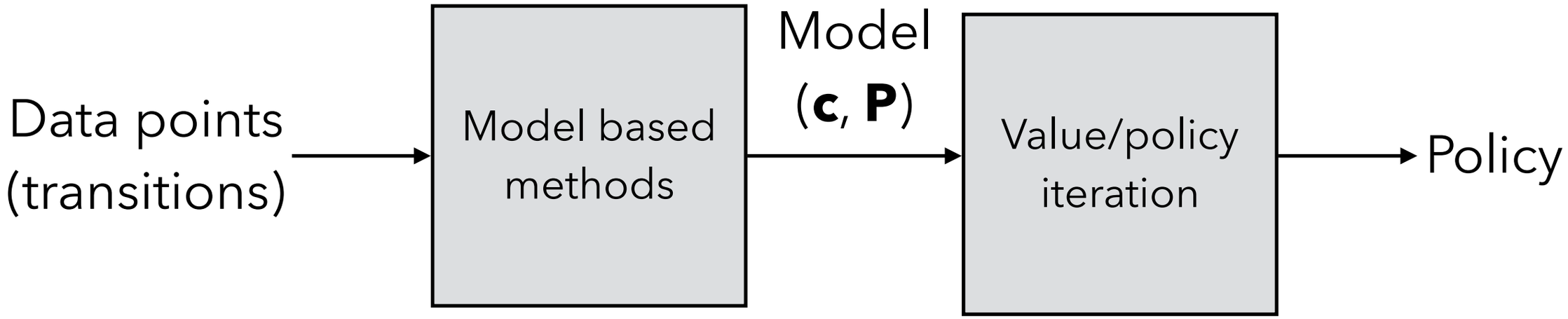
TÉCNICO LISBOA

Planning, Learning and Decision Making

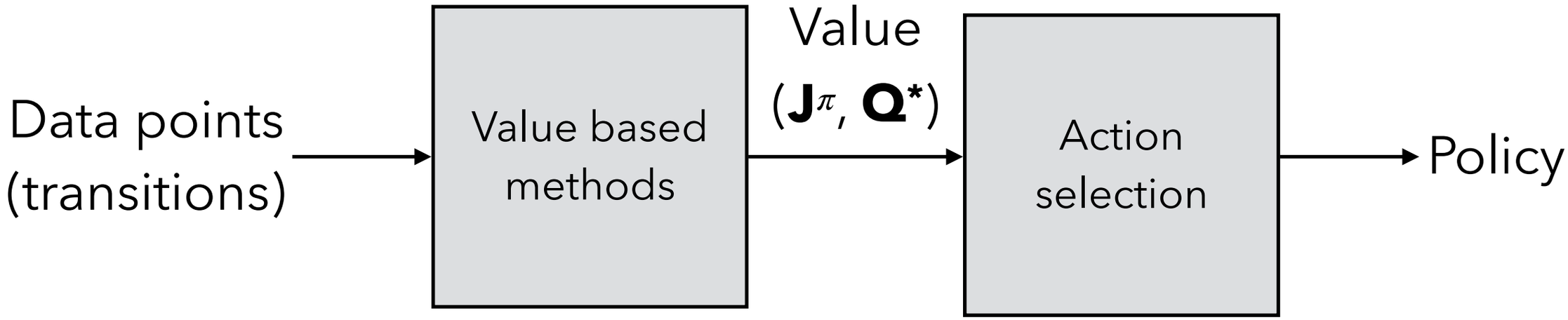
Lecture 19. Reinforcement learning: Q-learning and SARSA

Model based RL



Value based RL

- Value-based methods:



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 get TD(0) (the previous algorithm)

Value based RL
Episode II

Computing J^π

- From

$$J^\pi(x) = c_\pi(x) + \gamma \sum_{y \in \mathcal{X}} \mathsf{P}_\pi(y \mid x) J^\pi(y)$$

we derived

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

Stochastic approximation

What about Q^*?

Computing Q*

- We have that

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

which, back in lecture 8, we wrote as

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

Q^* is a fixed point

Computing Q*

- Alternatively, for each pair (*x*, *a*), we can write

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

Random variable

Computing Q*

- Alternatively, for each pair (x, a), we can write

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

Q* is the zero of this expression

Computing Q*

- Using the stochastic approximation/computation of the mean recipe

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

Can be seen as

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t [\mathbf{H} Q_t(x_t, a_t) - Q_t(x_t, a_t) + \varepsilon]$$

Q-learning

- This is the most famous RL algorithm and is known as **Q-learning**
- The quantity

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

  is also a **temporal difference**
- Therefore, we can say that Q-learning is also a temporal-difference learning algorithm (although not TD-learning)

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.

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”

"DO OR DO NOT.

THERE IS NO TRY."

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

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 - ε (exploitation)

ε may decay with time

Heuristics for E x E

- Boltzmann policy
  - Agent selects each action a ∈ $\mathscr{A}$ 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 ∈ $\mathscr{A}$ 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)

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?

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

Random variables

On-policy updates

- Using our stochastic approximation recipe,

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

State x_t — Action a_t — Reinforcement (cost) c_t — Next state x_{t+1} — Next action a_{t+1}

S A R S A

On-policy updates

- Using our stochastic approximation recipe,

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

SARSA

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

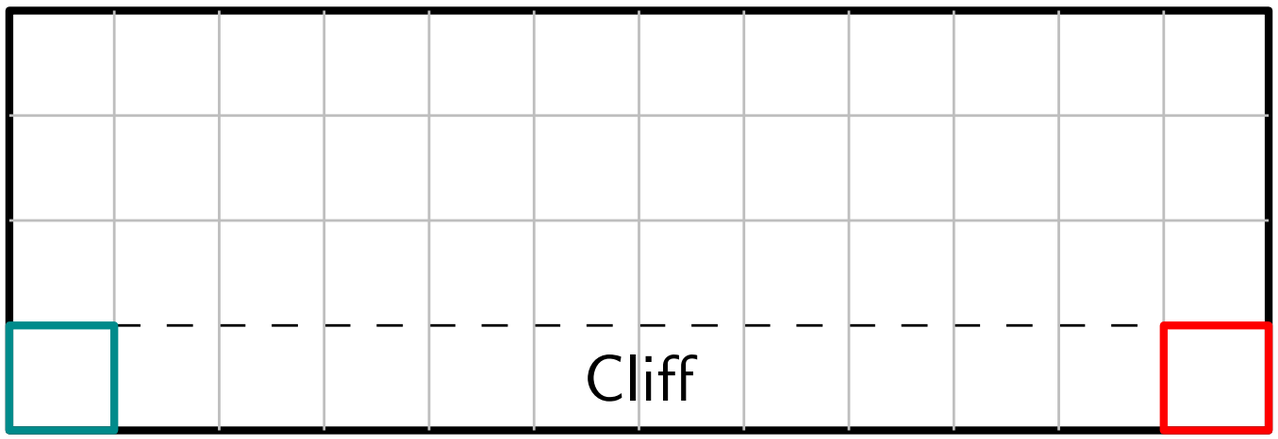
  - For example, ε-greedy with decaying ε

Q-learning vs SARSA

- SARSA often leads to more stable updates

Example

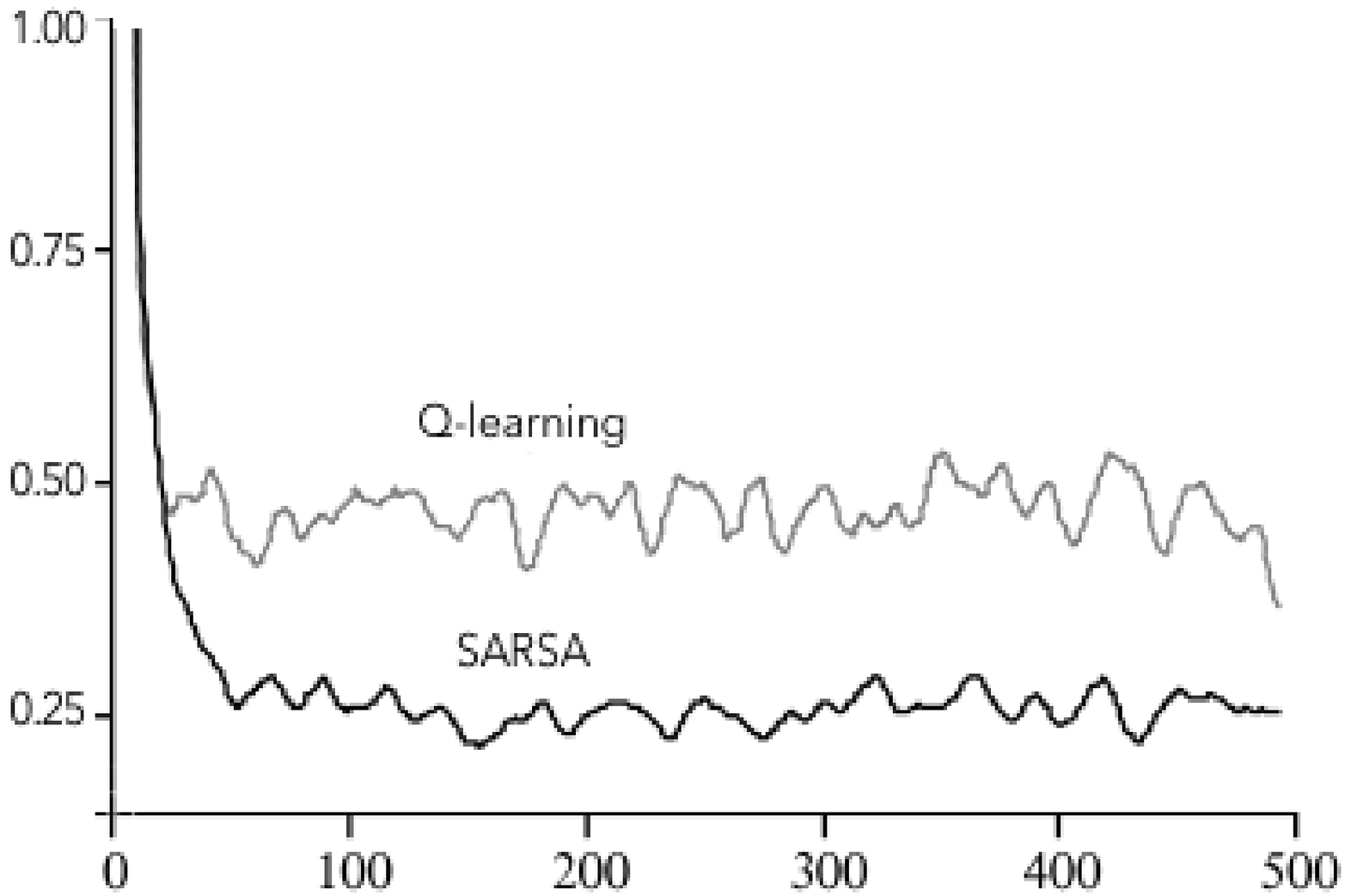
- A mouse must reach the cheese and avoid the cliff



- Cost cliff: 1
- Cost otherwise: 0.01

Example

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



Q-learning vs SARSA

- SARSA often leads to more stable updates

Large domains

Large domains

- How do these methods work in large domains (e.g., Tetris, Gammon, Go)?

Function approximation

- We can no longer represent J^{π} and Q^{*} exactly
- We must resort to some form of approximation
- Instead of allowing arbitrary functions, methods restrict to pre-specified families of functions
- Families provide good representations → methods perform well
- Families provide bad representations → methods perform poorly

Linear approximation

- Each state is described as a vector of "features"

$$\boldsymbol{\phi}(x)$$

- J and Q are represented as combinations of features

$$J^{\pi}(x) \approx \boldsymbol{\phi}^{\top}(x)\boldsymbol{w} \qquad Q^{*}(x,a) \approx \boldsymbol{\phi}^{\top}(x,a)\boldsymbol{w}$$

- Methods compute best weights for the combinations

TD(λ) with linear FA

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

$$\boldsymbol{z}_t = \lambda\gamma\boldsymbol{z}_{t-1} + \boldsymbol{\phi}(x_t)$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t \boldsymbol{z}_{t+1}[c_t + \gamma J_t(x_{t+1}) - J_t(x_t)]$$

where

$$J_t(x) = \boldsymbol{\phi}^\top(x)\boldsymbol{w}_t$$

Q-learning with linear FA

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

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

where

$$Q_t(x, a) = \boldsymbol{\phi}^\top(x, a) \boldsymbol{w}_t$$

SARSA with linear FA

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

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t \boldsymbol{\phi}(x, a) \left[c_t + \gamma Q_t(x_{t+1}, a_{t+1}) - Q_t(x_t, a_t) \right]$$

Unfortunately…

- TD(λ) does retain its convergence guarantees
- SARSA is more stable than Q-learning, as long as learning policy changes smoothly with Q_t
- Q-learning with linear function approximation does not retain its convergence guarantees…

Batch RL

Batch RL

- All methods seen so far are *online*
  - Agent collects samples and performs immediate updates
  - Sample is not used again

Batch RL

- Alternatively, agent could collect a big batch of data and use it to learn
  - Data is better used
  - Can take advantage of supervised learning
  - More stable (convergence problems “solved”)

Batch RL

- The idea:
  - Given current estimate,

$$Q_t$$

  - … given a batch of transitions

$$\{t_1 = (x_1, a_1, c_1, x'_1), \ldots, t_N(x_N, a_N, c_N, x'_N)\}$$

  - … build dataset

$$\left\{ (x_1, a_1, \underbrace{c_1 + \gamma \min_{a \in \mathcal{A}} Q_t(x'_1, a)}_{\text{Target}}), \ldots (x_N, a_N, \underbrace{c_N + \gamma \min_{a \in \mathcal{A}} Q_t(x'_N, a)}_{\text{Target}}) \right\}$$

Input (for (x_1, a_1)) Input (for (x_N, a_N))

Batch RL

- We can now use our preferred supervised learning algorithm and find Q to minimize

$$\mathcal{E} = \sum_{n=1}^{N} (c_n + \gamma \min_{a \in \mathcal{A}} Q_t(x'_n, a) - Q(x_n, a_n))^2$$

- The resulting Q will correspond to Q_{t+1}
- The process repeats

Batch RL

- Problem:

$$\mathcal{E} = \sum_{n=1}^{N} \boxed{(c_n + \gamma \min_{a \in \mathcal{A}} Q_t(x'_n, a)} - Q(x_n, a_n))^2$$

Target depends on Q_t

- Dataset changes from one iteration to the other
- … we're chasing a "moving target"

Batch RL and NN

- Two tricks to minimize error

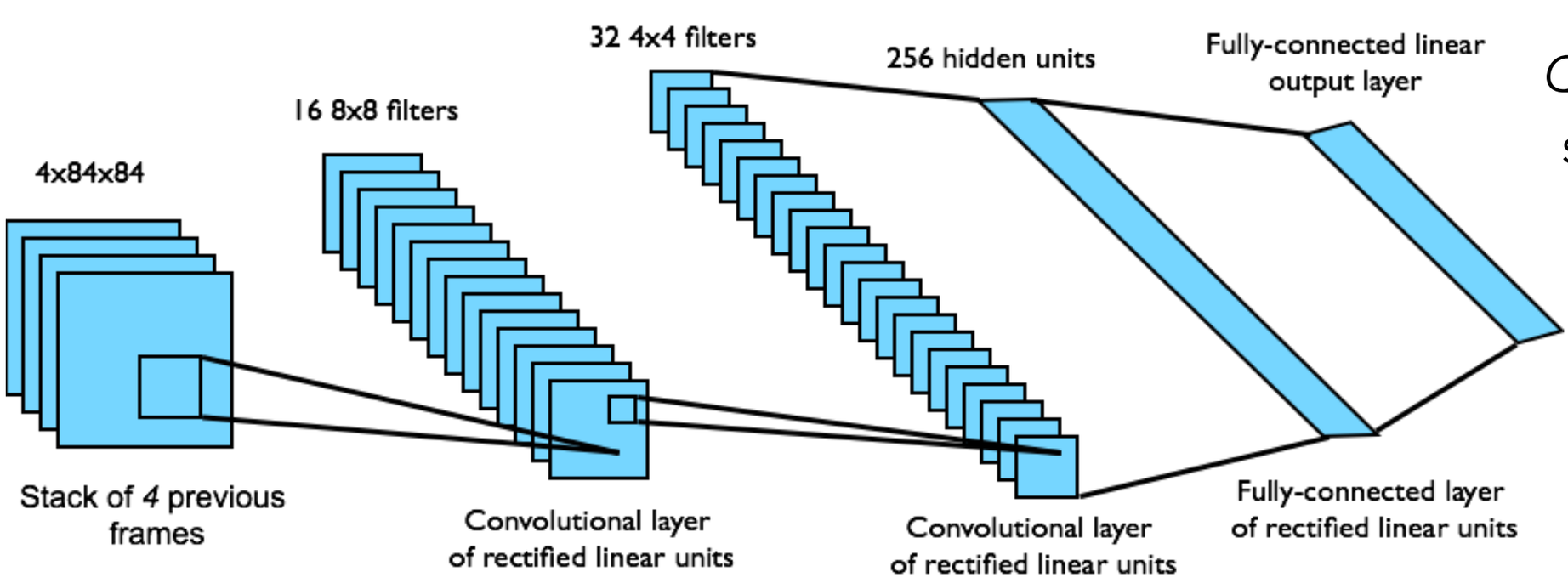
$$\mathcal{E} = \sum_{n=1}^{N} (c_n + \gamma \min_{a \in \mathcal{A}} Q_t(x'_n, a) - Q(x_n, a_n))^2$$

  - Q_t is held fixed for several iterations (in a **target network**)
  - Samples are not ordered, but selected randomly from a buffer (**experience replay**)

Batch RL and NN

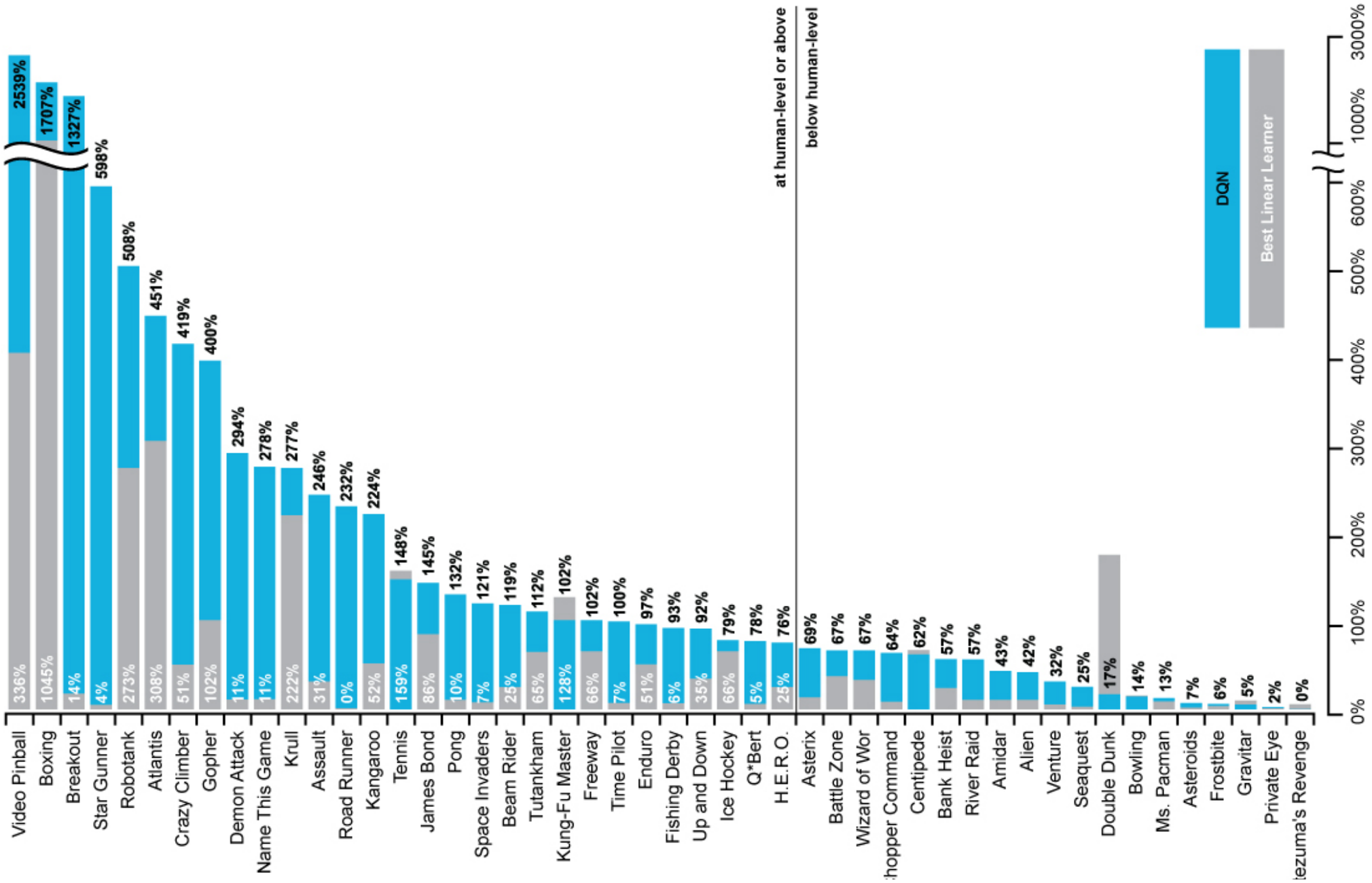
- For example, if the Q-values are represented using a neural network, we recover the architecture behind the ATARI playing program

Pixels of last 4 frames (state)



Q^* for current state and all 18 actions

Batch RL and NN



Batch RL

- A similar process can be used to compute J^{π}

Summarizing...

Fitted Q-iteration

- Given a parameterized family of functions $Q_{\mathbf{w}}$
- Given a batch of transitions

$$\{(x_1, a_1, c_1, x'_1), \ldots, (x_N, a_N, c_N, x'_N)\}$$

- Set, at each iteration t,

$$Q_{t+1} = Q_{\mathbf{w}^*}$$

- where

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n=1}^{N} (c_n + \gamma \min_{a \in \mathcal{A}} Q_t(x'_n, a) - Q_{\mathbf{w}}(x_n, a_n))^2$$

Fitted value-iteration

- Given a parameterized family of functions $J_{\mathbf{w}}$
- Given a batch of transitions

$$\{(x_1, c_1, x'_1), \ldots, (x_N, c_N, x'_N)\}$$

- Set, at each iteration t,

$$J_{t+1} = J_{\mathbf{w}^*}$$

- where

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n=1}^{N} (c_n + \gamma J_t(x'_n) - J_{\mathbf{w}}(x_n))^2$$