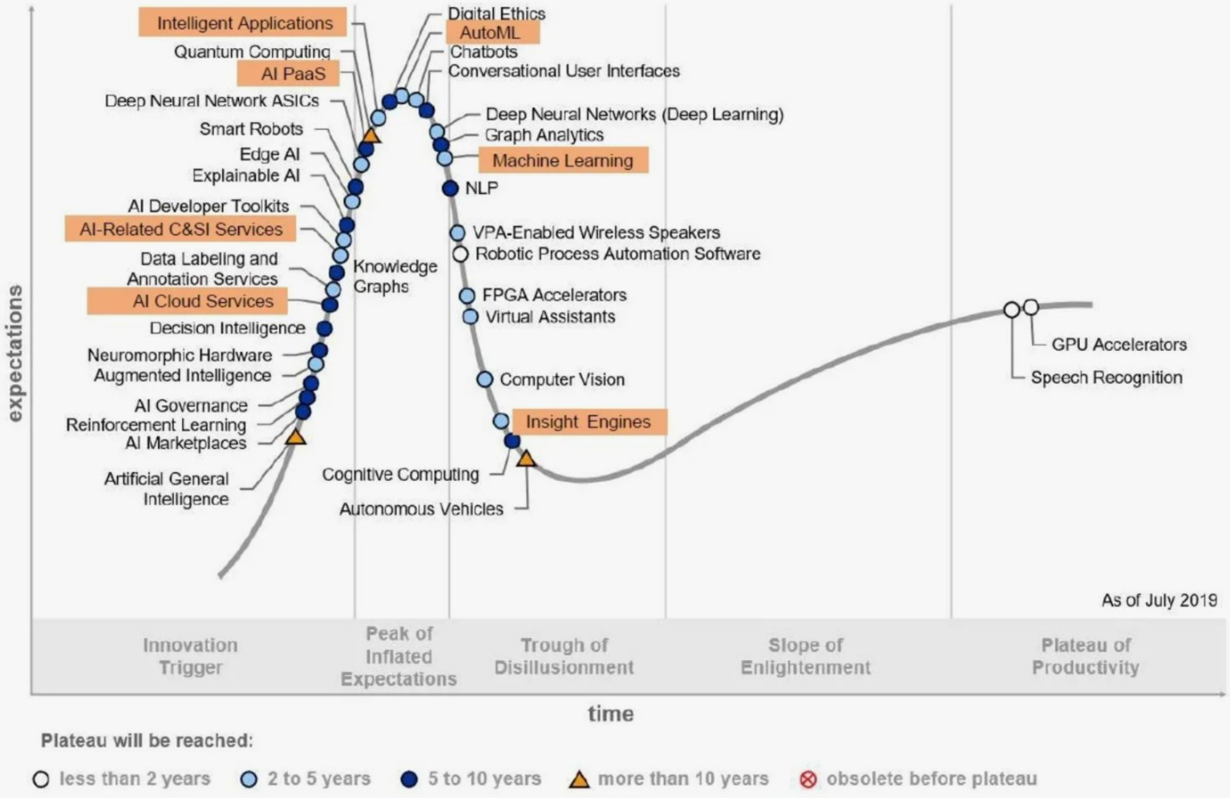
Intro to Reinforcement Learning

Lysyakov Arkadii

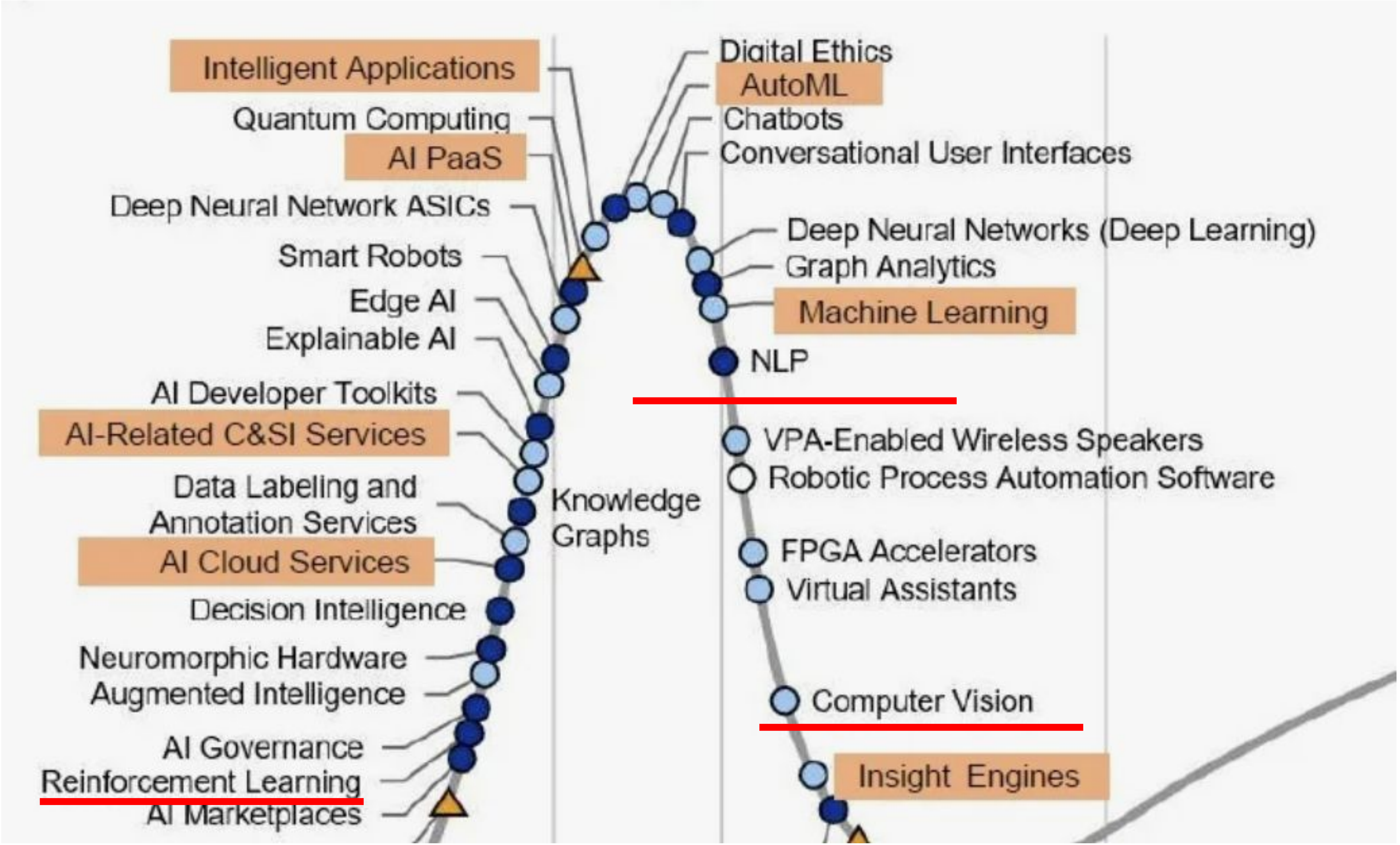
Based on: YSDA Practical RL course, Berkeley CS188 Intro to AI. Several icons by [Smashicons]

AI Is Entering Your Enterprise in Multiple Ways

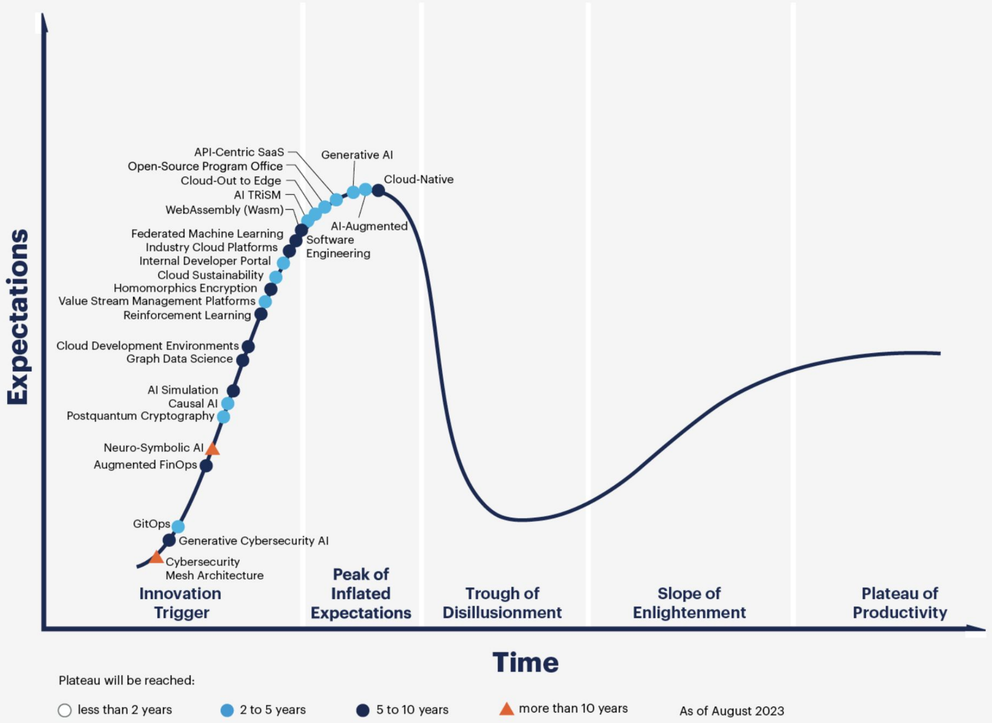
- ✓ Custom-developed AI solutions
- ✓ AI cloud services and APIs
- ✓ Search and insight engines
- ✓ AI embedded in ERP, CRM, HR applications
- ✓ Automated ML



From "Hype Cycle for Artificial Intelligence, 2019," 25 July 2019 (G00369840)


Intelligent Applications
Quantum Computing
AI PaaS
Deep Neural Network ASICs
Smart Robots
Edge AI
Explainable AI
AI Developer Toolkits
AI-Related C&SI Services
Data Labeling and Annotation Services
AI Cloud Services
Decision Intelligence
Neuromorphic Hardware
Augmented Intelligence
AI Governance
Reinforcement Learning
AI Marketplaces
Knowledge Graphs
Digital Ethics
AutoML
Chatbots
Conversational User Interfaces
Deep Neural Networks (Deep Learning)
Graph Analytics
Machine Learning
NLP
VPA-Enabled Wireless Speakers
Robotic Process Automation Software
FPGA Accelerators
Virtual Assistants
Computer Vision
Insight Engines

Hype Cycle for Emerging Technologies, 2023



gartner.com

Source: Gartner

Gartner

API-Centric SaaS
Open-Source Program Office
Cloud-Out to Edge
AI TRiSM
WebAssembly (Wasm)
Generative AI
Cloud-Native
AI-Augmented Software Engineering
Federated Machine Learning
Industry Cloud Platforms
Internal Developer Portal
Cloud Sustainability
Homomorphics Encryption
Value Stream Management Platforms
Reinforcement Learning
Cloud Development Environments
Graph Data Science
AI Simulation
Causal AI
Postquantum Cryptography
Neuro-Symbolic AI
Augmented FinOps

Differences

Key differences

Supervised Learning

- Learn to approximate reference answers
- Need reference answers
- Model does not affect the input data

Reinforcement Learning

- Learn optimal strategy by trial and error
- Need feedback on agent’s actions
- Agent actions affect the environment (so the observations)

Key differences

Unsupervised Learning

- Learn underlying data structure
- No feedback required
- Model does not affect the input data

Reinforcement Learning

- Learn optimal strategy by trial and error
- Need feedback on agent’s actions
- Agent actions affect the environment (so the observations)

Reinforcement Learning

problem statement

Supervised learning

- Given:
  - Objects and reference answers $\boxed{x \in \mathcal{X}, y \in \mathcal{Y}}$ Want them to be i.i.d.
  - Loss/objective function $\boxed{L(\hat{y}, y)}$ Usually differentiable
  - Model family $f \in \mathcal{F}, f : \mathcal{X} \longrightarrow \mathcal{Y}$
- Goal:
  - Find optimal mapping $f^* = \arg\min_f L(f(x), y)$

Reinforcement learning

- Given:

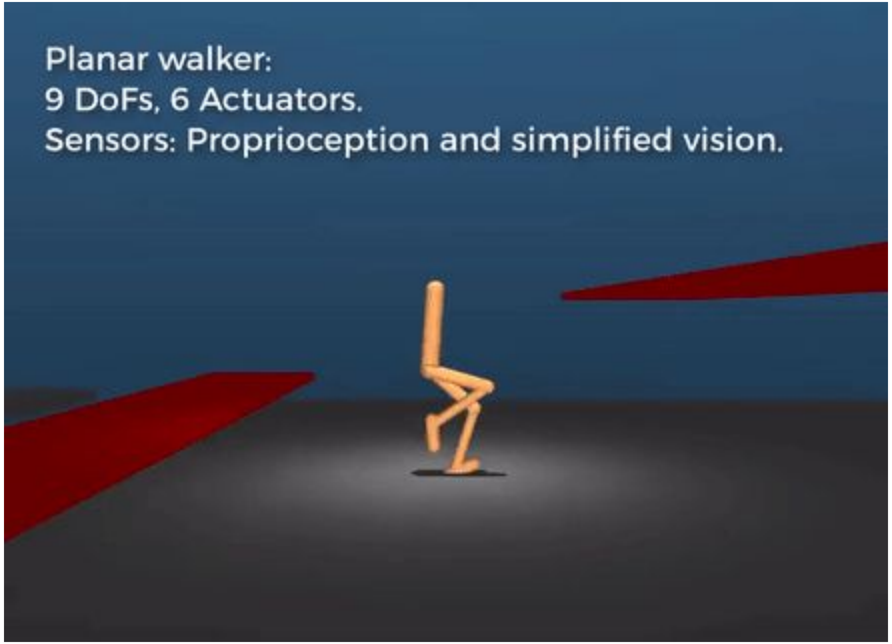
  Usually no reference answers

  E.g. want the robot to walk

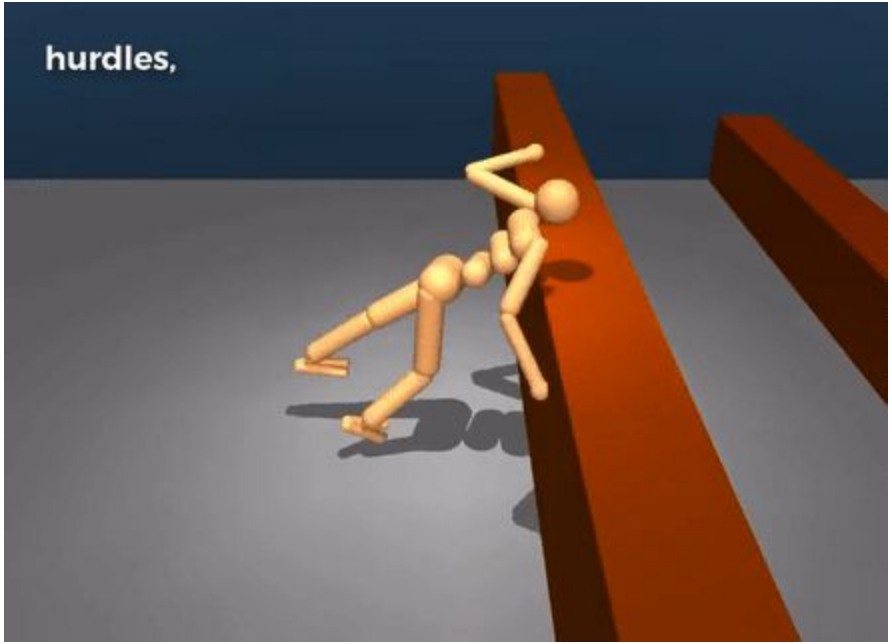
  - Objects ~~and reference answers~~ $x \in \mathcal{X}$
  - Loss/objective function $\boxed{L(\hat{y}, y)}$

    Usually even hard to formulate, non-differentiable

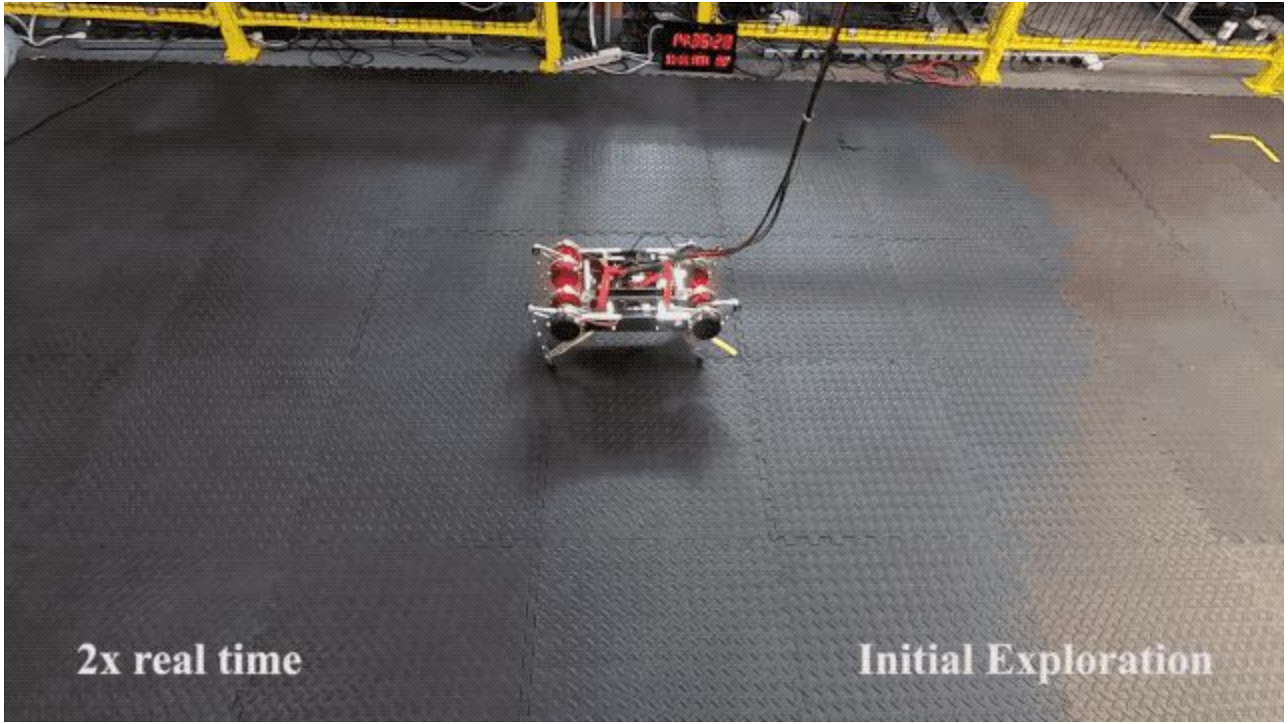
  - Model family $f \in \mathcal{F}, f : \mathcal{X} \longrightarrow \mathcal{Y}$
- Goal:
  - Find optimal mapping $f^* = \arg\min_f L(f(x), y)$



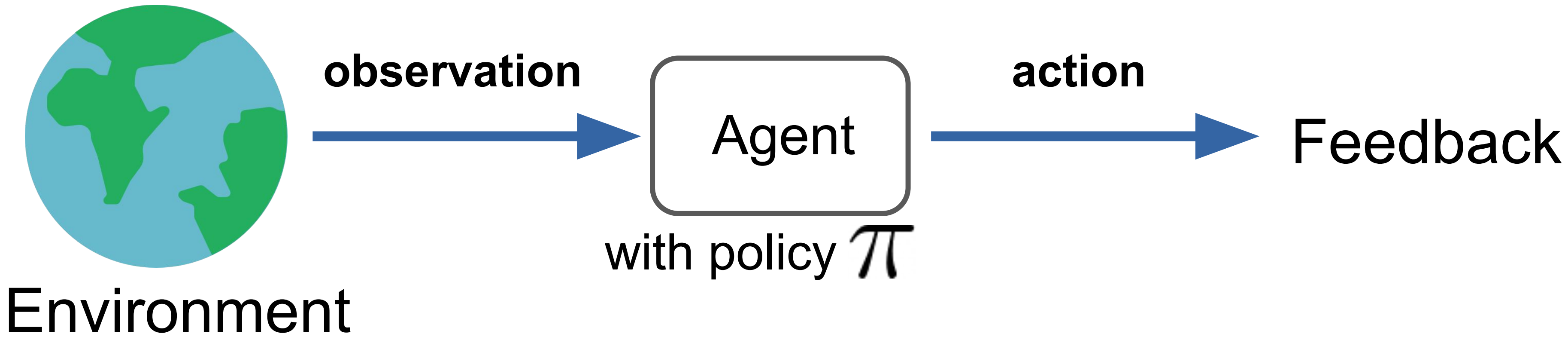
source: Emergence of Locomotion Behaviours in Rich Environments, DeepMind



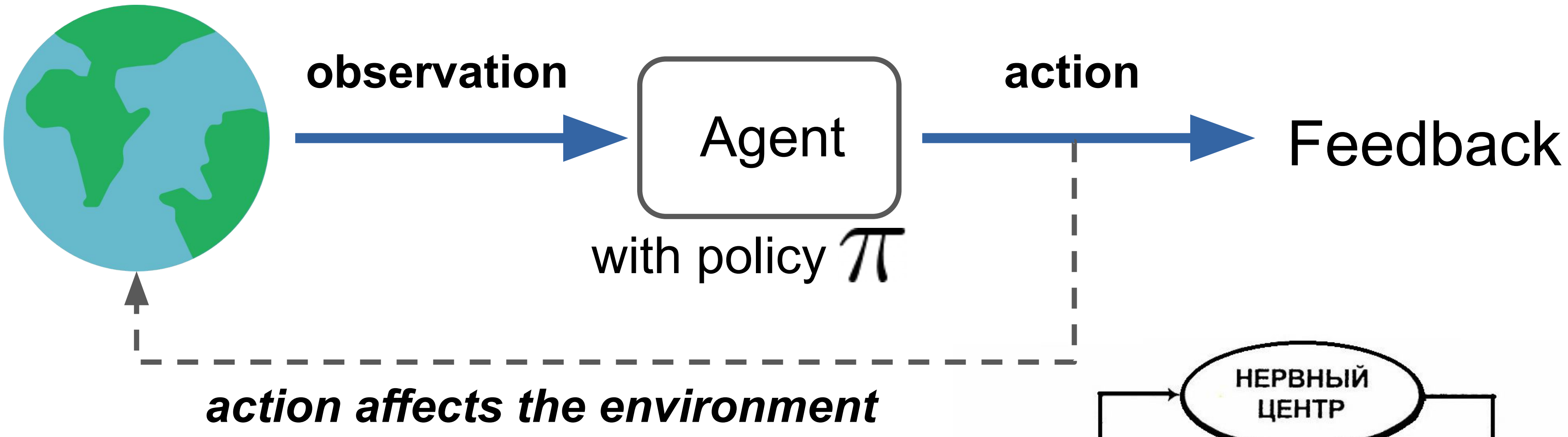
source: Emergence of Locomotion Behaviours in Rich Environments, DeepMind



source: [MIT Technology Review]

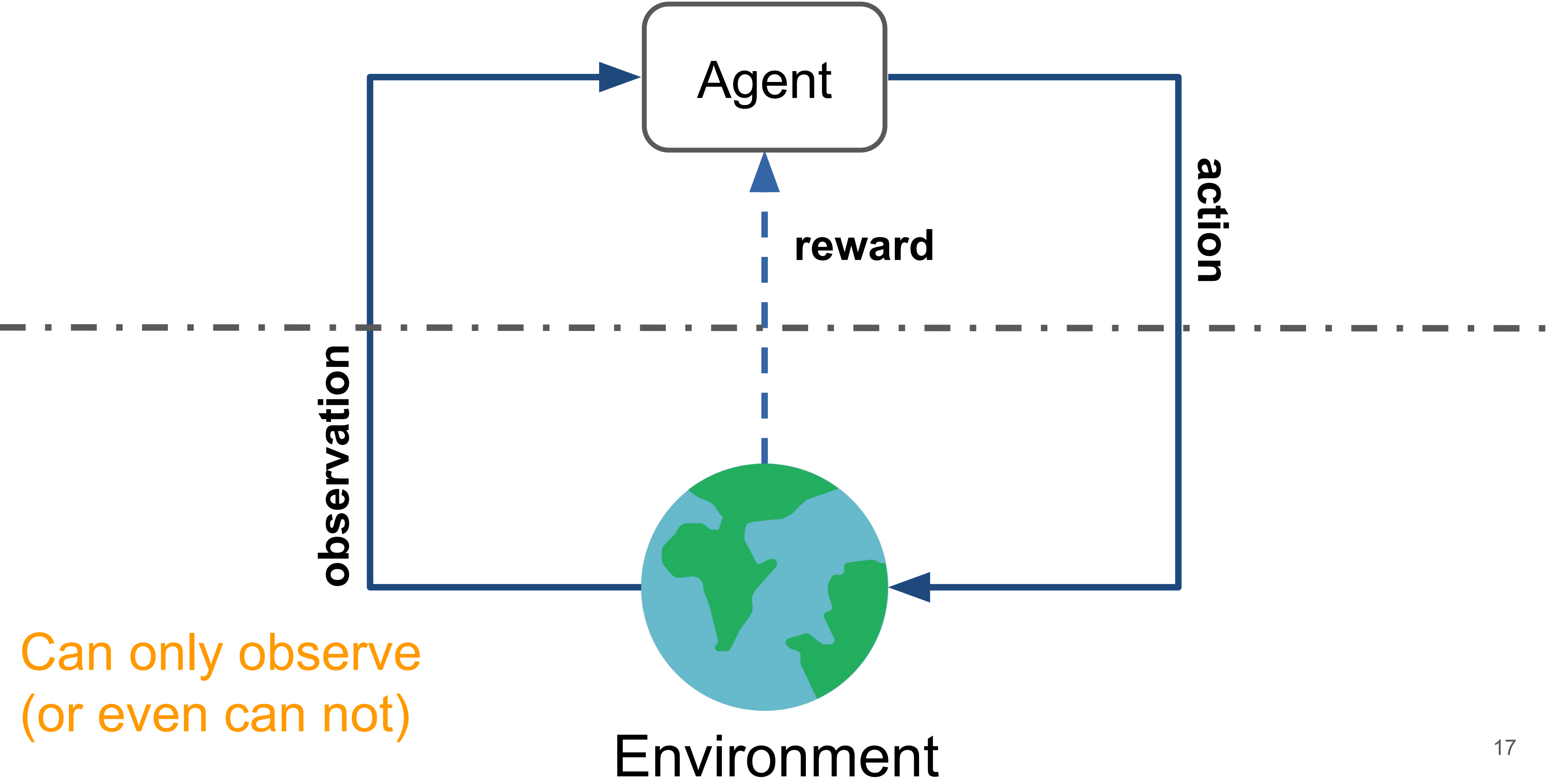


- Observation (state): vector or image or sequence … or *nothing*
- Policy: mapping from state to action
- Action
- Feedback (reward): usually a converted to a number



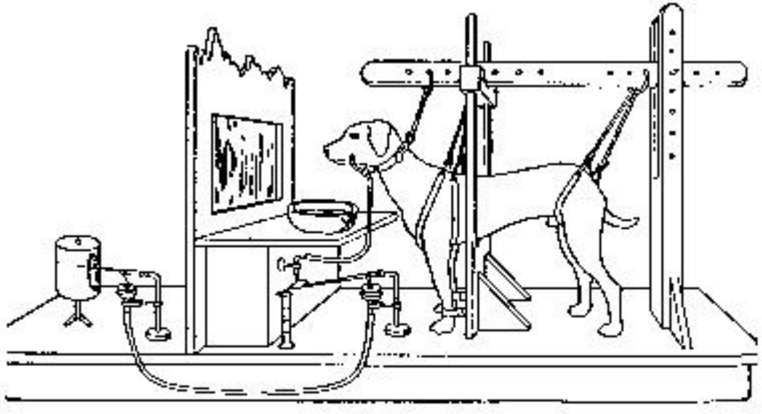
Рефлекторное кольцо введено А. Ф. Самойловым (ученик И.П. Павлова) в 1930, также исследовалось Н.А. Бернштейном

We are in control



Can only observe
(or even can not)

Psychology: point of view



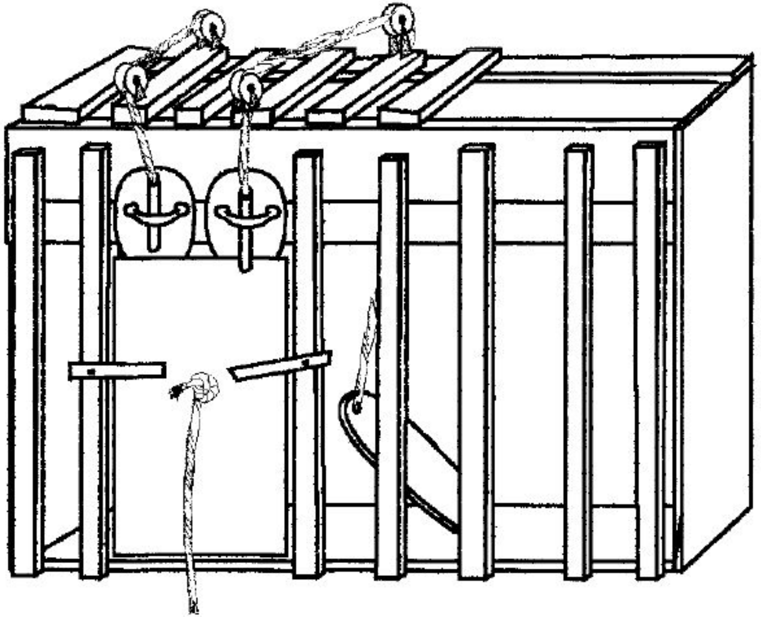
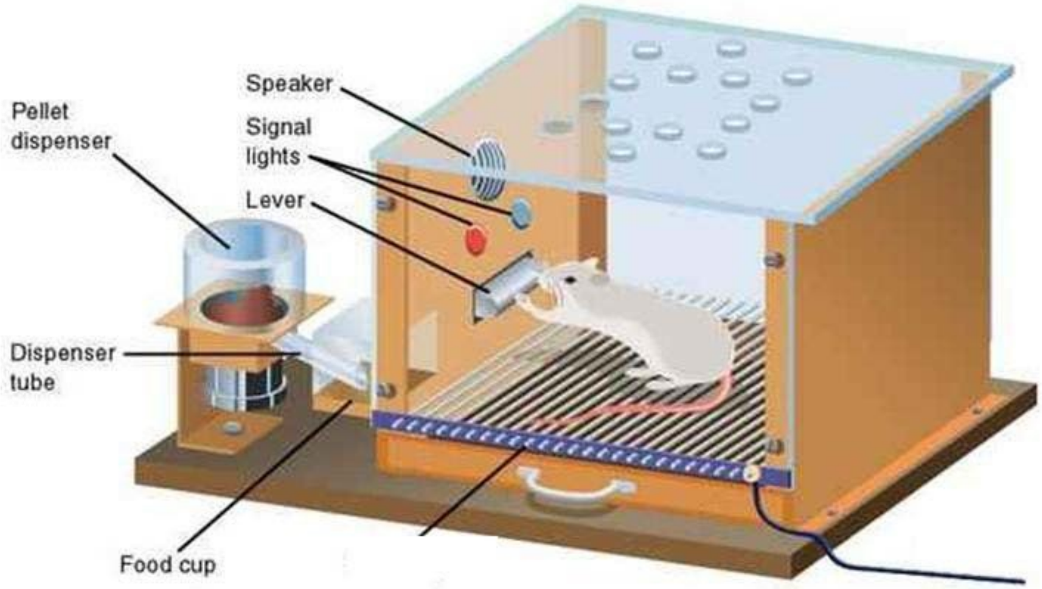


Fig. 4. Box K. The door is held in place by a weight suspended by a string. To open the door, a cat had to depress a treadle, pull on a string, and push a bar up or down. (After Thorndike, 1898, Figure 1, p. 8.)

source: Thorndike experiment: original paper (1898) and blog post
Pavlov's Classical conditioning on Wikipedia

Psychology: point of view



source: Operant conditioning chamber (Skinner box) on Wikipedia, Biomechatronic design and development of a legged rat robot paper

Reality check: dynamic control

Variety of papers on helicopter control:
heli.stanford.edu

Andrew Y. Ng PhD Thesis link: “Shaping and policy search in Reinforcement Learning”

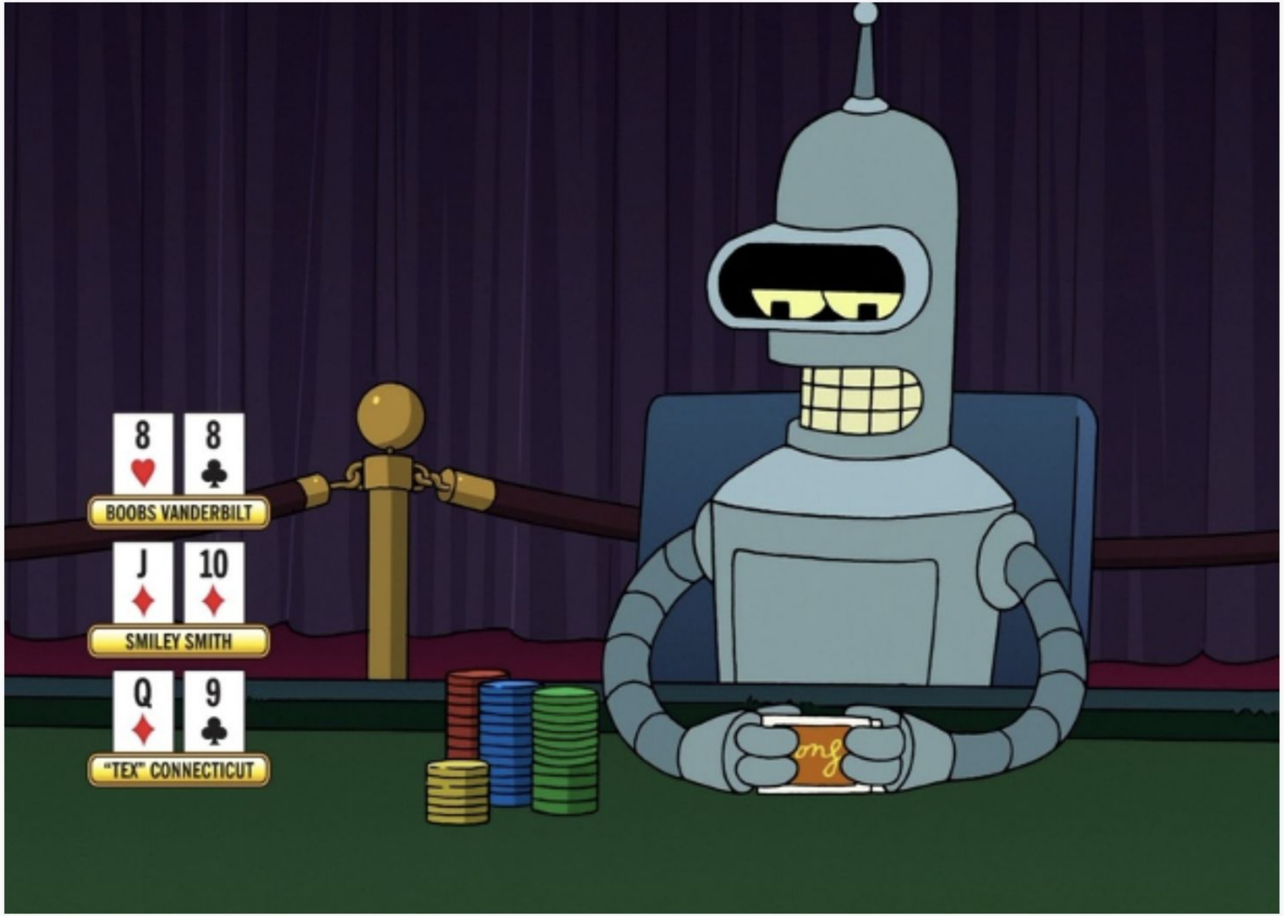
- Observation: accelerometer, gyroscope, engine data
- Action: change rotation speed, angle
- Feedback: some specific reward

source: heli.stanford.edu, photos by Ben Tse and Eugene Fratkin

~~Reality~~ check: video games

- Observation: image(s)
- Action: move, fire, turn
- Feedback: score/health/progres/…

source: gym.openai.com, DOOM



Futurama: Into the Wild Green Yonder / 20th Century Fox Home Entertainment, 2009

value-based Vs policy-based

Value-based

- Q-learning, SARSA, MCTS value-iteration

- Solves harder problem
- Artificial exploration
- Learns from partial experience (temporal difference)
- Evaluates strategy for free :)

Policy-based

- REINFORCE, CEM

- Solves easier problem
- Innate exploration
- Innate stochasticity
- Support continuous action space
- Learns from full session only?

Open questions

- What is *optimal* action?
  - Maximize the reward **on the next step**
  - Maximize the reward **in long term**

Open questions

- *Explore* or *exploit*?
  - Stepping of *current optimal* strategy may **decrease** the cumulative reward
  - Under *current optimal strategy* one may **never discover** something better

How to maximize the reward?

$\mathbb{E}_{\pi}[R]$ is an expected cumulative reward earned per session following policy π

Need to maximize the following objective:

$$\mathbb{E}_{\pi}[R] = \mathbb{E}_{s_0 \sim P(s_0)} \mathbb{E}_{a_0 \sim \pi(a|s_0)} \dots \mathbb{E}_{s_t, r_t \sim P(s, r | s_{t-1}, a_{t-1})} [r_0 + \dots + r_t]$$

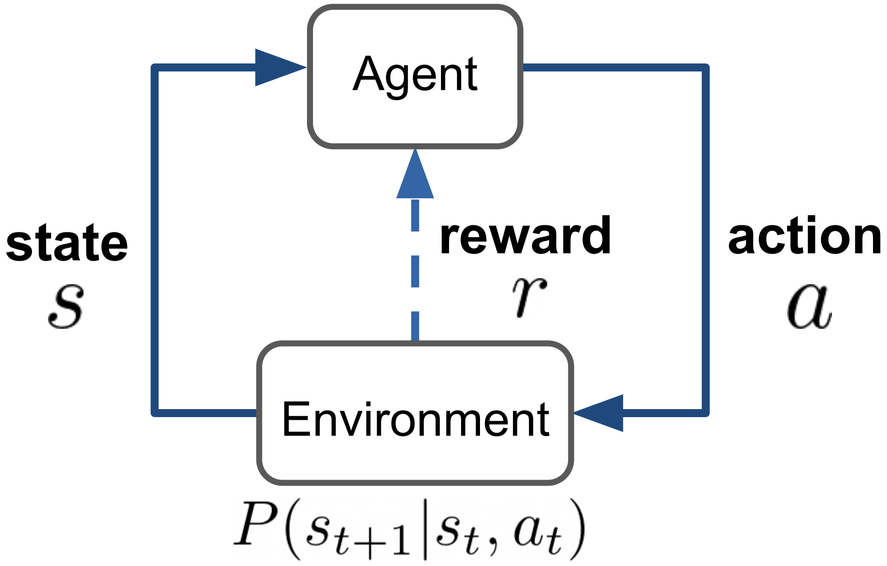
How to do it?

How to maximize the reward?

- Play a few sessions with existing policy
- Update the policy using new feedback
- Repeat

MDP formalism

- State: $s \in \mathcal{S}$
- Action: $a \in \mathcal{A}$
- Reward: $r \in \mathbb{R}$



- Dynamics: $P(s_{t+1}|s_t, a_t)$

Markov property:

$$P(s_{t+1}|s_t, a_t, \ldots, s_0, t_0) = P(s_{t+1}|s_t, a_t)$$

Total reward

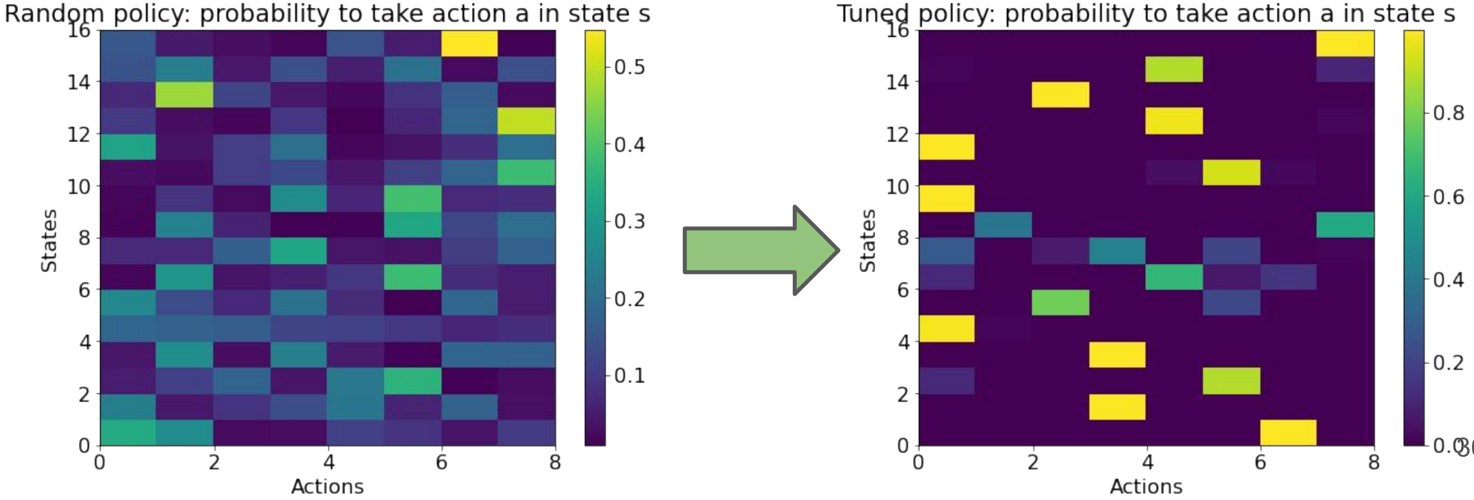
- Total reward for session: $R = \sum_t r_t$

- Policy: $\pi(a|s) = P(\text{take action } a \text{ in state } s)$

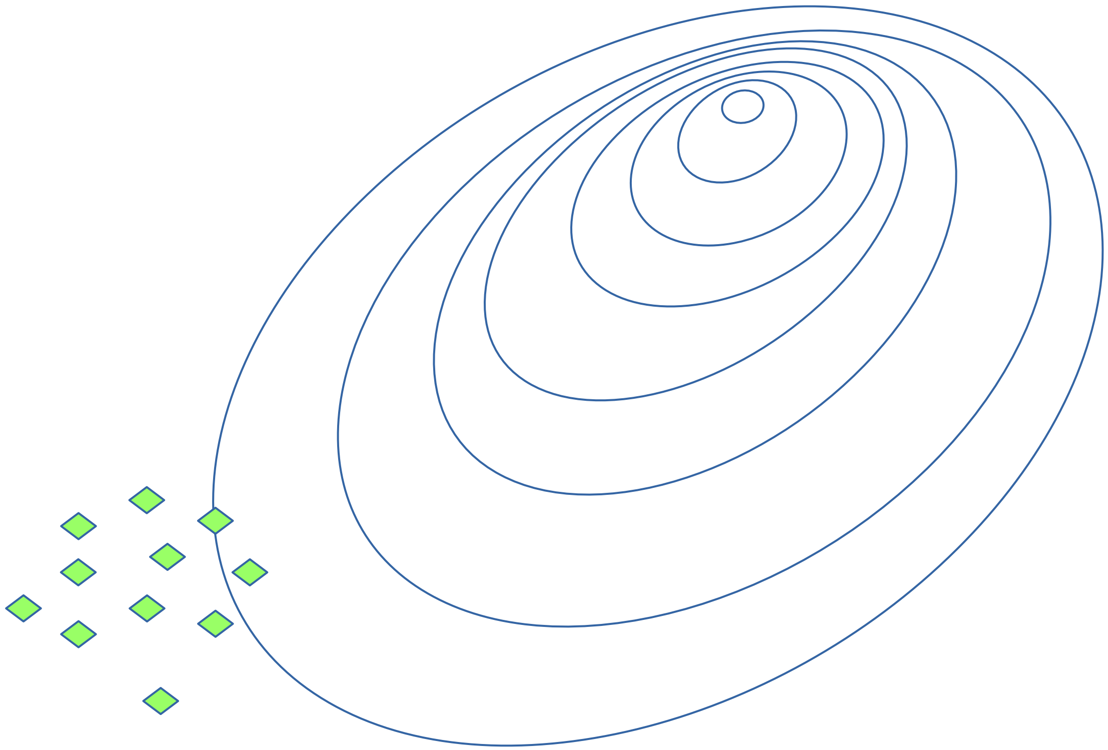
- Goal: maximize reward; $\pi^*(a|s) = \arg\max_\pi \mathbb{E}_\pi[R]$

Crossentropy method: tabular case

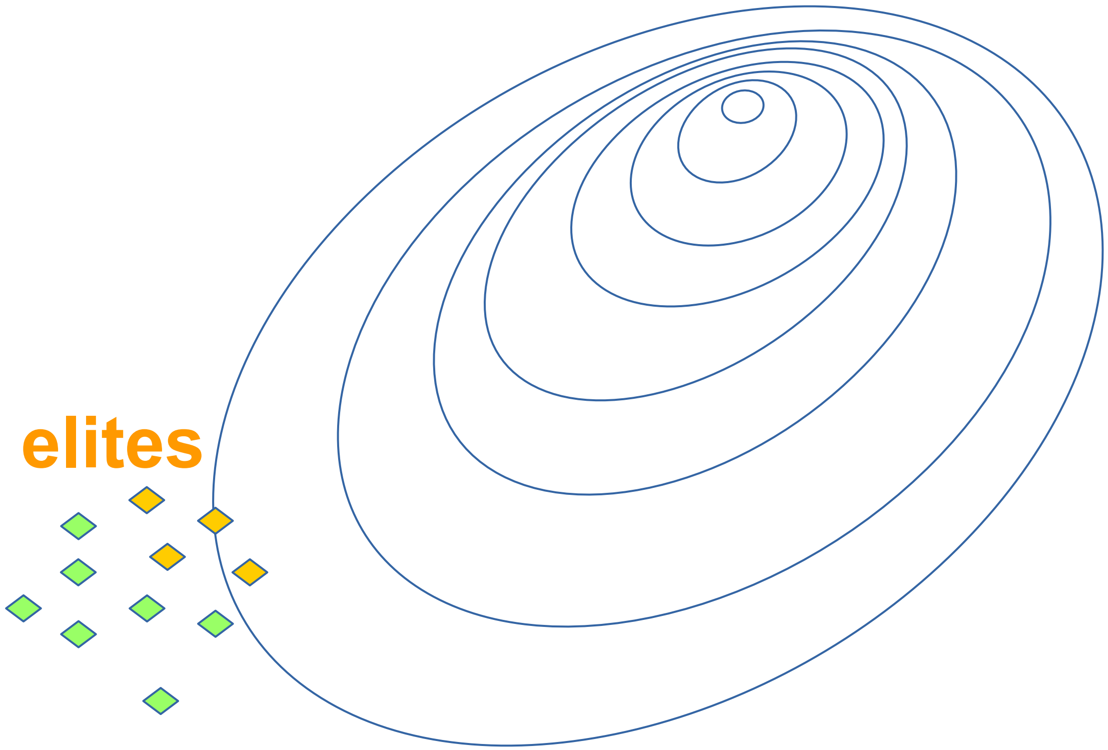
- Initialize policy (state-action matrix, every row sums up to 1)
- Sample N sessions
- Select M **elite** sessions with highest rewards
- Update policy using the elite session state-action sequences
- Repeat



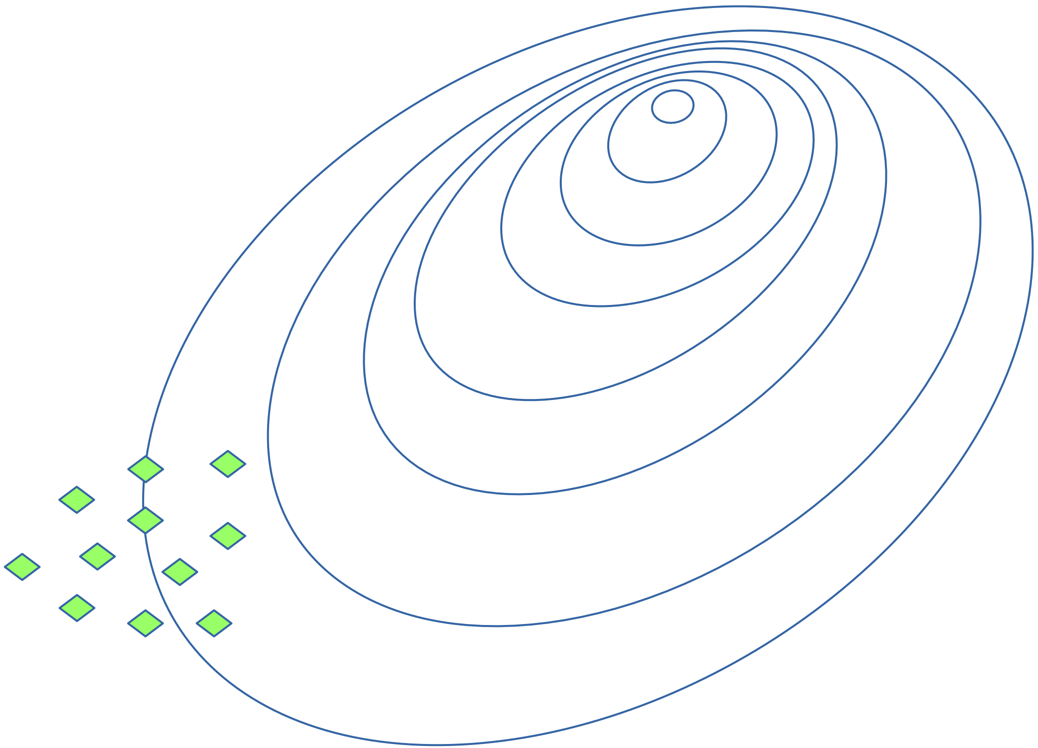
Crossentropy method: illustration



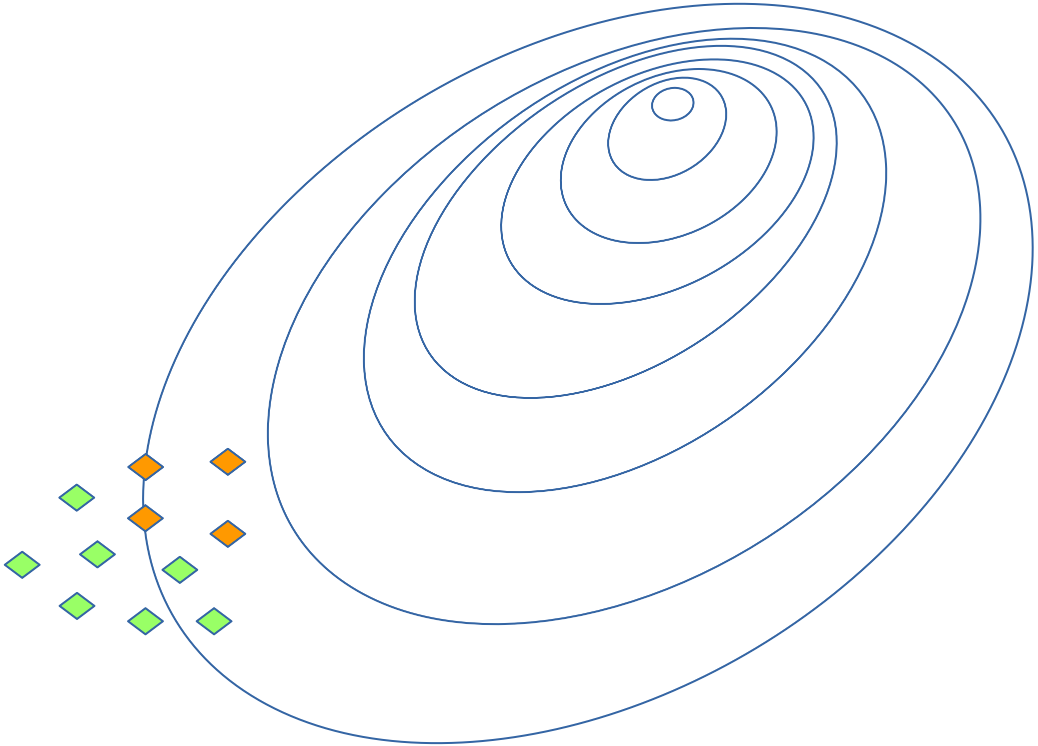
Crossentropy method: illustration



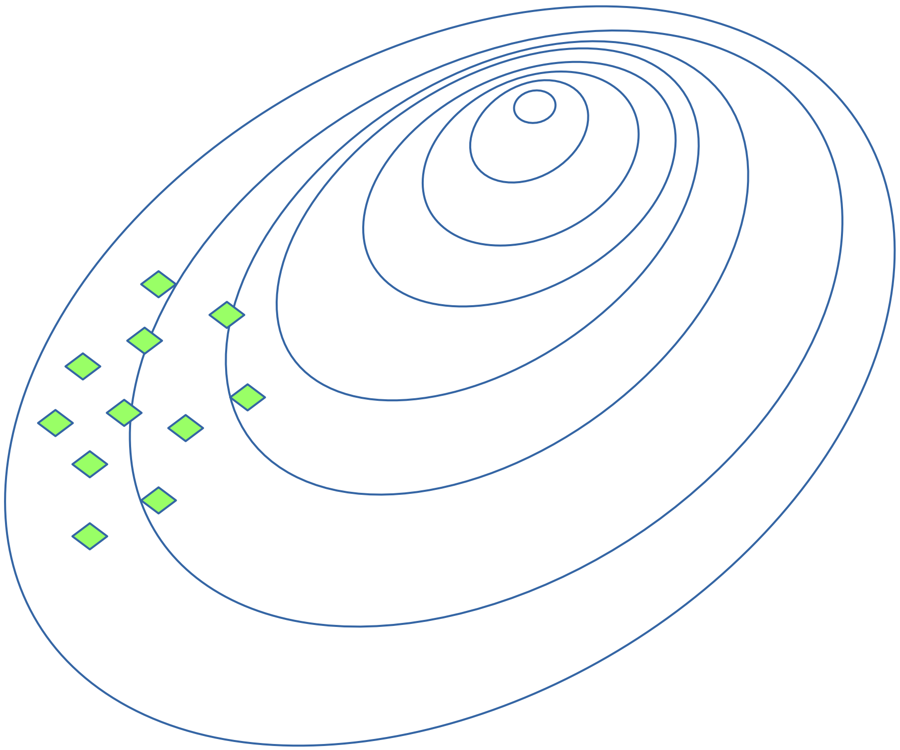
Crossentropy method: illustration



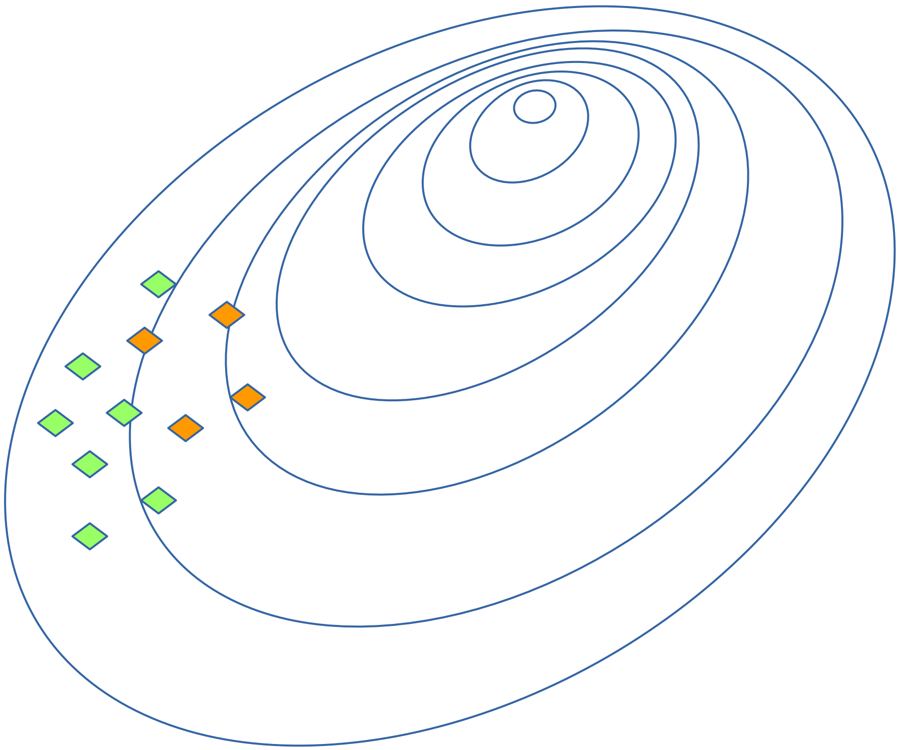
Crossentropy method: illustration



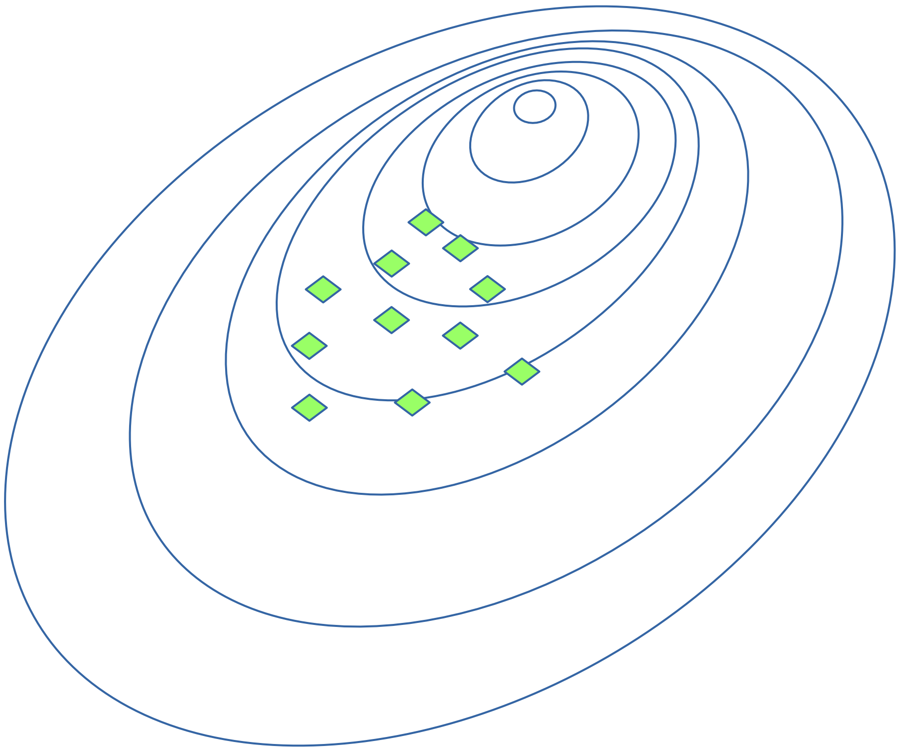
Crossentropy method: illustration



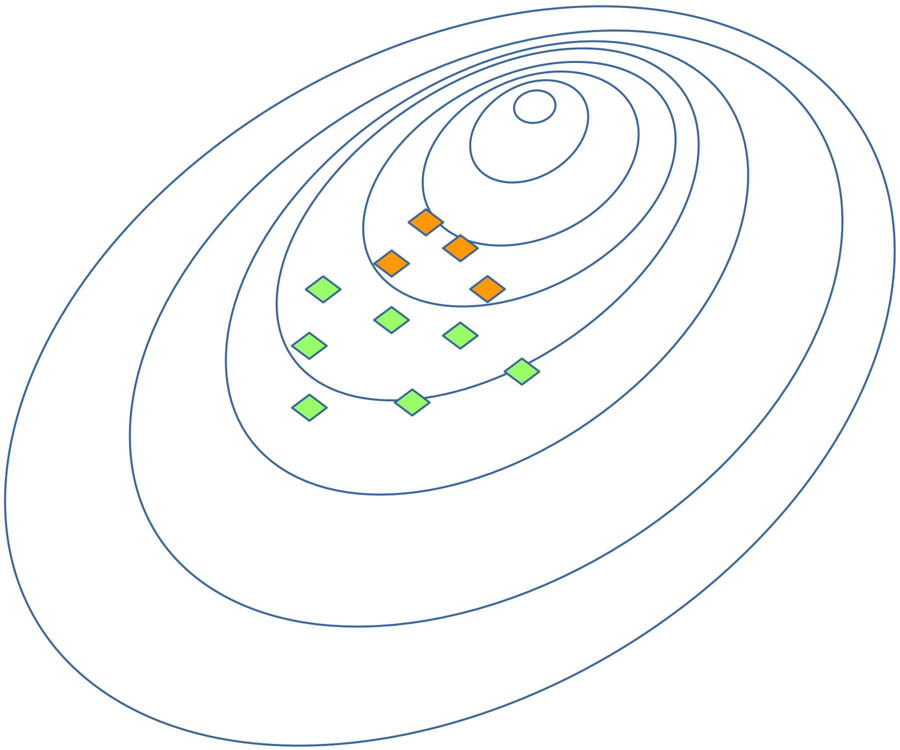
Crossentropy method: illustration



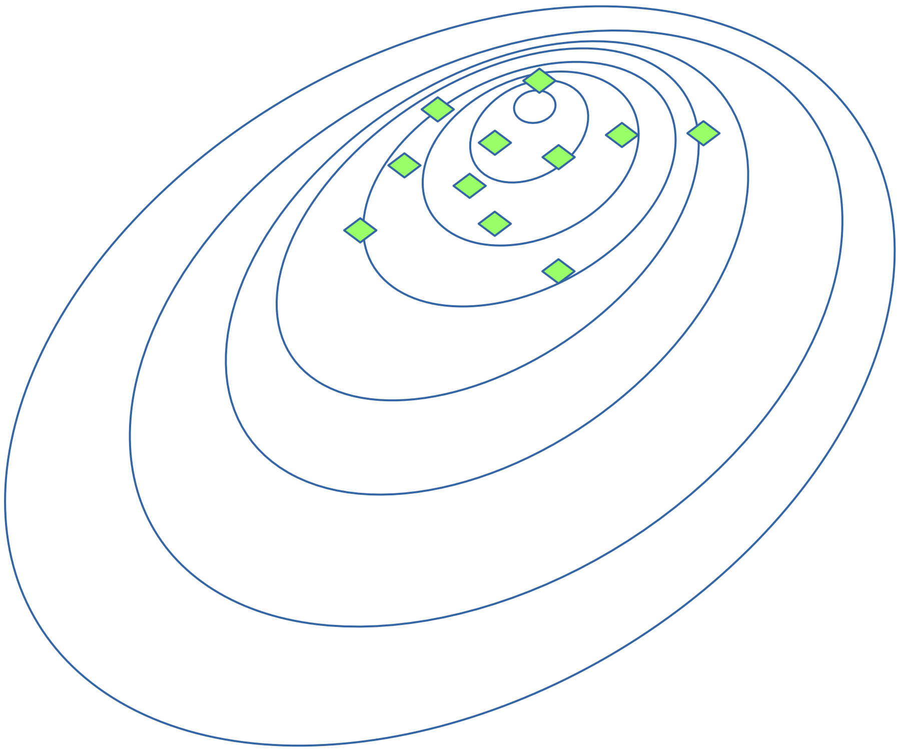
Crossentropy method: illustration



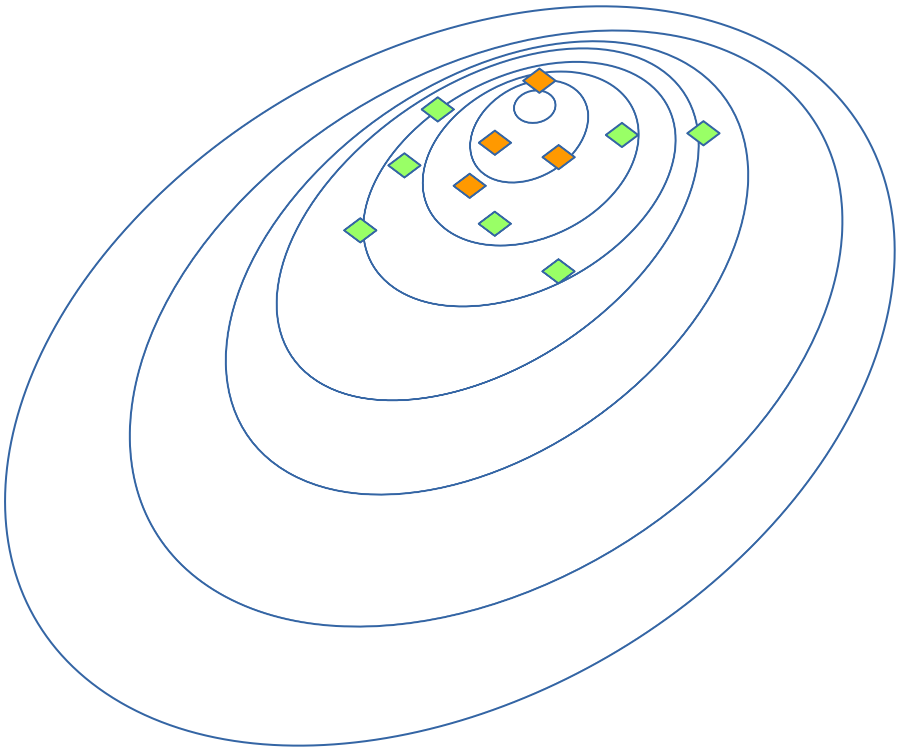
Crossentropy method: illustration



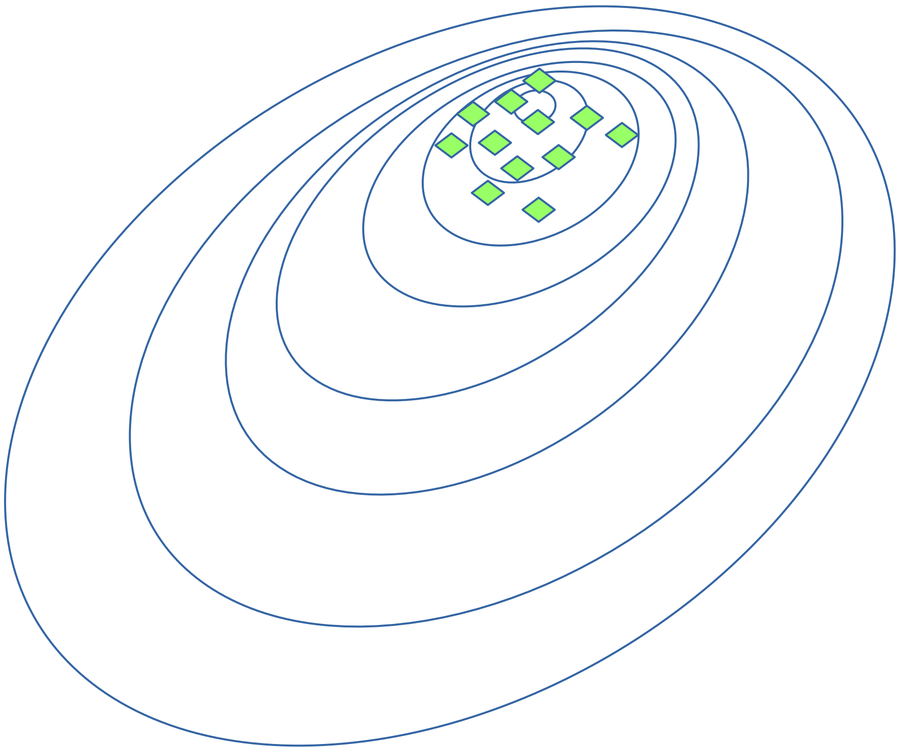
Crossentropy method: illustration



Crossentropy method: illustration



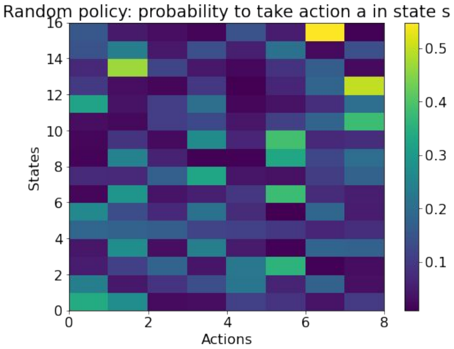
Crossentropy method: illustration



Crossentropy method: tabular case

- Policy is a matrix

$$\pi(a|s) = A_{s,a}$$



- Sample N games with this policy

- Select M **elite** sessions with highest rewards

$$\text{Elite} = [(s_0, a_0), (s_1, a_1), \ldots, (s_M, a_M)]$$

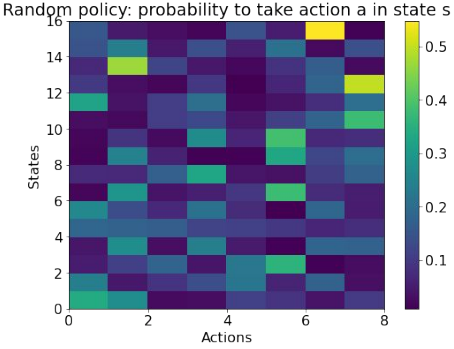
- Update policy:

$$\pi_{\text{new}}(a|s) = \frac{\sum\limits_{s_t, a_t \in \text{Elite}} [s_t = s][a_t = a]}{\sum\limits_{s_t, a_t \in \text{Elite}} [s_t = s]}$$

Crossentropy method: tabular case

- Policy is a matrix

$$\pi(a|s) = A_{s,a}$$



- Sample N games with this policy
- Select M **elite** sessions with highest rewards
- Update policy using the **elite** sessions:

$$\pi_{\text{new}}(a|s) = \frac{\text{how many times took action } a \text{ at state } s}{\text{how many times was at state } s}$$

Harsh reality

Some environments have huge or infinite number of states

How to fix it?

Approximate crossentropy method

- Model (e.g. parametric) predicts action probability given state:

$$\pi(a|s) = f_\theta(a, s)$$

```
model = RandomForestClassifier()
```

Random Forest Classifier, Logistic Regression, NN etc.

- Sample N sessions, select M **elite** sessions

$$\text{Elite} = [(s_0, a_0), (s_1, a_1), \ldots, (s_M, a_M)]$$

New training set; states are objects, actions are target values

- Maximize likelihood of actions in elite sessions:

$$\pi(a|s)_{\text{new}} = \arg\max_{\pi} \sum_{s_t, a_t \in \text{Elite}} \log \pi(a_i|s_i)$$

```
model.fit(elite_states, elite_actions)
```

Approximate crossentropy method

What if action space is continuous?

- Model samples actions from some appropriate distribution:

$$\pi(a|s) = \mathcal{N}(\mu_\theta(a,s), \sigma_\gamma(a,s))$$

One model — Another model (or constant)

It is just a regressor!

Approximate crossentropy method

What if action space is continuous?

- Model (e.g. parametric) predicts action given state:

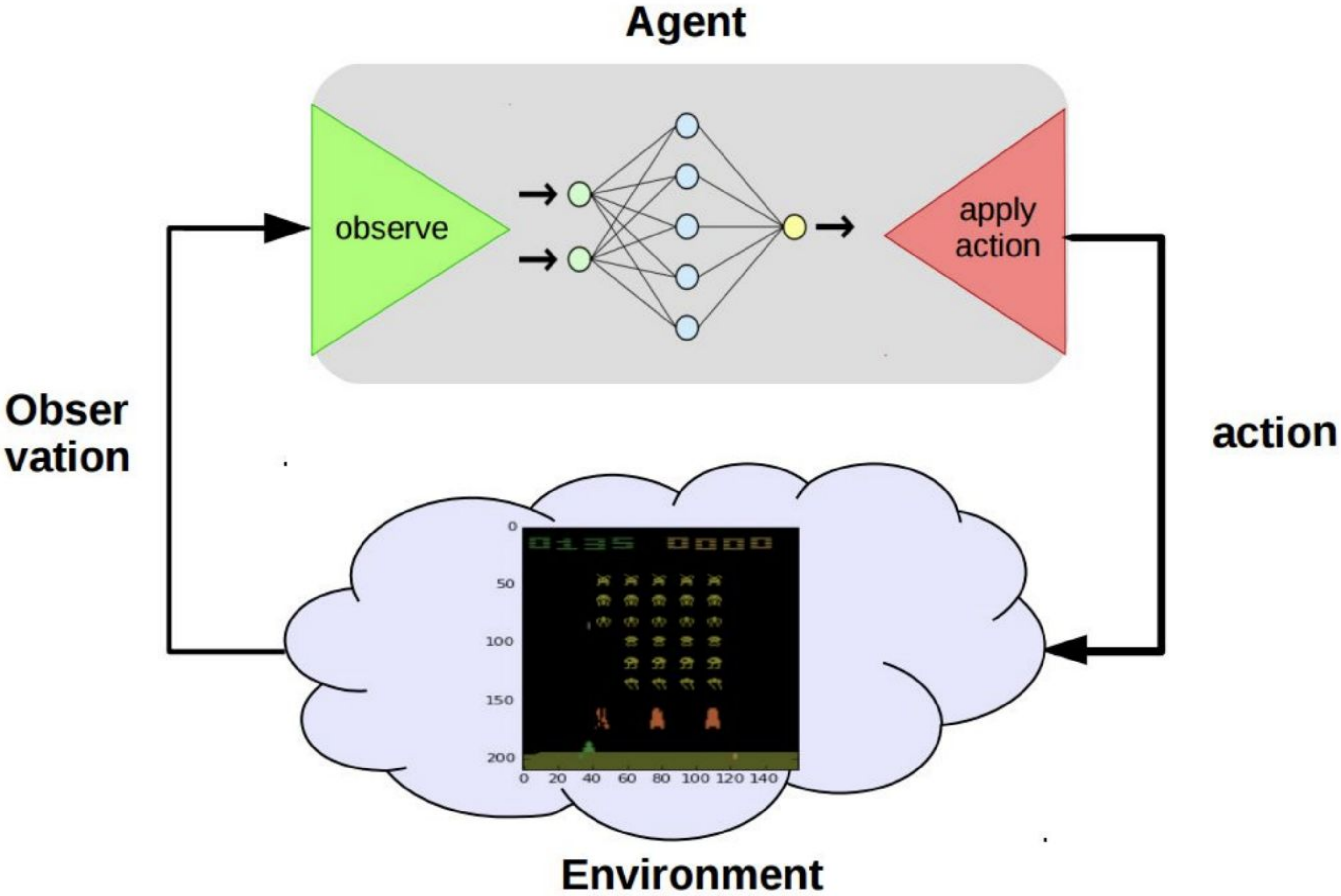
```
model = RandomForestRegressor()
```

- Sample N sessions, select M **elite** sessions

$$\text{Elite} = [(s_0, a_0), (s_1, a_1), \ldots, (s_M, a_M)]$$

- Maximize likelihood of actions in elite sessions:

```
model.fit(elite_states, elite_actions)
```


Agent
observe
apply
action
Obser
vation
action
Environment

Basic definitions

$$G_t = \sum_{t'=t}^{T} \gamma^{(t'-t)} r_{t'}$$

$$Q^\pi(s,a) = E_\pi[G_t | s_t = s, a_t = a]$$

$$V^\pi(s) = E_\pi[G_t | s_t = s]$$

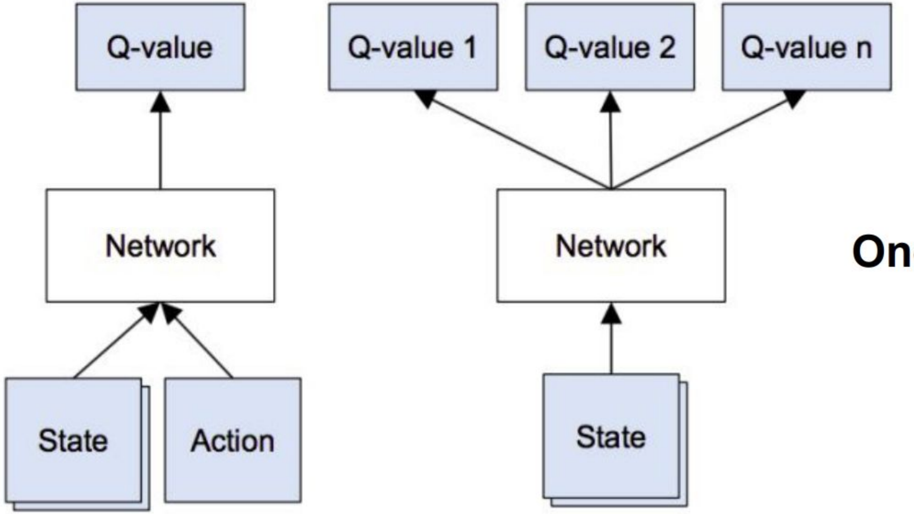
Recurrent relations

$$Q^\pi(s,a) = E_{s_{t+1}}[r_t + \gamma V^\pi(s_{t+1})]$$

$$Q^\pi(s,a) = E_{s_{t+1},a_{t+1}\sim\pi}[r_t + \gamma Q^\pi(s_{t+1}, a_{t+1})]$$

Possible architectures

Continuous control or large number of actions

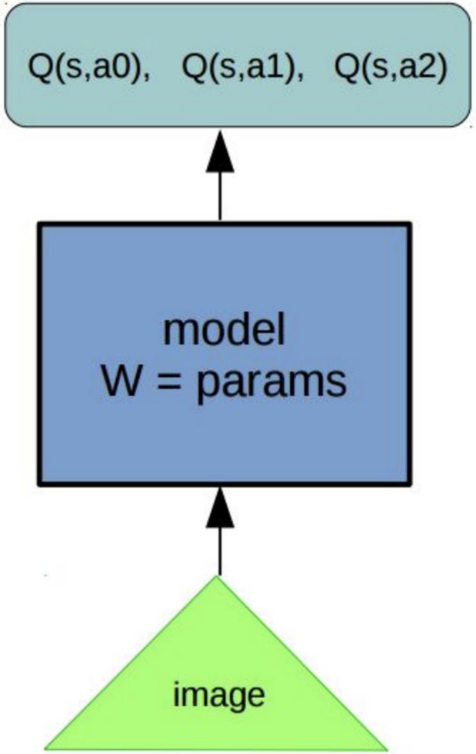


One pass for all actions

Given **(s,a)**
Predict Q(s,a)

Given **s** predict all q-values
Q(s,a0), Q(s,a1), Q(s,a2)

Approximate Q-learning



Q-values:

$$\hat{Q}(s_t, a_t) = r + \gamma \cdot max_{a'} Q(s_{t+1}, a')$$

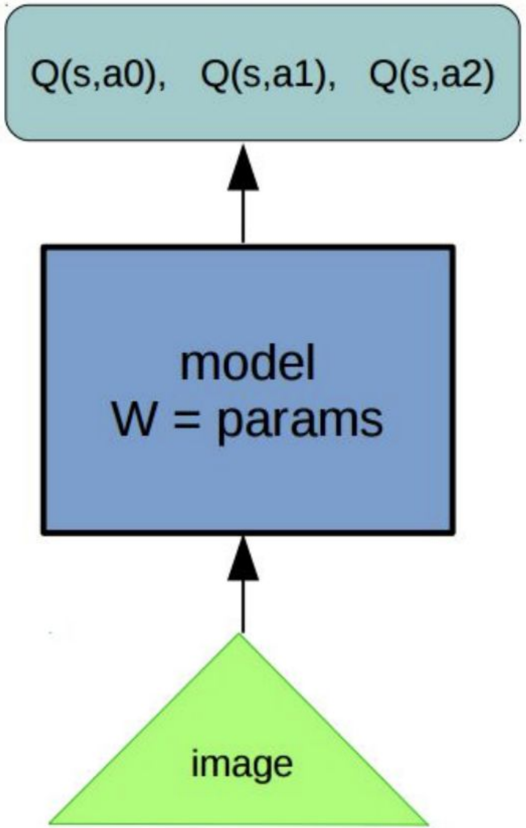
Objective:

$$L = (Q(s_t, a_t) - [r + \gamma \cdot max_{a'} \underbrace{Q(s_{t+1}, a')}_{\text{Const}}])^2$$

Gradient step:

$$w_{t+1} = w_t - \alpha \cdot \frac{\delta L}{\delta w}$$

Approximate Q-learning



Objective:

$$L=\left(Q(s_t,a_t)-\underbrace{\hat{Q}(s_t,a_t)}_{\text{consider const}}\right)^2$$

Q-learning:

$$\hat{Q}(s_t,a_t)=r+\gamma\cdot max_{a'}Q(s_{t+1},a')$$

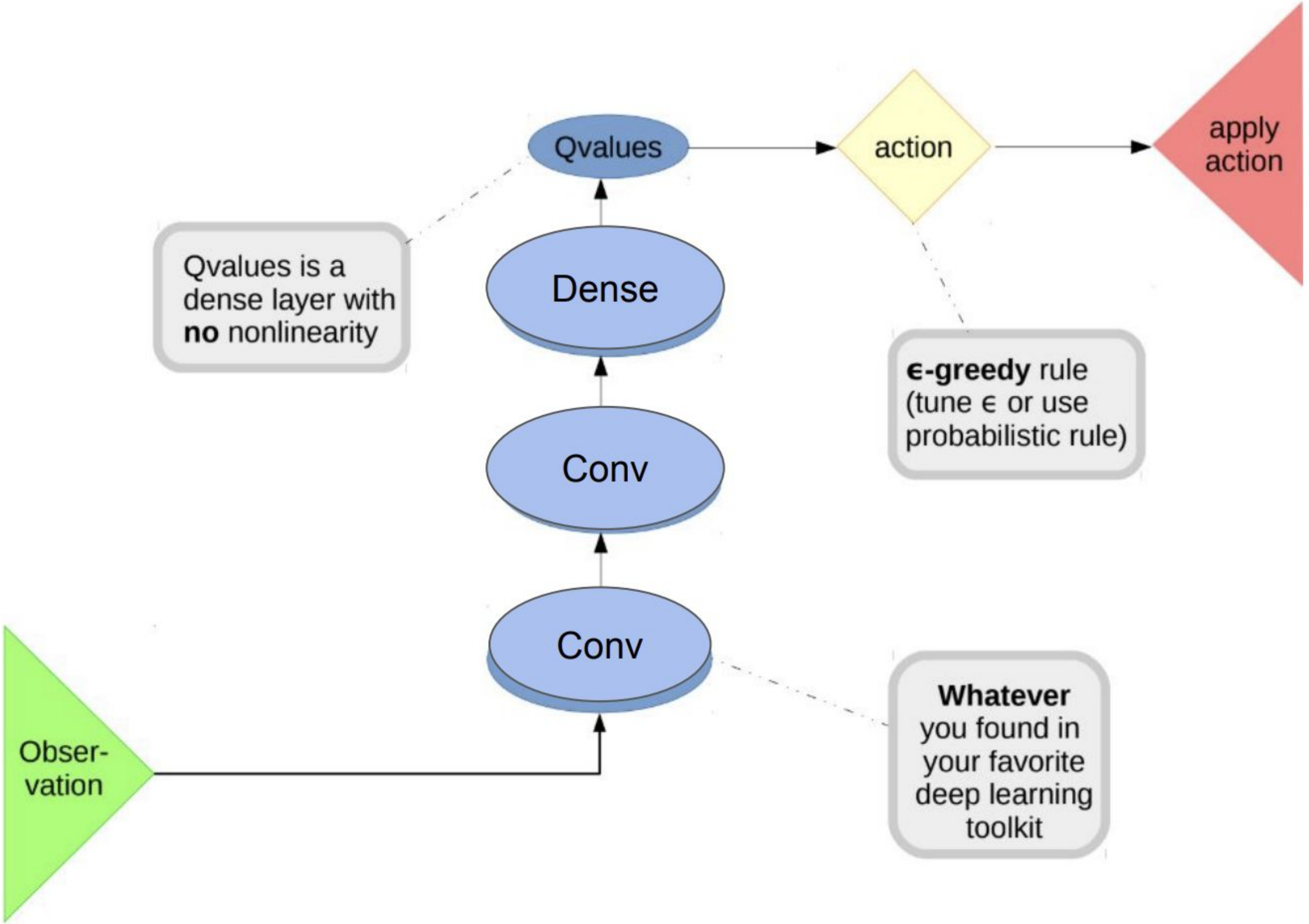
SARSA:

$$\hat{Q}(s_t,a_t)=r+\gamma\cdot Q(s_{t+1},a_{t+1})$$

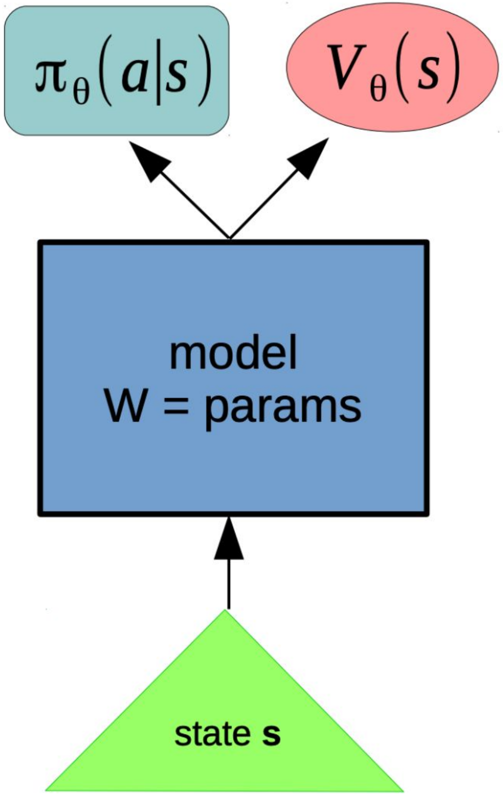
Expected Value SARSA:

$$\hat{Q}(s_t,a_t)=r+\gamma\cdot \underset{a'\sim\pi(a|s)}{E}Q(s_{t+1},a')$$

Basic deep Q-learning



Advantage actor-critic



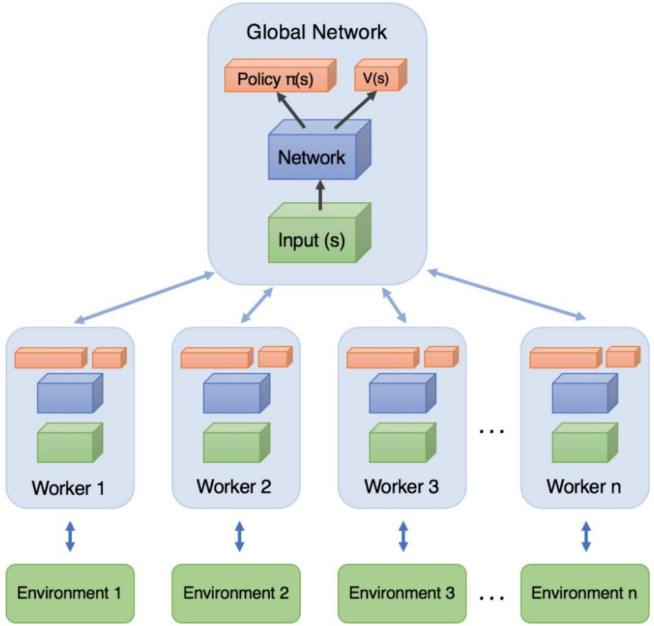
Improve policy:

$$\nabla J_{actor} \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{s,a \in z_i} \nabla \log \pi_\theta(a|s) \cdot A(s,a)$$

Improve value:

$$L_{critic} \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{s,a \in z_i} (V_\theta(s) - [r + \gamma \cdot V(s')])^2$$

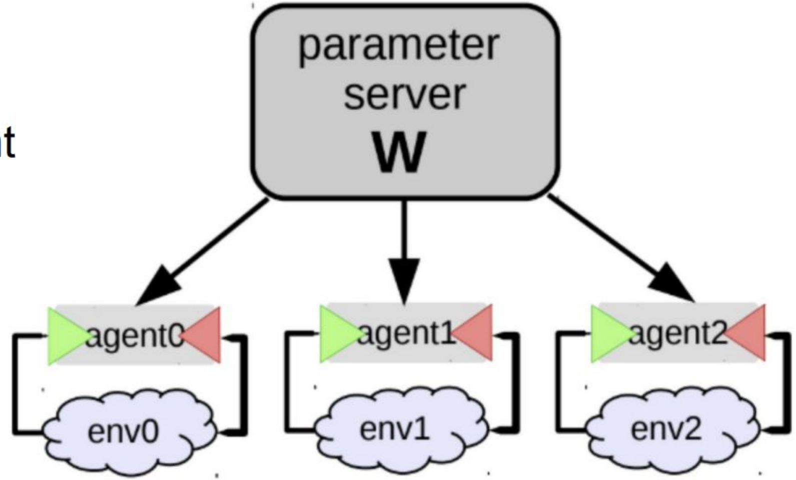
Asynchronous advantage actor-critic



Experience replay

Idea: Throw in several agents with shared **W**.

- Chances are, they will be exploring different parts of the environment
- More stable training
- Requires a lot of interaction

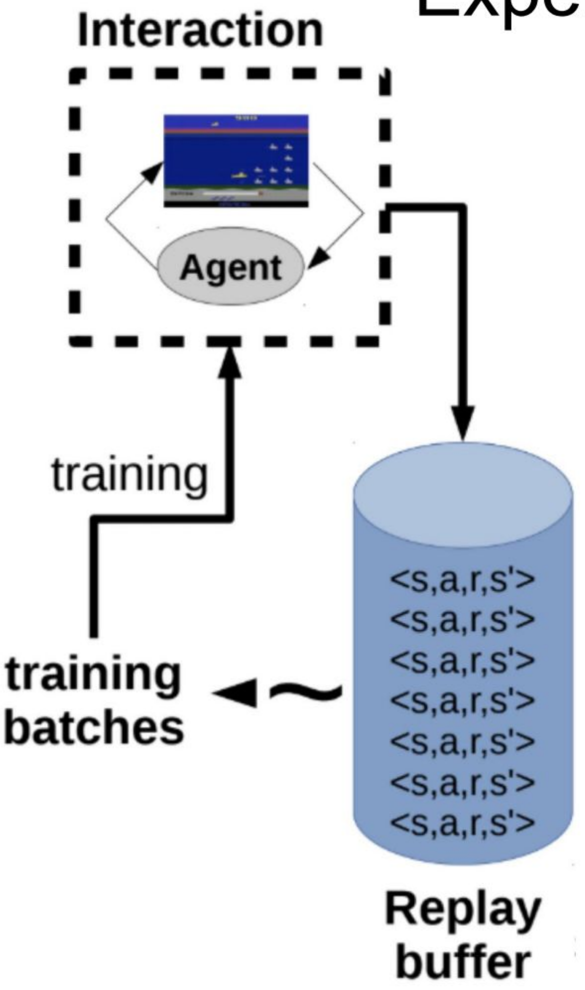


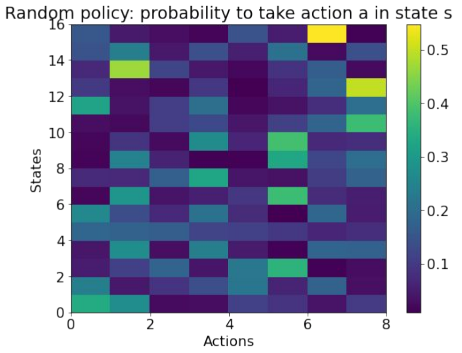
Experience replay

Idea: store several past interactions

<s,a,r,s'>

Train on random subsamples





Pseudocode

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence