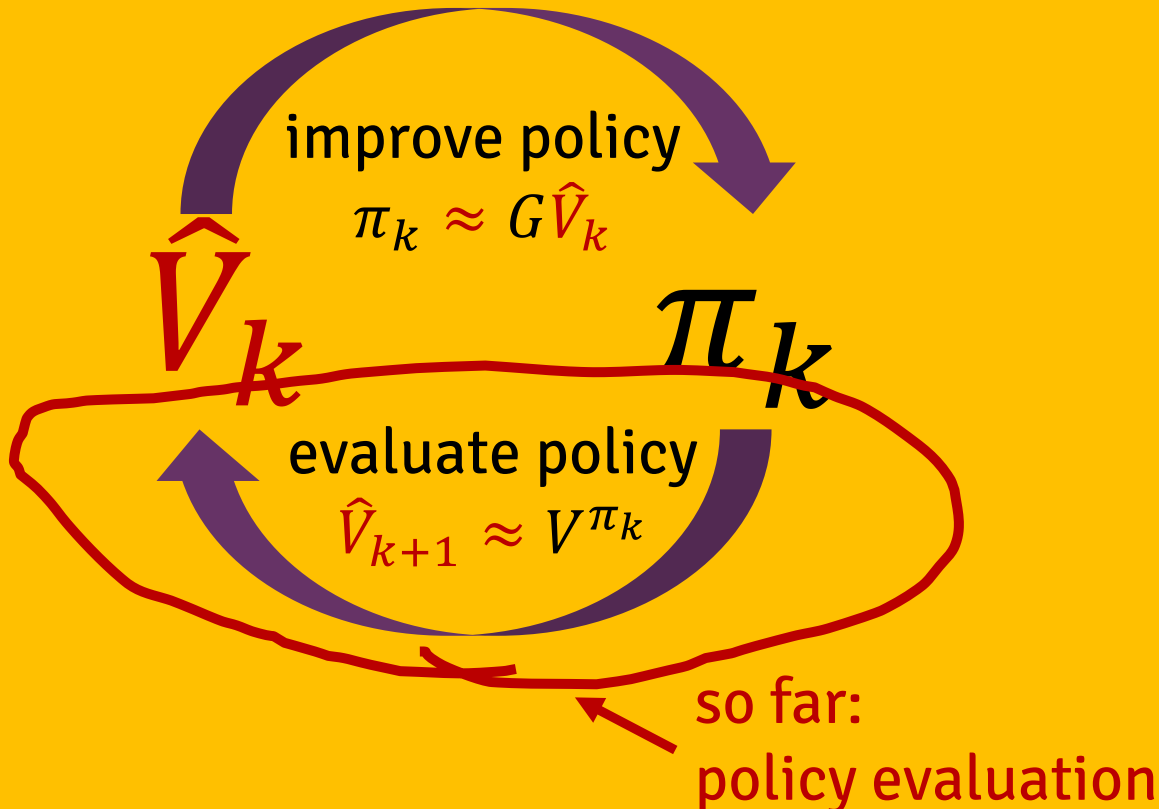


Somewhere between 0 and 1 ☺

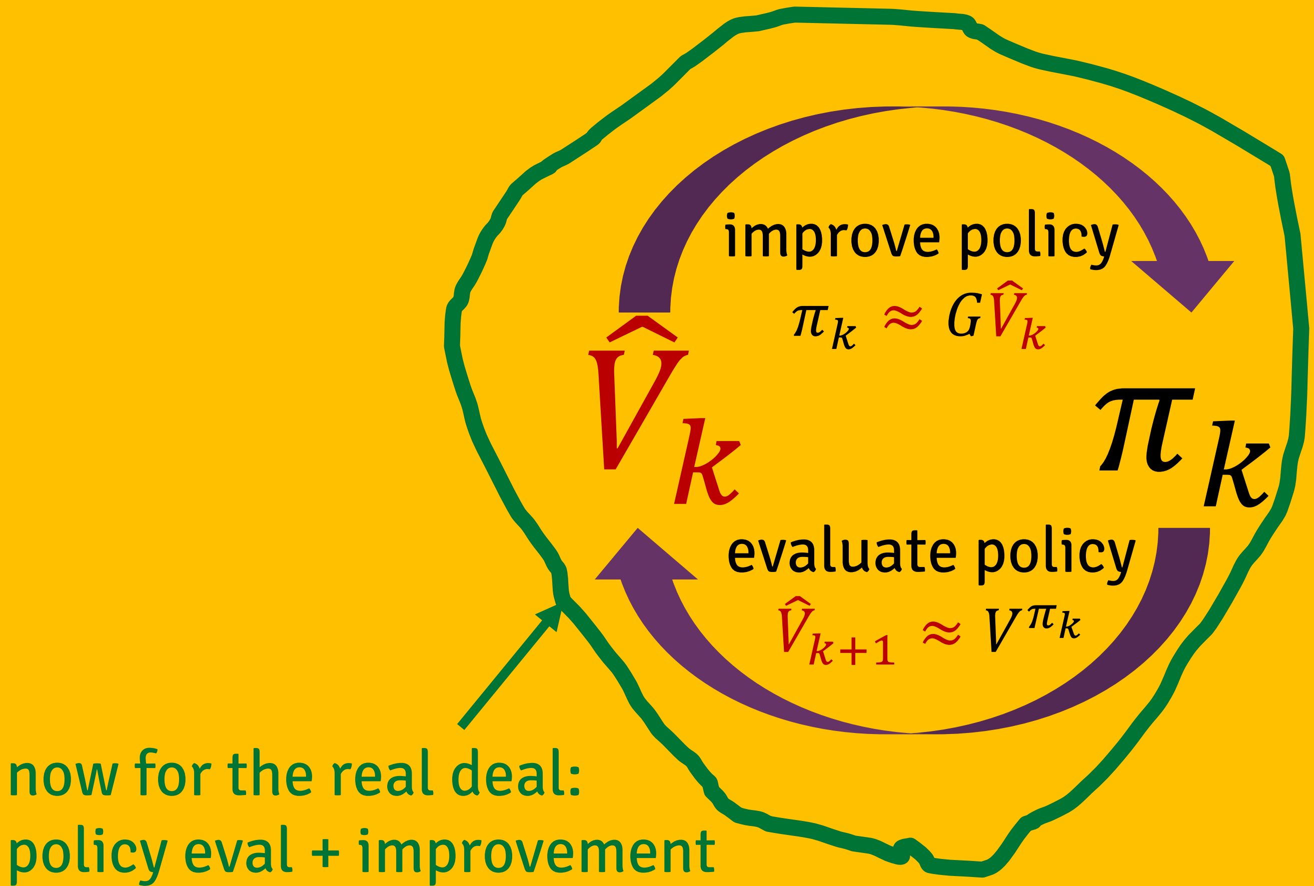
There's no universal recipe!

Basic RL algorithms

1. building blocks of RL methods
2. policy evaluation
   - Monte Carlo
   - temporal difference learning
3. policy evaluation and learning
   - SARSA
   - Q-learning



FROM POLICY EVALUATION POLICY IMPROVEMENT



FROM POLICY EVALUATION POLICY IMPROVEMENT

CONTROL WHILE LEARNING: SARSA

Idea: Let's try to improve the policy on the fly, based on the current value estimates!

CONTROL WHILE LEARNING: SARSA

Idea: Let's try to improve the policy on the fly, based on the current value estimates!

Changes to the TD recipe:

- Estimate the Q-function instead of V
- Compute ε-greedy policy w.r.t. $\hat{Q}_t$:

$$\pi_t(x) = \begin{cases} \arg\max \hat{Q}_t(x,a)\,, & \text{w.p. } 1-\varepsilon \\ \text{uniform random action,} & \text{w.p. } \varepsilon \end{cases}$$

CONTROL WHILE LEARNING: SARSA

Idea: Let's try to improve the policy on the fly, based on the current value estimates!

Changes to the TD recipe:

- Estimate the Q-function instead of V
- Compute ε-greedy policy w.r.t. $\hat{Q}_t$:

$$\pi_t(x) = \begin{cases} \arg\max \hat{Q}_t(x, a), & \text{w.p. } 1 - \varepsilon \\ \text{uniform random action,} & \text{w.p. } \varepsilon \end{cases}$$

- Optimize the Bellman error

$$\Delta_t = \mathbf{E}\left[r_t + \gamma \hat{Q}_t(x_{t+1}, \pi_t(x_{t+1}))\right] - \hat{Q}_t(x_t, a_t)$$

CONTROL WHILE LEARNING: SARSA

Idea: Let's try to improve the policy on the fly, based on the current value estimates!

Changes to the TD recipe:

- Estimate the Q-function instead of V
- Compute ε-greedy policy w

$$\pi_t(x) = \begin{cases} \arg\max \hat{Q}_t(x \\ \text{uniform rand} \end{cases}$$

- Optimize the Bellman error

$$\Delta_t = \mathbf{E}\left[r_t + \gamma \hat{Q}_t(x_{t+1}, \pi_t(x_{t+1}))\right] - \hat{Q}_t(x_t, a_t)$$

Drifting target policy!

CONTROL WHILE LEARNING: SARSA

SARSA

Input: arbitrary $\hat{Q}_0 : X \times A \to \mathbf{R}$

For $t = 0,1, \dots,$

- Choose action $a_t \sim \varepsilon$-greedy w.r.t. $\hat{Q}_t$
- Observe r_t, x_{t+1}, a'_{t+1}
- Compute

  $\delta_t = r_t + \gamma \hat{Q}_t(x_{t+1}, a'_{t+1}) - \hat{Q}_t(x_t, a_t)$

  $\hat{Q}_{t+1}(x_t, a_t) = \hat{Q}_t(x_t, a_t) + \alpha_t \delta_t$

CONTROL WHILE LEARNING: SARSA

SARSA

Input: arbitrary $\hat{Q}_0 : X \times A \to \mathbf{R}$

For $t = 0,1, \ldots,$

- Choose action $a_t \sim \varepsilon$-greedy w.r.t. $\hat{Q}_t$
- Observe r_t, x_{t+1}, a'_{t+1}
- Compute

  $\delta_t = r_t + \gamma \hat{Q}_t(x_{t+1}, a'_{t+1}) - \hat{Q}_t(x_t, a_t)$

  $\hat{Q}_{t+1}(x_t, a_t) = \hat{Q}_t(x_t, a_t) + \alpha_t \delta_t$

$a'_{t+1} \sim \pi_t$

CONTROL WHILE LEARNING: SARSA

SARSA

Input: arbitrary $\hat{Q}_0 : X \times A \to \mathbf{R}$

For $t = 0,1, \ldots,$

- Choose action $a_t \sim \varepsilon$-greedy w.r.t. $\hat{Q}_t$
- Observe r_t, x_{t+1}, a'_{t+1}
- Compute

$$\delta_t = r_t + \gamma \hat{Q}_t(x_{t+1}, a'_{t+1}) - \hat{Q}_t(x_t, a_t)$$
$$\hat{Q}_{t+1}(x_t, a_t) = \hat{Q}_t(x_t, a_t) + \alpha_t \delta_t$$

$a'_{t+1} \sim \pi_t$

SARSA = $(s_t, a_t, r_t, s_{t+1}, a'_{t+1})$

CONTROL WHILE LEARNING: SARSA

SARSA ~ XARXA

Input: arbitrary $\hat{Q}_0 : X \times A \to \mathbf{R}$

For $t = 0,1, \ldots,$

- Choose action $a_t \sim \varepsilon$-greedy w.r.t. $\hat{Q}_t$
- Observe r_t, x_{t+1}, a'_{t+1}
- Compute

$$\delta_t = r_t + \gamma \hat{Q}_t(x_{t+1}, a'_{t+1}) - \hat{Q}_t(x_t, a_t)$$
$$\hat{Q}_{t+1}(x_t, a_t) = \hat{Q}_t(x_t, a_t) + \alpha_t \delta_t$$

$a'_{t+1} \sim \pi_t$

SARSA = $(s_t, a_t, r_t, s_{t+1}, a'_{t+1})$

TRACKING A DRIFTING POLICY

Bellman error "optimized" by SARSA:

$$\Delta_t = \mathbf{E}\left[r_t + \gamma \hat{Q}_t(x_{t+1}, \pi_t(x_{t+1}))\right] - \hat{Q}_t(x_t, a_t)$$

TRACKING A DRIFTING POLICY

Bellman error "optimized" by SARSA:

$$\Delta_t = \mathbf{E}\left[r_t + \gamma \hat{Q}_t(x_{t+1}, \pi_t(x_{t+1}))\right] - \hat{Q}_t(x_t, a_t)$$

On-policy learning:

target policy $\pi_t =$ behavior policy π_t

TRACKING A DRIFTING POLICY

Bellman error "optimized" by SARSA:

$$\Delta_t = \mathbf{E}\left[r_t + \gamma \hat{Q}_t(x_{t+1}, \pi_t(x_{t+1}))\right] - \hat{Q}_t(x_t, a_t)$$

On-policy learning:

target policy π_t = behavior policy π_t

Multiple challenges for analysis:

- Data distribution drifts as policy keeps changing
- The target function itself drifts as well!
- Behavior policy needs to be increasingly greedy to converge to optimum ($\varepsilon \to 0$ for large t)

OFF-POLICY CONTROL: Q-LEARNING

Idea: Let's try to

- directly learn about Q^*, and
- improve the policy on the fly!

OFF-POLICY CONTROL: Q-LEARNING

Idea: Let's try to

- directly learn about Q^*, and
- improve the policy on the fly!

Off-policy learning:
target policy π^* $\neq$ behavior policy π_t

OFF-POLICY CONTROL: Q-LEARNING

Idea: Let's try to

- directly learn about Q^*, and
- improve the policy on the fly!

Off-policy learning:
target policy $\pi^* \neq$ behavior policy π_t

Bellman error "optimized" by Q-learning:

$$\Delta_t = \mathbf{E}\left[r_t + \gamma \max_a \hat{Q}_t(x_{t+1}, a)\right] - \hat{Q}_t(x_t, a_t)$$

OFF-POLICY CONTROL: Q-LEARNING

Q-learning

Input: arbitrary $\hat{Q}_0: X \times A \to \mathbf{R}$

For $t = 0,1, \ldots,$

- Choose action $a_t \sim \varepsilon$-greedy w.r.t. $\hat{Q}_t$
- Observe r_t, x_{t+1}
- Compute

$$\delta_t = r_t + \gamma \max_a \hat{Q}_t(x_{t+1}, a) - \hat{Q}_t(x_t, a_t)$$

$$\hat{Q}_{t+1}(x_t, a_t) = \hat{Q}_t(x_t, a_t) + \alpha_t \delta_t$$

ON-POLICY VS. OFF-POLICY

On-policy learning:
target policy π_t = behavior policy π_t

Multiple challenges for analysis:

- Data distribution drifts as policy keeps changing
- The target function itself drifts as well!
- Behavior policy needs to be increasingly greedy to converge to optimum

ON-POLICY VS. OFF-POLICY

On-policy learning:
target policy π_t = behavior policy π_t

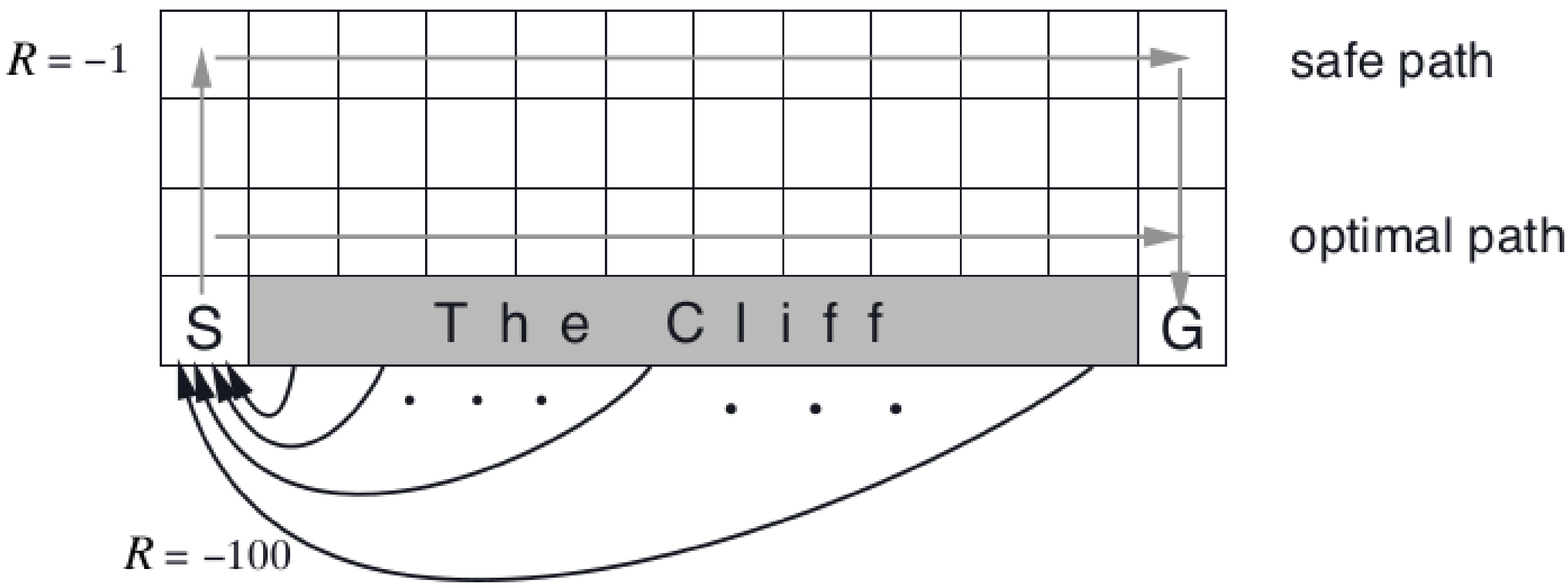
Multiple challenges for analysis:

- Data distribution drifts as policy keeps changing
- The target function itself drifts as well! – a little less
- Behavior policy needs to be increasingly greedy to converge to optimum – a little less

Off-policy learning:
target policy π^* $\neq$ behavior policy π_t

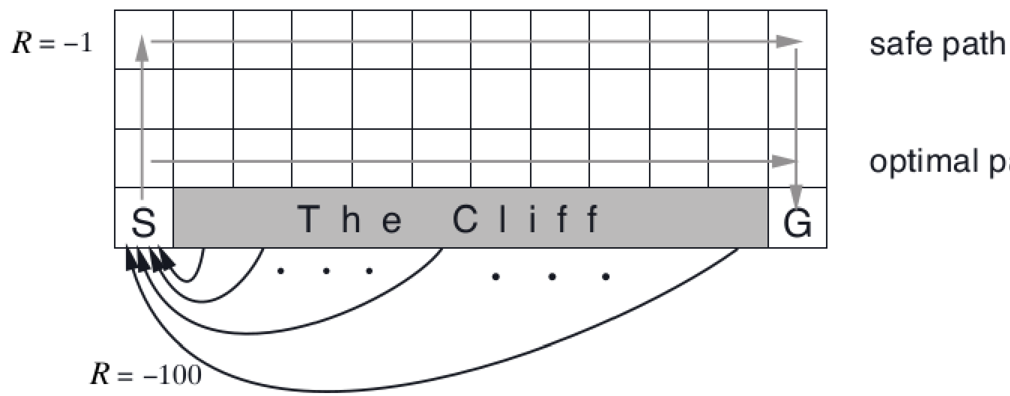
EXAMPLE: Q-LEARNING VS. SARSA

The cliff-walking task:



EXAMPLE: Q-LEARNING VS. SARSA

The cliff-walking task:



Q-learning

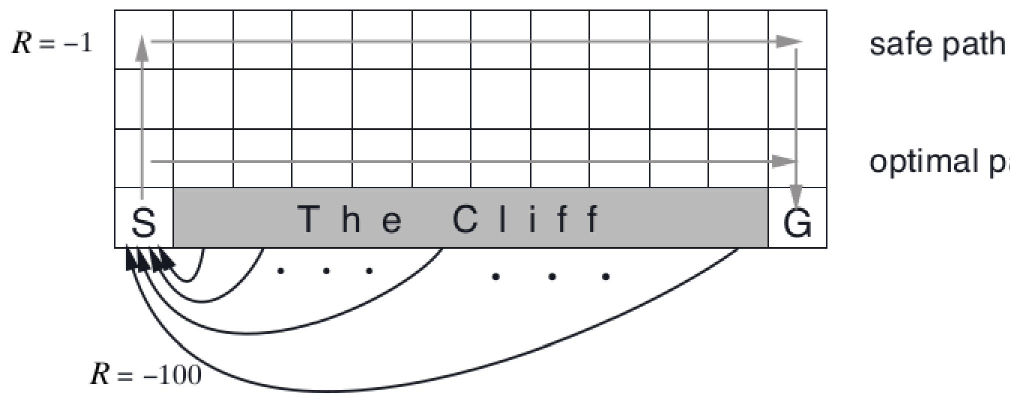
- Gathers less reward ☹

SARSA

- Gathers more reward ☺

EXAMPLE: Q-LEARNING VS. SARSA

The cliff-walking task:



Q-learning

- Gathers less reward ☹
- Learns the optimal policy ☺☺☺

SARSA

- Gathers more reward ☺
- Learns the longer path ☹☹☹

Q-LEARNING VS. SARSA WITH FUNCTION APPROXIMATION

- Both algorithms can be adapted to linear and non-linear FA by using the update rule
$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \nabla_\theta Q_\theta(x_t, a_t)$$
- Ditto with eligibility traces: SARSA(λ), $Q(\lambda)$

Q-LEARNING VS. SARSA WITH FUNCTION APPROXIMATION

- Both algorithms can be adapted to linear and non-linear FA by using the update rule
$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \nabla_\theta Q_\theta(x_t, a_t)$$
- Ditto with eligibility traces: SARSA(λ), $Q(\lambda)$
- SARSA guarantees bounded error and tends to behave well in practice (may not find optimal policy though)

Q-LEARNING VS. SARSA WITH FUNCTION APPROXIMATION

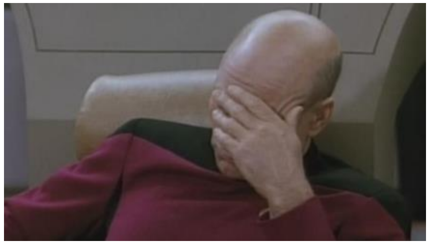
- Both algorithms can be adapted to linear and non-linear FA by using the update rule
$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \nabla_\theta Q_\theta(x_t, a_t)$$
- Ditto with eligibility traces: SARSA(λ), $Q(\lambda)$
- SARSA guarantees bounded error and tends to behave well in practice (may not find optimal policy though)
- Q-learning may diverge catastrophically

Q-LEARNING VS. SARSA WITH FUNCTION APPROXIMATION

- Both algorithms can be adapted to linear and non-linear FA by using the update rule
$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \nabla_\theta Q_\theta(x_t, a_t)$$
- Ditto with eligibility traces: SARSA(λ), $Q(\lambda)$
- SARSA guarantees bounded error and tends to behave well in practice (may not find optimal policy though)
- Q-learning may diverge catastrophically
  - Proposed fixes: gradient TD algorithms, emphatic TD algorithms, double Q-learning, soft Q-learning, G-learning,…

Q-LEARNING VS. SARSA WITH FUNCTION APPROXIMATION

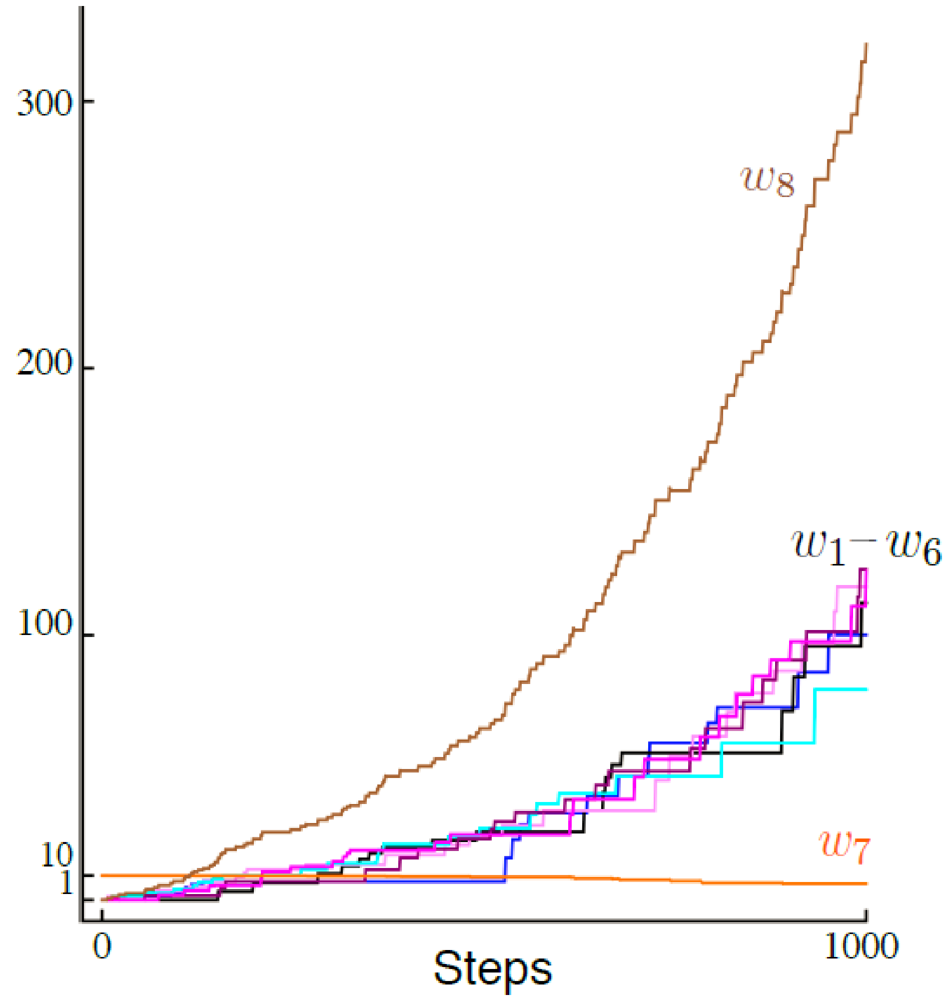
- Both algorithms can be adapted to linear and non-linear FA by using the update rule
$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \nabla_\theta Q_\theta(x_t, a_t)$$
- Ditto with eligibility traces: SARSA(λ), $Q(\lambda)$
- SARSA guarantees bounded error and tends to behave well in practice (may not find optimal policy though)
- Q-learning may diverge catastrophically
  - Proposed fixes: gradient TD algorithms, emphatic TD algorithms, double Q-learning, soft Q-learning, G-learning,…
  - Practical solution: tune it until it works



DIVERGENCE OF OFF-POLICY TD LEARNING

The "deadly triad":

- Function approximation
- Bootstrapping
- Off-policy learning



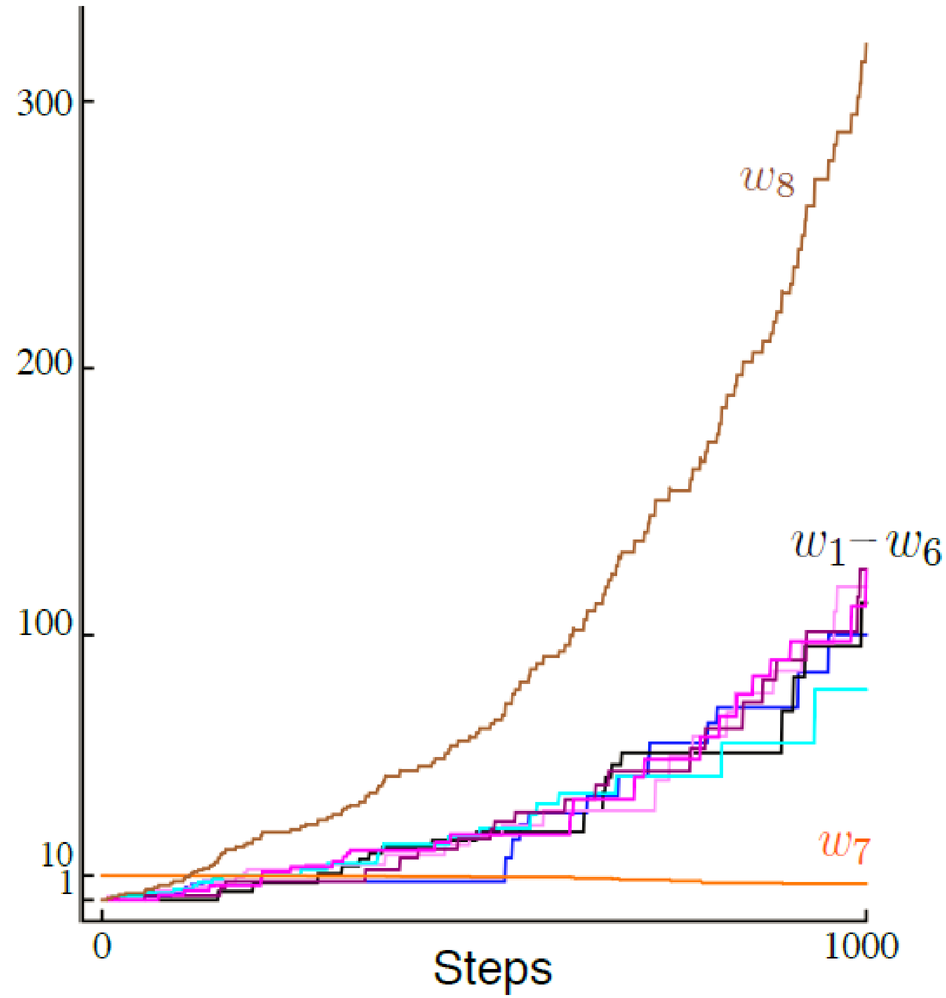
DIVERGENCE OF OFF-POLICY TD LEARNING

The "deadly triad":

- Function approximation
- Bootstrapping
- Off-policy learning

BUT

Divergence is typically not too extreme when behavior policy is close to evaluation policy and FA is linear



Next week:
Batch RL and
policy optimization