TÉCNICO LISBOA

Planning, Learning and Intelligent Decision Making

Lecture 10

PADInt 2024

Stochastic approximation

- Iterative algorithms to compute the solution to the equation

$$\mathbb{E}[\mathrm{H}(\theta)] = 0$$

where H is some function that can be queried

- Take the general form

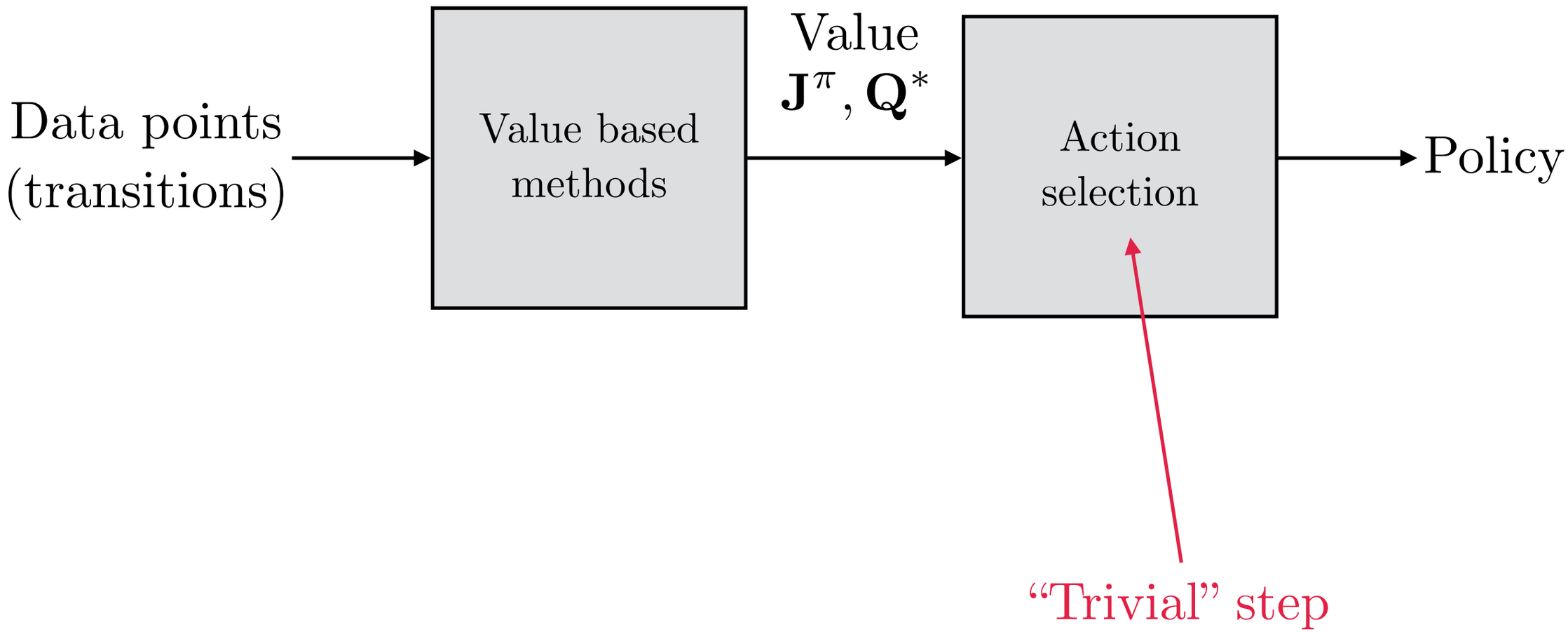
Sample of $\mathrm{H}(\theta)$

$$\theta_{n+1} = \theta_n + \alpha_n H(\theta_n)$$

TÉCNICO LISBOA

Value based RL

Value-based methods



Monte Carlo Policy Evaluation

- Given a trajectory obtained with policy π

$$\tau_k = \{x_{k,0}, c_{k,0}, x_{k,1}, c_{k,1}, \ldots, c_{k,T-1}, x_{k,T}\}$$

- Compute loss

$$L(\tau_k) = \sum_{t=0}^{T-1} \gamma^t c_{k,t}$$

- Update

$$J_{k+1}(x_{k,0}) = J_k(x_{k,0}) + \alpha_k(L(\tau_k) - J_k(x_{k,0}))$$

Monte Carlo Policy Evaluation

- Given a trajectory obtained with policy π

$$\tau_k = \{x_{k,0}, a_{k,0}, c_{k,0}, x_{k,1}, a_{k,1}, c_{k,1}, \ldots, a_{k,T-1}, c_{k,T-1}, x_{k,T}\}$$

- Compute loss

$$L(\tau_k) = \sum_{t=0}^{T-1} \gamma^t c_{k,t}$$

- Update

$$Q_{k+1}(x_{k,0}, a_{k,0}) = Q_k(x_{k,0}, a_{k,0}) + \alpha_k(L(\tau_k) - Q_k(x_{k,0}, a_{k,0}))$$

Monte Carlo Policy Optimization

- We use the principle of policy iteration
  - We start with random policy π_0
  - We evaluate the policy (compute Q^{π_0})
  - We compute an improved policy π_1
  - Repeat

TD(0)

- Given a sample (x_t, c_t, x_{t+1}), where the action was selected from π

- Compute

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t \left[c_t + \gamma J_t(x_{t+1}) - J_t(x_t)\right]$$

Why TD(0)? What's the 0 for?

Fixed points

- In vector form,

$$\mathbf{J}^\pi = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \boxed{\mathbf{J}^\pi}$$

We can replace this one

Fixed points

- In vector form,

$$\mathbf{J}^{\pi} = \mathbf{c}_{\pi} + \gamma \mathbf{P}_{\pi}[\mathbf{c}_{\pi} + \gamma \mathbf{P}_{\pi}\mathbf{J}^{\pi}]$$

Fixed points

- In vector form,

$$\mathbf{J}^{\pi} = \mathbf{c}_{\pi} + \gamma \mathbf{P}_{\pi} \mathbf{c}_{\pi} + \gamma^2 \mathbf{P}_{\pi}^2 \mathbf{J}^{\pi}$$

We can replace this one

Fixed points

- In vector form,

$$\mathbf{J}^{\pi} = \mathbf{c}_{\pi} + \gamma \mathbf{P}_{\pi} \mathbf{c}_{\pi} + \gamma^2 \mathbf{P}_{\pi}^2 [\mathbf{c}_{\pi} + \gamma \mathbf{P}_{\pi} \mathbf{J}^{\pi}]$$

Fixed points

- In vector form,

$$\mathbf{J}^{\pi} = \mathbf{c}_{\pi} + \gamma \mathbf{P}_{\pi} \mathbf{c}_{\pi} + \gamma^2 \mathbf{P}_{\pi}^2 \mathbf{c}_{\pi} + \gamma^3 \mathbf{P}_{\pi}^3 \mathbf{J}^{\pi}$$

... many steps later...

Fixed points

- In vector form,

$$\mathbf{J}^{\pi} = \sum_{n=0}^{N} \gamma^{n} \mathbf{P}_{\pi}^{n} \mathbf{c}_{\pi} + \gamma^{N+1} \mathbf{P}_{\pi}^{N+1} \mathbf{J}^{\pi}$$

Fixed points

- So we have all these versions:

$$\mathbf{J}^\pi = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{J}^\pi$$

$$\mathbf{J}^\pi = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{c}_\pi + \gamma^2 \mathbf{P}_\pi^2 \mathbf{J}^\pi$$

$$\mathbf{J}^\pi = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{c}_\pi + \gamma^2 \mathbf{P}_\pi^2 \mathbf{c}_\pi + \gamma^3 \mathbf{P}_\pi^3 \mathbf{J}^\pi$$

$$\dots$$

$$\mathbf{J}^\pi = \sum_{n=0}^{N} \gamma^n \mathbf{P}_\pi^n \mathbf{c}_\pi + \gamma^{N+1} \mathbf{P}_\pi^{N+1} \mathbf{J}^\pi$$

Variations

- We can build an algorithm out of each...

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t[c_t + \gamma J_t(x_{t+1}) - J_t(x_t)]$$

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t[c_t + \gamma c_{t+1} + \gamma^2 J_t(x_{t+2}) - J_t(x_t)]$$

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t[c_t + \gamma c_{t+1} + \gamma^2 c_{t+2} + \gamma^3 J_t(x_{t+3}) - J_t(x_t)]$$

...

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t \left[\sum_{n=0}^{N} \gamma^n c_{t+n} + \gamma^{N+1} J_t(x_{t+N+1}) - J_t(x_t)\right]$$

N-step TD

- These algorithms are generally called N-step TD:

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t \left[\sum_{n=0}^{N} \gamma^n c_{t+n} + \gamma^{N+1} J_t(x_{t+N+1}) - J_t(x_t) \right]$$

- Good points:
  - Each update uses informations from multiple steps

N-step TD

- These algorithms are generally called N-step TD:

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t \left[\sum_{n=0}^{N} \gamma^n c_{t+n} + \gamma^{N+1} J_t(x_{t+N+1}) - J_t(x_t) \right]$$

- Good points:
  - Each update uses informations from multiple steps
  - Updates are more informative
  - Converges (possibly) faster

N-step TD

- These algorithms are generally called N-step TD:

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t \left[\sum_{n=0}^{N} \gamma^n c_{t+n} + \gamma^{N+1} J_t(x_{t+N+1}) - J_t(x_t) \right]$$

- Not-so-good points:
  - Updates require "looking into the distant future"
  - Updates require "long" transitions $(x_t, c_t, c_{t+1}, \ldots, c_{t+N}, x_{t+N+1})$
  - Updates ignore intermediate state information

What N should we choose?

Revisited fixed points

- So we have all these versions:

$$(1-\lambda) \xrightarrow{\text{Multiply}} \mathbf{J}^\pi = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{J}^\pi$$

$$(1-\lambda)\lambda \xrightarrow{\text{Multiply}} \mathbf{J}^\pi = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{c}_\pi + \gamma^2 \mathbf{P}_\pi^2 \mathbf{J}^\pi$$

$$(1-\lambda)\lambda^2 \xrightarrow{\text{Multiply}} \mathbf{J}^\pi = \mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{c}_\pi + \gamma^2 \mathbf{P}_\pi^2 \mathbf{c}_\pi + \gamma^3 \mathbf{P}_\pi^3 \mathbf{J}^\pi$$

$$\vdots \qquad \cdots$$

$$(1-\lambda)\lambda^N \xrightarrow{\text{Multiply}} \mathbf{J}^\pi = \sum_{n=0}^{N} \gamma^n \mathbf{P}_\pi^n \mathbf{c}_\pi + \gamma^{N+1} \mathbf{P}_\pi^{N+1} \mathbf{J}^\pi$$

Revisited fixed points

- So we have all these versions:

Add them all

$$(1-\lambda)\mathbf{J}^\pi = (1-\lambda)[\mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{J}^\pi]$$

$$(1-\lambda)\lambda\mathbf{J}^\pi = (1-\lambda)\lambda[\mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{c}_\pi + \gamma^2 \mathbf{P}_\pi^2 \mathbf{J}^\pi]$$

$$(1-\lambda)\lambda^2\mathbf{J}^\pi = (1-\lambda)\lambda^2[\mathbf{c}_\pi + \gamma \mathbf{P}_\pi \mathbf{c}_\pi + \gamma^2 \mathbf{P}_\pi^2 \mathbf{c}_\pi + \gamma^3 \mathbf{P}_\pi^3 \mathbf{J}^\pi]$$

$$\ldots$$

$$(1-\lambda)\lambda^N\mathbf{J}^\pi = (1-\lambda)\lambda^N \left[\sum_{n=0}^{N} \gamma^n \mathbf{P}_\pi^n \mathbf{c}_\pi + \gamma^{N+1} \mathbf{P}_\pi^{N+1} \mathbf{J}^\pi\right]$$

Revisited fixed points

- We get:

$$\underbrace{(1-\lambda)\sum_{N=0}^{\infty}\lambda^N}_{=1}\mathbf{J}^{\pi} = (1-\lambda)\sum_{N=0}^{\infty}\lambda^N\left[\sum_{n=0}^{N}\gamma^n\mathbf{P}_{\pi}^{n}\mathbf{c}_{\pi}+\gamma^{N+1}\mathbf{P}_{\pi}^{N+1}\mathbf{J}^{\pi}\right]$$

Revisited fixed points

- We get:

$$\mathbf{J}^{\pi} = (1-\lambda)\sum_{N=0}^{\infty}\lambda^{N}\left[\sum_{n=0}^{N}\gamma^{n}\mathbf{P}_{\pi}^{n}\mathbf{c}_{\pi} + \gamma^{N+1}\mathbf{P}_{\pi}^{N+1}\mathbf{J}^{\pi}\right]$$

Chewing on this for a bit

$$\mathbf{J}^{\pi} = \sum_{n=0}^{\infty}\lambda^{n}\gamma^{n}\mathbf{P}_{\pi}^{n}\left[\mathbf{c}_{\pi} + \gamma\mathbf{P}_{\pi}\mathbf{J}^{\pi} - \mathbf{J}^{\pi}\right] + \mathbf{J}^{\pi}$$

Revisited fixed points

- We get:

$$\mathbf{J}^{\pi} = (1-\lambda)\sum_{N=0}^{\infty}\lambda^{N}\left[\sum_{n=0}^{N}\gamma^{n}\mathbf{P}_{\pi}^{n}\mathbf{c}_{\pi} + \gamma^{N+1}\mathbf{P}_{\pi}^{N+1}\mathbf{J}^{\pi}\right]$$

Chewing on this for a bit

$$\sum_{n=0}^{\infty}\lambda^{n}\gamma^{n}\mathbf{P}_{\pi}^{n}\left[\mathbf{c}_{\pi} + \gamma\mathbf{P}_{\pi}\mathbf{J}^{\pi} - \mathbf{J}^{\pi}\right] = 0$$

Finally...

- We have a new algorithm:

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t \sum_{n=0}^{\infty} \lambda^n \gamma^n [c_{t+n} + \gamma J_t(x_{t+n+1}) - J_t(x_{t+n})]$$

- We no longer ignore intermediate states
- We no longer need to select an N

However...

- We now need an infinite trajectory!

DON'T PANIC

What does this mean?

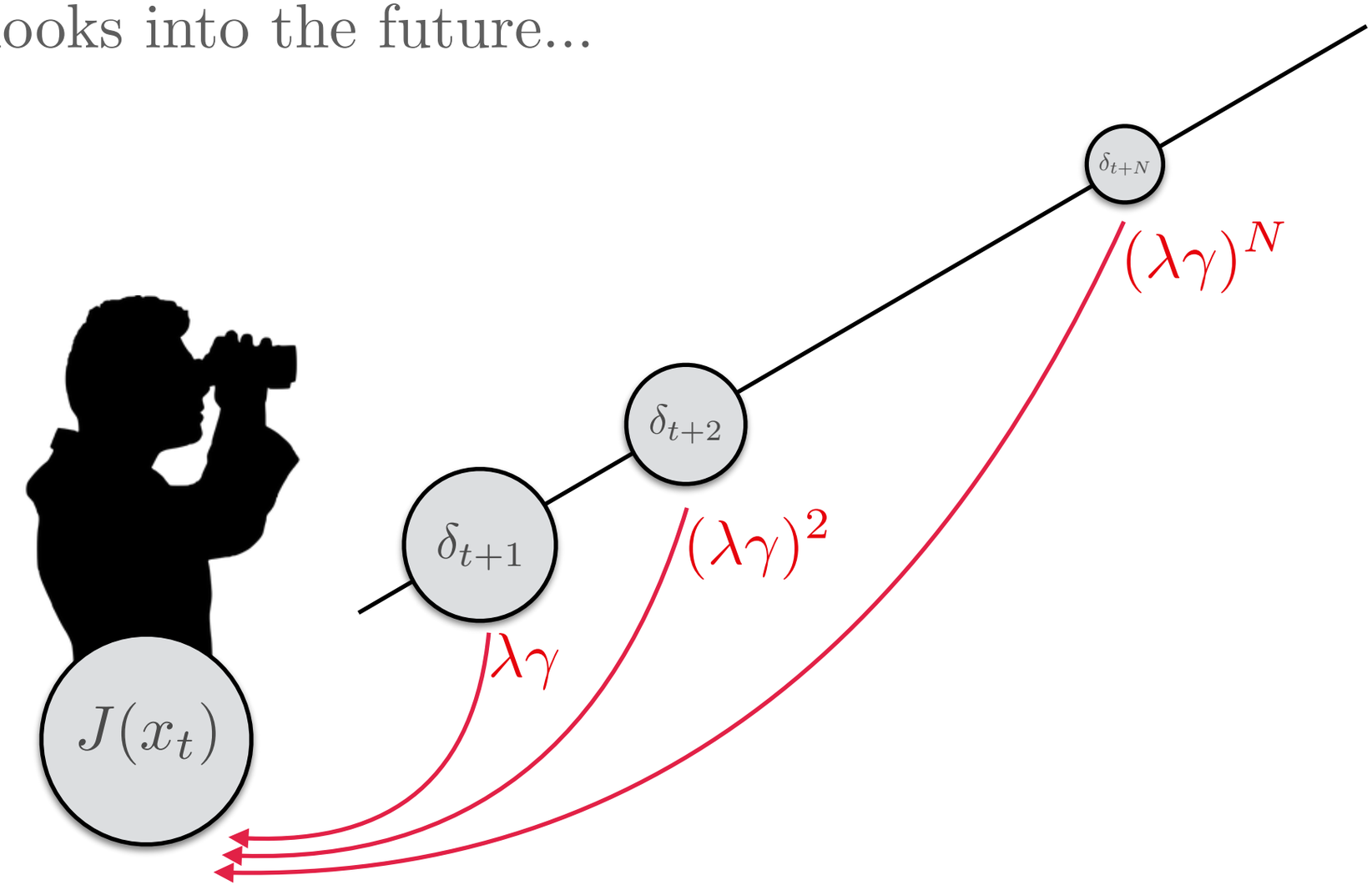
- Let's look at this carefully:

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t \sum_{n=0}^{\infty} \lambda^n \gamma^n [c_{t+n} + \gamma J_t(x_{t+n+1}) - J_t(x_{t+n})]$$

- All future temporal differences contribute to current value
- Differences further away contribute less (weighted down by $\gamma < 1$ and $\lambda \leq 1$)

Forward view

- Agent looks into the future...

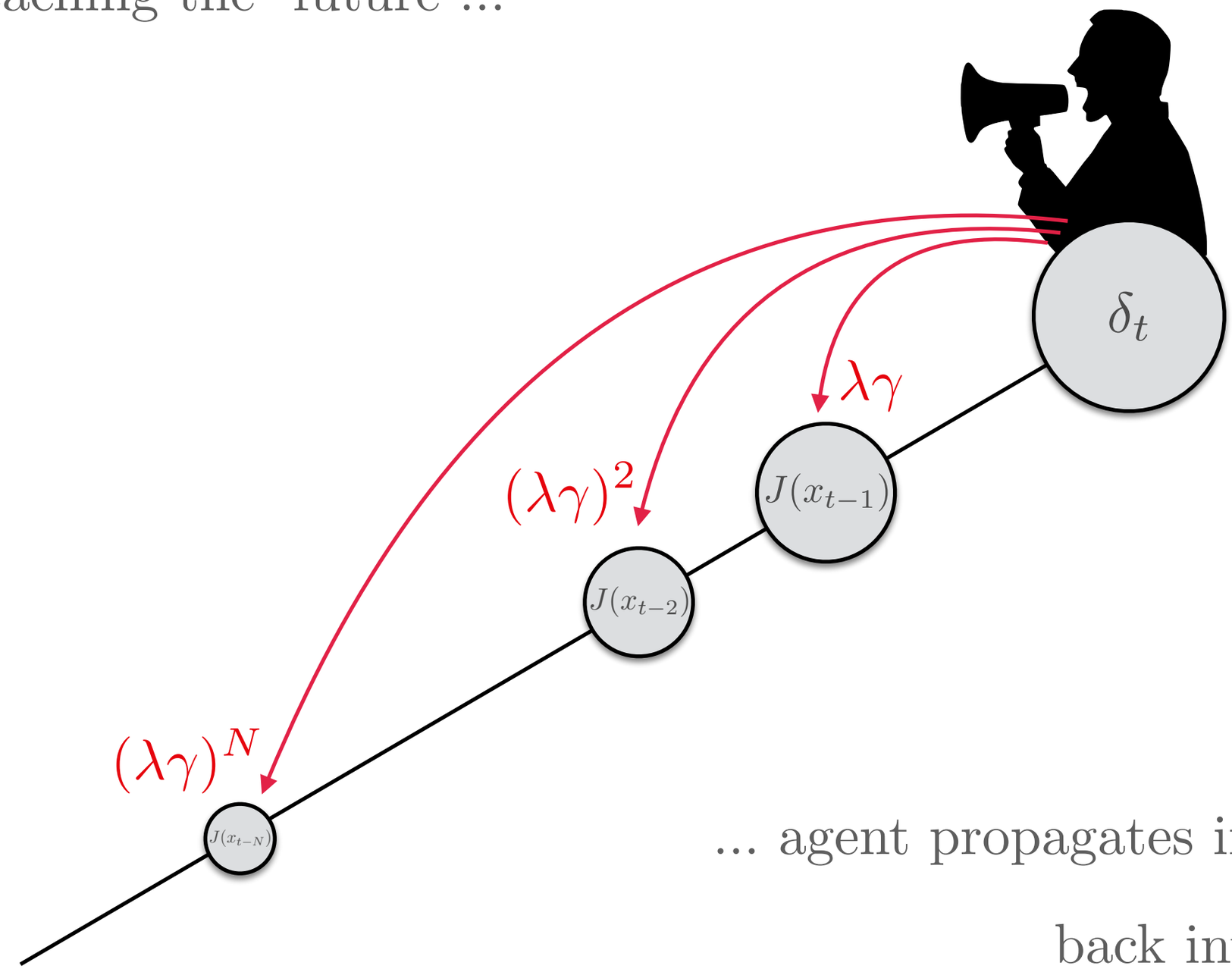


... and weights in all future information

... but we can look at it the other way around...

Backward view

- Upon reaching the "future"...



... agent propagates information

back into the past

What does this mean?

- We track how much current state contributes to previous states:
  - We store how long ago previous states were visited
  - Weight current temporal difference accordingly

What does this mean?

- Algorithmically,

$$J_{t+1}(x) = J_t(x) + \alpha_t z_{t+1}(x)[c_t + \gamma J_t(x_{t+1}) - J_t(x_t)]$$

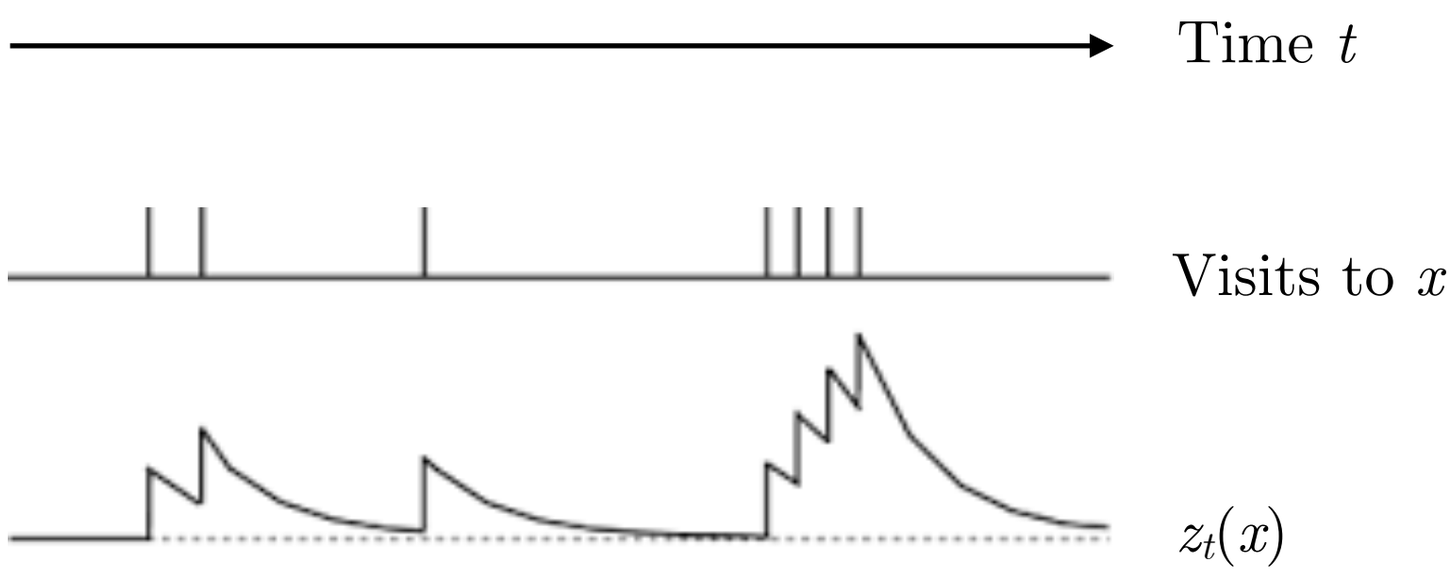
This factor traces how much $J_t(x)$ should "receive" from δ_t

What does this mean?

- Algorithmically,

$$J_{t+1}(x) = J_t(x) + \alpha_t z_{t+1}(x)[c_t + \gamma J_t(x_{t+1}) - J_t(x_t)]$$

Eligibility trace



What does this mean?

- Algorithmically,

$$J_{t+1}(x) = J_t(x) + \alpha_t z_{t+1}(x)[c_t + \gamma J_t(x_{t+1}) - J_t(x_t)]$$

$$z_{t+1}(x) = \lambda\gamma z_t(x) + \mathbb{I}(x = x_t)$$

- In this algorithm:
  - Each update uses informations from multiple steps
  - No looking in the future
  - No “long transitions” required

TD(λ)

- Given a sample (x_t, c_t, x_{t+1}), where the action was selected from π
- Compute

$$z_{t+1}(x) = \lambda\gamma z_t(x) + \mathbb{I}(x = x_t)$$

$$J_{t+1}(x) = J_t(x) + \alpha_t z_{t+1}(x)[c_t + \gamma J_t(x_{t+1}) - J_t(x_t)]$$

- For $\lambda = 0$, we recover TD(0) (the previous algorithm)

... hence the 0 in TD(0)

Does this work?

Theorem: For any $0 \leq \lambda \leq 1$, as long as every state is visited infinitely often, TD(λ) converges to J^π w.p.1.

What about Q^*?

Computing Q^*

- We have that

$$Q^*(x,a) = c(x,a) + \gamma \sum_{y \in \mathcal{X}} \mathbf{P}_a(y \mid x) \min_{a' \in \mathcal{A}} Q^*(y,a')$$

which, back in lecture 6, we wrote as

$$\mathbf{Q}^* = \mathbf{H}\mathbf{Q}^*$$

$\mathbf{Q}^*$ is a fixed point

Computing Q^*

- Alternatively, for each state x, we can write

$$Q^*(x,a) = \mathbb{E}_{\mathrm{y}\sim\mathbf{P}(x,a)}\left[c(x,a) + \gamma \min_{a'\in\mathcal{A}} Q^*(\mathrm{y},a')\right]$$

Random variable

Computing Q^*

- Alternatively, for each state x, we can write

$$\mathbb{E}_{\mathrm{y}\sim\mathbf{P}(x,a)}\left[c(x,a)+\gamma\min_{a'\in\mathcal{A}}Q^*(\mathrm{y},a')-Q^*(x,a)\right]=0$$

Q^* is the zero of this equation

Computing Q^*

- Alternatively, for each state x, we can write

$$\mathbb{E}_{\mathrm{y}\sim\mathbf{P}(x,a)}\left[c(x,a)+\gamma\min_{a'\in\mathcal{A}}Q^*(\mathrm{y},a')-Q^*(x,a)\right]=0$$

- Using our stochastic approximation trick,

$$Q_{t+1}(x_t,a_t)=Q_t(x_t,a_t)+\alpha_t\left[c_t+\gamma\min_{a'\in\mathcal{A}}Q_t(x_{t+1},a')-Q_t(x_t,a_t)\right]$$

Sample of $\mathrm{H}(\theta)$

Q-learning

- The algorithm

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t \left[c_t + \gamma \min_{a' \in \mathcal{A}} Q_t(x_{t+1}, a') - Q_t(x_t, a_t) \right]$$

  is called **Q-learning**, and is the most widely known RL algorithm

- The quantity

$$\delta_t = c_t + \gamma \min_{a' \in \mathcal{A}} Q_t(x_{t+1}, a') - Q_t(x_t, a_t)$$

  is also a **temporal difference** at time step t

Q-learning

- Given a sample (x_t, a_t, c_t, x_{t+1}),

- Compute

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t \left[c_t + \gamma \min_{a' \in \mathcal{A}} Q_t(x_{t+1}, a') - Q_t(x_t, a_t) \right]$$

Does this work?

Theorem: As long as every state-action pair is visited infinitely often, Q-learning converges to Q^* w.p.1.

Let's revisit this

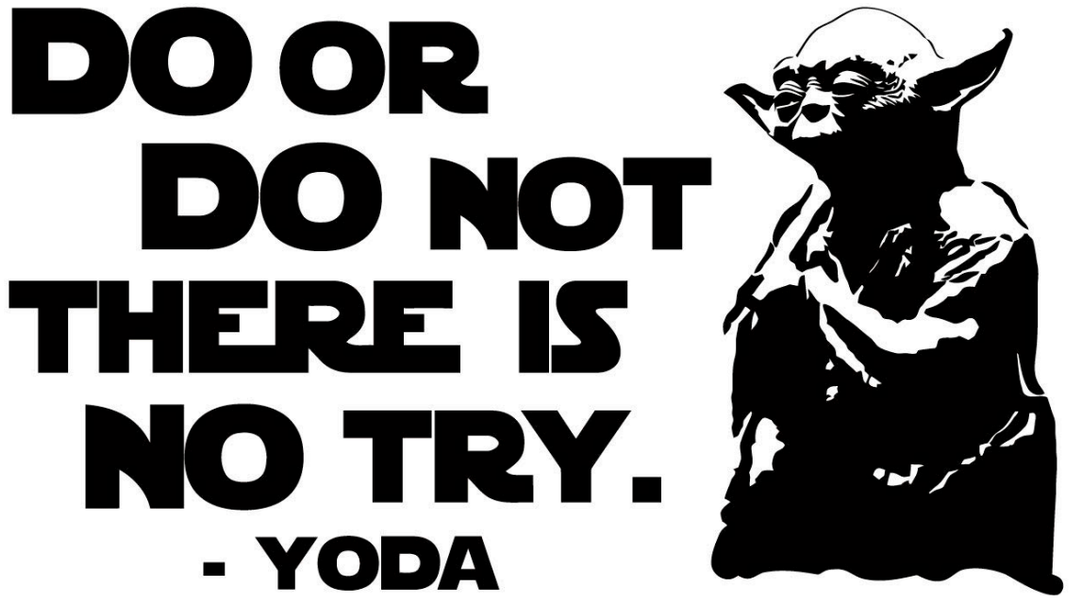
Exploration vs exploitation

- How can we visit every state action pair **infinitely often**?
  - In practice, "infinitely often" means a "large number of times"
  - The agent needs to try all actions in all states many times
  - **But this means that the agent will keep doing sub-optimal actions for a long time!**

On the other hand...

Exploration vs exploitation

- We want the agent to start acting "reasonably" as soon as possible
  - In practice, this means using the knowledge already available
  - The agent needs to stop "trying" and start "doing"

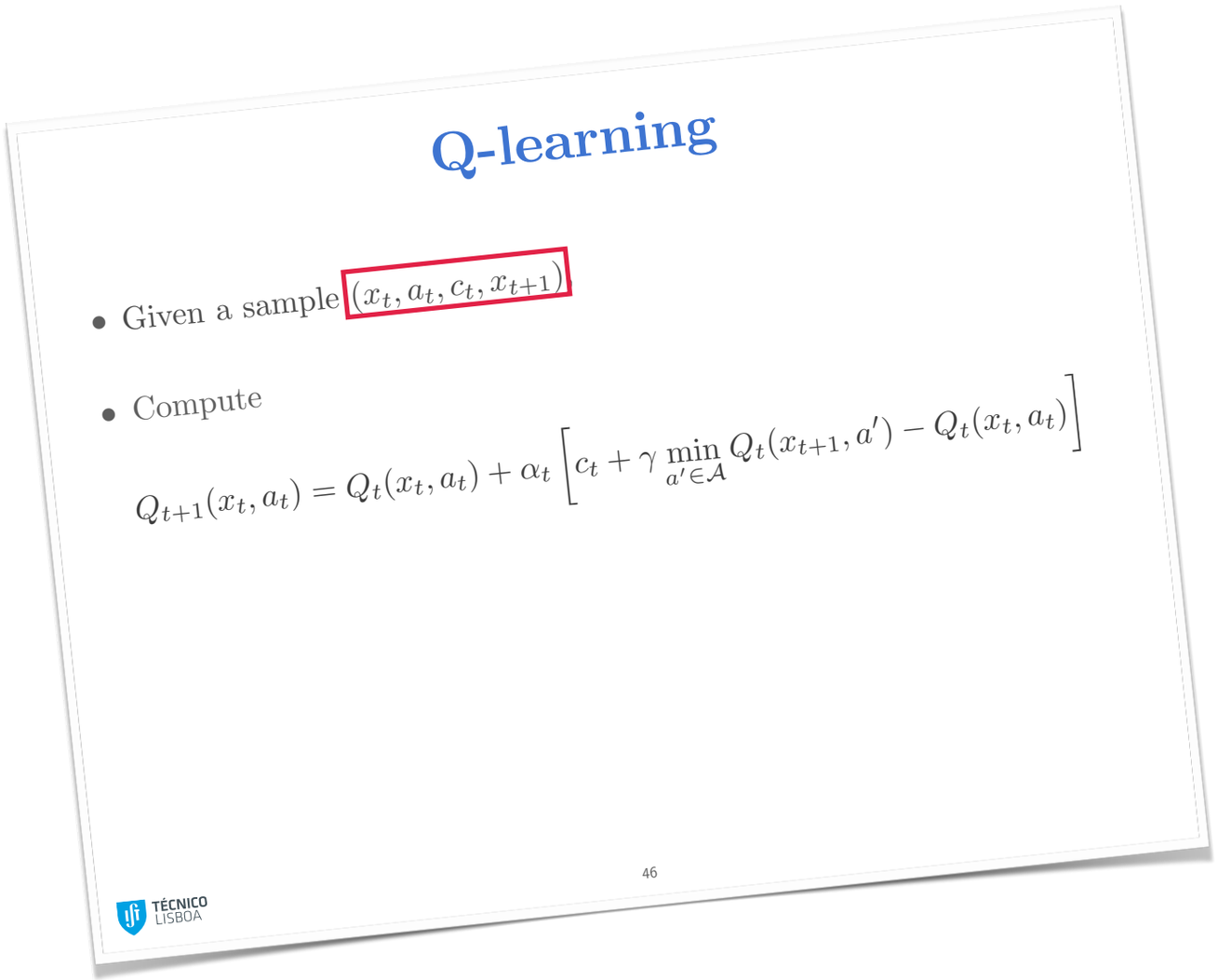


Exploration vs exploitation

- The agent needs to balance:
  - **Exploration**, i.e., trying new actions (or actions that have been less experimented with)
  - **Exploitation**, i.e., using the knowledge already acquired to select the seemingly better actions

Exploration vs exploitation tradeoff

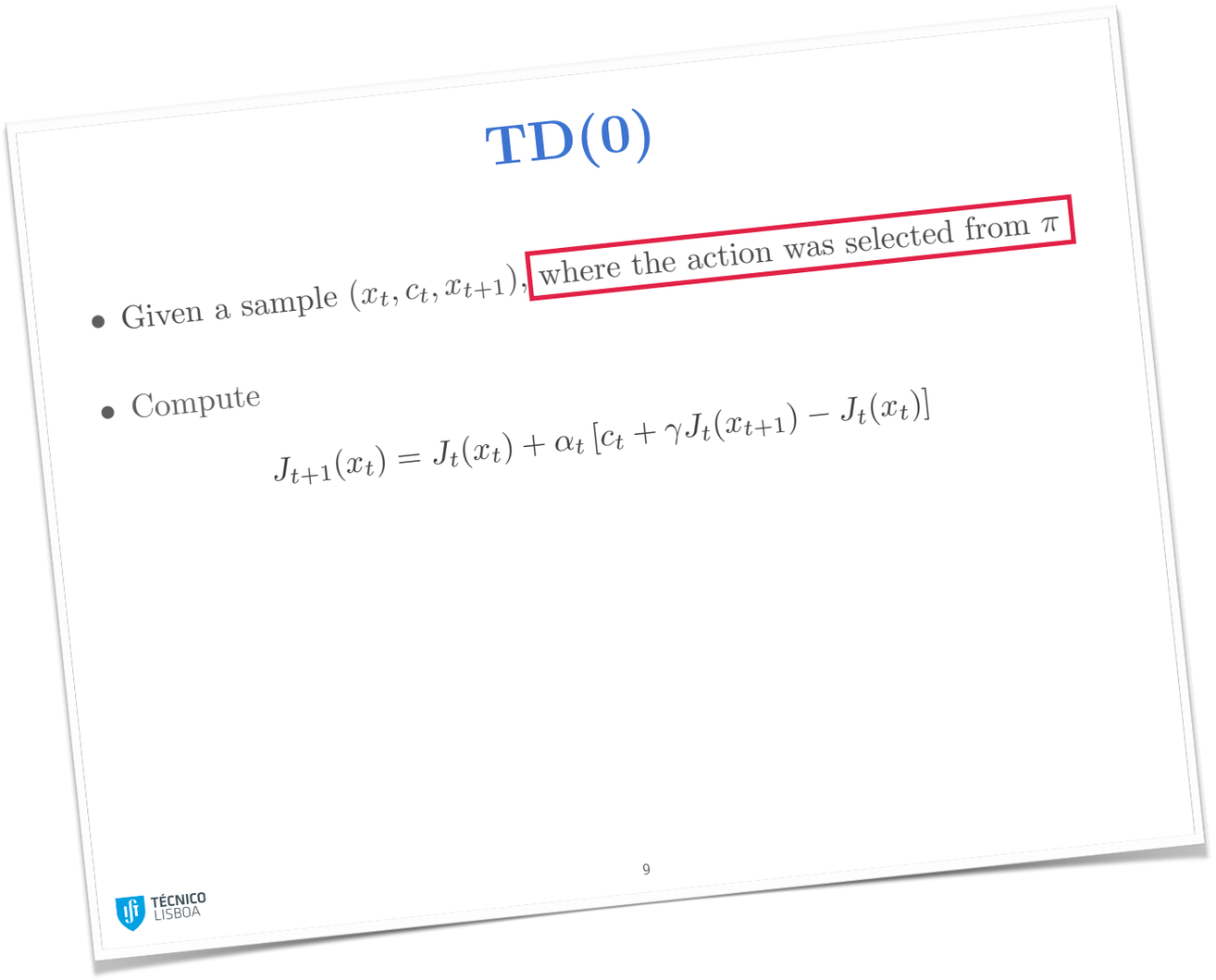
Implications of exploration



- When computing Q^*, we use samples (x_t, a_t, c_t, x_{t+1})
- It is "irrelevant" how these samples are obtained

... however...

Implications of exploration



- When computing J^π, we use samples (x_t, c_t, x_{t+1}) obtained when the action was selected from π

Implications of exploration

- This means that...
  - We cannot just "explore"
  - We must be able to select x_t arbitrarily (c_t and x_{t+1} then come from the environment)
- This is a restrictive condition known as "exploring starts"
  - Requires ability to "reset" the environment to any desired state

Heuristics for good E x E tradeoff

Heuristics for E x E

- ϵ-greedy
  - Agent selects a random action with probability ϵ (exploration)
  - Agent selects the greedy action (action with smallest Q-value) with probability $1 - \epsilon$ (exploitation)

ϵ may decay with time

Heuristics for E x E

- Boltzmann policy
  - Agent selects each action $a \in \mathcal{A}$ with probability

$$\pi(a \mid x) = \frac{e^{-\eta Q_t(x,a)}}{\sum_{a' \in \mathcal{A}} e^{-\eta Q_t(x,a')}}$$

Actions with large Q (cost-to-go) have smaller probability

η controls how greedy the policy is

Heuristics for E x E

- Boltzmann policy
  - Agent selects each action $a \in \mathcal{A}$ with probability

$$\pi(a \mid x) = \frac{e^{-\eta Q_t(x,a)}}{\sum_{a' \in \mathcal{A}} e^{-\eta Q_t(x,a')}}$$

- The parameter η controls the exploration and may grow with time

Heuristics for E x E

- Optimistic initialization
  - All Q-values are initialized to 0 or negative value
  - Agent always selects greedy action (in case of tie, it randomizes)
    - Initially, all actions are equally good → agent randomizes (exploration)
    - As more knowledge is available, suboptimal actions become less interesting → agent approaches optimal (exploitation)

What's the impact of the learning policy?

Learning policy

- What is the impact of the learning policy in the algorithm?

Learning policy

- Q-learning is independent of the learning policy

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t \left[c_t + \gamma \min_{a' \in \mathcal{A}} Q_t(x_{t+1}, a') - Q_t(x_t, a_t) \right]$$

Learn Q-values
for optimal policy

On-policy updates

- What about the Q-values for the actual learning policy?

$$Q^{\pi}(x,a) = \mathbb{E}_{\mathrm{y}\sim\mathbf{P}(x,a),\mathrm{a}'\sim\pi(\mathrm{y})}\left[c(x,a) + \gamma Q^{\pi}(\mathrm{y},\mathrm{a}')\right]$$

Random variables

On-policy updates

- What about the Q-values for the actual learning policy?

$$\boxed{\mathbb{E}_{\mathrm{y}\sim\mathbf{P}(x,a),\mathrm{a}'\sim\pi(\mathrm{y})}\left[c(x,a)+\gamma Q^{\pi}(\mathrm{y},\mathrm{a}')-Q^{\pi}(x,a)\right]}=0$$

Q^{π} is the zero of this equation

On-policy updates

- What about the Q-values for the actual learning policy?

$$\mathbb{E}_{\mathrm{y}\sim\mathbf{P}(x,a),\mathrm{a}'\sim\pi(\mathrm{y})}\left[c(x,a)+\gamma Q^{\pi}(\mathrm{y},\mathrm{a}')-Q^{\pi}(x,a)\right]=0$$

- Using our stochastic approximation trick,

$$Q_{t+1}(x_t,a_t)=Q_t(x_t,a_t)+\alpha_t\left[c_t+\gamma Q_t(x_{t+1},a_{t+1})-Q_t(x_t,a_t)\right]$$

Sample of $\mathrm{H}(\theta)$

On-policy updates

- To run the previous update, we need the following information:

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t \left[c_t + \gamma Q_t(x_{t+1}, a_{t+1}) - Q_t(x_t, a_t) \right]$$

State — Action — Reinforcement signal (cost) — Next state — Next action

S A R S A

On-policy updates

- To run the previous update, we need the following information:

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t \left[c_t + \gamma Q_t(x_{t+1}, a_{t+1}) - Q_t(x_t, a_t)\right]$$

S A R S A

SARSA

- The algorithm

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t \left[c_t + \gamma Q_t(x_{t+1}, a_{t+1}) - Q_t(x_t, a_t)\right]$$

  is called **SARSA**

- The quantity

$$\delta_t = c_t + \gamma Q_t(x_{t+1}, a_{t+1}) - Q_t(x_t, a_t)$$

  is also a **temporal difference** at time step t

SARSA

- Given a sample $(x_t, a_t, c_t, x_{t+1}, a_{t+1})$,

- Compute

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t \left[c_t + \gamma Q_t(x_{t+1}, a_{t+1}) - Q_t(x_t, a_t)\right]$$

Q-learning vs SARSA

- Q-learning is an **off-policy** algorithm

  Learns the value of one policy while following another

- SARSA (like TD(λ)) is an **on-policy** algorithm

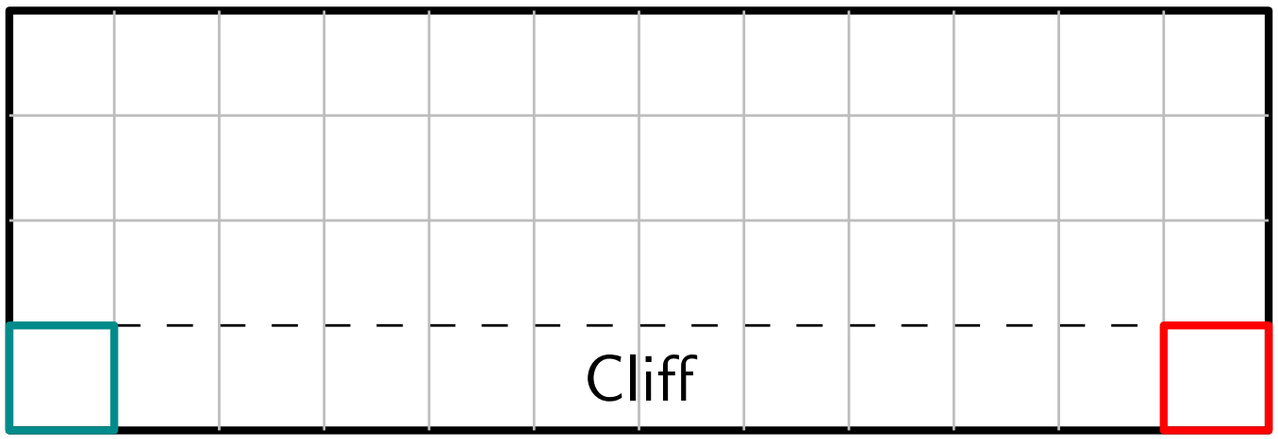
  Learns the value of the policy that it follows

- For SARSA to learn Q^* must be combined with policy improvement

  - For example, ε-greedy with decaying ε

Example

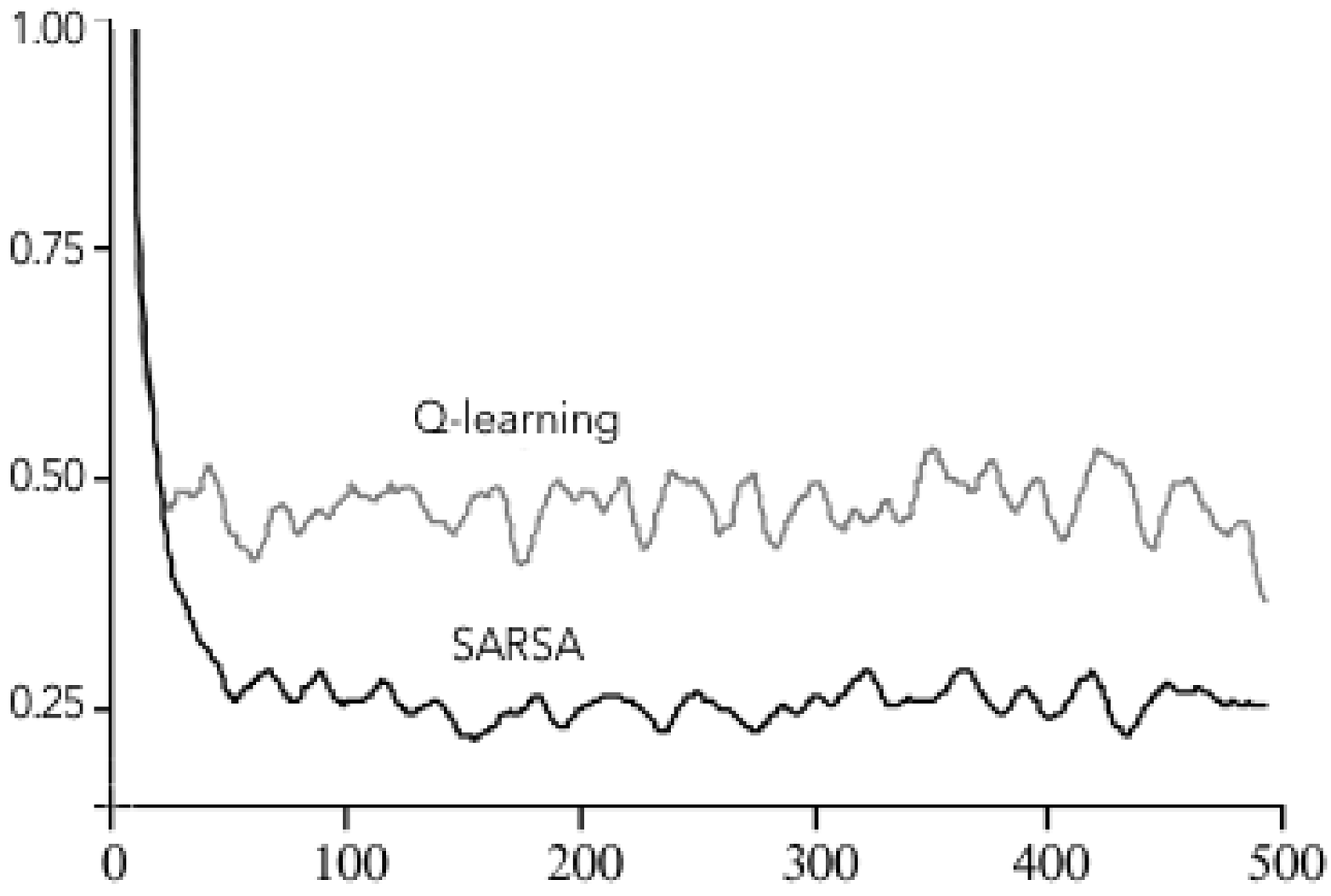
- A mouse must reach the cheese and avoid the cliff



- Cost cliff: 1
- Cost otherwise: 0.01

Example

- Run Q-learning and SARSA with ε-greedy exploration, ε=0.1



Q-learning vs SARSA

- Q-learning always computes the optimal Q-function
- SARSA computes the Q-values for its current policy
  - Needs policy improvements to converge to Q^*
- SARSA often leads to more stable updates