Optimal and Learning Control

for

Autonomous Robots

Lecture 9

A D R L

Jonas Buchli
Agile & Dexterous Robotics Lab

FNSNF

ETH Zürich

Class logistics

Exercise 2 out today (available online)

due - 6.5.2015

interview - 8.5.2015

Lecture 9 Goals

★Model free RL,
★Monte Carlo,
★Q learning

ETH Zürich

Sample based RL

Monte Carlo Method

Monte-Carlo Methods

Monte-Carlo Method (Sutton definition): average (values) over random samples of actual returns

Episodic learning

$$R_t = r_t + r_{t+1} + r_{t+2} + \ldots + r_N$$

$$V^{\pi}(x) = E\{R_n \mid x_n = x\} = E\left\{\sum_{k=0}^{\infty} \alpha^k r_{n+k} \mid x_n = x\right\}$$

Expectation is a weighted average!

Approximate E by 'sampling'

$$E(x) = \sum_i P(x_i)x_i \approx \sum_s \frac{1}{N} x_s \qquad x_s \sim P(x)$$

Approximate V by sampling

- Do N rollouts
- Average return observed after first visit of each state

$$\widetilde{V}_N^{\pi}(x) \approx \frac{1}{N}\sum_{i=1}^{N} R_n^i(x,u) = \frac{1}{N}\sum_{i=1}^{N}\left(r_n^i + \alpha\, r_{n+1}^i + \alpha^2\, r_{n+2}^i + \dots\right)$$

$$V^{\pi}(x) = E\{R_n \mid x_n = x\} = E\left\{\sum_{k=0}^{\infty} \alpha^k\, r_{n+k} \mid x_n = x\right\}$$

$$\widetilde{V}_N^{\pi}(x) \approx \frac{1}{N}\sum_{i=1}^{N_R}\sum_{k=0}^{N_T} \alpha^k\, r_{n+k}$$

'sampling approach to calculate expectation'

ETH Zürich

MC Policy Evaluation

Expectation is an average!

$$R_t = r_t + r_{t+1} + r_{t+2} + \ldots + r_N$$

$$V^\pi(x) = E\{R_n \mid x_n = x\} = E\left\{\sum_{k=0}^{\infty} \alpha^k r_{n+k} \mid x_n = x\right\}$$

Initialize:

. $\pi \leftarrow$ policy to be evaluated

. $V \leftarrow$ an arbitrary state-value function

. $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat Forever:

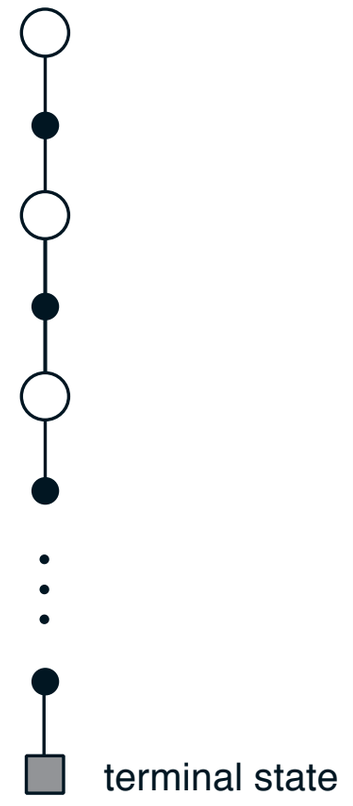
. (a) Generate an episode using π

. (b) For each state s appearing in the episode:

. . $R \leftarrow$ return following the first occurrence of s

. . Append R to $Returns(s)$

. . $V(s) \leftarrow$ average($Returns(s)$)



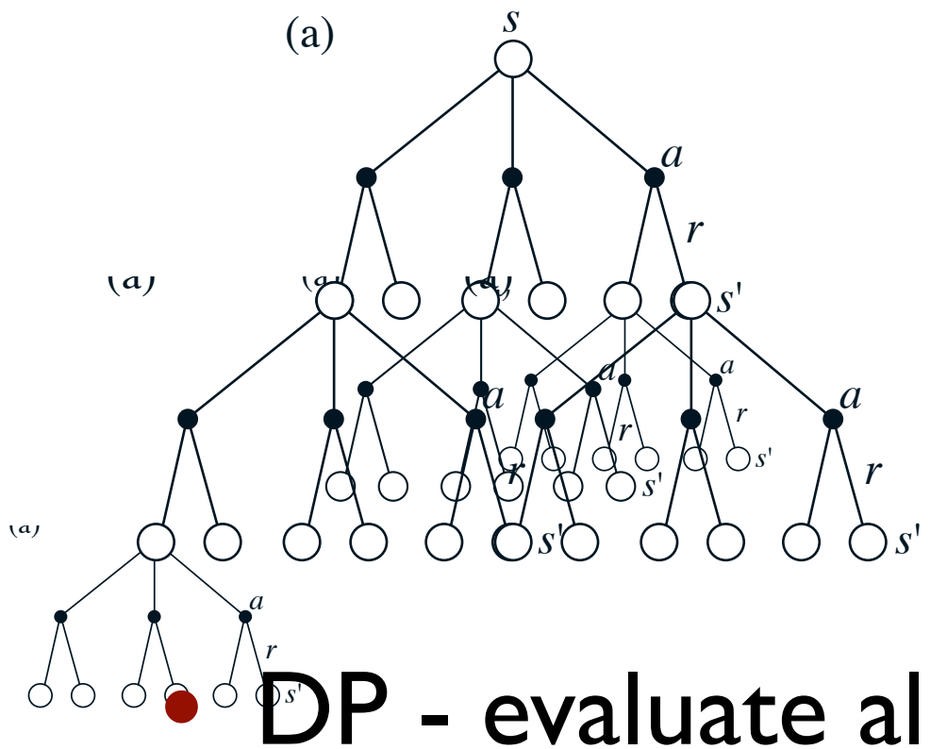
$$E(x) = \sum_i P(x_i)x_i \approx \sum_s \frac{1}{N} x_s$$

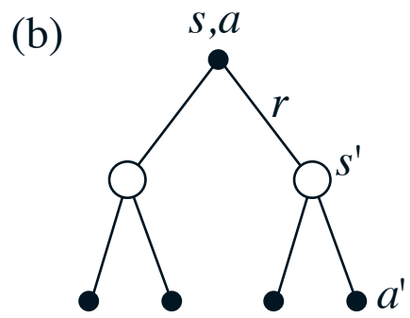
Buchli - OLCAR - 2015

ETH Zürich

Tree view on DP/MC

terminal state



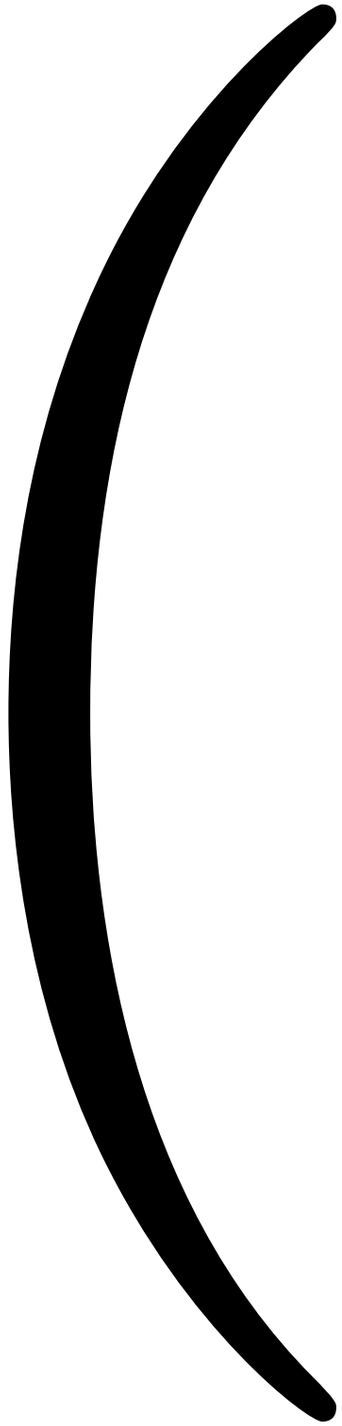


- DP - evaluate all states and/or all choices: full backups
- MC - only evaluate states seen in an episode

opportunity and problem: can ‘focus’ on relevant states, might not explore...

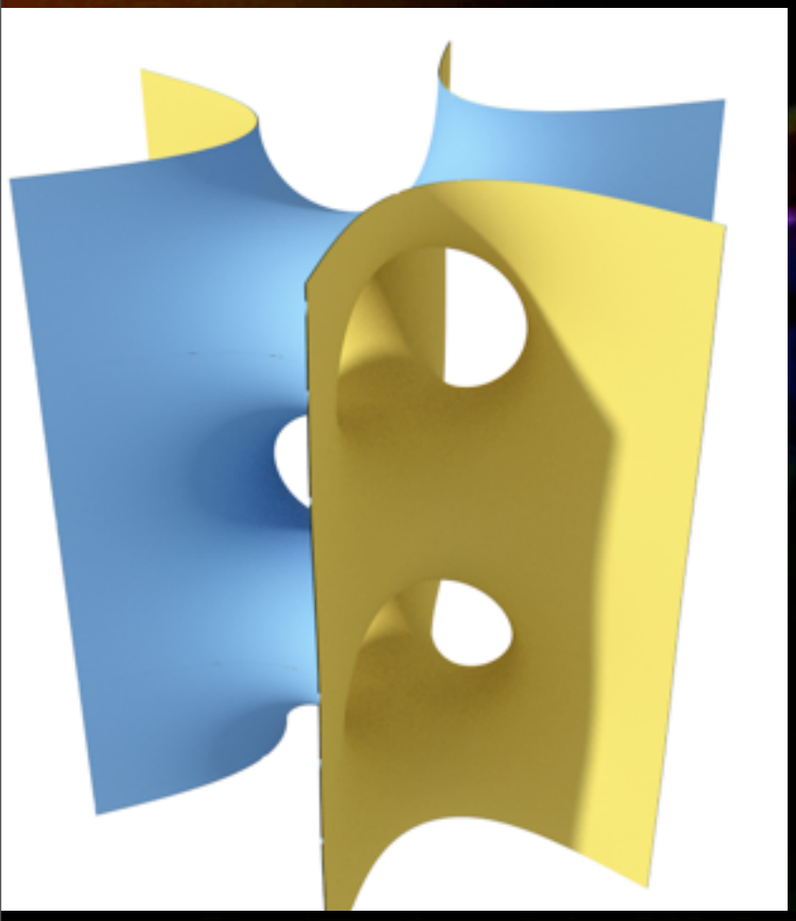
ADRL

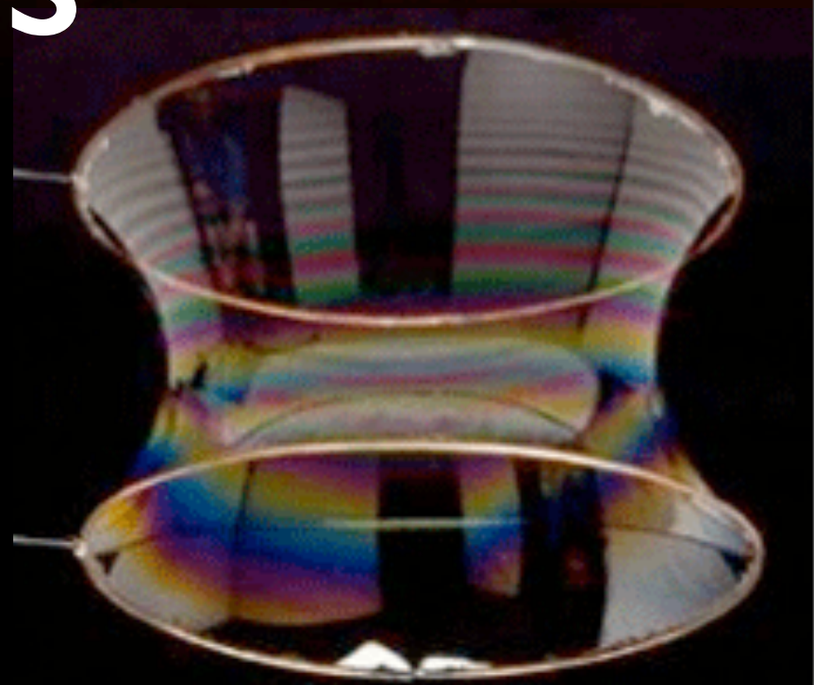
ETH Zürich

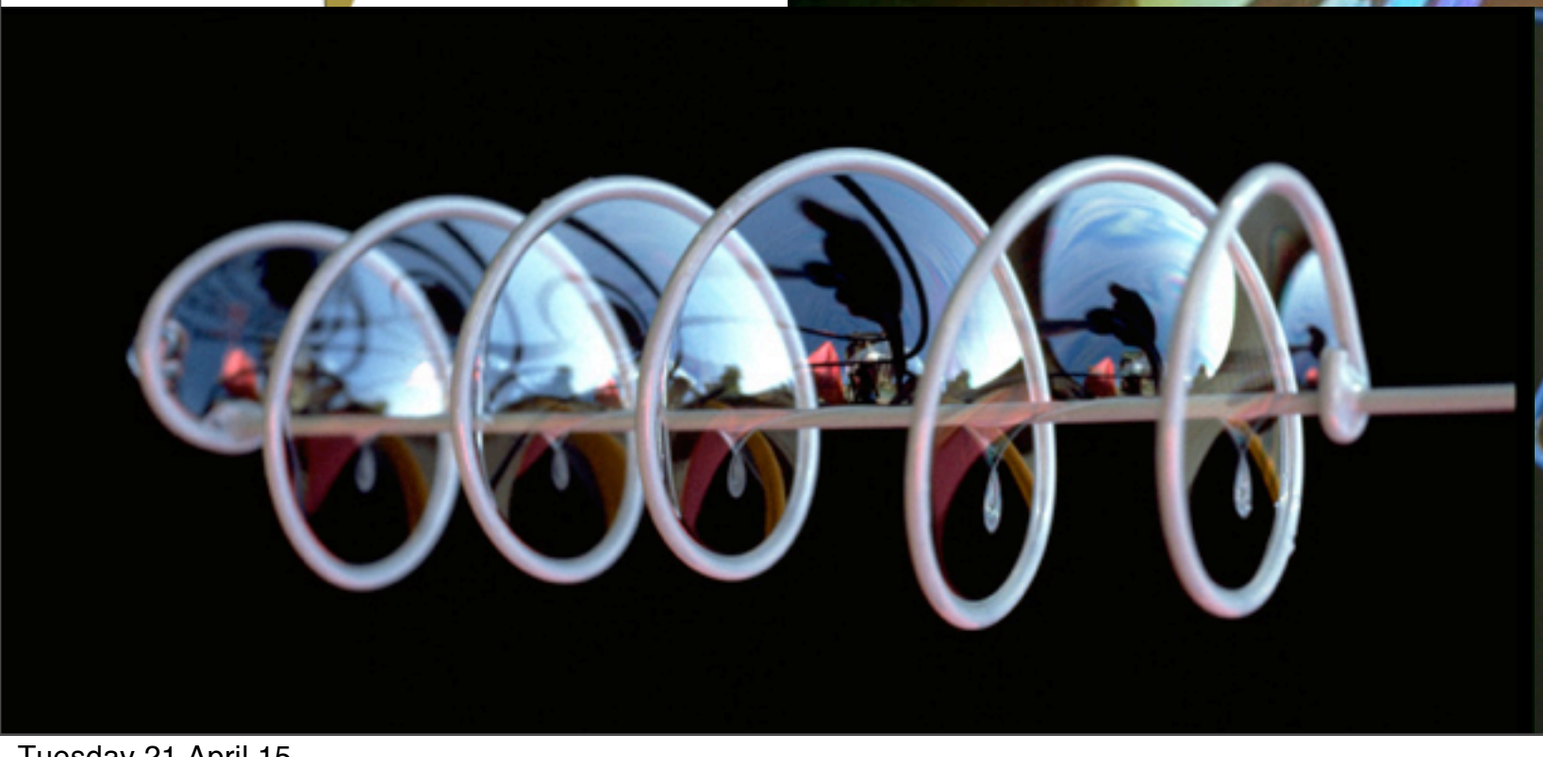


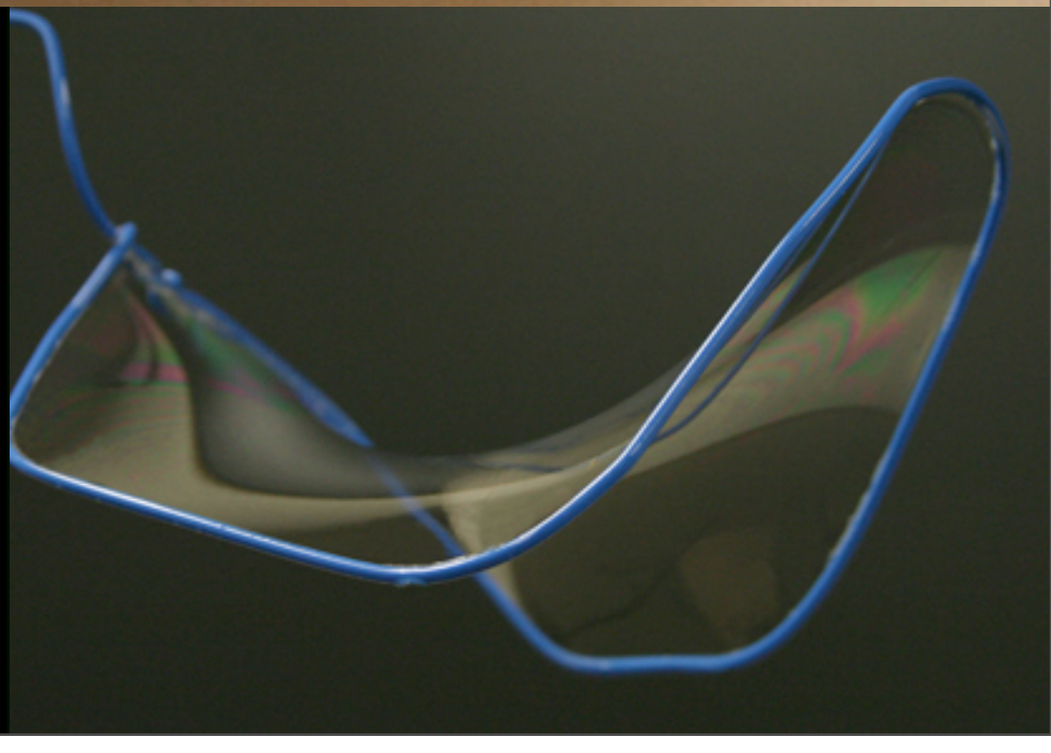
‘Monte Carlo’

Soap bubbles









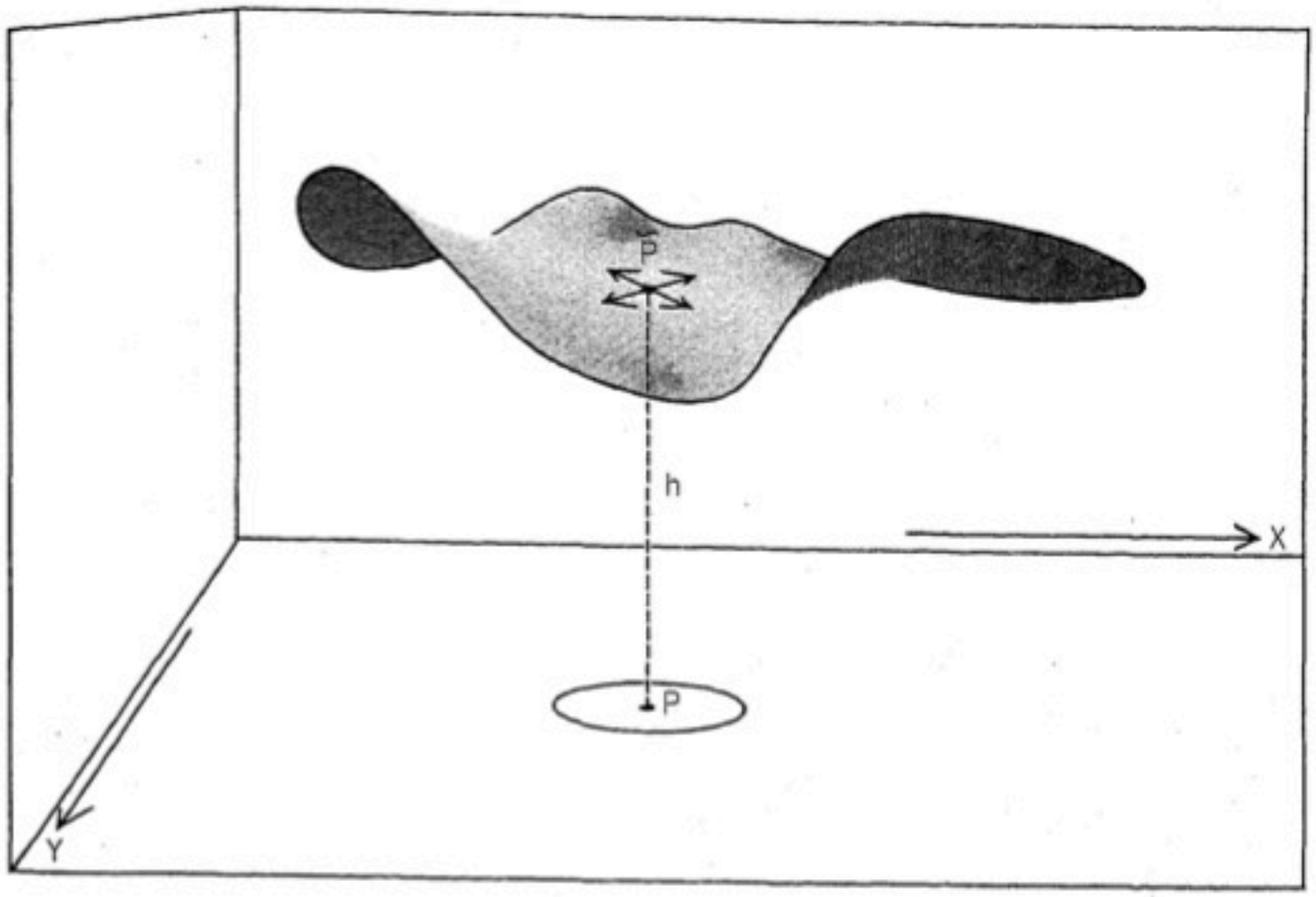
Brownian Motion and Potential Theory

The discovery that these two apparently unrelated branches of physics are in some sense mathematically equivalent has led to a new subject known as probabilistic potential theory

by Reuben Hersh and Richard J. Griego

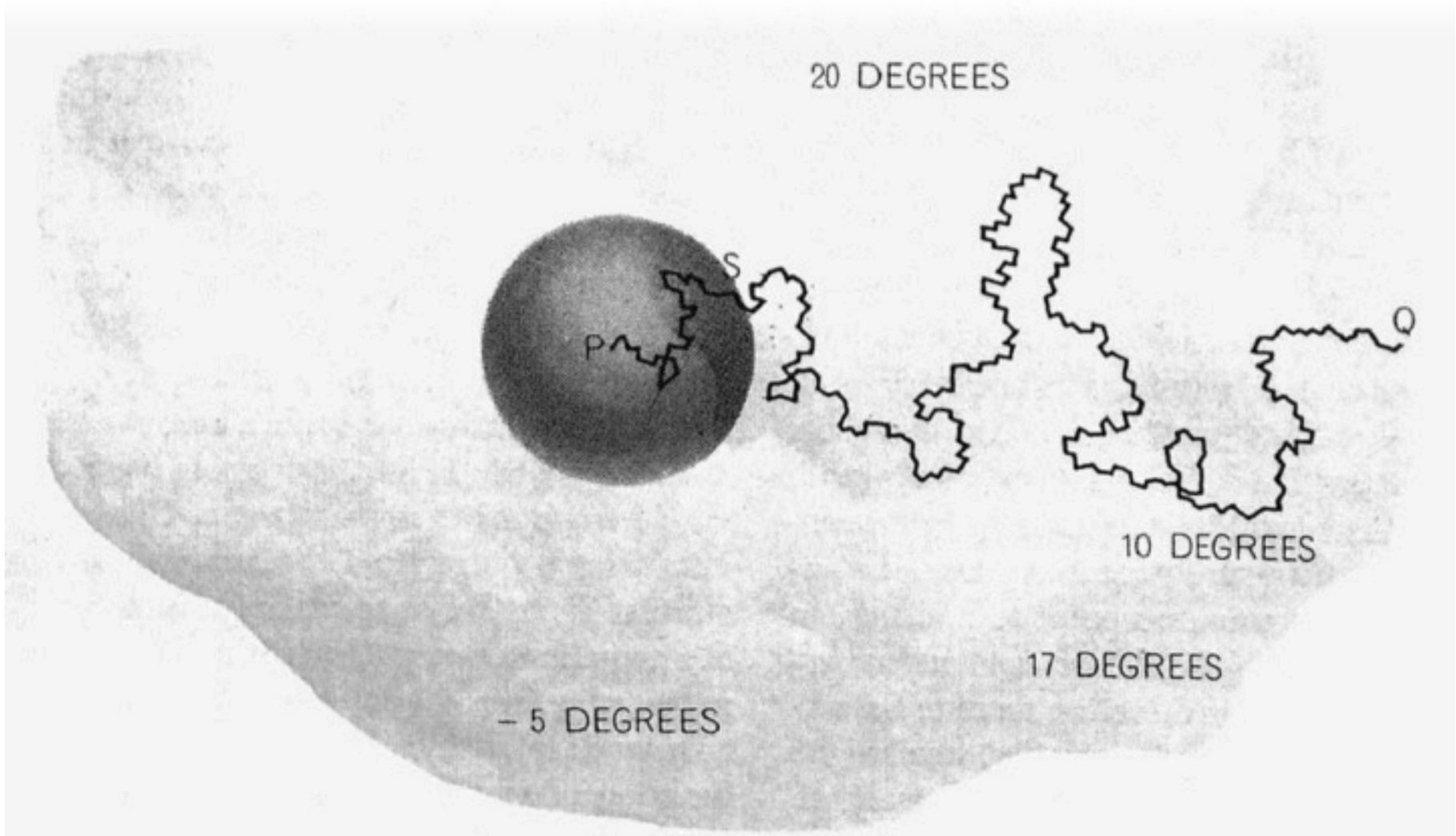
[Hersh and Griego, Scientific American, 1969]

Harmonic functions



ELASTIC MEMBRANE is stretched across a stiff, closed frame that is twisted into some fixed shape in space in this illustration of the role of harmonic functions in potential theory. The configuration of such a membrane is given by the height h of each point $\tilde{P}$ on the surface of the membrane. Directly below each $\tilde{P}$ on the membrane is a point P on the base plane, which has coordinates x, y. Besides being continuous, the function $h(x, y)$ has the following simple property: If P is a point in the x, y plane, and Γ is a small circle with its center at P, then the value of h at P (that is, the height of the membrane above P) equals the average of the values of h for all points on the circle Γ. This is called the mean-value property, and a continuous function h possessing this property is called a harmonic function. In this case the position of $\tilde{P}$ (the point on the membrane above P) is determined by the sum of the tension forces exerted on $\tilde{P}$ by the surrounding portion of the membrane (*arrows*). If the membrane is in equilibrium, these forces must cancel, so that the number of nearby elevations greater than that of $\tilde{P}$ must be matched by corresponding elevations lower than that of $\tilde{P}$, and the average must be just equal to the elevation of $\tilde{P}$, namely, the function h at P.

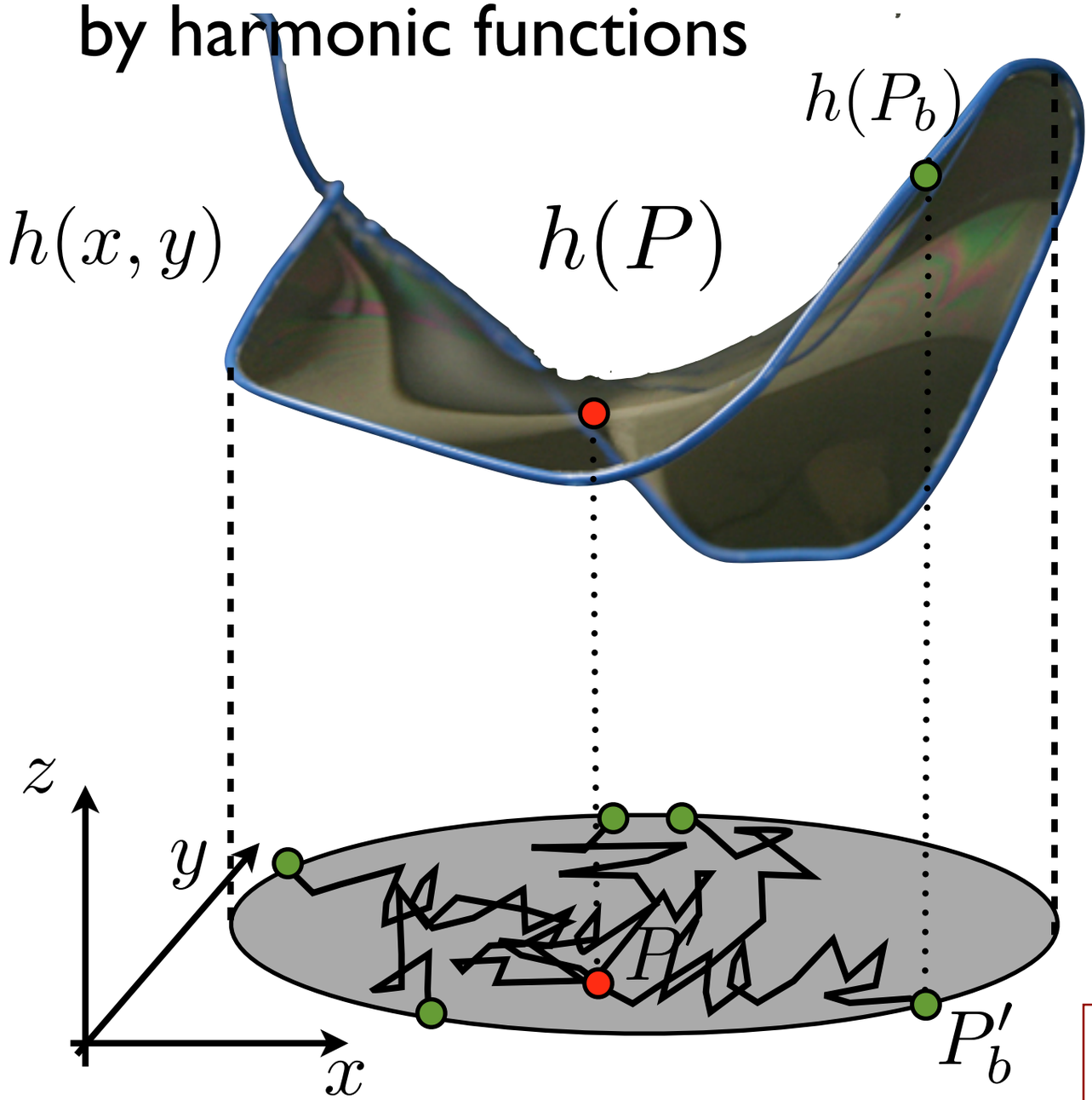
Random Walks



DISTRIBUTION OF HEAT in a homogeneous solid body at thermal equilibrium is another physical problem that involves a harmonic function. Since the temperature at any given point in such a body does not change with time, the temperature at that point must equal the average temperature over the surface of a small surrounding sphere. In other words, the temperature T is a harmonic function of the coordinates x, y, z of the point P. The problem can be solved by means of the probabilistic theory of the Brownian motion of a hypothetical particle starting at P and hitting the surface of the body at a random point Q.

Soap bubble MC

Soap bubbles can be described by harmonic functions



$h(P)$? find height on of arbitrary point on surface

... can be expressed as expectation!

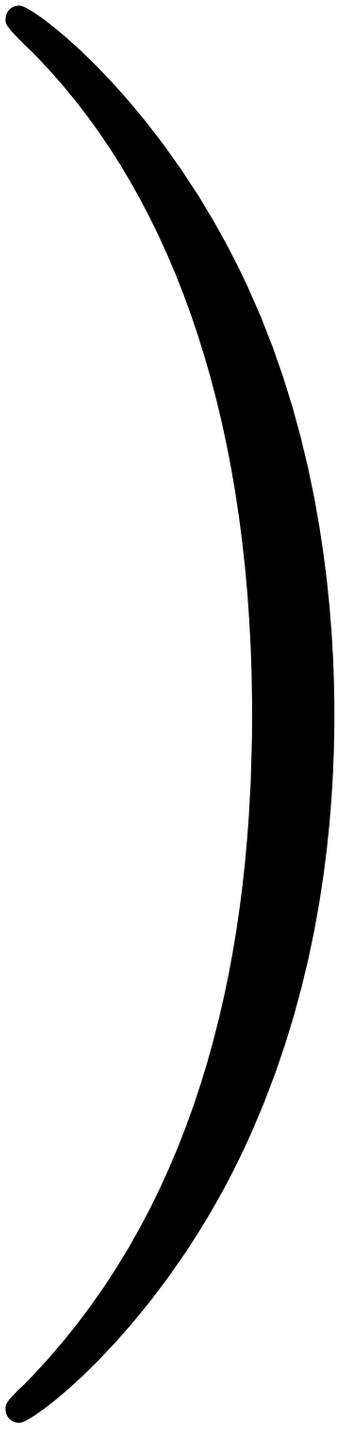
$$h(P) = E(h(P_{b,i}))$$

$h(P_b)$ height of point where random walk in x,y plane started in P' crosses boundary

expectation approximated by sampling

$$E\left(h(P_b)\right) \approx \sum_i^N \frac{1}{N} h(P_{b,i})$$

If interested only in part of the surface, only need to 'play' MC 'game' for this region



Remember Policy evaluation (part of DP solution)

$$V_{k+1}(s) = E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s\}$$
$$= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

requires transition probabilities

Initialize:
. $\pi \leftarrow$ policy to be evaluated
. $V \leftarrow$ an arbitrary state-value function
. $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$
Repeat Forever:
. (a) Generate an episode using π
. (b) For each state s appearing in the episode:
. . $R \leftarrow$ return following the first occurrence of s
. . Append R to $Returns(s)$
. . $V(s) \leftarrow$ average($Returns(s)$)

Need only 'experienced' rewards

Yields state value function, still need model to find policy

MC Estimation of Action Values?

Have seen MC estimation for state value Function V

Recall: To get controls from V need model (complete transition probabilities)

Can we find state-value function Q with MC Methods?

Problem: Might not visit all (relevant) *state-action pairs*

Approximate Q by sampling

- Do N rollouts
- Average return observed after first visit of each *state-action* pair

$$\widetilde{Q}_N^{\pi}(x,u) \approx \frac{1}{N}\sum_{i=1}^{N} R_n^i(x,u) = \frac{1}{N}\sum_{i=1}^{N}\left(r_n^i + \alpha\, r_{n+1}^i + \alpha^2\, r_{n+2}^i + \dots\right)$$

'sampling approach to calculate expectation'

Model free Q evaluation

1-step update / recursive

$$Q^{\pi}(x,u) = E_{\pi}\{R_n \mid x_n = x, u_n = u\}$$

$$\widetilde{Q}_N^{\pi}(x,u) \approx \frac{1}{N}\sum_{i=1}^{N} R_n^i(x,u) = \frac{1}{N}\sum_{i=1}^{N}\left(r_n^i + \alpha\, r_{n+1}^i + \alpha^2\, r_{n+2}^i + \dots\right)$$

(N+1)th sample:

$$\widetilde{Q}_{N+1}^{\pi}(x,u) \approx \frac{1}{N+1}\sum_{i=1}^{N+1} R_n^i(x,u)$$

$$\approx \frac{1}{N+1}\left(\sum_{i=1}^{N} R_n^i(x,u) + R_n^{N+1}(x,u)\right)$$

$$\approx \frac{N}{N+1}\cdot \widetilde{Q}_N^{\pi}(x,u) + \frac{1}{N+1}\cdot R_n^{N+1}(x,u)$$

$$\approx \widetilde{Q}_N^{\pi}(x,u) + \underbrace{\frac{1}{N+1}}_{=:\omega_{N+1}}\left(R_n^{N+1} - \widetilde{Q}_N^{\pi}(x,u)\right)$$

$$\lim_{N\to\infty} \widetilde{Q}_N^{\pi}(x,u) = Q^{\pi}(x,u)$$

Q Update Rule

Update ('learning') rule:

$$\widetilde{Q}^{\pi}_{N+1}(x,u) = Q_N + \omega_{N+1} \cdot \left(R_{N+1} - \widetilde{Q}^{\pi}_{N}(x,u) \right)$$

Update step/Learning rate $\omega_{N+1} = 1/(N+1)$

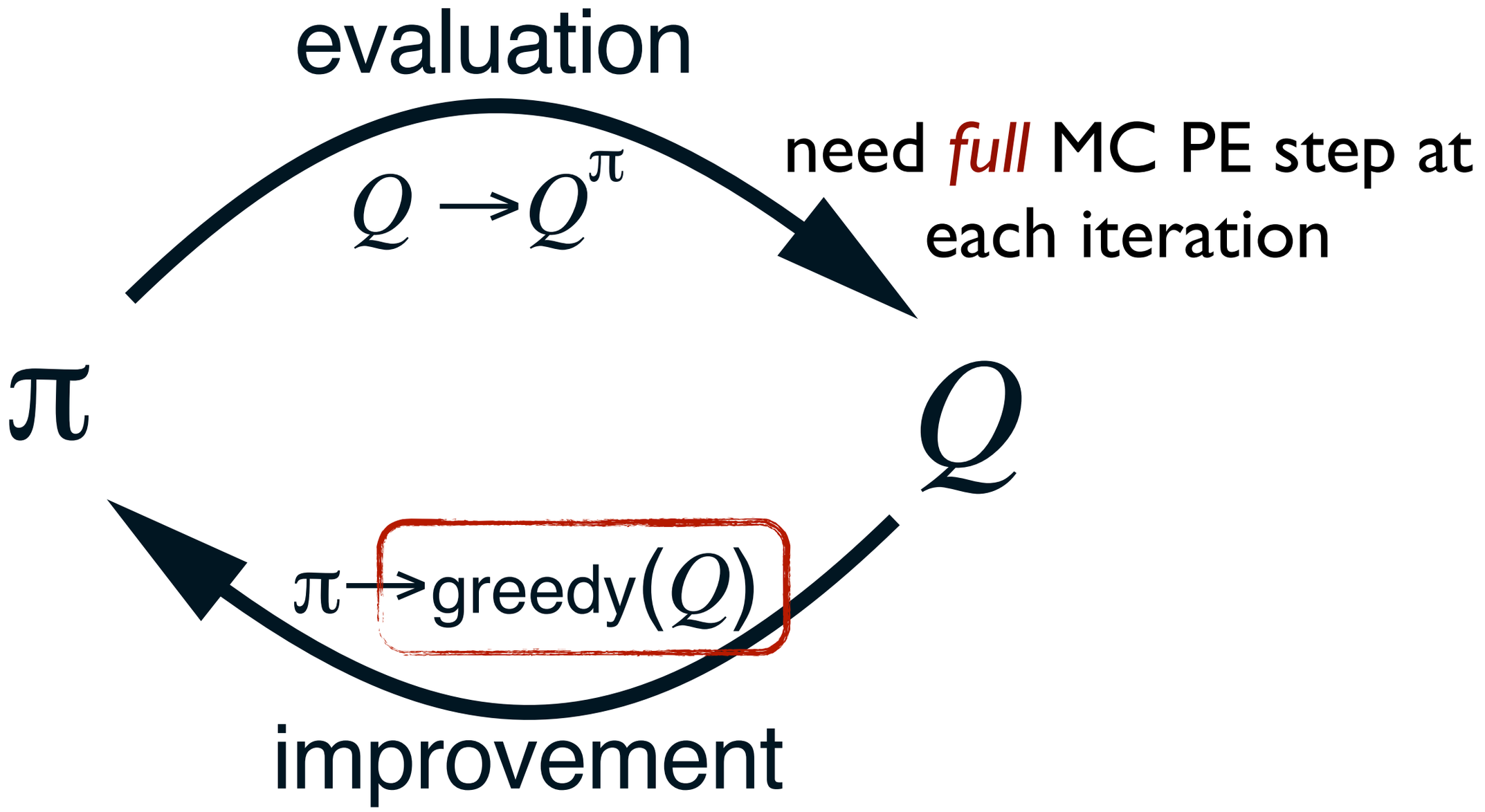
In practice can keep learning rate constant
... or decrease with a 'schedule'

ETH Zürich

Have action-value function (Q), but need policy

Model free GPI using Q?

MC GPI



$$\pi_0 \xrightarrow{E} Q^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} Q^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi^* \xrightarrow{E} Q^*$$

A D R L

ETH Zürich

(Reminder)

Algorithm 4 Generalized Policy Iteration (GPI) using the action value function $Q^\pi(x,u)$

1. Initialization

$Q^\pi(x,u) \in \mathbb{R}$

2. Policy Evaluation (PE)

repeat

for select a pair (x,u) with $x \in \mathcal{X},\ u \in \mathcal{U}$ **do**

$v \leftarrow Q^\pi(x,u)$

$Q^\pi(x,u) \leftarrow \sum_{x'} \mathcal{P}^u_{xx'} [\mathcal{R}^u_{xx'} + \alpha \sum_u \pi(x',u) Q^\pi(x',u)]$

end for

until "individual PE criterion satisfied"

3. Policy Improvement

policy-stable $\leftarrow$ true

for $x \in \mathcal{X}$ **do**

$b \leftarrow \pi(x)$

$\pi(x) \leftarrow \underset{\pi}{\operatorname{argmax}} \sum_{x'} \mathcal{P}^u_{xx'} [\mathcal{R}^u_{xx'} + \alpha V^*(x')]$

if $b \neq \pi(x)$ **then**

policy-stable$\leftarrow$false

end if

end for

if (policy-stable == true) **then**

stop;

else

go to 2.

end if

model required

Can we do this evaluation model free?

Model free policy improvement

In a state x, try all controls

Update policy as

$$\pi(x) = \arg\max_u Q^\pi(x, u)$$

ETH Zürich

Naive Monte Carlo Policy Improvement

Expectation is an average!

$$R_t = r_t + r_{t+1} + r_{t+2} + \ldots + r_N$$

$$Q^\pi(x,u) = E_\pi\{R_n \mid x_n = x, u_n = u\} = E\left\{\sum_{k=0}^{\infty} \alpha^k r_{n+k} \mid x_n = x, u_n = u\right\}$$

Algorithm 5 Naive Monte Carlo Algorithm Assuming Exploring Starts

Initialize, for all $x \in \mathcal{X}$, $u \in \mathcal{U}$

$Q(x,u) \leftarrow$ arbitrary

$\pi \leftarrow$ an arbitrary deterministic policy

Repeat forever:

(a) Exploring start: select random pair (x,u)

(b) Select a policy π and generate an episode: $x_0, u_0, r_0, x_1, u_1, r_1, \ldots, x_N, u_N, r_N$. *episodic*

(c) Sample-based **Policy Evaluation**:

for each pair x, u appearing in the episode:

$R \leftarrow$ return following the first occurrence of x, u

$Q(x,u) \leftarrow Q + \omega \cdot (R - Q(x,u))$

(d) **Policy improvement**:

$\pi(x) \leftarrow \arg\max_u Q(x,u)$ *greedy*

Monte Carlo PI assuming exploring starts!

A D R L

ETH Zürich

Issues of MC Control

Two unrealistic assumptions in 'vanilla' MC GPI

- #1 - Infinite number of rollouts in PE step
- #2 - Exploring starts: start from all state-action pairs

Issues of MC Control

Two unrealistic assumptions in 'vanilla' MC GPI

- #1 - Infinite number of rollouts in PE step

Don't do infinitely many roll-outs
Update value function and policy after one episode

ETH Zürich

Maintaining exploration

Non-realistic assumption #2: exploring starts!
How to maintain exploration?

Consider: What's best policy given an estimated Value function?
⇒ Greedy, deterministic

How will the 'sampled' roll-outs look like?
⇒ always the 'same' (or at least very similar)

ETH Zürich

Exploration vs. Exploitation

Important: Exploring policy should not be optimal!

Why?

Exploring starts: Start at all possible state-action pairs

even better: Exploration policy should not be deterministic, non-zero probability for all x-u pairs

ETH Zürich

On-Policy Monte Carlo Control with ε-soft Policy

★ On-policy

★Soft

On/off policy

2 ways to ensure exploration:

- On-policy: Use the same policy for exploration and execution

  ⇒ 'special requirements for policy'

- Off-policy: Use a different policy for exploration and for execution

  ⇒ need to ensure 'coherence' between executing and exploring policy

ε-greedy Policy

$$\pi(u \mid x) = \begin{cases} \frac{\varepsilon}{|\mathcal{U}(x)|} & \text{for the non-greedy action} \\ 1 - \varepsilon \left(1 - \frac{1}{|\mathcal{U}(x)|}\right) & \text{for the greedy action} \end{cases}$$

- With high probability choose action that maximizes est. action value (greedy)
- with small (but not 0) prob. choose a non-greedy action

ε-greedy Policy

Replacement for exploring start: Soft Exploration Policy

Soft policy: $\pi(x,u) > 0, \quad \forall x \in X, \forall u \in \mathcal{U}$

p(non-greedy) = $\frac{\varepsilon}{|\mathcal{U}(x)|} \quad \varepsilon > 0$

otherwise greedy

$$\Leftrightarrow \quad \pi(x,u) \leftarrow \begin{cases} \frac{\varepsilon}{|\mathcal{U}(x)|} & \text{if } u \neq u^* \\ 1 - \varepsilon\left(1 - \frac{1}{|\mathcal{U}(x)|}\right) & \text{if } u = u^* \end{cases}$$

$\Leftrightarrow \quad \pi(x,u) \geq \frac{\varepsilon}{|\mathcal{U}|}$ for all states and actions

policy is ϵ-soft

A D R L

ETH Zürich

Proof of policy improvement with ε-greedy policy

$$E_{\substack{u_n \sim \pi' \\ u_{[n+1,\dots]} \sim \pi}} \{R_n \mid x_n = x\} = \sum_u \pi'(x,u) Q^\pi(x,u)$$

$$= \frac{\varepsilon}{|\mathcal{U}(x)|} \sum_u Q^\pi(x,u) + (1-\varepsilon) \max_u Q^\pi(x,u)$$

avg. is less or equal max:

$$\max_u Q^\pi(x,u) \geq \sum_u \frac{\pi(x,u) - \frac{\varepsilon}{|\mathcal{U}(x)|}}{1-\varepsilon} Q^\pi(x,u)$$

'weights'

$$\left\{ \frac{\pi(x,u) - \frac{\varepsilon}{|\mathcal{U}(x)|}}{1-\varepsilon} \right\}$$

'non-negative weights', sum to 1

$$\left\{ \frac{\pi(x,u) - \frac{\varepsilon}{|\mathcal{U}(x)|}}{1-\varepsilon} \right\}$$ 'non-negative weights', sum to 1

Proof:

$$\sum_u \frac{\pi(x,u) - \frac{\varepsilon}{|\mathcal{U}(x)|}}{1-\varepsilon} = \sum_u \frac{\pi(x,u)}{1-\varepsilon} - \sum_u \frac{\frac{\varepsilon}{|\mathcal{U}(x)|}}{1-\varepsilon}$$

$$= \frac{1}{1-\varepsilon} \sum_u \pi(x,u) - \frac{\frac{\varepsilon}{|\mathcal{U}(x)|}}{1-\varepsilon} \sum_u 1$$

$$= \frac{1}{1-\varepsilon} - \frac{\frac{\varepsilon}{|\mathcal{U}(x)|}}{1-\varepsilon} |\mathcal{U}(x)| = 1$$

ETH Zürich

$$E_{\substack{u_n \sim \pi' \\ u_{[n+1,\dots]} \sim \pi}} \{R_n \mid x_n = x\} = \sum_u \pi'(x,u) Q^\pi(x,u)$$

$$= \frac{\varepsilon}{|\mathcal{U}(x)|} \sum_u Q^\pi(x,u) + (1-\varepsilon) \max_u Q^\pi(x,u)$$

$$\max_u Q^\pi(x,u) \geq \sum_u \frac{\pi(x,u) - \frac{\varepsilon}{|\mathcal{U}(x)|}}{1-\varepsilon} Q^\pi(x,u)$$

$$\Rightarrow \quad E_{\substack{u_n \sim \pi' \\ u_{[n+1,\dots]} \sim \pi}} \{R_n \mid x_n = x\} \geq \frac{\varepsilon}{|\mathcal{U}(x)|} \sum_u Q^\pi(x,u) + (1-\varepsilon) \sum_u \frac{\pi(x,u) - \frac{\varepsilon}{|\mathcal{U}(x)|}}{1-\varepsilon} Q^\pi(x,u)$$

RHS

$$\frac{\varepsilon}{|\mathcal{U}(x)|} \sum_u Q^\pi(x,u) - \frac{\varepsilon}{|\mathcal{U}(x)|} \sum_u Q^\pi(x,u) + \sum_u \pi(x,u) Q^\pi(x,u) = \sum_u \pi(x,u) Q^\pi(x,u)$$

$$\Rightarrow \quad E_{\substack{u_n \sim \pi' \\ u_{[n+1,\dots]} \sim \pi}} \{R_n \mid x_n = x\} \geq \sum_u \pi(x,u) Q^\pi(x,u)$$

$$\Rightarrow \quad E_{\substack{u_n \sim \pi' \\ u_{[n+1,\dots]} \sim \pi}} \{R_n \mid x_n = x\} \geq V^\pi(x)$$

QED (1)

ETH Zürich

2nd part of proof

In the second step of the proof, we should show that the expected accumulated reward for the case that we perform the first two steps according to the policy π' then following the policy π is greater or equal to $E_{\substack{u_n \sim \pi' \\ u_{[n+1,\dots]} \sim \pi}} \{R_n \mid x_n = x\}$. Since this proof is very similar to the one, we just did, we will skip this proof and just use the result.

$$E_{\substack{u_{[n,n+1]} \sim \pi' \\ u_{[n+2,\dots]} \sim \pi}} \{R_n \mid x_n = x\} \geq E_{\substack{u_n \sim \pi' \\ u_{[n+1,\dots]} \sim \pi}} \{R_n \mid x_n = x\}$$

$$\geq V^{\pi}(x)$$

Following the same procedure, we can replace the policy to the end.

$$E_{u_{[n,n+1,\dots]} \sim \pi'} \{R_n \mid x_n = x\} \geq V^{\pi}(x)$$

$$V^{\pi'}(x) \geq V^{\pi}(x)$$

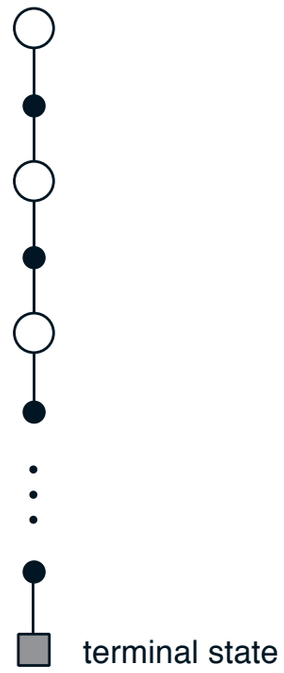
Algorithm 6 ε-soft, On-Policy Monte Carlo Algorithm

choose a constant learning rate, ω
choose a positive $\varepsilon \in (0, 1]$
$Q^\pi(x, u) \leftarrow$ arbitrary
$\pi \leftarrow$ an arbitrary ε-soft policy
Repeat forever:
(a) generate an episode using π
(b) Policy Evaluation
for each pair (x, u) appearing in the episode
$R \leftarrow$ return following the first occurrence of (x, u)
$Q^\pi(x, u) \leftarrow Q(x, u) + \omega\left(R - Q^\pi(x, u)\right)$
(c) Policy Improvement
for each: x in the episode:
$u^* \leftarrow \arg\max_u Q^\pi(x, u)$
For all $a \in \mathcal{U}(x)$:

$$\pi(x, u) \leftarrow \begin{cases} \frac{\varepsilon}{|\mathcal{U}(x)|} & \text{if } u \neq u^* \\ 1 - \varepsilon\left(1 - \frac{1}{|\mathcal{U}(x)|}\right) & \text{if } u = u^* \end{cases}$$

(d) (*optional*) decrease ε.

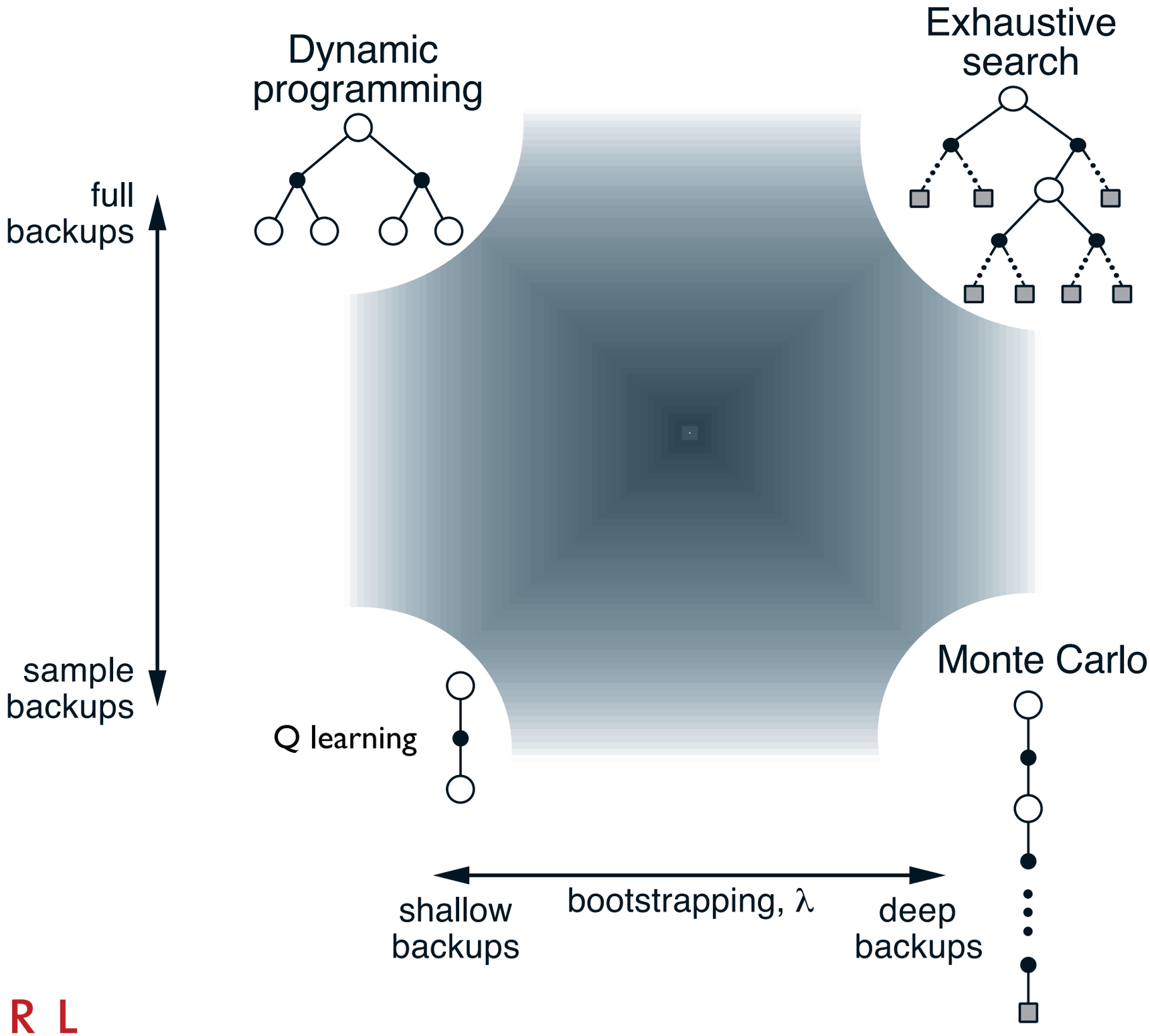
episodic learning



Q-learning

Combine ideas from Monte Carlo and Dynamic Programming

MC: Learn direct from samples
DP: Update estimate based on neighboring estimates (don't wait for full episode)


Dynamic programming
Exhaustive search
full backups
sample backups
Q learning
Monte Carlo
shallow backups
bootstrapping, λ
deep backups

Q-learning

Optimal Bellman Equation

$$Q^*(x_n, u_n) = E\left\{r_n + \alpha \max_{u'} Q^*(x', u') \mid x_n = x, u_n = u\right\}$$

sample Q to estimate expectation

$$\widetilde{Q}(x, u) = \frac{1}{N} \sum_{i=1}^{N} \left(r_n^i + \alpha \max_{u} Q(x_i', u)\right)$$

formulated recursively (as before):

$$\widetilde{Q}^{i+1}(x_n, u_n) = \widetilde{Q}^i(x_n, u_n) + \omega_{i+1}\left[r_n^{i+1} + \alpha \max_{u_n'} \widetilde{Q}^i(x_n', u_n') - \widetilde{Q}^i(x_n, u_n)\right]$$

estimates Q independent of policy: off-policy!

(as long as Q explores sufficiently)

A D R L

ETH Zürich

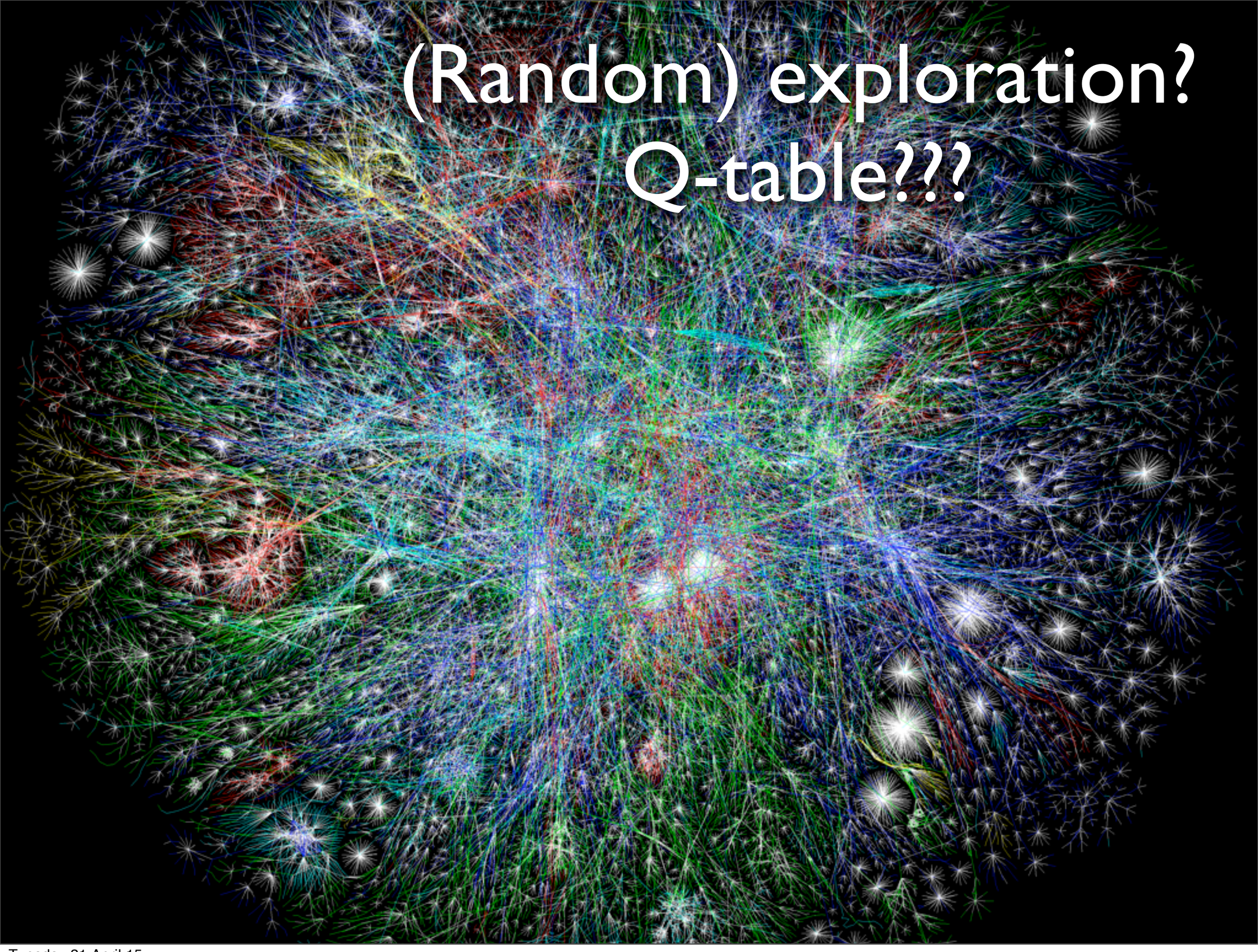
Q-learning

Algorithm 7 Q-Learning

Initialize $Q(x, u)$ arbitrarily
Repeat for each episode:
  Initialize x
repeat (for each step of episode):
    Choose u from x using policy derived from Q
    (e.g., $\varepsilon -$ greedy)
  Take action u, observe r, x'
  $Q(x, u) \leftarrow Q(x, u) + \omega[r + \gamma \max_{u'} Q(x', u') - Q(x, u)]$
  $x \leftarrow x'$
until x is terminal

http://www.cse.unsw.edu.au/~cs9417ml/RL2/index.html

Random exploration


(Random) exploration?
Q-table???

Value function approximation

Instead of learning full Q(x,u), approximate it.

$$Q(s,a;\theta) \approx Q^*(s,a)$$

Linear approximator
Neural Network
...

Learn θ using reward samples

ETH Zürich


INNOVATIONS IN
The microbiome
nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
LEARNING CURVE
Self-taught AI software attains human-level performance in video games
PAGES 486 & 529
EPIDEMIOLOGY
SHARE DATA IN OUTBREAKS
Forge open access to sequences and more
PAGE 477
COSMOLOGY
A GIANT IN THE EARLY UNIVERSE
A supermassive black hole at a redshift of 6.3
PAGES 490 & 512
QUANTUM PHYSICS
TELEPORTATION FOR TWO
Transferring two properties of a single photon
PAGES 491 & 516
NATURE.COM/NATURE
26 February 2015 £10
Vol. 518, No. 7540

26 FEBRUARY 2015 | VOL 518 | NATURE | 529

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

1) Basic idea: Q-learning ...
2) ... using Value function approximation (using deep network)
3) using emulator to generate samples
4) ... a couple of other tricks... (preprocessing, experience replay, ...)


HIGH SCORE
540
SONY
ATARI

Goal - RL Problem

The goal of the agent is to interact with the emulator by selecting actions in a way that maximizes future rewards. We make the standard assumption that future rewards are discounted by a factor of γ per time-step (γ was set to 0.99 throughout), and define the future discounted return at time t as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, in which T is the time-step at which the game terminates. We define the optimal action-value function $Q^*(s,a)$ as the maximum expected return achievable by following any policy, after seeing some sequence s and then taking some action a, $Q^*(s,a) = \max_\pi \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$ in which π is a policy mapping sequences to actions (or distributions over actions).

Basic RL

The optimal action-value function obeys an important identity known as the Bellman equation. This is based on the following intuition: if the optimal value $Q^*(s',a')$ of the sequence s' at the next time-step was known for all possible actions a', then the optimal strategy is to select the action a' maximizing the expected value of $r+\gamma Q^*(s',a')$:

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r+\gamma \max_{a'} Q^*(s',a')|s,a\right]$$

The basic idea behind many reinforcement learning algorithms is to estimate the action-value function by using the Bellman equation as an iterative update, $Q_{i+1}(s,a) = \mathbb{E}_{s'}[r+\gamma \max_{a'} Q_i(s',a')|s,a]$. Such value iteration algorithms converge to the optimal action-value function, $Q_i \rightarrow Q^*$ as $i \rightarrow \infty$. In practice, this basic approach

Value approximation

reward. More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \mid s_t = s,\, a_t = a,\, \pi\right],$$

The basic idea behind many reinforcement learning algorithms is to estimate the action-value function by using the Bellman equation as an iterative update, $Q_{i+1}(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q_i(s',a') \mid s,a]$. Such value iteration algorithms converge to the optimal action-value function, $Q_i \rightarrow Q^*$ as $i \rightarrow \infty$. In practice, this basic approach is impractical, because the action-value function is estimated separately for each sequence, without any generalization. Instead, it is common to use a function approximator to estimate the action-value function, $Q(s,a;\theta) \approx Q^*(s,a)$. In the reinforcement learning community this is typically a linear function approximator, but sometimes a nonlinear function approximator is used instead, such as a neural network. We refer to a neural network function approximator with weights θ as a Q-network. A Q-network can be trained by adjusting the parameters θ_i at iteration i to reduce the mean-squared error in the Bellman equation, where the optimal target values $r + \gamma \max_{a'} Q^*(s',a')$ are substituted with approximate target values $y = r + \gamma \max_{a'} Q(s',a';\theta_i^-)$, using parameters θ_i^- from some previous iteration. This leads to a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i,

$$\begin{aligned} L_i(\theta_i) &= \mathbb{E}_{s,a,r}\left[\left(\mathbb{E}_{s'}[y \mid s,a] - Q(s,a;\theta_i)\right)^2\right] \\ &= \mathbb{E}_{s,a,r,s'}\left[\left(y - Q(s,a;\theta_i)\right)^2\right] + \mathbb{E}_{s,a,r}\left[\mathbb{V}_{s'}[y]\right]. \end{aligned}$$

Loss function

Note that the targets depend on the network weights; this is in contrast with the targets used for supervised learning, which are fixed before learning begins. At each stage of optimization, we hold the parameters from the previous iteration θ_i^- fixed when optimizing the ith loss function $L_i(\theta_i)$, resulting in a sequence of well-defined optimization problems. The final term is the variance of the targets, which does not depend on the parameters θ_i that we are currently optimizing, and may therefore be ignored. Differentiating the loss function with respect to the weights we arrive at the following gradient:

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right].$$

Rather than computing the full expectations in the above gradient, it is often computationally expedient to optimize the loss function by stochastic gradient descent. The familiar Q-learning algorithm[19] can be recovered in this framework by updating the weights after every time step, replacing the expectations using single samples, and setting $\theta_i^- = \theta_{i-1}$.

A D R L

More details on policy

(basic Q-learning)

Note that this algorithm is model-free: it solves the reinforcement learning task directly using samples from the emulator, without explicitly estimating the reward and transition dynamics $P(r,s'|s,a)$. It is also off-policy: it learns about the greedy policy $a = \text{argmax}_{a'} Q(s,a';\theta)$, while following a behaviour distribution that ensures adequate exploration of the state space. In practice, the behaviour distribution is often selected by an ε-greedy policy that follows the greedy policy with probability $1-\varepsilon$ and selects a random action with probability ε.

Problem size

210x160x128

Partially observed system

Because the agent only observes the current screen, the task is partially observed[33] and many emulator states are perceptually aliased (that is, it is impossible to fully understand the current situation from only the current screen x_t). Therefore, sequences of actions and observations, $s_t = x_1, a_1, x_2, \ldots, a_{t-1}, x_t$, are input to the algorithm, which then learns game strategies depending upon these sequences. All sequences in the emulator are assumed to terminate in a finite number of time-steps. This formalism gives rise to a large but finite Markov decision process (MDP) in which each sequence is a distinct state. As a result, we can apply standard reinforcement learning methods for MDPs, simply by using the complete sequence s_t as the state representation at time t.

Preprocessing

Preprocessing. Working directly with raw Atari 2600 frames, which are 210 × 160 pixel images with a 128-colour palette, can be demanding in terms of computation and memory requirements. We apply a basic preprocessing step aimed at reducing the input dimensionality and dealing with some artefacts of the Atari 2600 emulator. First, to encode a single frame we take the maximum value for each pixel colour value over the frame being encoded and the previous frame. This was necessary to remove flickering that is present in games where some objects appear only in even frames while other objects appear only in odd frames, an artefact caused by the limited number of sprites Atari 2600 can display at once. Second, we then extract the Y channel, also known as luminance, from the RGB frame and rescale it to 84 × 84. The function ϕ from algorithm 1 described below applies this preprocessing to the m most recent frames and stacks them to produce the input to the Q-function, in which $m = 4$, although the algorithm is robust to different values of m (for example, 3 or 5).

Value function approximation

Instead of learning full Q(x,u), approximate it.

$$Q(s,a;\theta) \approx Q^*(s,a)$$

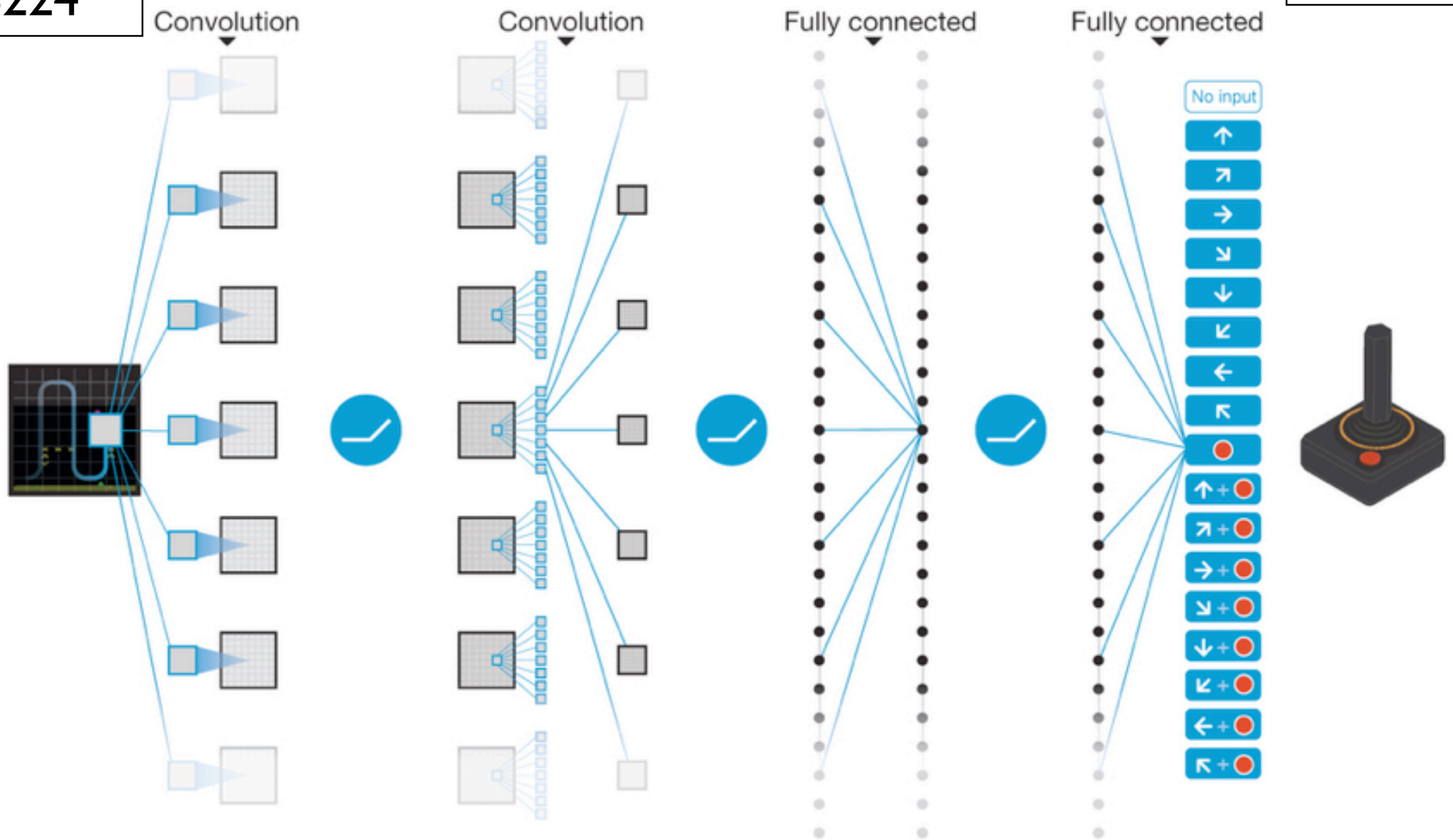
concepts such as object categories directly from raw sensory data. We use one particularly successful architecture, the deep convolutional network[17], which uses hierarchical layers of tiled convolutional filters to mimic the effects of receptive fields—inspired by Hubel and Wiesel's seminal work on feedforward processing in early visual cortex[18]—thereby exploiting the local spatial correlations present in images, and building in robustness to natural transformations such as changes of viewpoint or scale.

Learn θ using reward samples

Input:
84x84x4=
28224

‘Deep network’

Actions:
4 ... 18



note special architecture

Model architecture. There are several possible ways of parameterizing Q using a neural network. Because Q maps history–action pairs to scalar estimates of their Q-value, the history and the action have been used as inputs to the neural network by some previous approaches[24,26]. The main drawback of this type of architecture is that a separate forward pass is required to compute the Q-value of each action, resulting in a cost that scales linearly with the number of actions. We instead use an architecture in which there is a separate output unit for each possible action, and only the state representation is an input to the neural network. The outputs correspond to the predicted Q-values of the individual actions for the input state. The main advantage of this type of architecture is the ability to compute Q-values for all possible actions in a given state with only a single forward pass through the network.

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1$,T **do**

With probability ε select a random action a_t

otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

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 D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

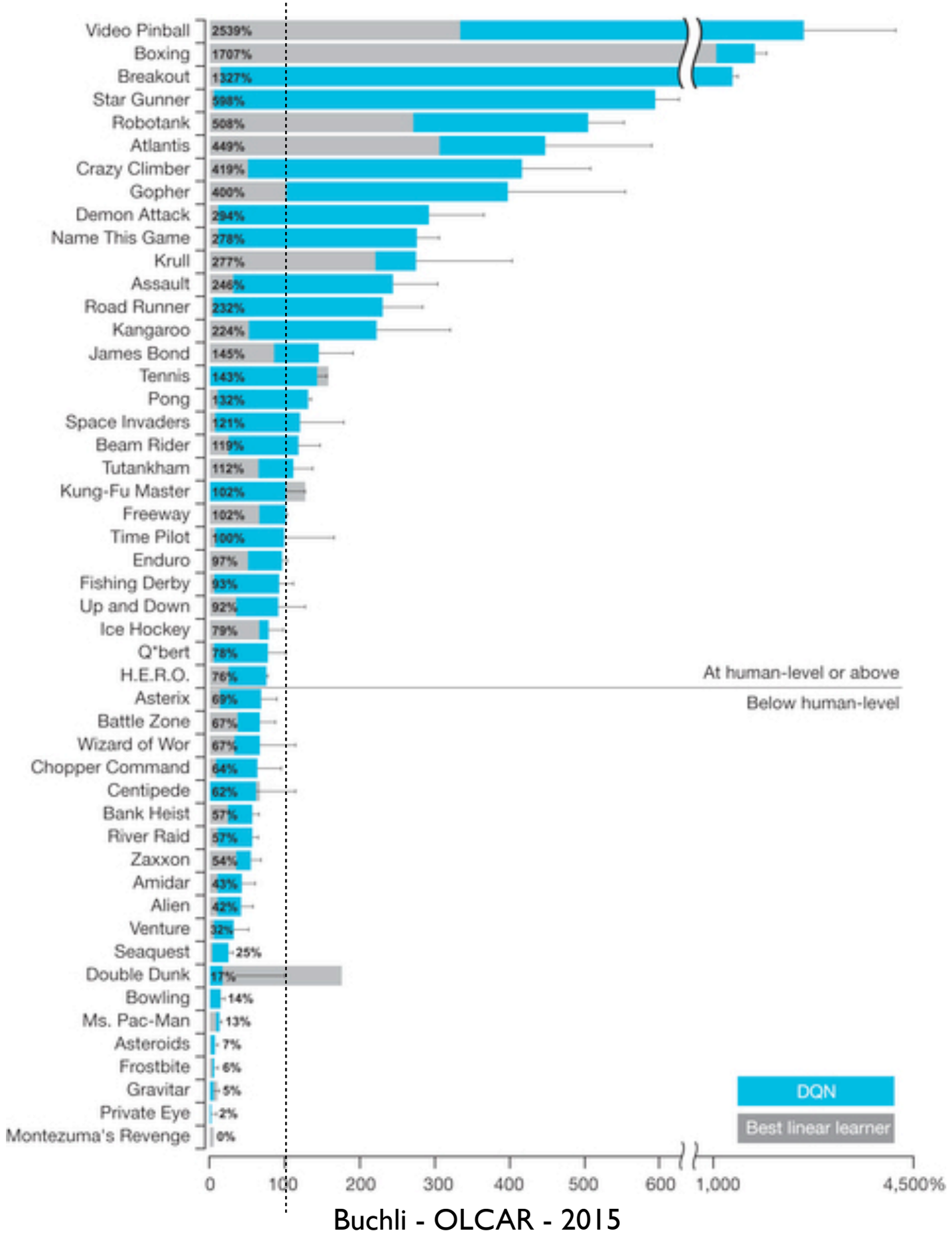
Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

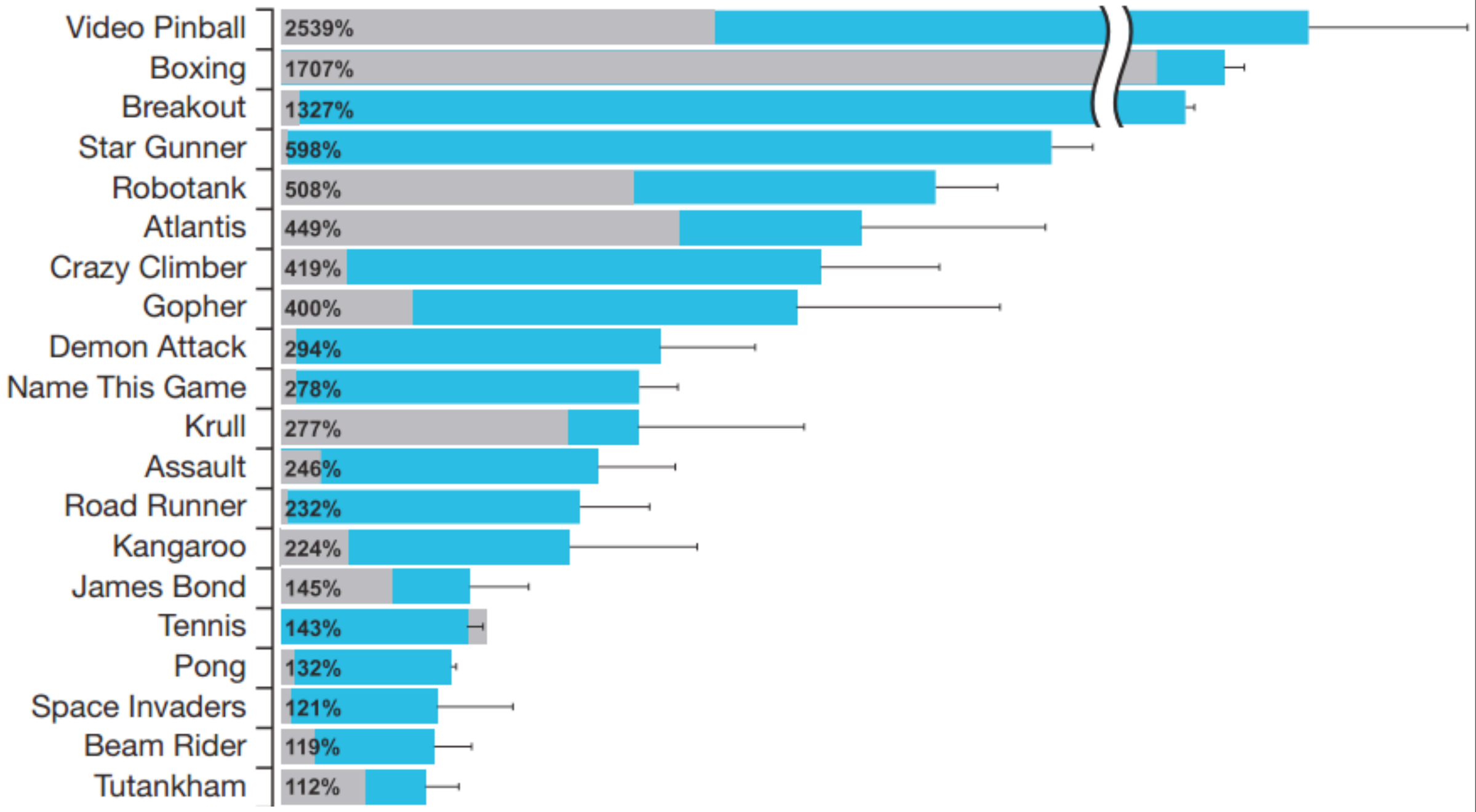
Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

End For

End For


Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%


Video Pinball
2539%
Boxing
1707%
Breakout
1327%
Star Gunner
598%
Robotank
508%
Atlantis
449%
Crazy Climber
419%
Gopher
400%
Demon Attack
294%
Name This Game
278%
Krull
277%
Assault
246%
Road Runner
232%
Kangaroo
224%
James Bond
145%
Tennis
143%
Pong
132%
Space Invaders
121%
Beam Rider
119%
Tutankham
112%

DQN - discussion

★ state dimensionality DQN vs robotics problems
★ randomness/predictability/repeatability?
★ signal of reward / task specification
★ difficulty of task
★ use of emulator vs. sampling in the physical world

mance over the course of training). Nevertheless, games demanding more temporally extended planning strategies still constitute a major challenge for all existing agents including DQN (for example, Montezuma's Revenge).

Credits

some material from:

Pieter Abbeel's Fall 2012: CS 287 Advanced Robotics
@ UC Berkeley

Sutton & Barto's book: http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html

Mnih, et al "Human-level control through deep reinforcement learning", Nature, Vol 518, p 529, 2015
http://dx.doi.org/10.1038/nature14236

ETH Zürich