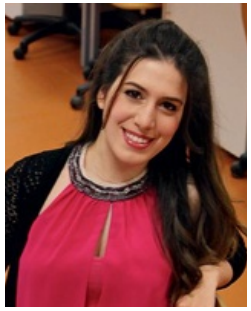
A Tutorial on RL

Benjamin Van Roy
Stanford University and DeepMind


Maria Dimakopolou
Netflix

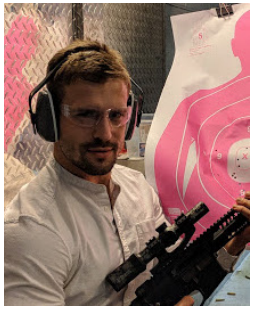

Shi Dong
Stanford


Vikranth Dwaracherla
Stanford


Morteza Ibrahimi
DeepMind


Abbas Kazerouni
Stanford


Xiuyuan Lu
Stanford


Ian Osband
DeepMind


Dan Russo
Columbia


Zheng Wen
DeepMind


Bill Zhu
Facebook

Objectives

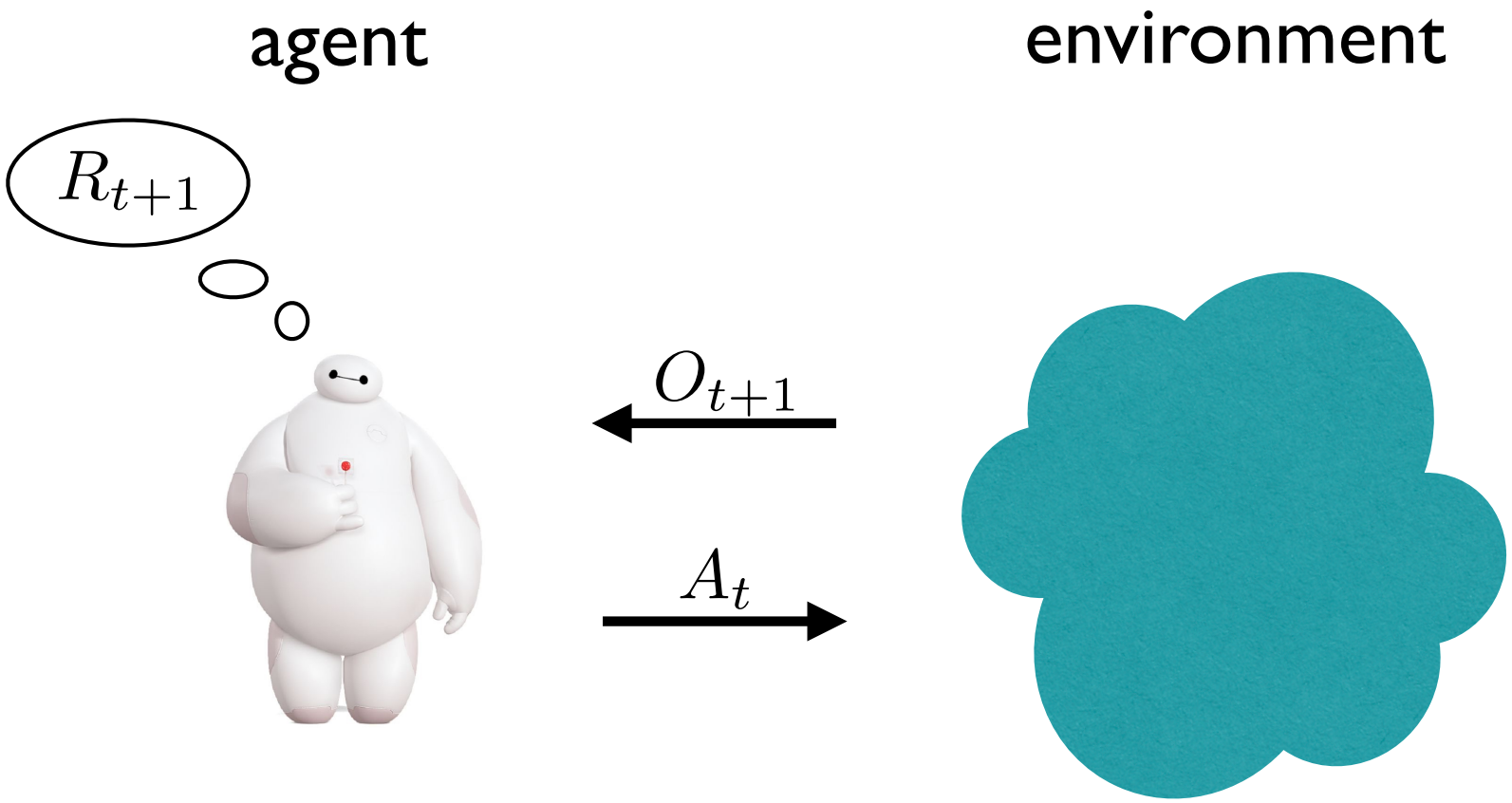
- Introduce you to reinforcement learning
  - Problem, formalism, and examples
  - Algorithms
  - Some current research directions

- Prerequisites
  - Probability / stochastic processes
  - Supervised learning
    - e.g., neural networks, SGD

What is reinforcement learning?

- A problem
- A community who work on the problem
- Methods produced by that community
  - Some of which are also useful for other problems

The Reinforcement Learning Problem



"other learning paradigms are about **minimization**, reinforcement learning is about **maximization**"

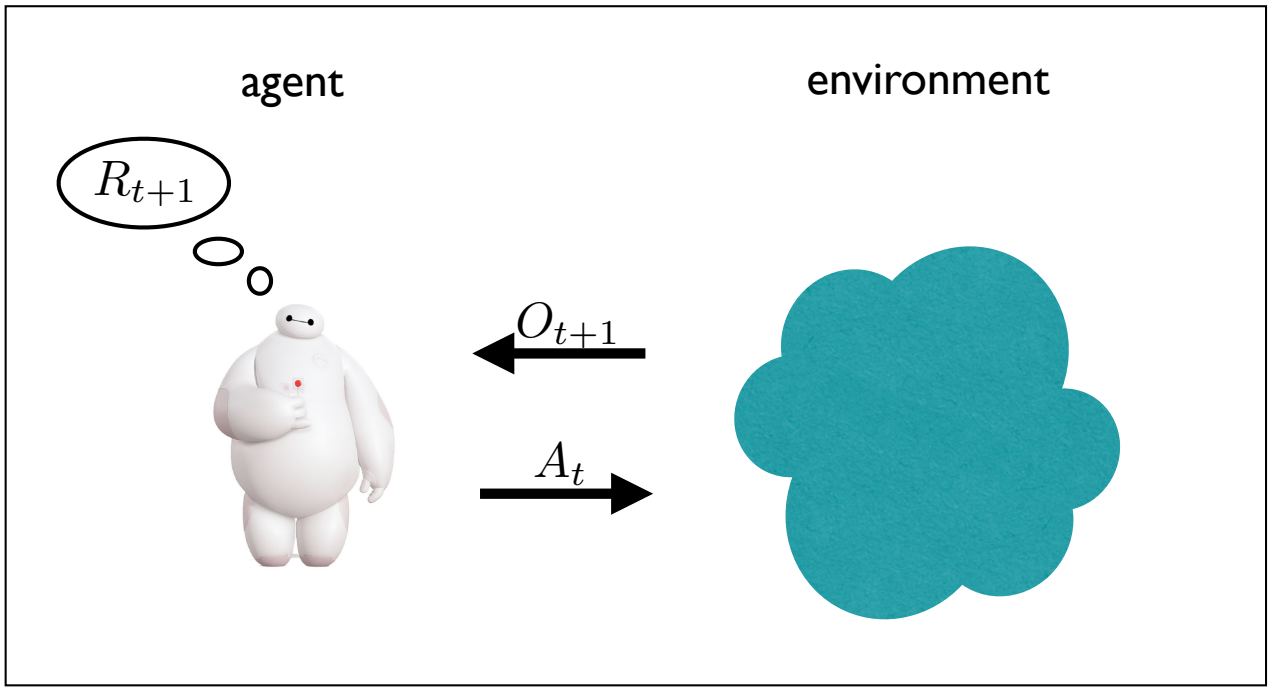
- Klopf

Artificial General Intelligence

- Develop a system that can achieve human-level competence across *all* intellectual tasks

- Can RL achieve AGI?

Episodic Version of the RL Problem



data from episodes 1 through ℓ

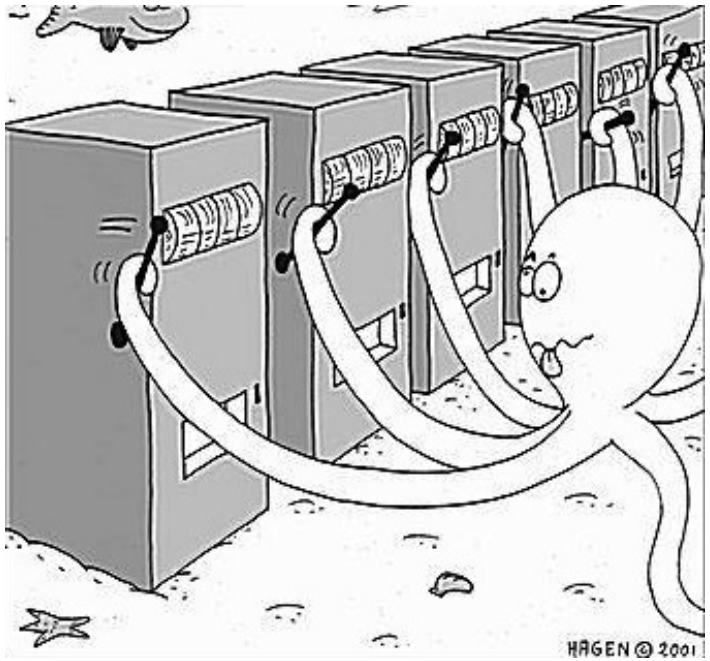
$$O_0^1, A_0^1, O_1^1, \ldots, A_{\tau_1-1}^1, O_{\tau_1}^1$$

$$\vdots$$

$$O_0^\ell, A_0^\ell, O_1^\ell, \ldots, A_{\tau_\ell-1}^\ell, O_{\tau_\ell}^\ell$$

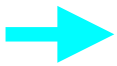
→ **RL algorithm** → policy for next episode π_ℓ

RL versus Bandits



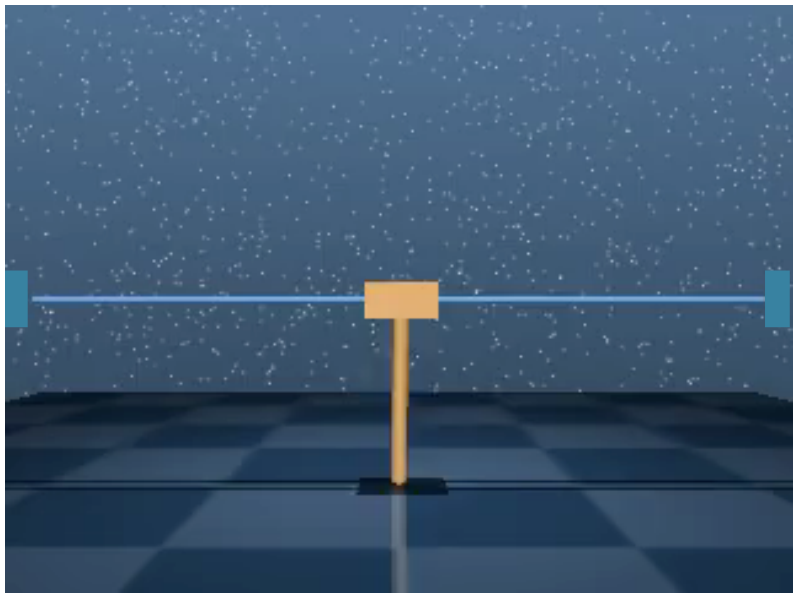
- Bandit problem = RL with episode of duration 1
- Bandit algorithms are RL algorithms applied to bandits
- Bandits much better understood
- RL formulation can be overkill for specific applications
- RL can be critical for delayed consequences
- RL is critical for AGI

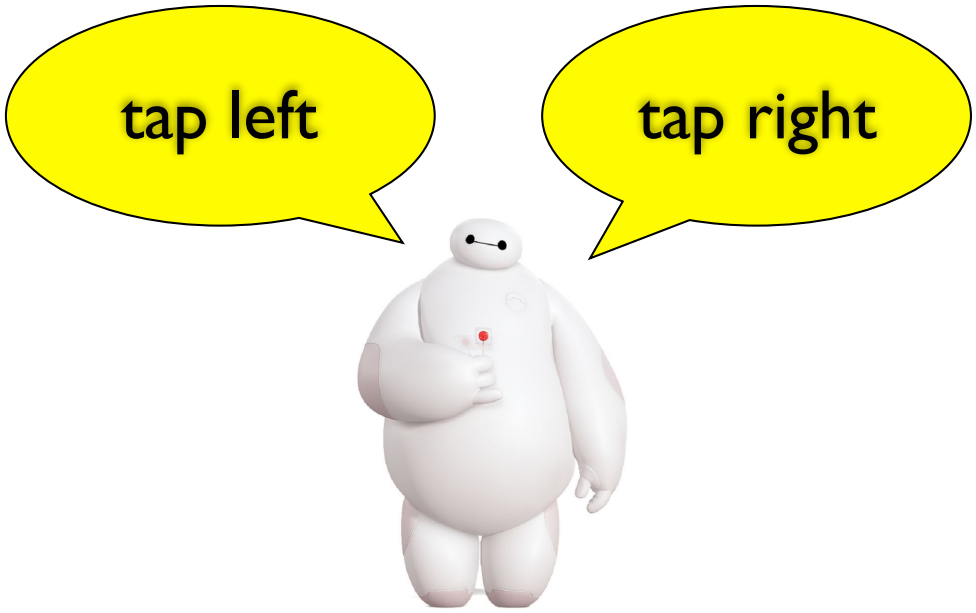
Outline



- Examples
- Mathematical formalism
- Common algorithms
- Challenges of data efficiency

Example: Cart-Pole Swingup

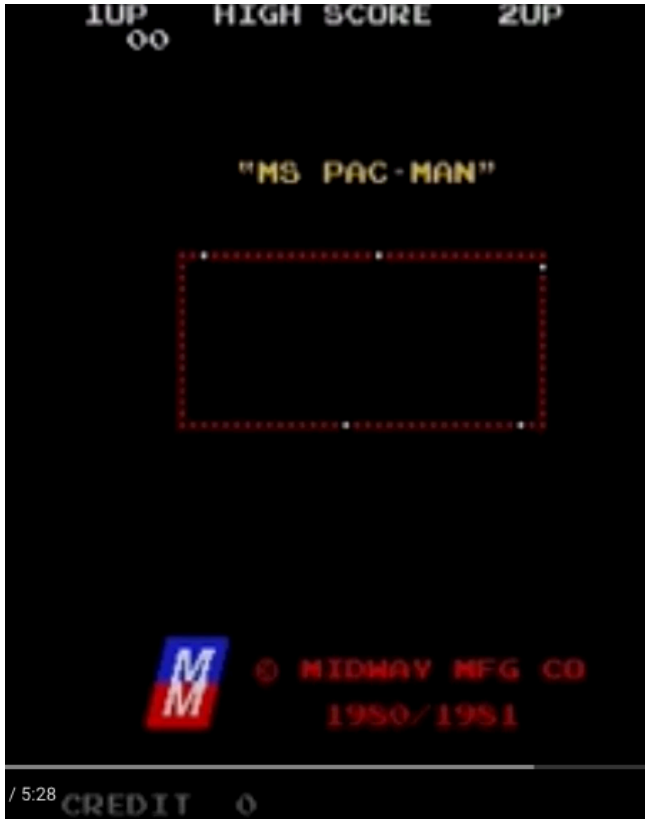


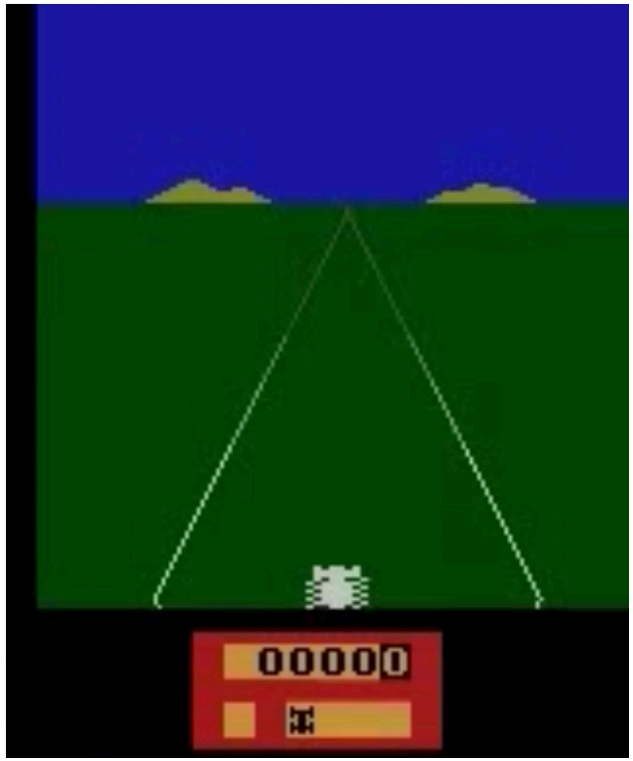


20 seconds per episode

Example: Robotics

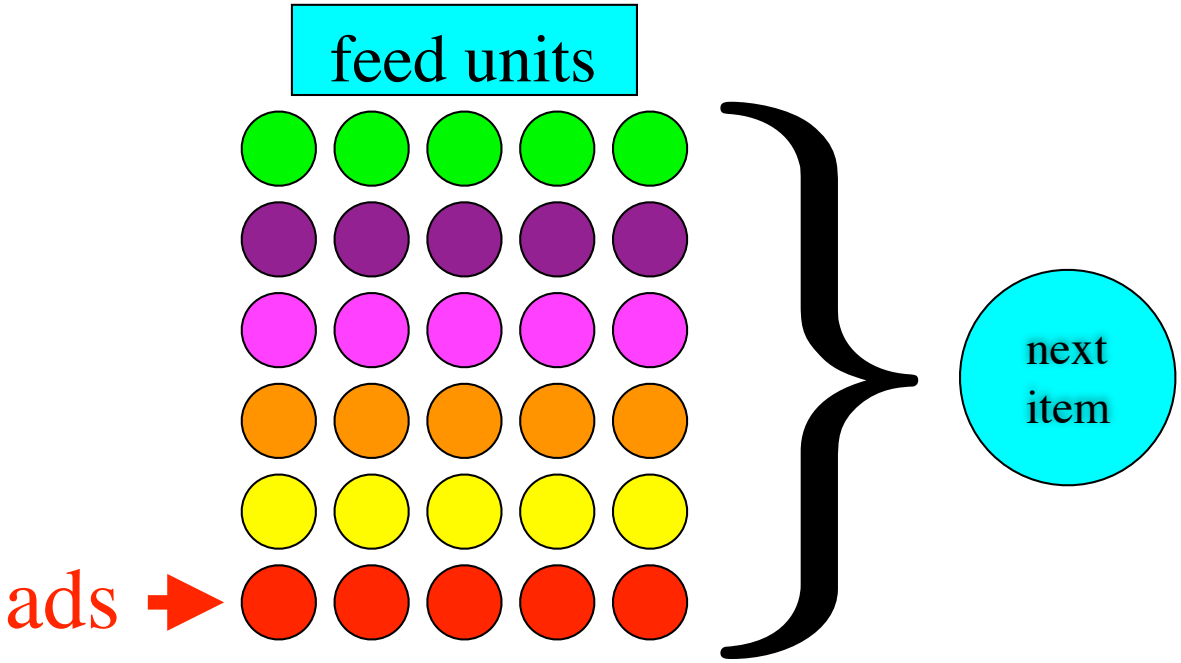
Example: Arcade Games

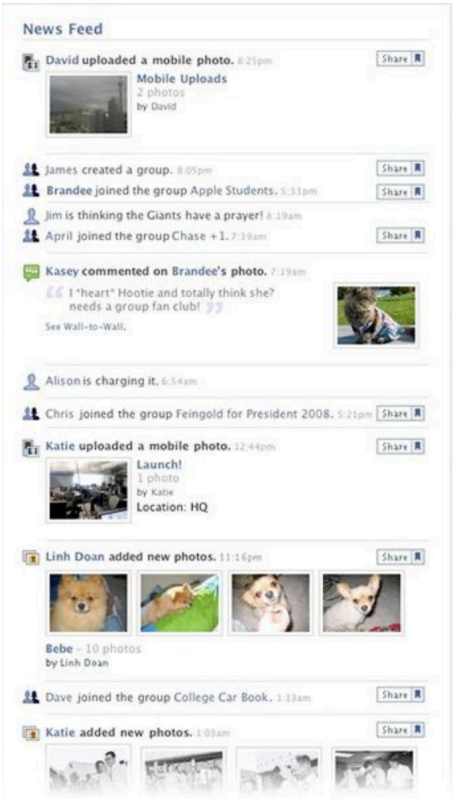




action: joystick actuation
observation: screen shot

Example: News Feed Engagement





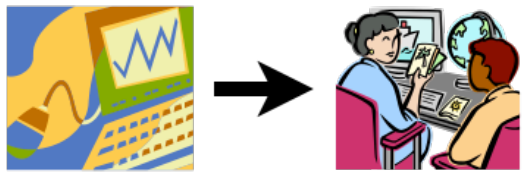
action: next item

observation: click, like, share, comment, scroll, terminate

Example: Revenue Management

seat inventory

Fare Product	Specification	Price
1	Economy SF-LA 8AM 12/11/10 LA-SF 1PM 15/11/10 Nonrefundable ...	$120
2	First SF-Ch 8AM 12/24/10 Ch-NY 4PM 12/24/10 Refundable ...	$2500
...	...	...

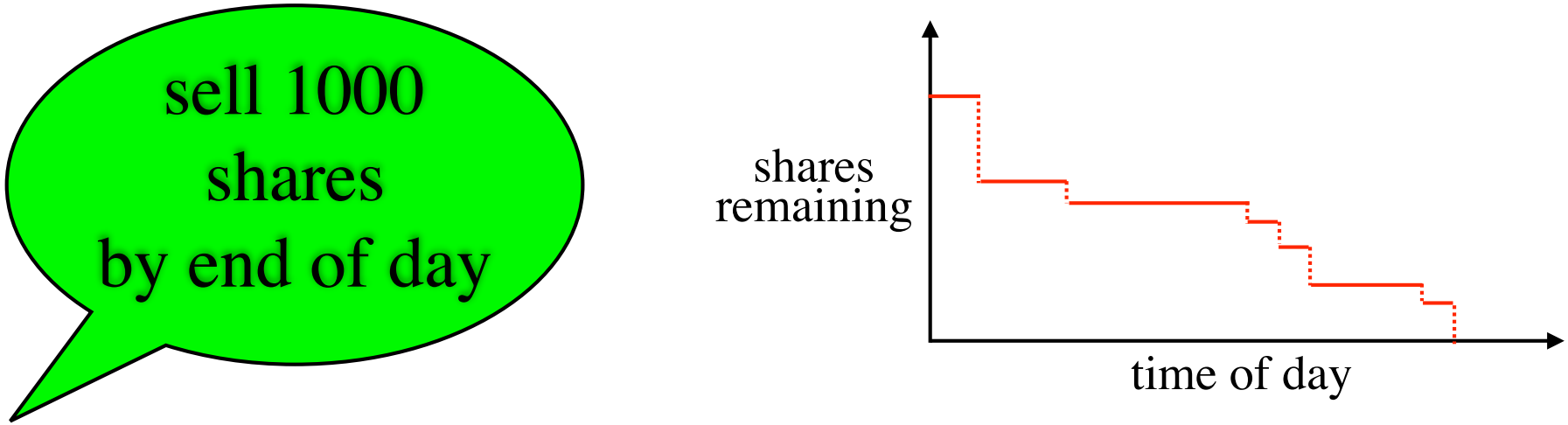


determine
"availability"

action: accept/reject

observation: requests, cancellations, forecasts, etc.

Example: Trade Execution



action: orders to exchange
observation: order book, other market data

Outline

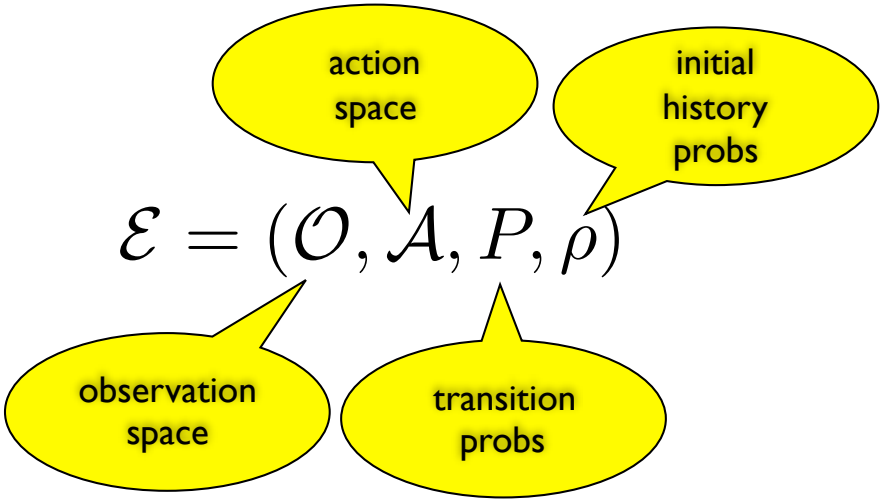
- Examples
- Mathematical formalism
- Common algorithms
- Challenges of data efficiency

The Role of Formalism

- Clarify the class of problems under consideration
- Provide notation for making rigorous statements
- Establish properties that apply across the class
- Design algorithms that apply across the class
- Guide modeling
  - Epistemic versus aleatoric uncertainty

Formalism: Environment

- Environment



- Assume finite
  - observation space
  - action space

- News feed model

$\mathcal{E} = (\mathcal{O}, \mathcal{A}, P, \rho)$

$\mathcal{A} \in \{\text{feed unit}\}$

$$\mathcal{O} \in \left\{ \begin{array}{c} (\text{inventory, user profile}) \\ (\text{like/swipe/quit}) \end{array} \right\}$$

$\rho \rightarrow (\text{inventory, user profile})$

$P \rightarrow (\text{like/swipe/quit})$

Formalism: Dynamics

- History

$$H_0 = (O_o)$$

$$H_t = (O_0, A_0, \ldots, O_t)$$

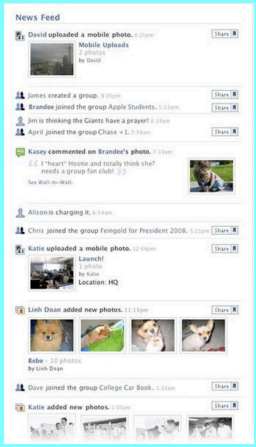
$$H_t \in \mathcal{H} = \mathcal{H}^+ \cup \mathcal{H}^-$$

terminal histories

$$H_0 = ((\text{inventory}, \text{user profile}))$$

$$H_t = (\ldots, \text{feed unit } \#22, (\text{like}, \text{swipe}))$$

$$\mathcal{H}^- = \{(\cdots, (\cdots, \text{quit}))\}$$



- Dynamics

$$\rho_h = \mathbb{P}(H_0 = h | \mathcal{E})$$

$$P_{ahh'} = \mathbb{P}(H_{t+1} = h' | \mathcal{E}, H_t = h, A_t = a)$$

Formalism: Policies

- Policy

$$\pi_h(a) = \mathbb{P}(A_t = a | \mathcal{E}, H_t = h)$$

feed unit $\leftarrow \pi_h$

- Transition probs

$$P_{\pi hh'} = \sum_{a \in \mathcal{A}} P_{ahh'}$$

like/swipe/quit $\leftarrow P_{\pi h}$.

- Finite episode duration τ

total # units displayed

Formalism: Rewards

- Period reward

$$R_{t+1} = r(H_{t+1})$$

$$r(H_{t+1}) = \begin{cases} 1 & \text{if } A_t \in \text{ads} \\ 0 & \text{otherwise} \end{cases}$$

- Episode reward

$$\sum_{t=0}^{\tau-1} R_{t+1}$$

total # ads displayed

How will an optimal policy behave?

Value Functions

- Value functions $V^\pi : \mathcal{H}^+ \to \Re$

$$V^\pi(h) = \mathbb{E}\left[\sum_{k=t}^{\tau-1} R_{k+1} \middle| \mathcal{E}, H_t = h, \pi\right]$$

$$V^*(h) = \sup_\pi V^\pi(h)$$

remaining # ads under π

remaining # ads under π^*

- Value per episode $\overline{V}^\pi \in \Re$

$$\overline{V}^\pi = \sum_{h \in \mathcal{H}^+} \rho_h V^\pi(h)$$

$$\overline{V}^* = \sum_{h \in \mathcal{H}^+} \rho_h V^*(h)$$

total # ads under π

total # ads under π^*

Action Value Functions

- Value conditioned on immediate action $Q^\pi : \mathcal{H}^+ \times \mathcal{A} \to \Re$

$$Q^\pi(h,a) = \mathbb{E}\left[\sum_{k=t}^{\tau-1} R_{k+1} \middle| \mathcal{E}, H_t = h, A_t = a, \pi\right]$$

$$Q^\pi(h,a) = \sum_{h' \in \mathcal{H}} P_{ahh'} r(h') + \sum_{h' \in \mathcal{H}^+} P_{ahh'} V^\pi(h')$$

- Optimal action value function

$$Q^*(h,a) = \sup_\pi Q^\pi(h,a)$$

From Value to Policy

- Greedy policy with respect to Q

$$\pi_h \sim \text{unif}\left(\arg\max_{a \in \mathcal{A}} Q(h, a)\right)$$

- Optimal policy is greedy with respect to Q^*

$$\pi_h^* \sim \text{unif}\left(\arg\max_{a \in \mathcal{A}} Q^*(h, a)\right)$$

- Actually, any distribution over the argmax will work…

Bellman's Equation

- Dynamic programming operators

$$(TV)(h) = \max_{a \in \mathcal{A}} \left(\sum_{h' \in \mathcal{H}} P_{ahh'} r(h') + \sum_{h' \in \mathcal{H}^+} P_{ahh'} V(h') \right)$$

$$(FQ)(h,a) = \sum_{h' \in \mathcal{H}} P_{ahh'} r(h') + \sum_{h' \in \mathcal{H}^+} P_{ahh'} \max_{a' \in \mathcal{A}} Q(h', a')$$

- Bellman's equation

$$TV^* = V^*$$

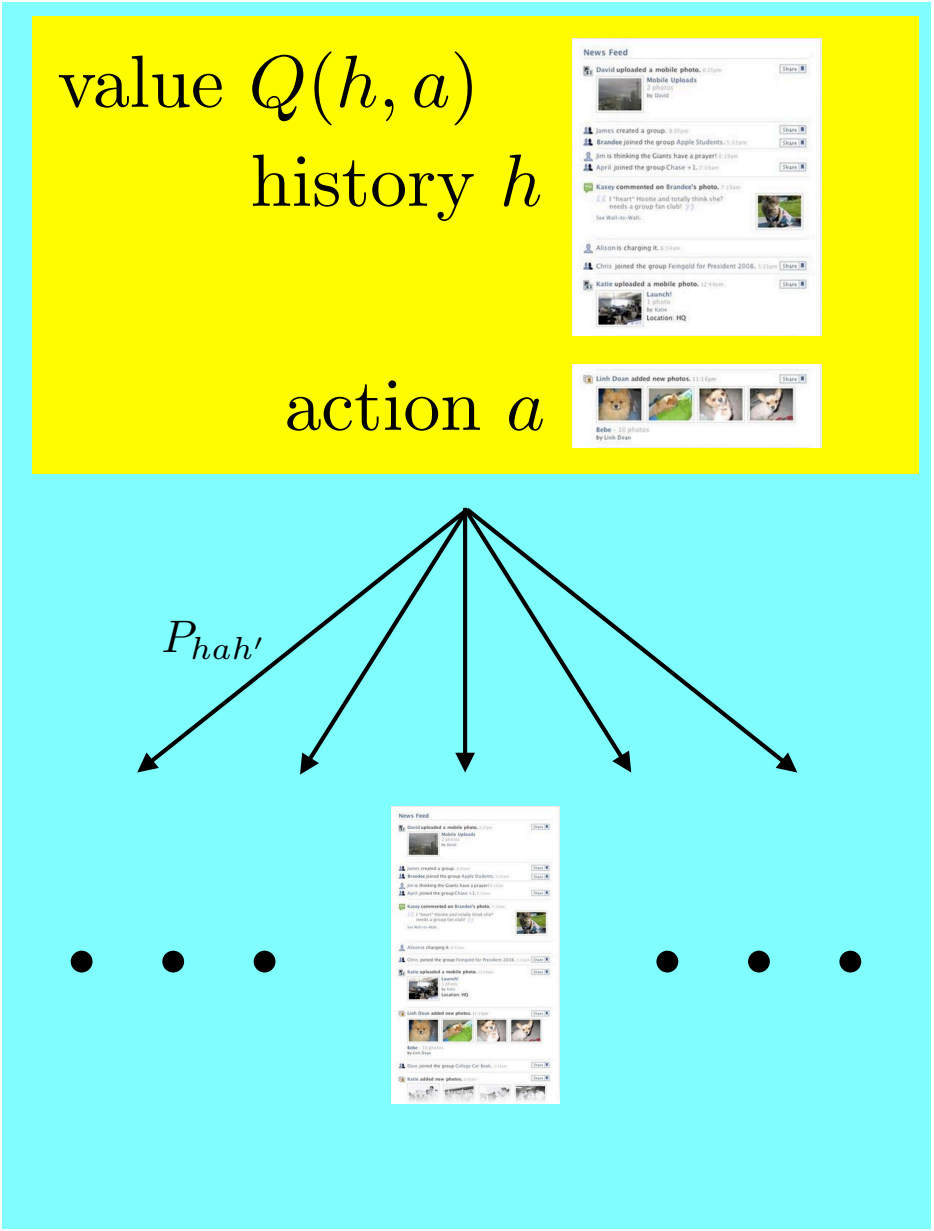
$$FQ^* = Q^*$$

Optimization via Value Iteration

- Computing V^*
  - Repeat $V \leftarrow TV$ forever
- Computing Q^*
  - Repeat $Q \leftarrow FQ$ forever

- VI converges, but
  - Computationally intractable
  - Requires knowledge of P

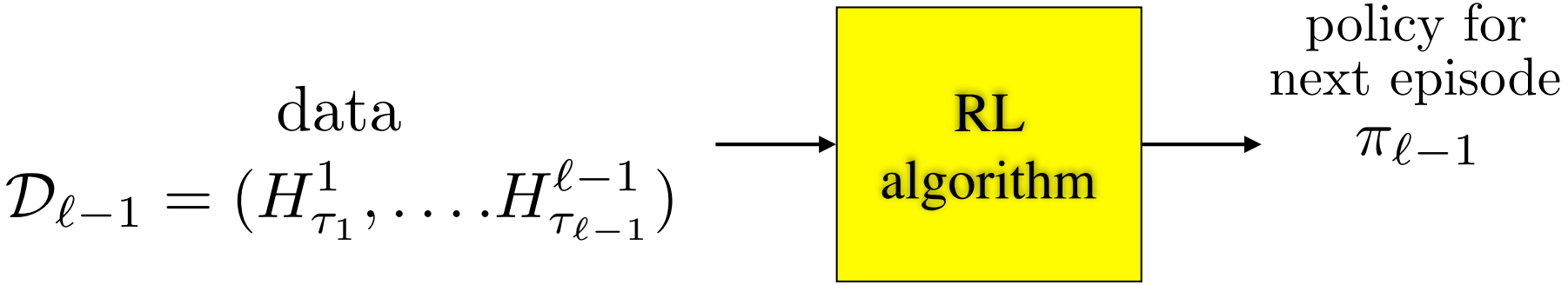
- RL algorithms address these issues



Formalism: Episodic RL

- Environment $\mathcal{E}$ is a random variable
  - In particular, ρ and P are random variables
  - *Epistemic* versus *aleatoric* uncertainty

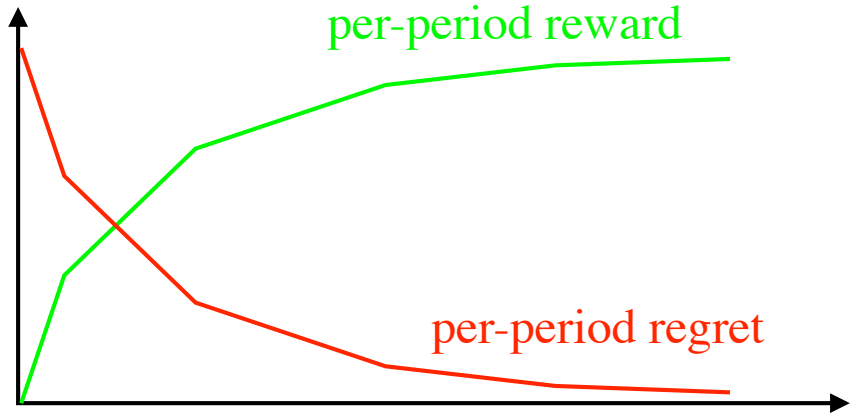
- Episodes $\ell = 1, 2, \ldots, L$



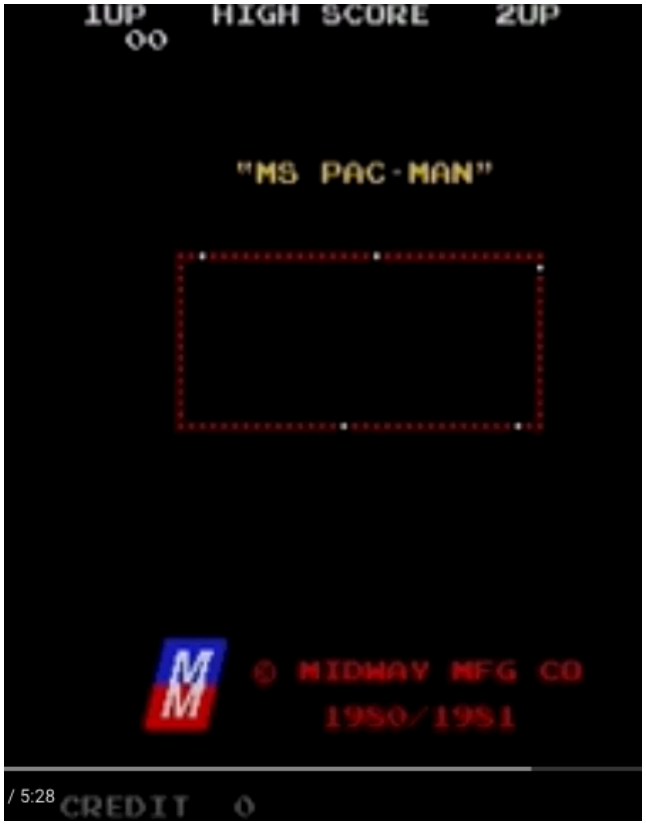
- RL algorithms address
  - Computational issues
  - Statistical issues, like exploration versus exploitation

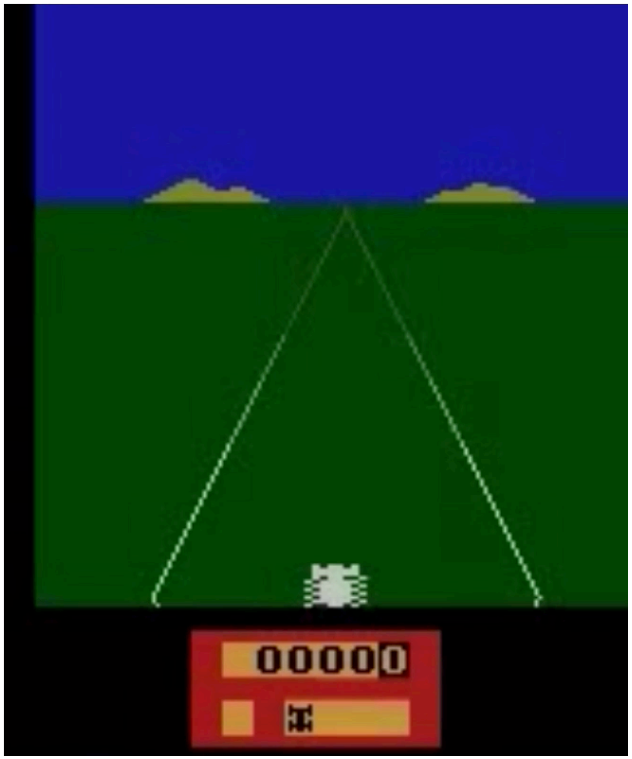
Formalism: Objective

- Expected reward $\quad \mathrm{Reward}(\mathrm{alg}) = \mathbb{E}\left[\sum_{\ell=1}^{L} \overline{V}^{\pi_{\ell-1}}\right]$
- "Bayesian decision problem"
  - RL algorithms can be viewed as heuristics for this problem
- Regret $\quad \mathrm{Regret}(\mathrm{alg}) = \mathbb{E}\left[\sum_{\ell=1}^{L} \left(\overline{V}^{*} - \overline{V}^{\pi_{\ell-1}}\right)\right]$



Atari Arcade Environment





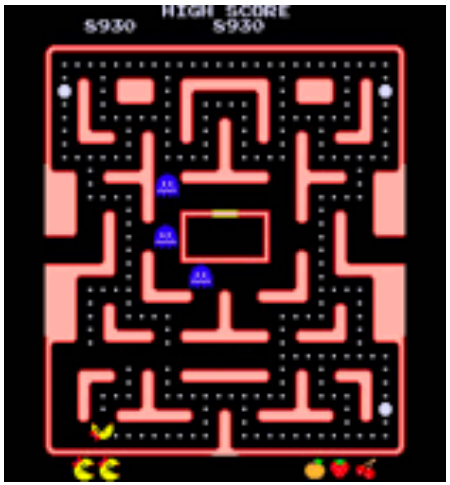
Atari 2600 supports 49 different games

develop an RL algorithm that learns to play

all games of this flavor

Arcade Games: Formulation

- Each game is an environment $\mathcal{E} = (\mathcal{O}, \mathcal{A}, P, \rho)$
  - Time increment: 1/60 of a second

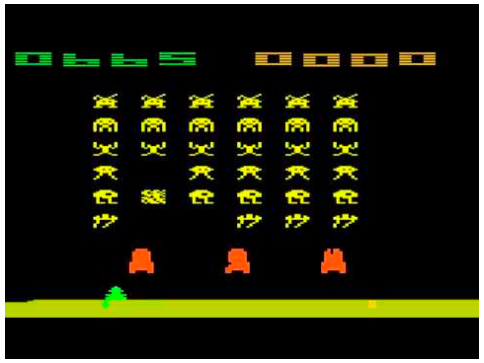


- Observations $O_t \in \mathcal{O}$
  - Video frame
- Actions $A_t \in \mathcal{A}$
  - 18 joystick modes
- Probabilities ρ, P
  - As produced by game

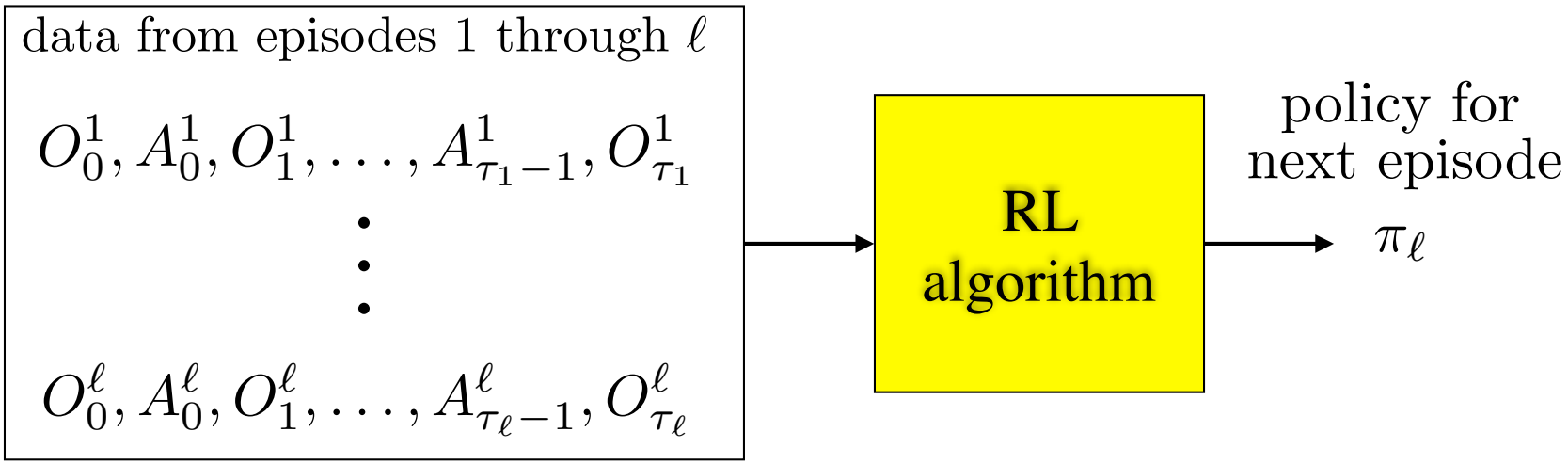
- Reward $R_{t+1} = r(H_{t+1})$
  - -1 if score decreasing
  - +1 if score increasing
  - 0 otherwise

Arcade Games: RL

- Epistemic uncertainty
  - ρ and P are random variables



- RL Algorithm applied over episodes of the same game



Outline

- Examples
- Mathematical formalism
- Common algorithms
- Challenges of data efficiency

Knowledge Representation

- Model learning: learn P
  - Left with a computationally intractable planning problem
  - Exponential explosion of roll-forward errors
- Policy learning: learn π^*
- Value function learning: learn Q^*

Generic RL Loop

Algorithm 1 `live`

Input: `agent`
`environment`

```
for ℓ in (1, 2, ...) do
    agent.learn_from_buffer()
    environment.reset()
    while environment.history is not terminal do
        action ← agent.act(environment.history)
        environment.step(action)
    end while
    agent.update_buffer(environment.history)
end for
```

object	`environment`
attributes	`history`
methods	`reset`
	`step`

object	`agent`
methods	`learn_from_buffer`
	`act`
	`update_buffer`

Agent Methods

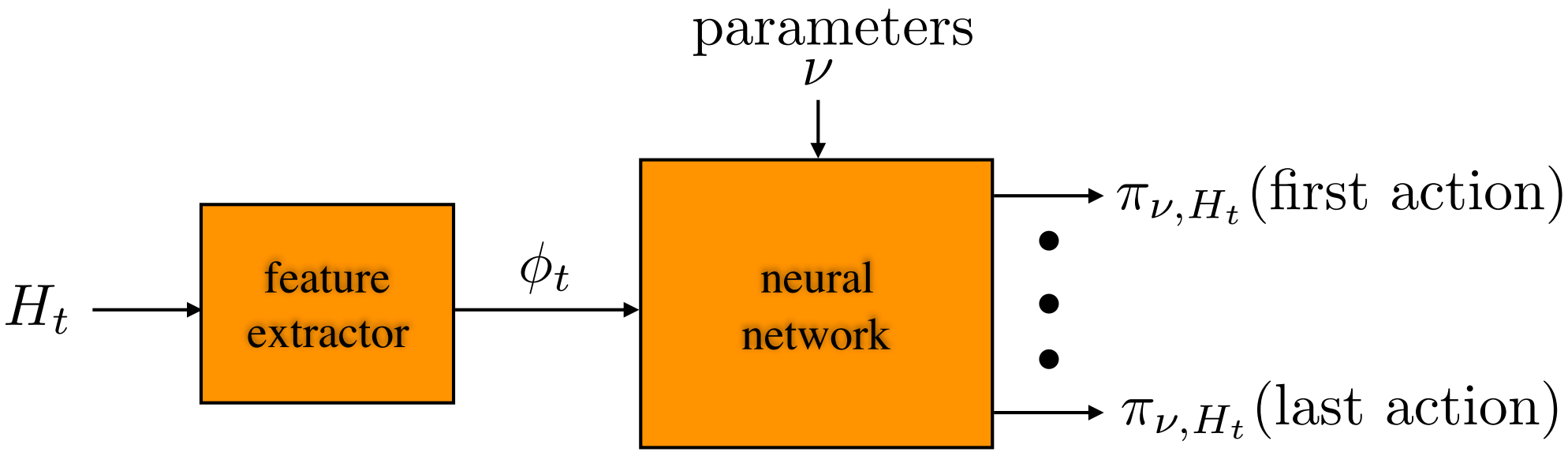
- act
  - policy learning: use stored policy
  - value function learning: greedy
  - possibly dither, e.g., ϵ-greedy

- update_buffer
  - append history
  - possibly subsample or drop very old samples
  - possibly compress to what is needed for learning

- learn_from_buffer
  - update parameters of policy and/or value function
  - sophisticated algorithms
  - we will focus on this part

Policy Representation

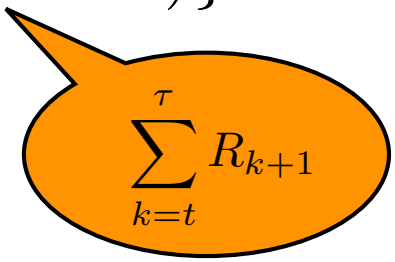
- Policy parameterized by vector ν
- Example
  - neural network with multinomial logistic output



The *Reinforce* Algorithm

$$\texttt{policy} = \tilde{\pi}_{\texttt{parameters}=\cdot,\texttt{features}=\cdot}(\texttt{action} = \cdot)$$

$$\texttt{buffer} = \{(\texttt{features}, \texttt{action}, \texttt{subsequent reward})\}$$



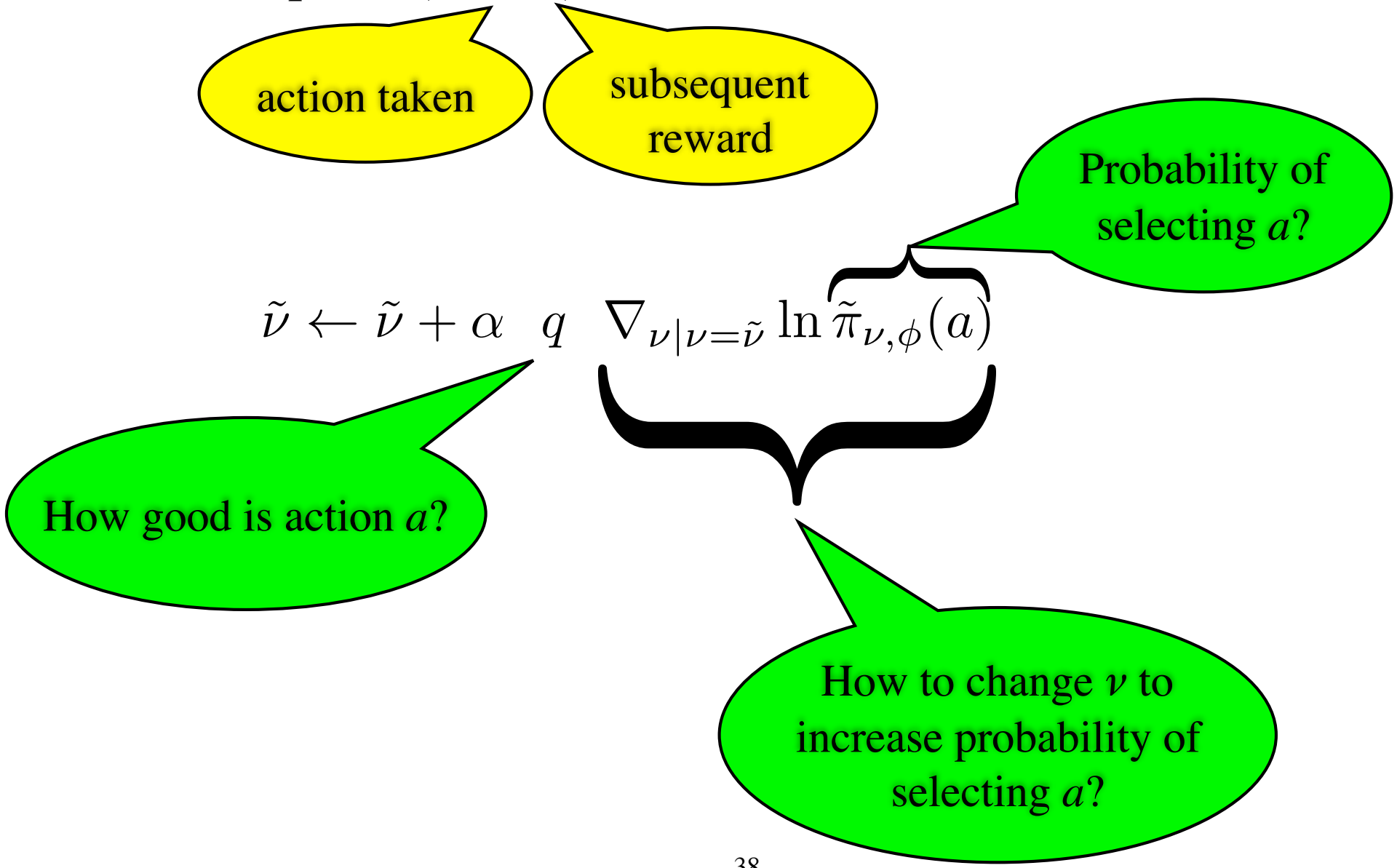
Algorithm 2 `learn_reinforce`

Input: `agent`

$\tilde{\pi} \leftarrow$ `agent.policy`
$\tilde{\nu} \leftarrow$ `agent.parameters`
`buffer` $\leftarrow$ `agent.buffer`
$N \leftarrow$ `agent.num_iters`
$\alpha \leftarrow$ `agent.learning_rate`
for n in $(1, 2, \ldots, N)$ **do**
    $\tilde{\mathcal{D}} \leftarrow$ `buffer.sample_minibatch()`
    $\tilde{\nu} \leftarrow \tilde{\nu} + \alpha \sum_{(\phi,a,q)\in\tilde{\mathcal{D}}} q \nabla_{\nu|\nu=\tilde{\nu}} \ln \tilde{\pi}_{\nu,\phi}(a)$
end for
`agent.parameters` $= \tilde{\nu}$

Intuition about the *Reinforce* Parameter Update

- Consider case with minibatch of size 1
  - Data point (ϕ, a, q)



Reinforce is a *Policy Gradient* Algorithm

- Reinforce is a *policy gradient* method
  - The mean field follows the gradient of $\overline{V}^{\pi_\nu}$
    - Under certain technical conditions…
  - Agent performance improves over episodes
  - Popular due to reliable improvement

- An *on-policy* algorithm
  - Estimates gradient at buffered policy
  - To be effectiveness, only retain recent data
  - Lack of *counterfactual analysis*
    - As opposed to *off-policy* algorithms

Policy Gradient Methods: Obstacles

- SGD can be slow and converge to local minima
  - Use adaptive step size rules as with supervised learning

- Statistically inefficient
  - On-policy learning
  - Naive exploration
  - High variance of gradient estimates
    - Variance reduction techniques
    - Baselining

Variance Reduction Through Baselining

- Thrashing if q is large
  - Reducing step size makes learning too slow when q is small

$$\tilde{\nu} \leftarrow \tilde{\nu} + \alpha \;\; q \;\; \nabla_{\nu|\nu=\tilde{\nu}} \ln \tilde{\pi}_{\nu,\phi}(a)$$

- Baselined update with any $B(\phi)$ retains policy gradient

$$\tilde{\nu} \leftarrow \tilde{\nu} + \alpha \;\; (q - B(\phi)) \;\; \nabla_{\nu|\nu=\tilde{\nu}} \ln \tilde{\pi}_{\nu,\phi}(a)$$

- A good baseline is given by $V^{\tilde{\pi}_{\tilde{\nu}}}$
  - Use approximation $\tilde{V}_\theta(\phi) \approx V^{\tilde{\pi}_{\tilde{\nu}}}$

Learning a Baseline

Algorithm 3 `learn_reinforce_baseline`

Input: `agent`

$\tilde{\pi} \leftarrow$ `agent.policy`
$\tilde{V} \leftarrow$ `agent.value_function`
$(\tilde{\nu}, \tilde{\theta}) \leftarrow$ `agent.parameters`
`buffer` $\leftarrow$ `agent.buffer`
$N \leftarrow$ `agent.num_iters`
$\alpha \leftarrow$ `agent.learning_rate`
for n in $(1, 2, \ldots, N)$ **do**
  $\tilde{\mathcal{D}} \leftarrow$ `buffer.sample_minibatch()`
  $\tilde{\nu} \leftarrow \tilde{\nu} + \alpha \sum_{(\phi,a,q)\in\tilde{\mathcal{D}}} (q - \tilde{V}_{\tilde{\theta}}(\phi)) \nabla_{\nu|\nu=\tilde{\nu}} \ln \tilde{\pi}_{\nu,\phi}(a)$
  $\tilde{\theta} \leftarrow \tilde{\theta} + \alpha \sum_{(\phi,a,q)\in\tilde{\mathcal{D}}} \left(\nabla_{\theta|\theta=\tilde{\theta}} \tilde{V}_{\theta}(\phi)\right) \left(q - \tilde{V}_{\tilde{\theta}}(\phi)\right)$
end for
`agent.parameters` $= (\tilde{\nu}, \tilde{\theta})$

adapt baseline to approximate $V^{\tilde{\pi}_{\tilde{\nu}}}$

- *Actor*: parameterized policy
- *Critic*: parameterized value function
- Broader class of *actor-critic methods*…

Actor-Critic Algorithm

- Actor can learn directly from the action-value critic rather than observed rewards

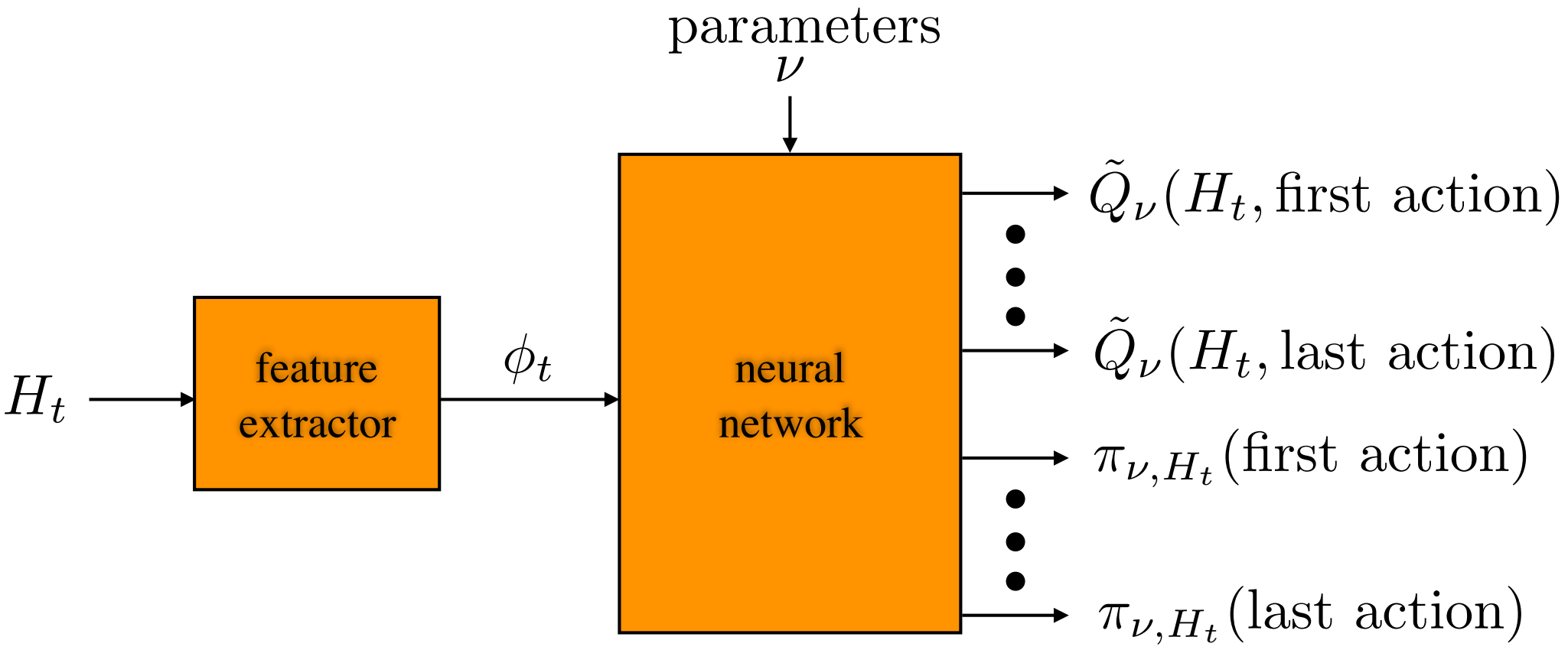
Algorithm 4 `learn_actor_critic`

Input: `agent`

$\tilde{\pi} \leftarrow$ `agent.policy`
$\tilde{Q} \leftarrow$ `agent.value_function`
$(\tilde{\nu}, \tilde{\theta}) \leftarrow$ `agent.parameters`
`buffer` $\leftarrow$ `agent.buffer`
$N \leftarrow$ `agent.num_iters`
$\alpha \leftarrow$ `agent.learning_rate`
for n in $(1, 2, \ldots, N)$ **do**
    $\tilde{\mathcal{D}} \leftarrow$ `buffer.sample_minibatch()`
    $\tilde{\nu} \leftarrow \tilde{\nu} + \alpha \sum_{(\phi,a,q)\in\tilde{\mathcal{D}}} \left(\tilde{Q}_{\tilde{\theta}}(\phi, a) - \sum_{a'\in\mathcal{A}} \tilde{\pi}_{\tilde{\nu},\phi}(a') \tilde{Q}_{\tilde{\theta}}(a') \right) \nabla_{\nu|\nu=\tilde{\nu}} \ln \tilde{\pi}_{\nu,\phi}(a)$
    $\tilde{\theta} \leftarrow \tilde{\theta} + \alpha \nabla_{\theta|\theta=\tilde{\theta}} \sum_{(\phi,a,q)\in\tilde{\mathcal{D}}} \tilde{Q}_{\theta}(\phi, a) \left(q - \tilde{Q}_{\tilde{\theta}}(\phi, a) \right)$
end for
`agent.parameters` $= (\tilde{\nu}, \tilde{\theta})$

Combined Actor-Critic Architecture

- Single neural network
  - Shared features
    - Policy-gradient theorem
  - Shared hidden layers

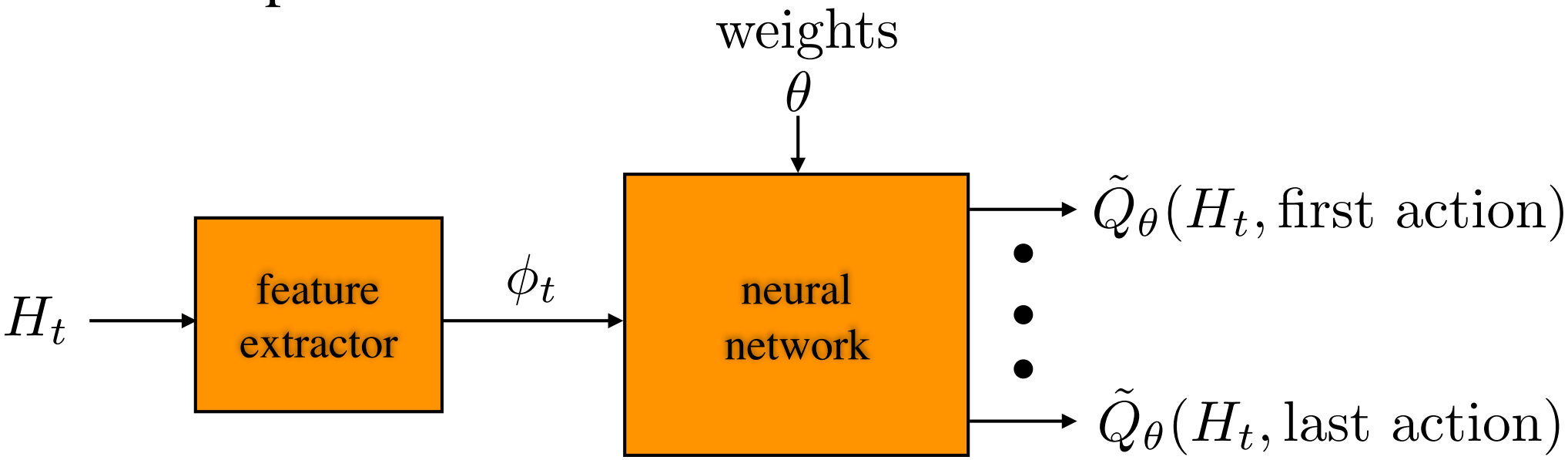


Value Function Learning

- Represent knowledge as parameterized value function
  - Possibly the optimal value function
  - Enables off-policy learning

$$\tilde{Q}_\theta(h, a) \approx Q^*(h, a)$$

- Example



Incremental TD

Algorithm 5 `learn_incremental_td`

Input: `agent`

$\tilde{Q} \leftarrow$ `agent.value_function`
$\tilde{\theta} \leftarrow$ `agent.parameters`
`buffer` $\leftarrow$ `agent.buffer`
$N \leftarrow$ `agent.num_iters`
$\alpha \leftarrow$ `agent.learning_rate`
for n in $(1, 2, \ldots, N)$ **do**
    $\tilde{\mathcal{D}} \leftarrow$ `buffer.sample_minibatch()`
    $\delta \leftarrow$ `buffer.minibatch_size` / `buffer.size`
    $\tilde{\theta} \leftarrow \tilde{\theta} - \alpha \nabla_{\theta|\theta=\tilde{\theta}} \left(\mathcal{L}(\theta, \tilde{\theta}, \tilde{\mathcal{D}}) + \delta \mathcal{R}(\theta) \right)$ ▷ TD step
end for

TD loss $$\mathcal{L}(\theta, \theta^-, \mathcal{D}) = \frac{1}{\sigma^2_{\text{outcome}}} \sum_{(\phi,a,r,\phi') \in \mathcal{D}} \left(r + \max_{a' \in \mathcal{A}} \tilde{Q}_{\theta^-}(\phi', a') - \tilde{Q}_\theta(\phi, a) \right)$$

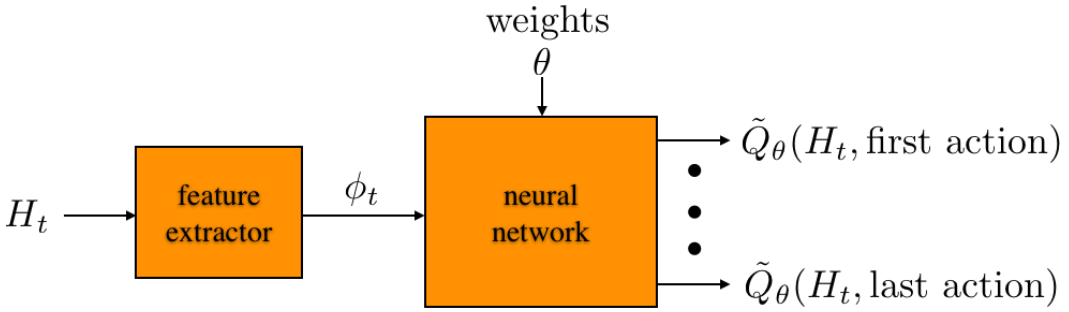
regularizor $$\mathcal{R}(\theta) = \frac{1}{\sigma^2_{\text{prior}}} \|\theta\|_2^2$$

Off-Policy Learning

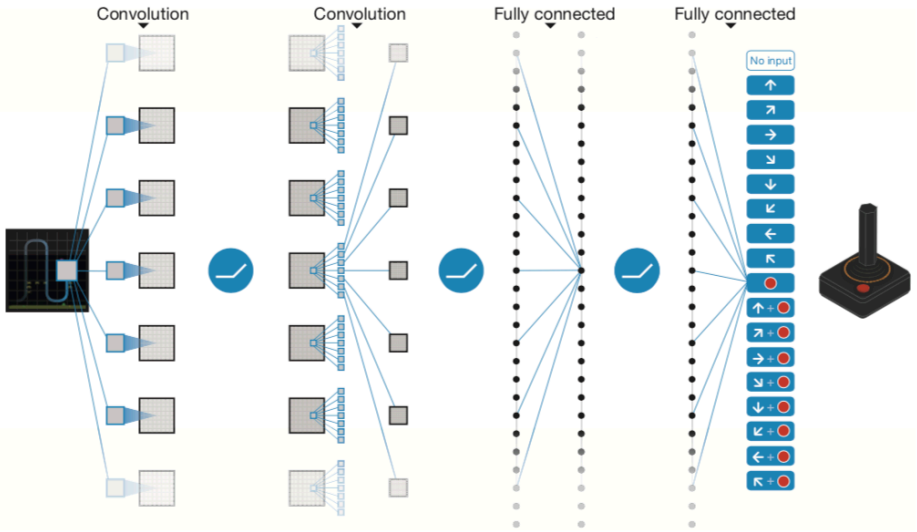
- TD error aligns prediction with max at next step
  - Not total reward under current policy

$$\text{TD loss} \qquad \mathcal{L}(\theta, \theta^-, \mathcal{D}) = \frac{1}{\sigma^2_{\text{outcome}}} \sum_{(\phi,a,r,\phi') \in \mathcal{D}} \left(r + \max_{a' \in \mathcal{A}} \tilde{Q}_{\theta^-}(\phi', a') - \tilde{Q}_\theta(\phi, a) \right)$$

Representation for Arcade RL Algorithm

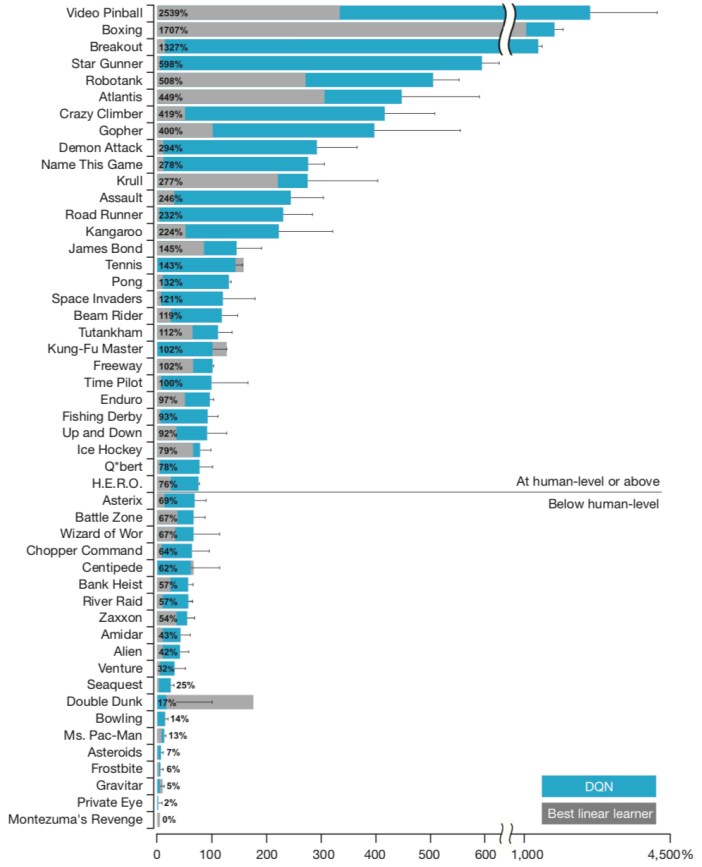


- Features
  - Last four frames, with some preprocessing
- Neural network architecture
  - Convolutional, RELUs, one linear output node per action



Performance in Atari Arcade Environment

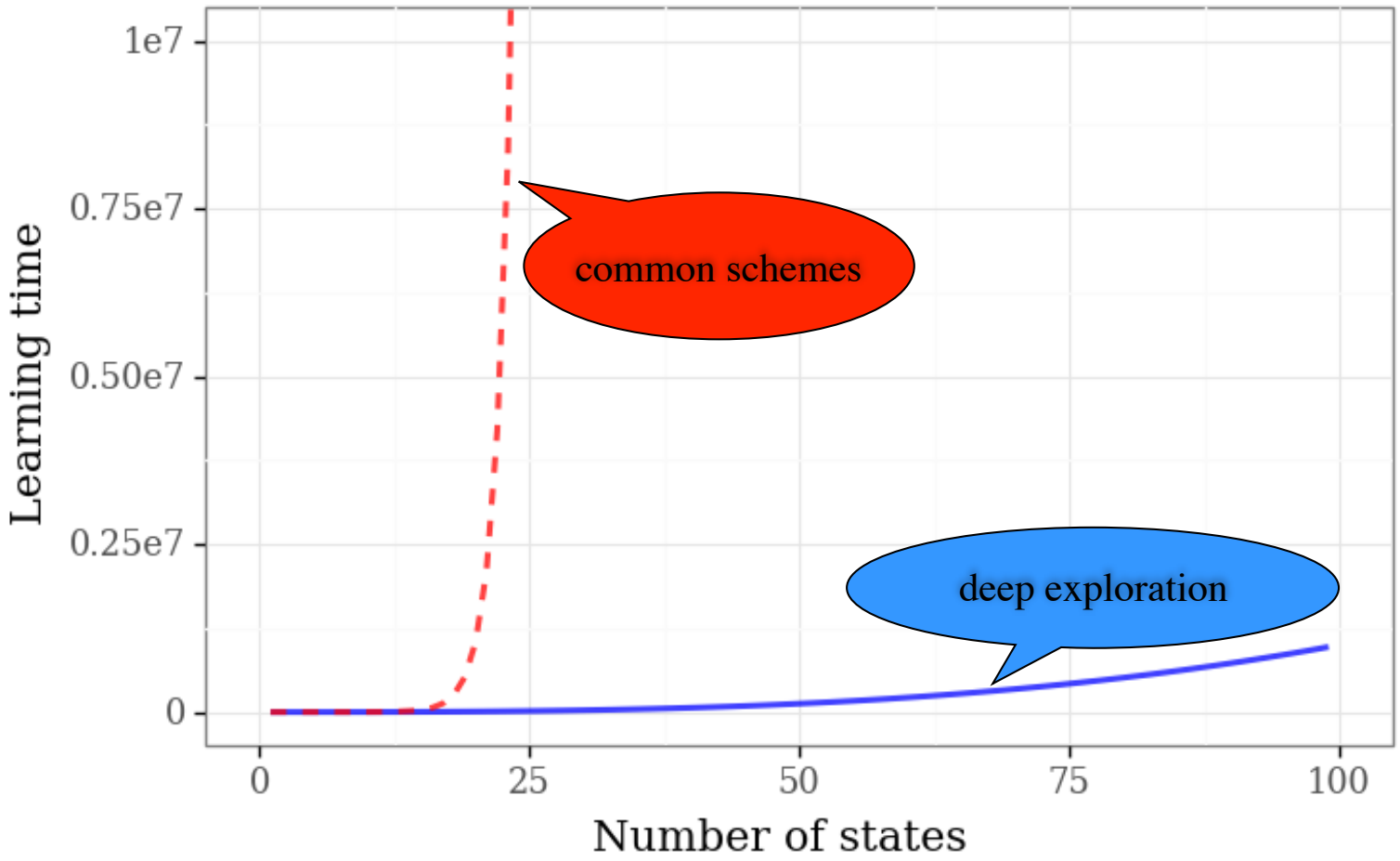
- DQN (incremental TD)
  - outperforms humans in most games
- Policy gradients approaches
  - perform even better
  - require much more data



Outline

- Examples
- Mathematical formalism
- Common algorithms
- Challenges of data efficiency

Big Data vs Smart Data

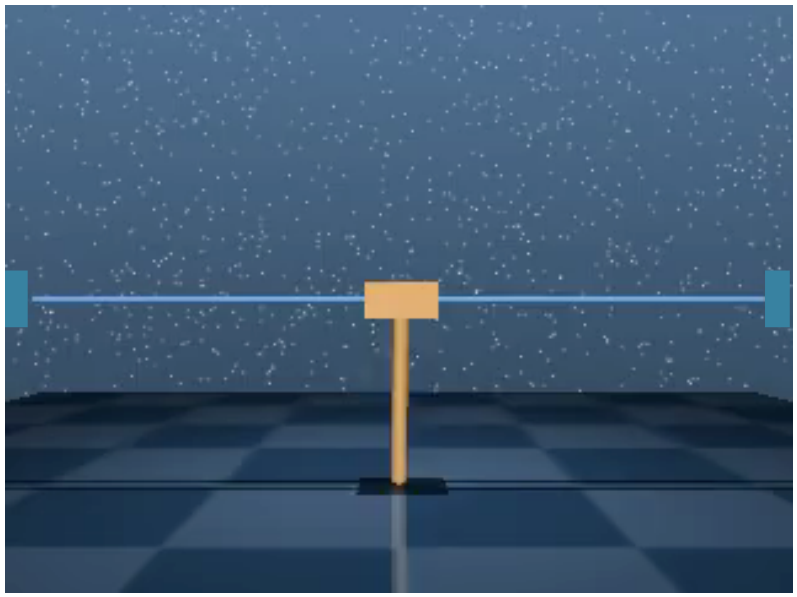


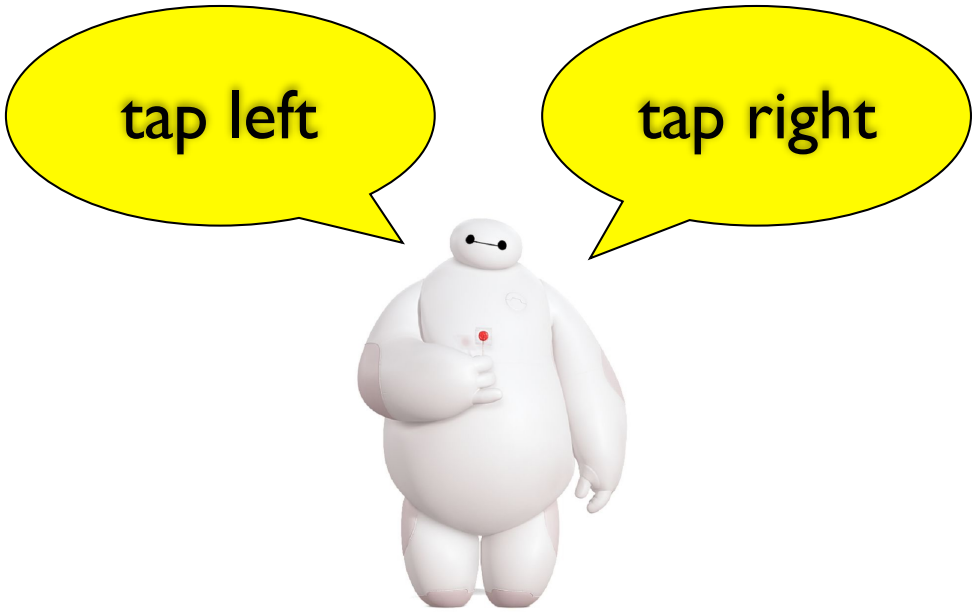
common exploration schemes can require exponentially large data sets even for simple reinforcement learning tasks

Representing and Resolving Uncertainty

- Many common RL methods learn point estimates
- Efficient exploration requires uncertainty representation
- How to learn neural networks that do this

Cart-Pole Problem

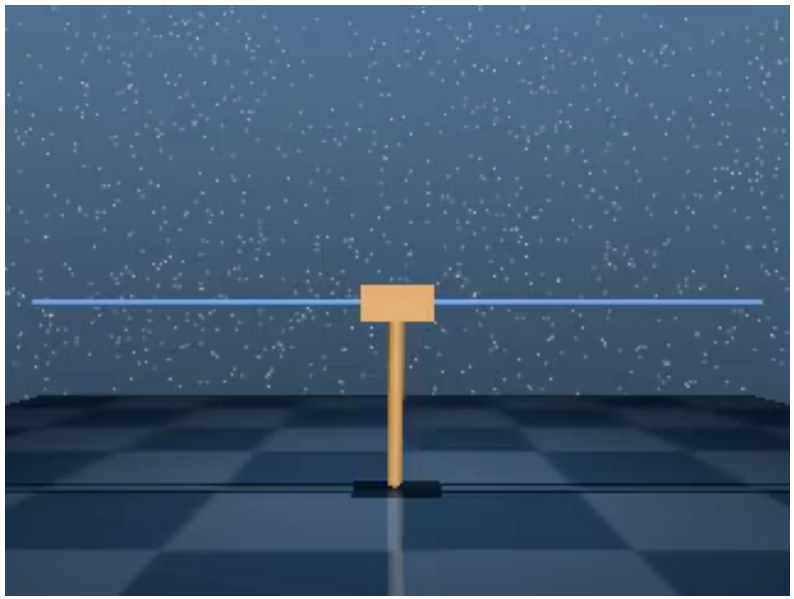




20 seconds per episode

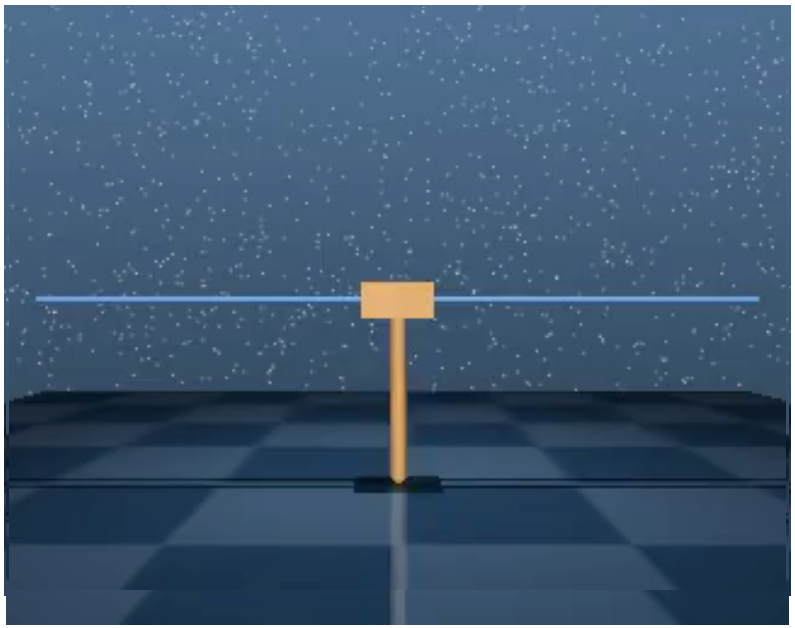
Cart-Pole Swing-Up

ε-greedy



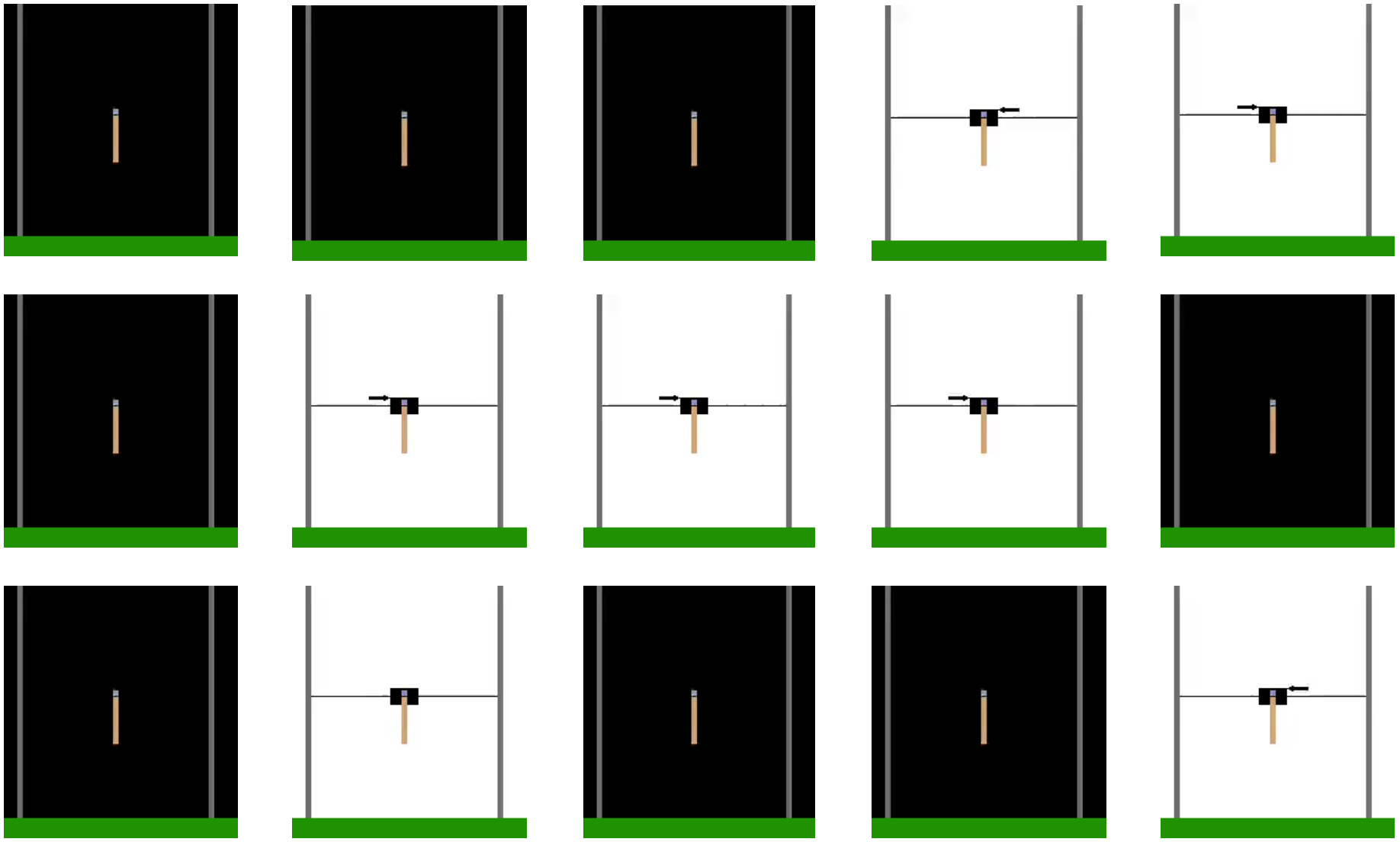
after 1000 episodes

deep exploration



after 1000 episodes

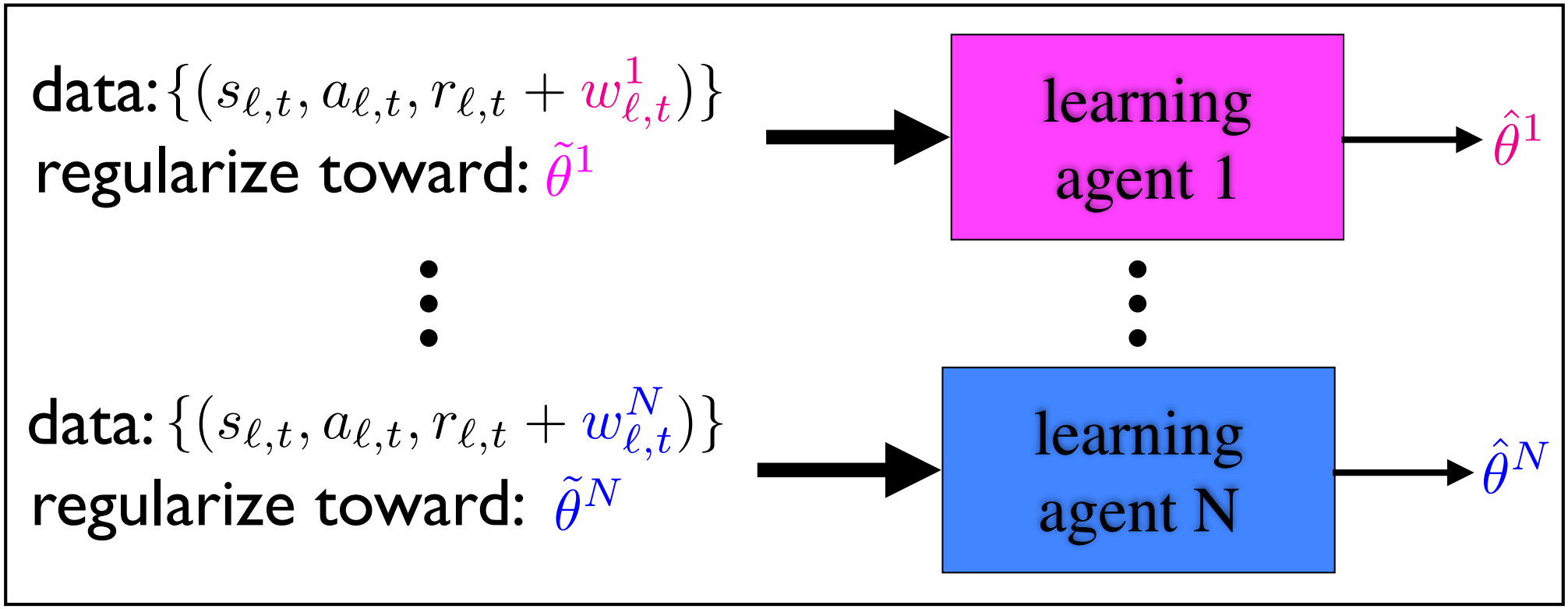
Information Sharing and Coordination



100 agents total

Ensemble Sampling

- Learn ensemble of N value functions, streaming data



- Before each episode, sample one uniformly, then argmax
- Other incremental approaches…

Hierarchical Representations

- Specific model classes, like linear, are too restrictive
- Can we do better than "flat" neural networks?
- CNNs induce useful inductive biases for vision
- What inductive biases are right for RL?
- HRL: effective policies decompose hierarchically

Leveraging Rich Observations

- Common methods often focus on predicting and maximizing future reward
- Richer observations can dramatically accelerate learning
- Don’t learn to optimize GDP based only on GDP
- Robots should learn from outcomes, not only rewards


Hello again,
and thank you!