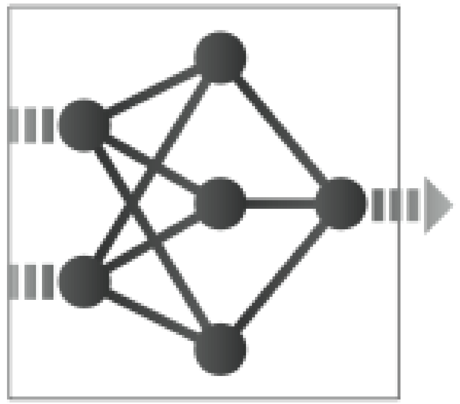
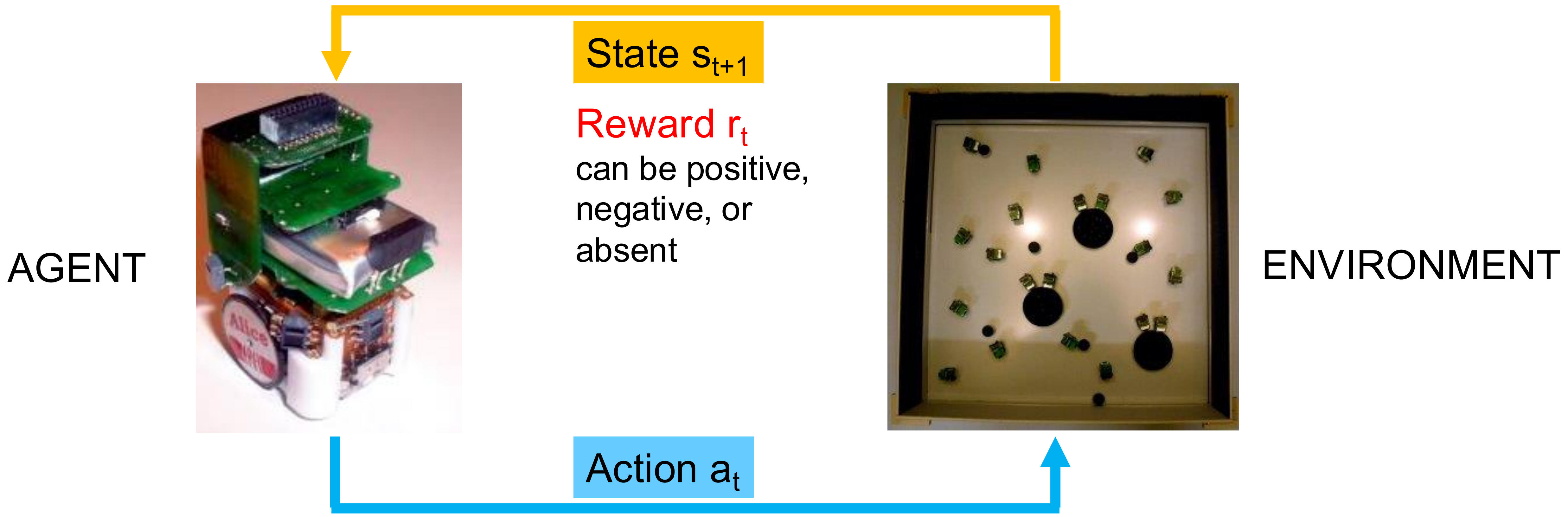
Reinforcement Learning and Evolutionary Computation



What you will learn in this class

- The Reinforcement Learning Framework
- Reward and Total Return
- The state-action value function (Q function)
- Value Learning: Deep Q Learning
- Policy Learning: Policy Gradient Learning
- Evolutionary Algorithms vs Reinforcement Learning

Reinforcement learning framework



The agent wants to find a mapping from states to actions (the *policy*) that maximizes the total future reward (the *Total Return*)

$$R_t = \sum_{i=t}^{\infty} r_i$$

Reward discount and Episode/Rollout

Temporal credit assignment problem: which actions are responsible for a delayed reward (or punishment)?

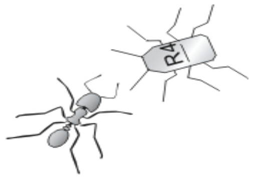
The *discount* factor γ is used to distribute reward over a sequence of actions and gives more importance to immediate rewards than to remote future rewards

$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i \qquad 0 < \gamma < 1$$

Episode: the finite number of steps *n* during which the agent interacts with the environment until a natural terminal event (e.g., game win).

$$R_t = \gamma^t r_t + \gamma^{t+1} r_{t+1} + \gamma^{t+2} r_{t+2} \cdots + \gamma^{t+n} r_{t+n}$$

Rollout: max number of steps (can be equal to *episode* duration*)*



The Q Function

The total return R_t is the discounted sum of all future rewards

$$R_t = \gamma^t r_t + \gamma^{t+1} r_{t+1} + \gamma^{t+2} r_{t+2} \cdots + \gamma^{t+n} r_{t+n}$$

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

The Q function describes the *expected* total return that an agent in state s can receive by performing a certain action a. It can be visualized as a look-up table that the agent gradually builds by summing up the observed rewards in several rollouts; for example (*fictitious numbers!*):

Rewards	**Action A**	**Action B**
State A	3	-3
State B	1	0
State C	2	0

Q values	**Action A**	**Action B**
State A	0	0
State B	-2	4
State C	-6	0

Finding the optimal policy

s_a, a?
s_b, a?
s_c, a?
s_d, a?
…

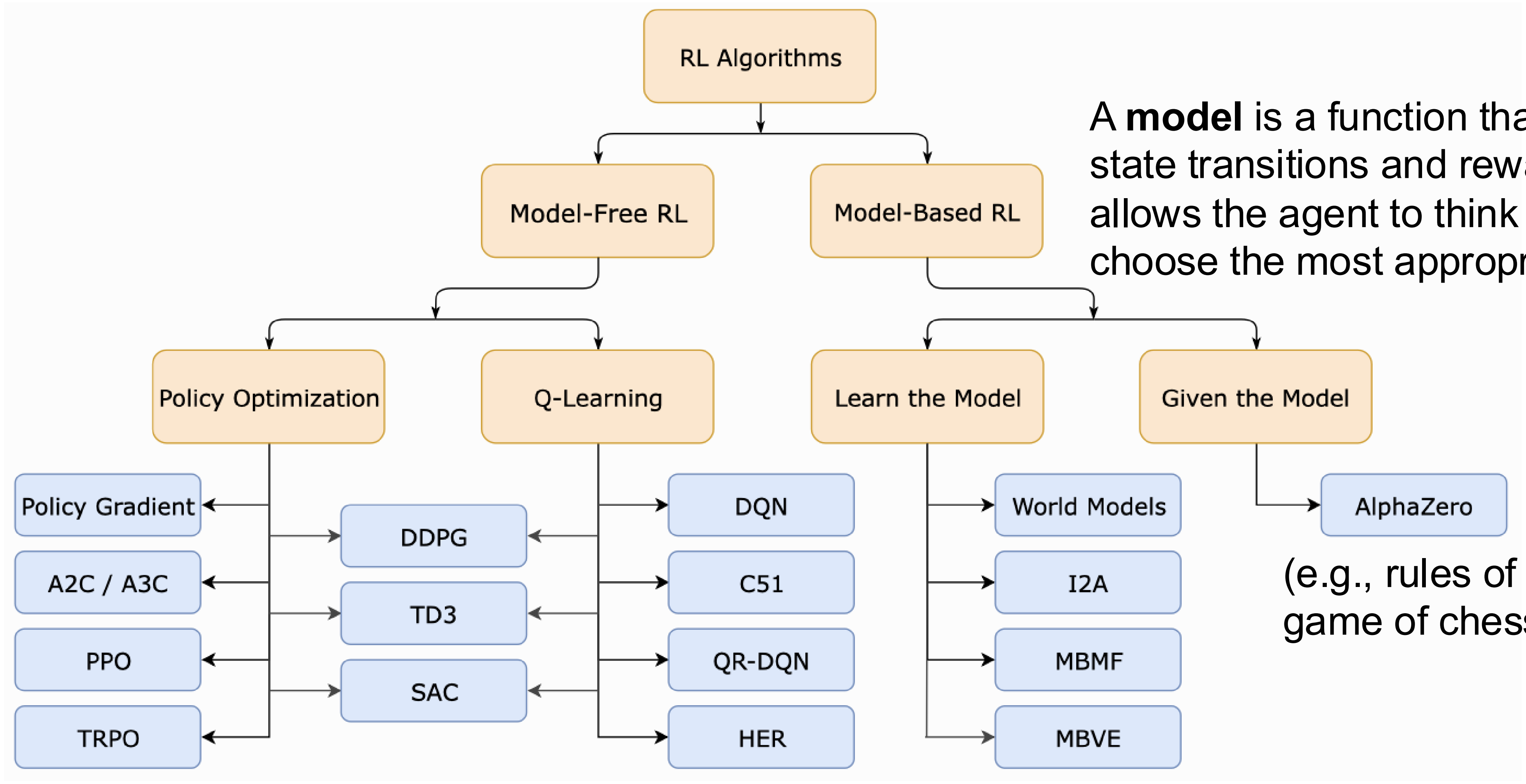
A policy $\pi(s)$ is a strategy to select an action a for a state s

The optimal policy $\pi^*(s)$ is a policy that maximizes the expected total return, which is described by the Q function

If the agent knows the Q function, the optimal policy consists in finding for each state s the best action a over all possible actions that maximize the Q function

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q(s,a)$$

A taxonomy of modern RL algorithms (2018)



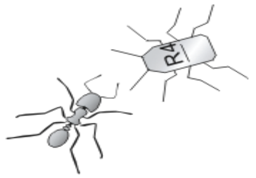
A **model** is a function that predicts state transitions and rewards: it allows the agent to think ahead and choose the most appropriate action

(e.g., rules of the game of chess)

Source: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

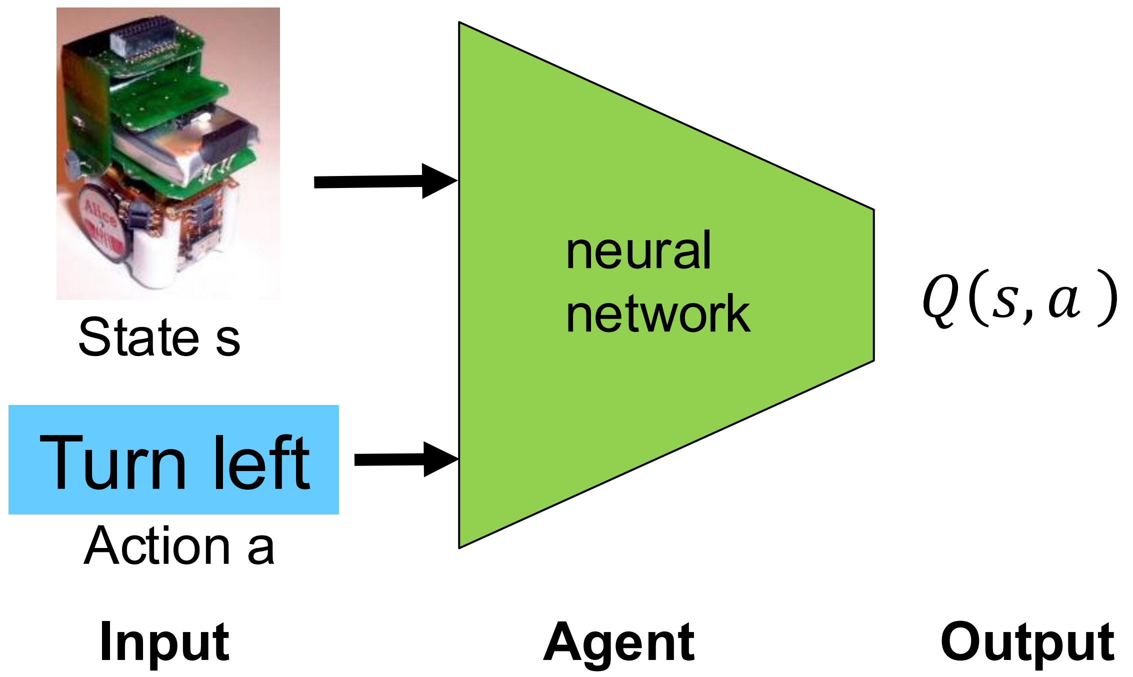
Model-free RL Methods

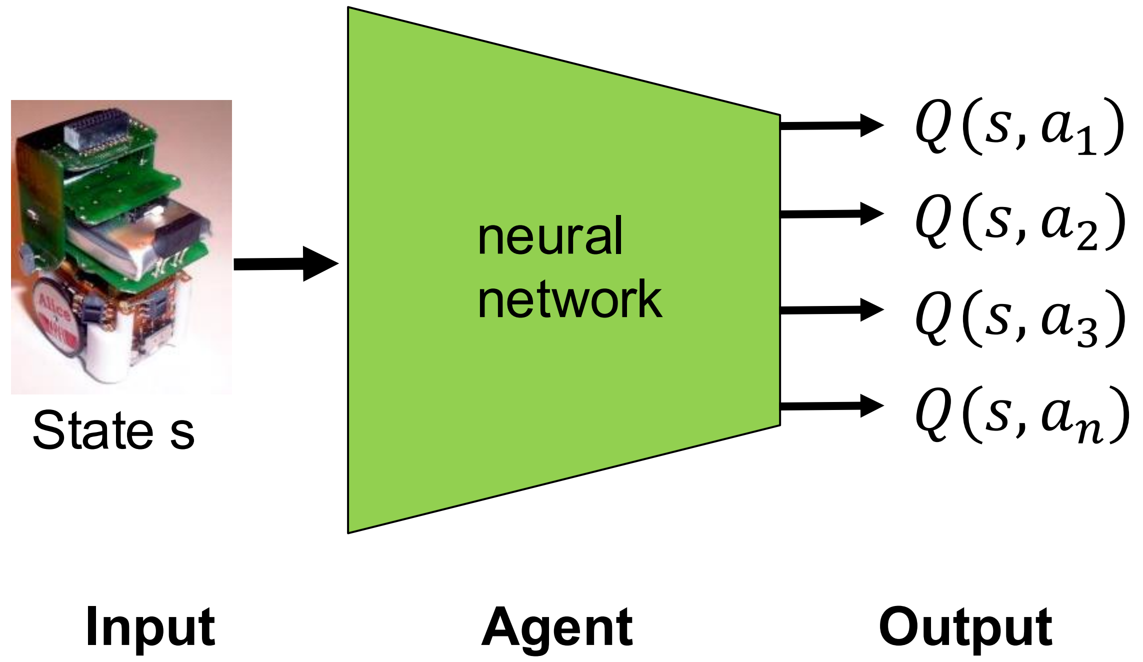
Q-VALUE LEARNING	POLICY LEARNING
Find $Q(s, a)$	Directly find $\pi(s)$
and pick best action $a = \underset{a}{\operatorname{argmax}} Q(s, a)$	and sample (try) action $a \sim \pi(s)$



Deep Q-Networks (DQN)

DQN assumes a discrete action space

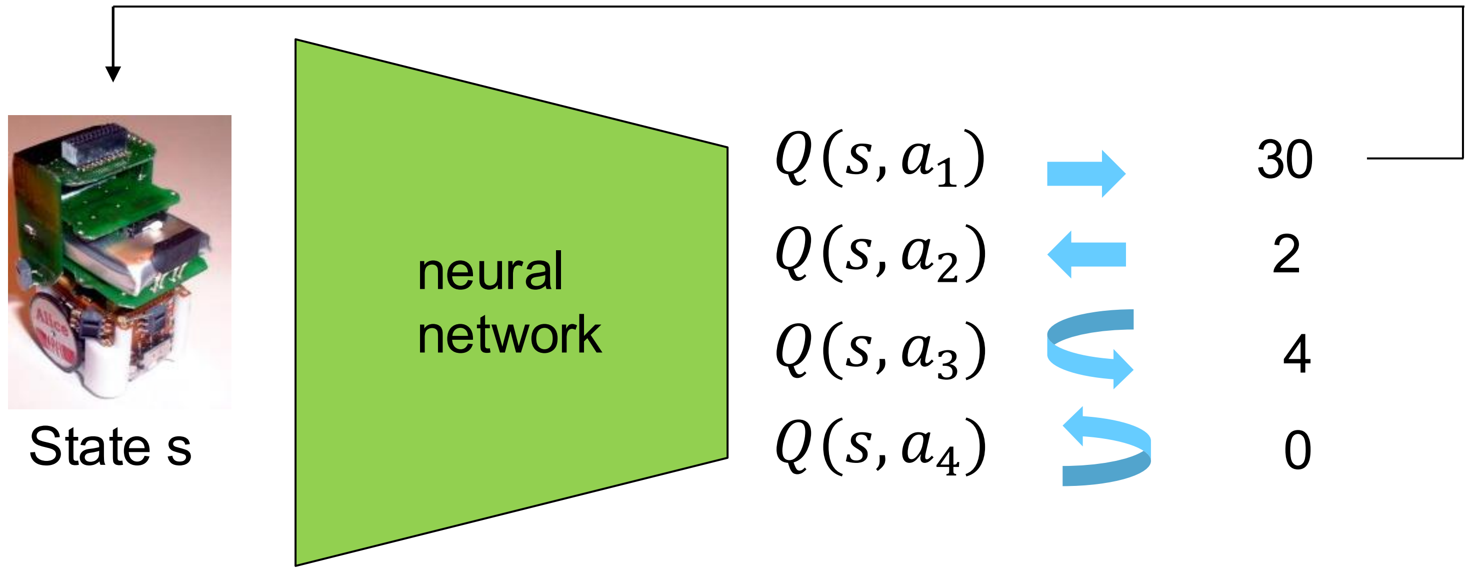




Problem: Q value must be recomputed for all possible actions at input state s

Solution: ask network to compute Q values for all possible actions of input state s

DQN learning



- Initialize random weights
- Toss a biased coin with high probability of selecting action with highest prediction value, otherwise randomly pick an action
- After termination event, compute Q loss and perform gradient descent on weights

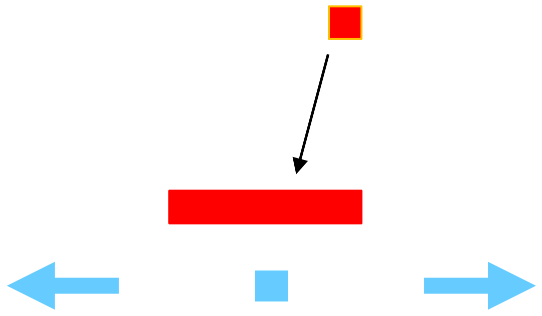
$$\text{Q-loss} = \mathbb{E}\left[\left\|\left(\overbrace{r + \gamma \max_{a'} Q(s',a')}^{\text{Observation}}\right) - \overbrace{Q(s,a)}^{\text{Prediction}}\right\|^{2}\right]$$

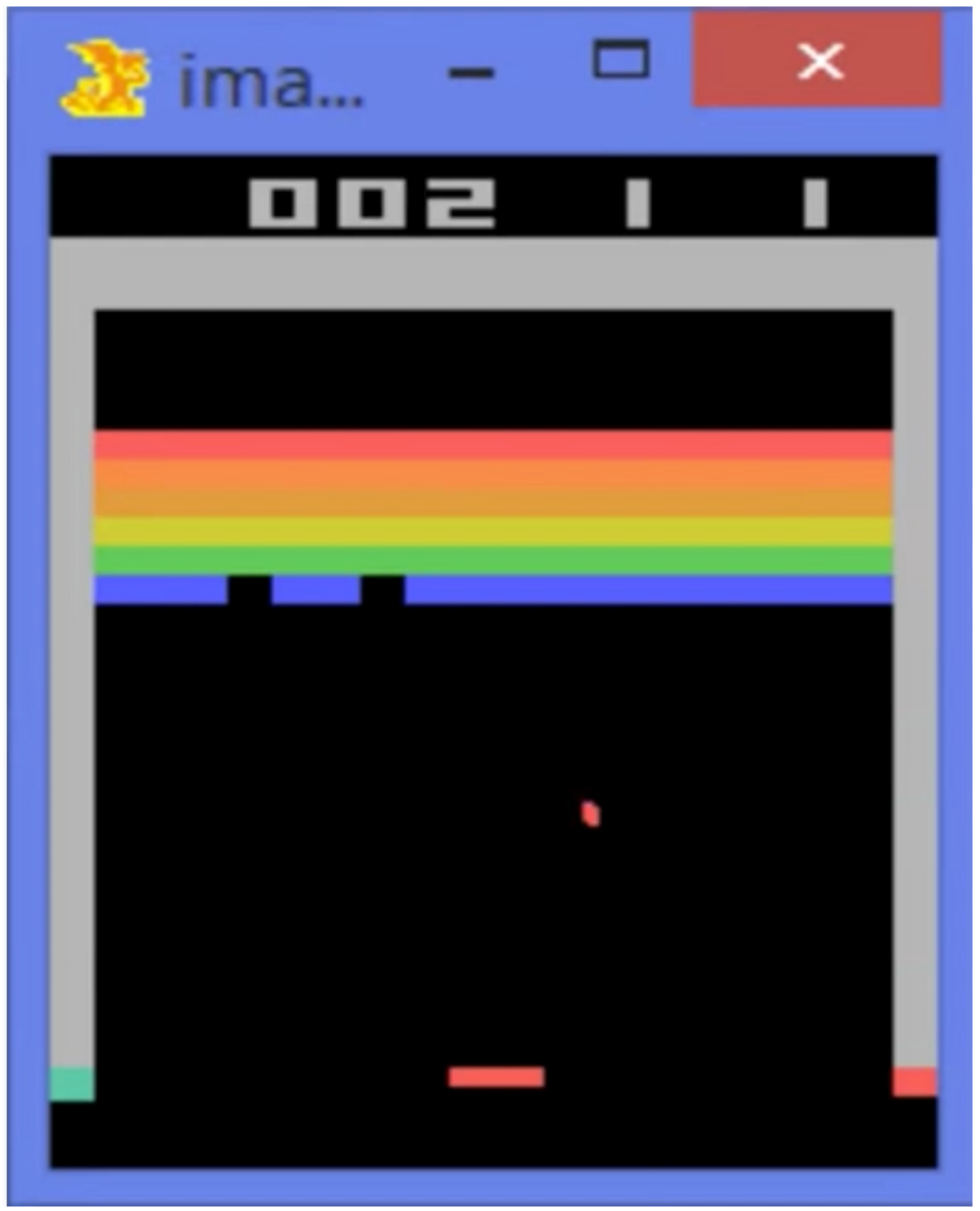
Use back-propagation of error to adapt network weights

DQN learning to play Atari Breakout game

State = screen image

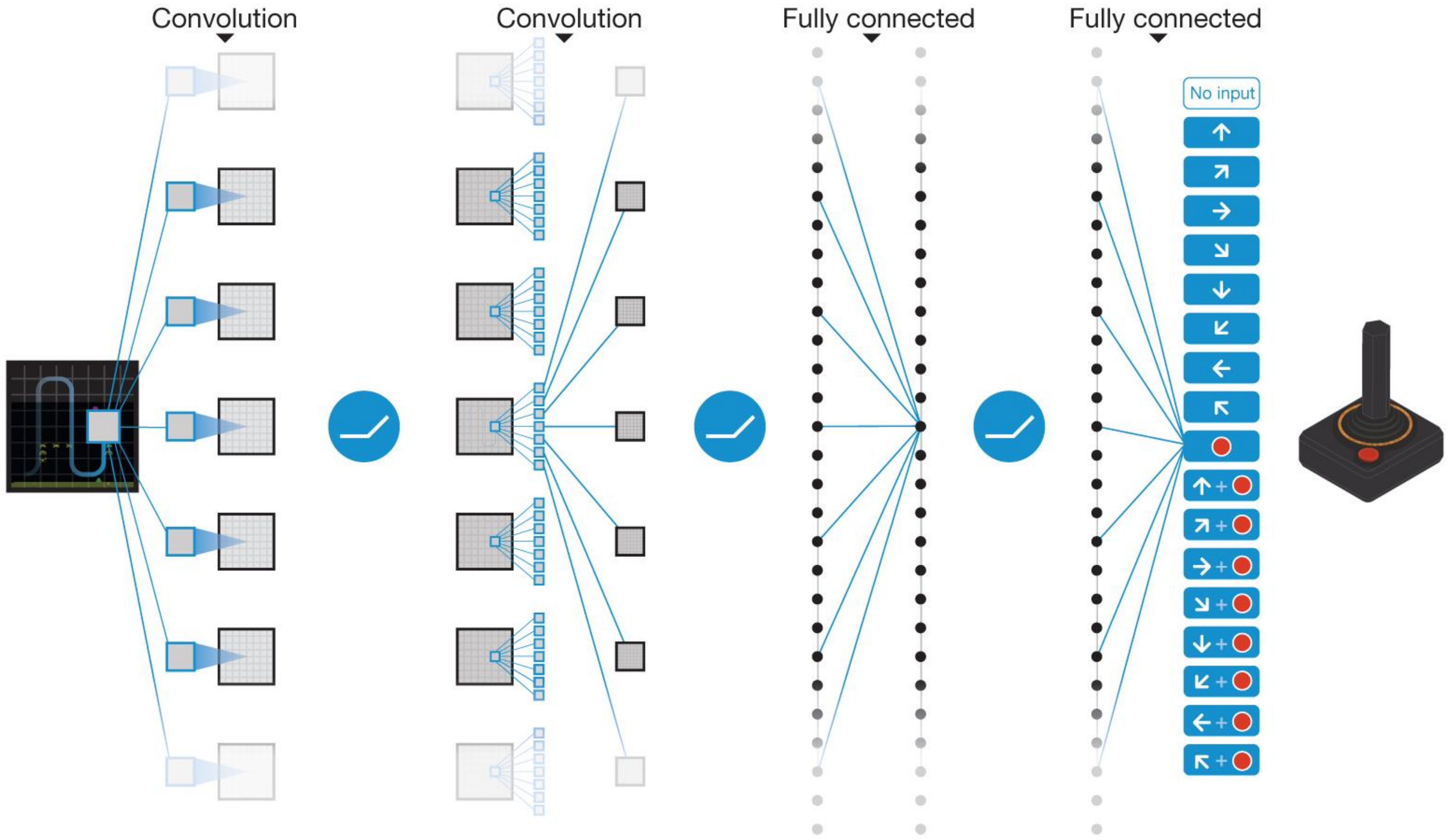
Paddle actions = left, stay, right

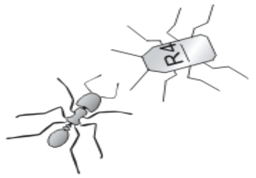


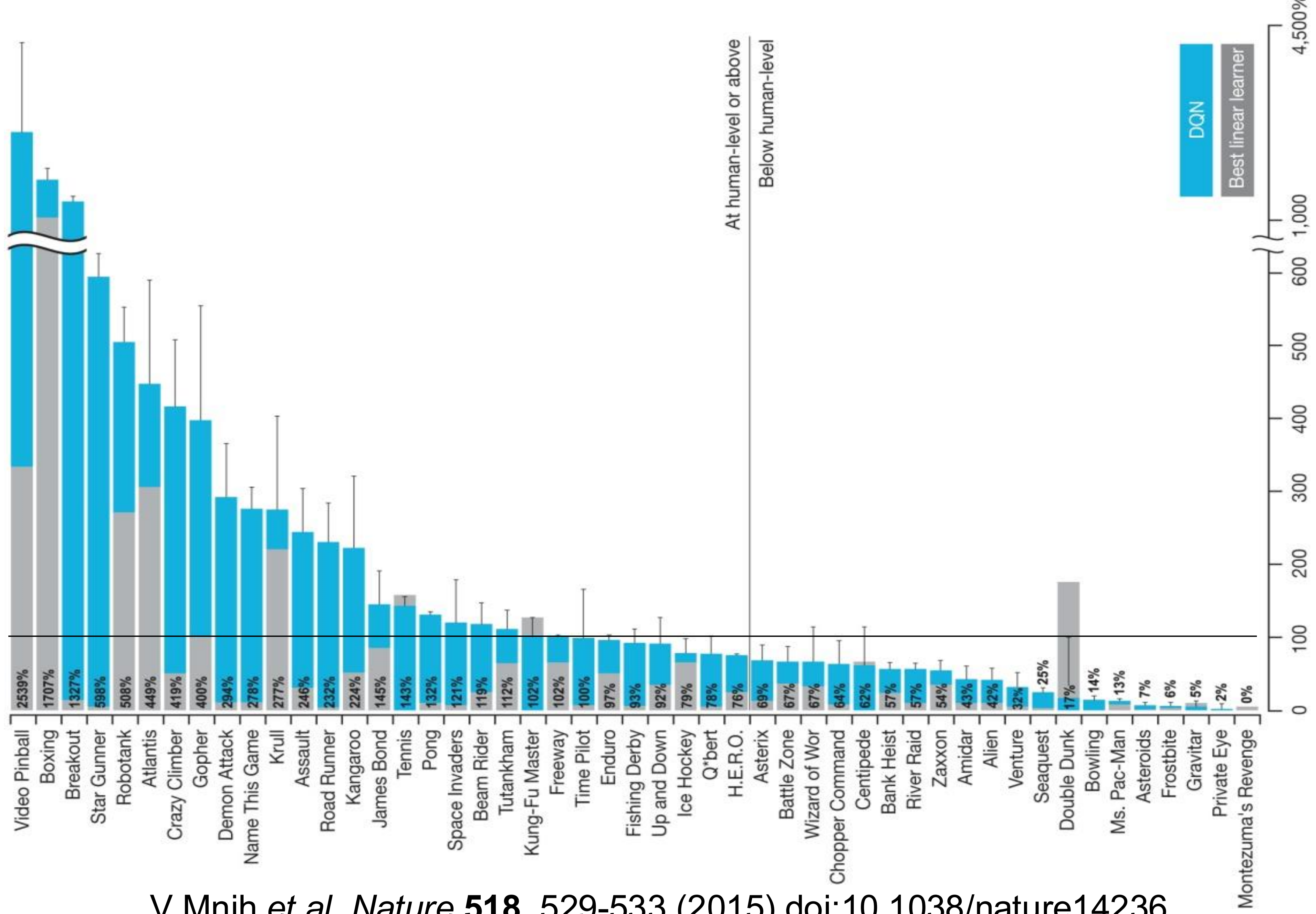


https://www.youtube.com/watch?v=V1eYniJ0Rnk

DQN playing Atari games







Human professional game tester

Q learning: strengths and limitations

It guarantees the possibility of identifying the optimal policy if the Q function is learned

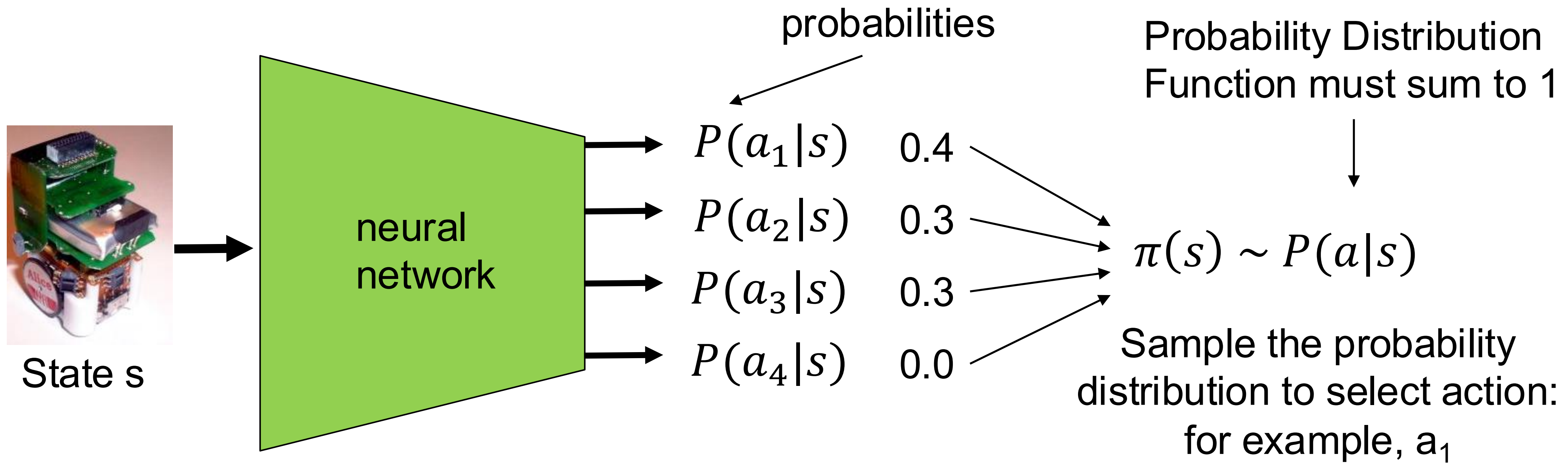
BUT

It requires a discrete action space (turn left, go forward, stay, etc.)

It only works for deterministic situations (it cannot learn stochastic policies)

Policy learning

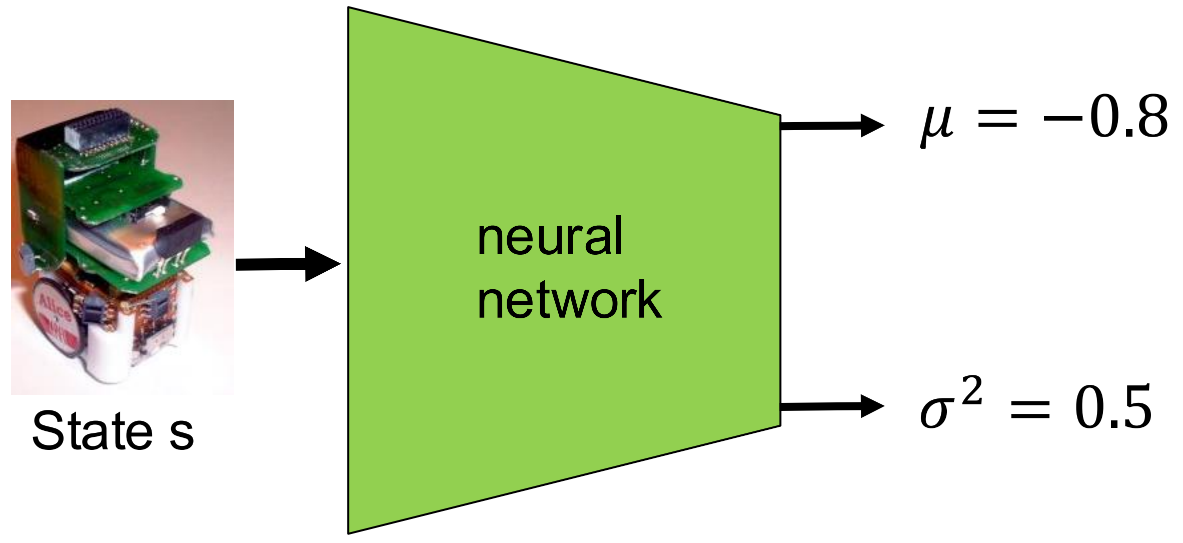
Directly learn the policy $\pi(s)$: <u>discrete action space</u>


probabilities
Probability Distribution Function must sum to 1
neural network
State s
$P(a_1|s)$ 0.4
$P(a_2|s)$ 0.3
$P(a_3|s)$ 0.3
$P(a_4|s)$ 0.0
$\pi(s) \sim P(a|s)$
Sample the probability distribution to select action: for example, a_1

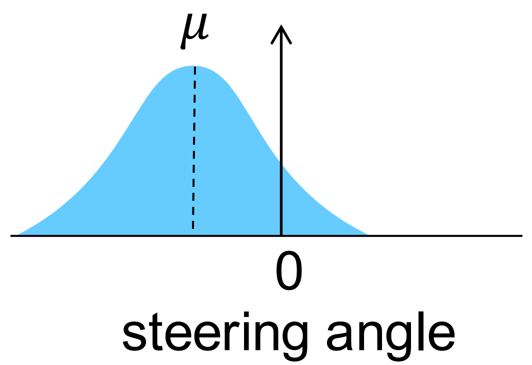
Input **Agent** **Output**

Policy learning

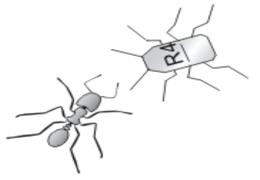
Directly learn the policy $\pi(s)$: <u>continuous action space</u>



$P(a|s) = \mathcal{N}(\mu, \sigma^2)$

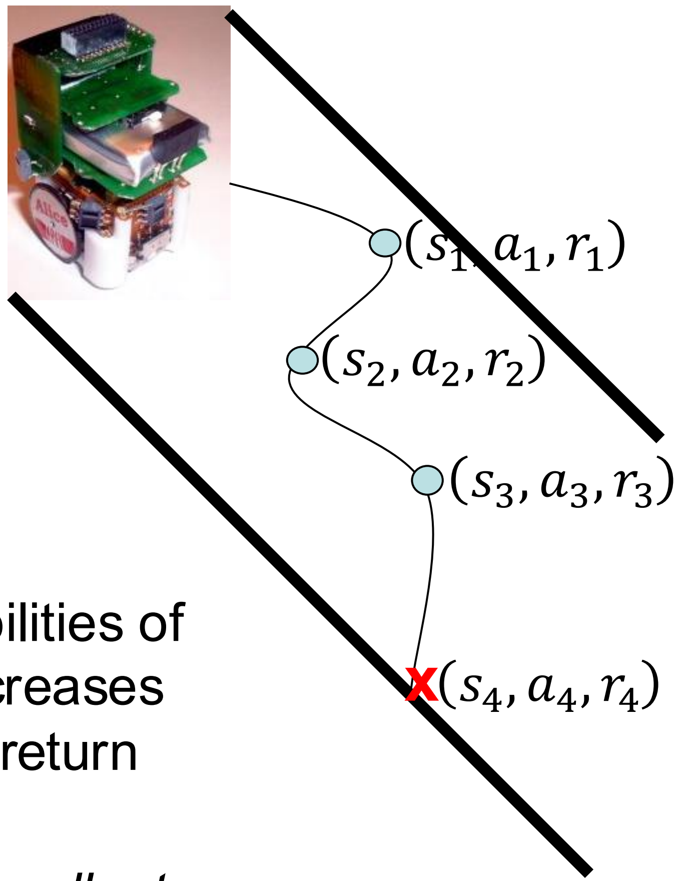


Input **Agent** **Output**



Policy Gradient Learning

1. Initialize weights of the agent
2. Run the agent (*policy*) until termination (*rollout*)
3. At each time step of the rollout, record the triplet (s_t, a_t, r_t)
4. Increase probability of actions that led to high reward
5. Decrease probability of actions that led to low reward



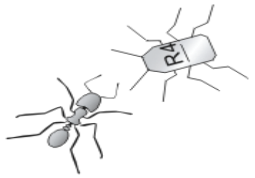
$$\boxed{loss = -\log P(a_t|s_t)\, R_t}$$

The loss function increases the probabilities of actions with higher total return and decreases probabilities of actions with lower total return

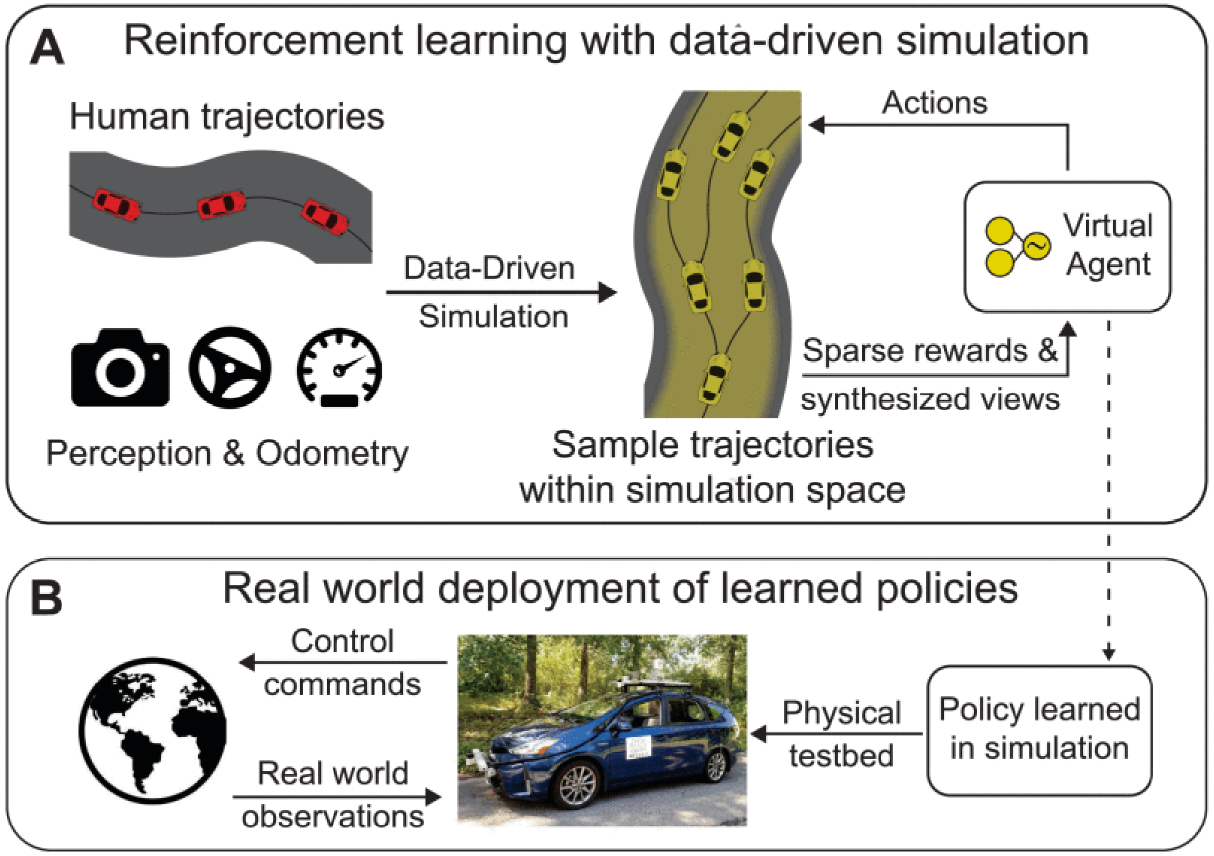
$$\Delta w = -\nabla loss$$
$$\Delta w = \nabla \log P(a_t|s_t)\, R_t$$

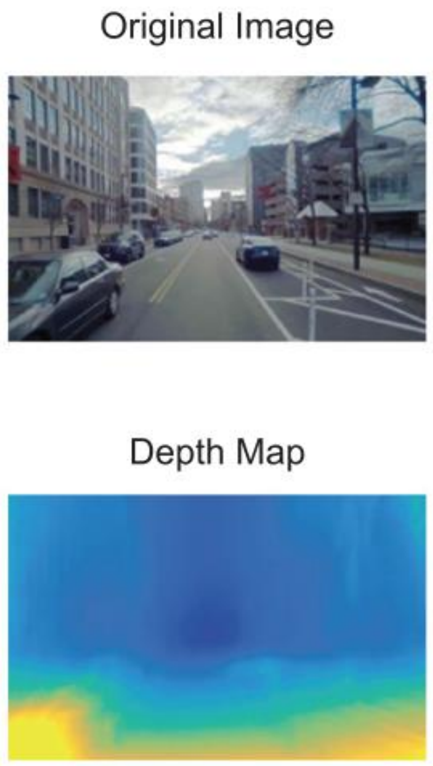
Weight change is performed after each *rollout*

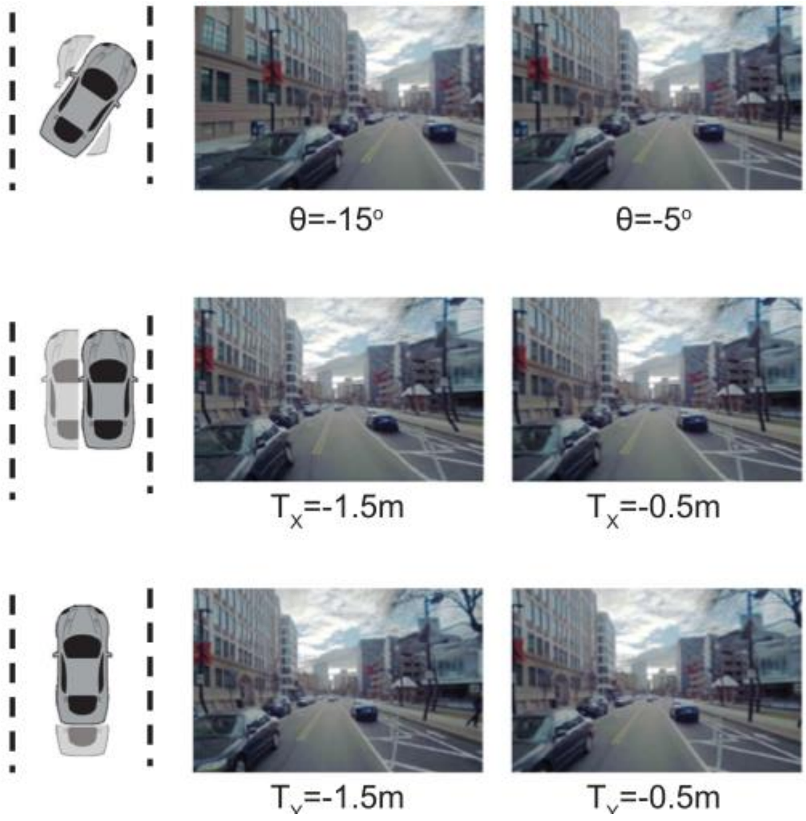
For full derivation; https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html



Autonomous driving by Policy Gradient Learning


A
Reinforcement learning with data-driven simulation
Human trajectories
Data-Driven Simulation
Perception & Odometry
Actions
Virtual Agent
Sparse rewards & synthesized views
Sample trajectories within simulation space
B
Real world deployment of learned policies
Control commands
Real world observations
Physical testbed
Policy learned in simulation





A. Amini *et al*., Learning Robust Control Policies for End-to-End Autonomous Driving From Data-Driven Simulation, (2020) *IEEE Robotics and Automation Letters*, 5(2), 1143-1150

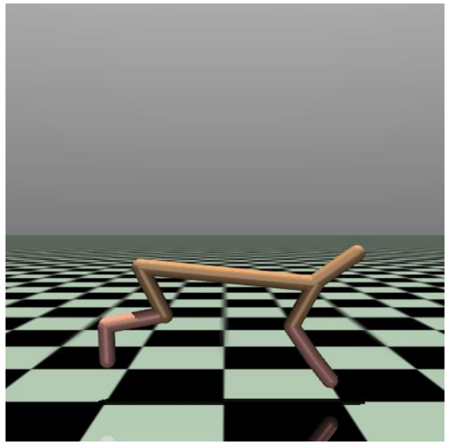
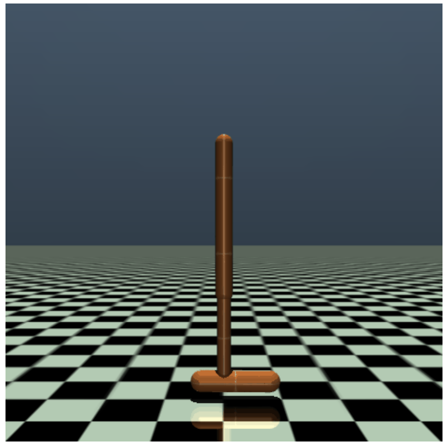
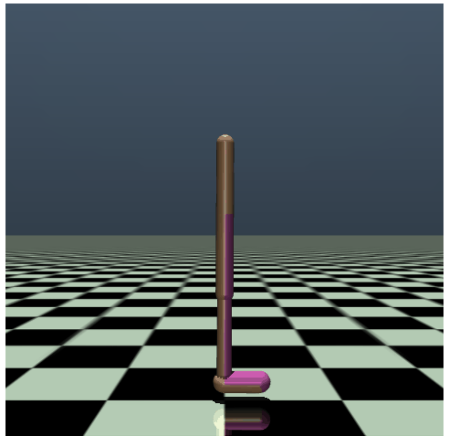
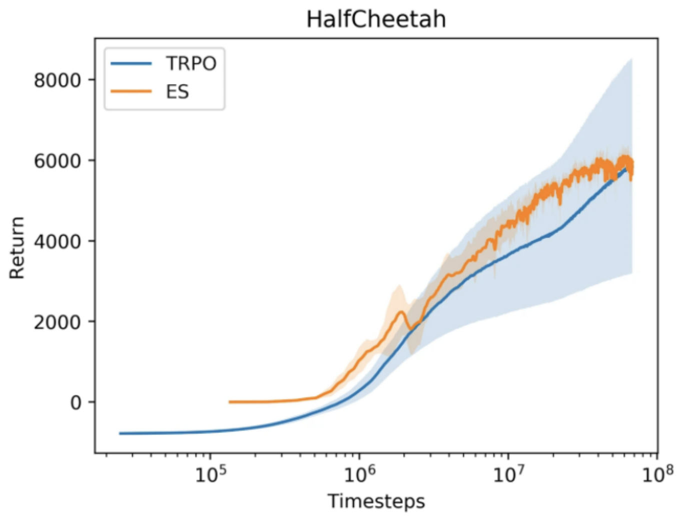
Contributions

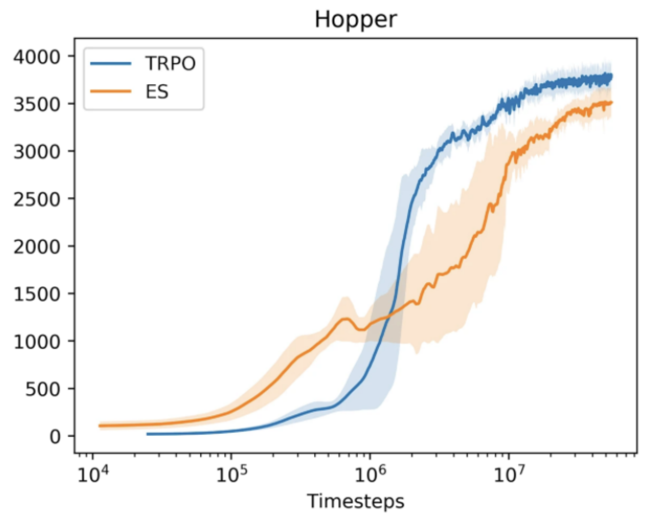
Our paper makes the following contributions

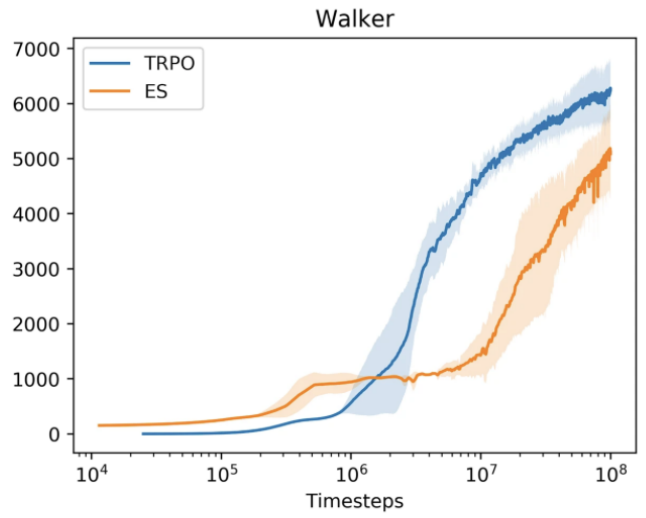
SUBSCRIBE

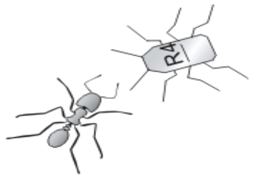
Reinforcement Learning vs Evolutionary Strategies

Both algorithms achieve similar results in similar step numbers for same neural network







https://openai.com/index/evolution-strategies/

RL vs ES: different smoothing of problem space

Both RL and ES search for a function *F*= $\pi(s;\theta)$ of the policy parameters θ that maximizes the return *R* over a sequence of actions **a**(θ)

$$F(\theta) = R(\mathbf{a}(\theta))$$

In hard decision problems the space can be discontinous, which makes gradient estimation difficult. Both RL and ES smooth the problem space by adding noise to the estimation of the policy, but:

- RL algorithms (such as Q-learning and Policy Gradient) add noise ε to the action space **a** of the policy $\pi(s;\theta)$

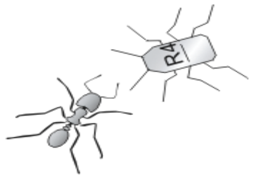
$$\mathbf{a}(\epsilon, \theta)$$

stochastic

- Evolution Strategies instead add noise ξ to the parameter space $\boldsymbol{\theta}$ of the policy $\pi(s;\tilde{\theta})$

$$\tilde{\theta} = \theta + \xi \qquad \mathbf{a}(\xi, \theta)$$

deterministic



https://openai.com/index/evolution-strategies/

RL vs ES: different parameter adjustment methods

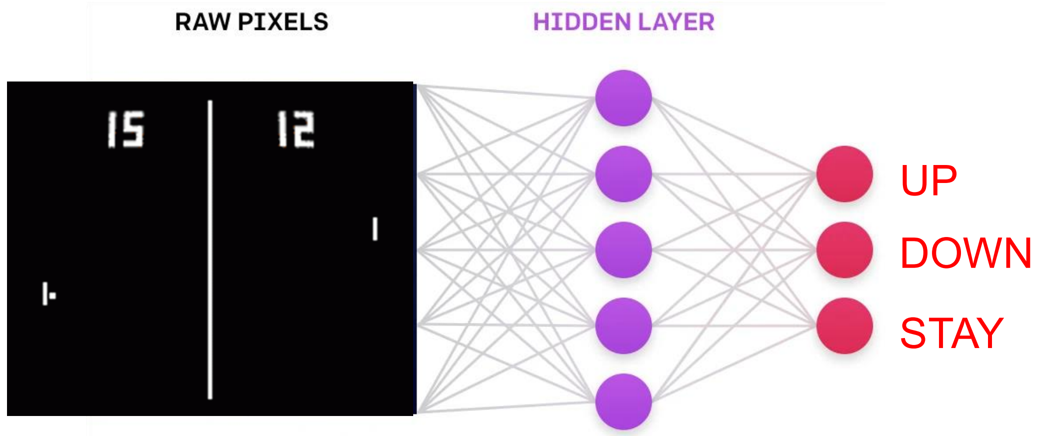
Both RL and ES move the policy parameters in the direction of the gradient, but:

- RL algorithms use Gradient Descent (back propagation) to minimize the loss of the *discounted* reward function over the parameter space θ

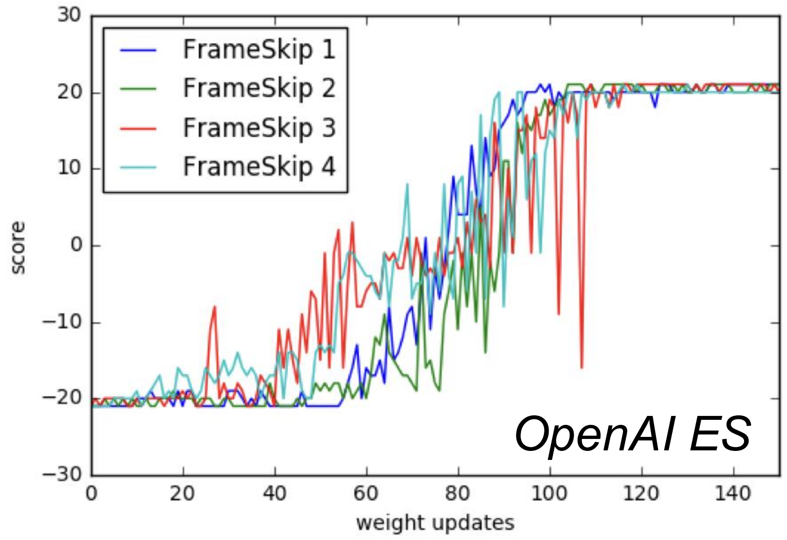
$\longrightarrow$ θ must be differentiable; requires FrameSkip to mitigate performance degradation in long rollouts

- Evolution Strategies use Finite Difference methods to estimate gradient from few perturbations of the parameter space θ and does not require a discounted reward function

$\longrightarrow$ θ must not be differentiable (e.g., binary neural networks); insensitive to rollout length

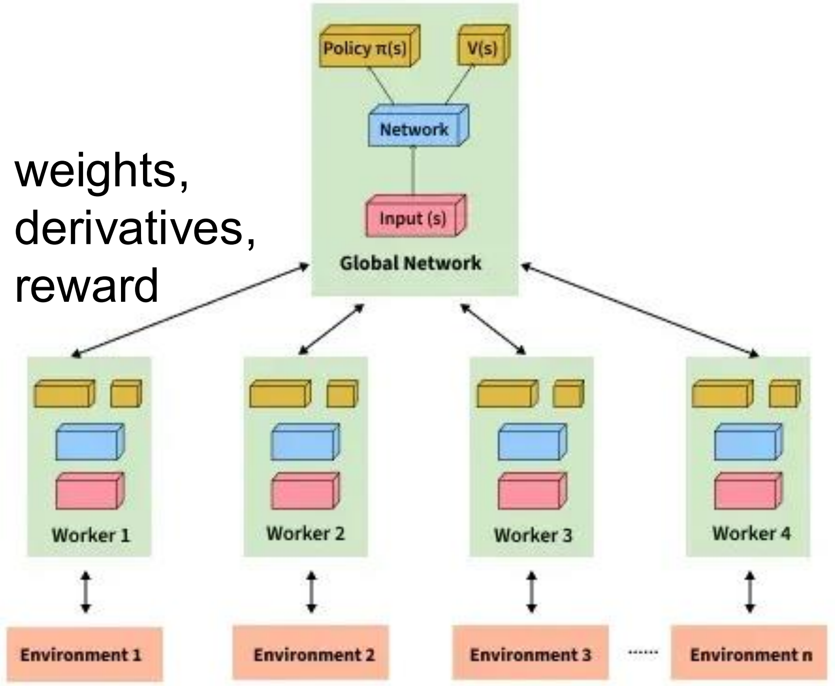


Atari Pong game



https://openai.com/index/evolution-strategies/

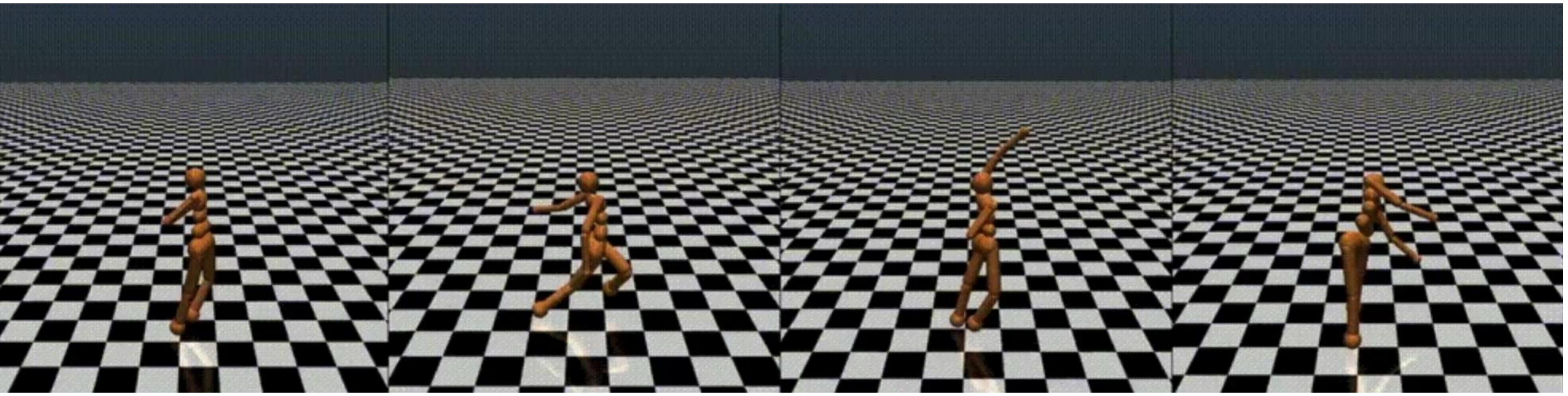
RL vs ES: computational efficiency on multiple cores



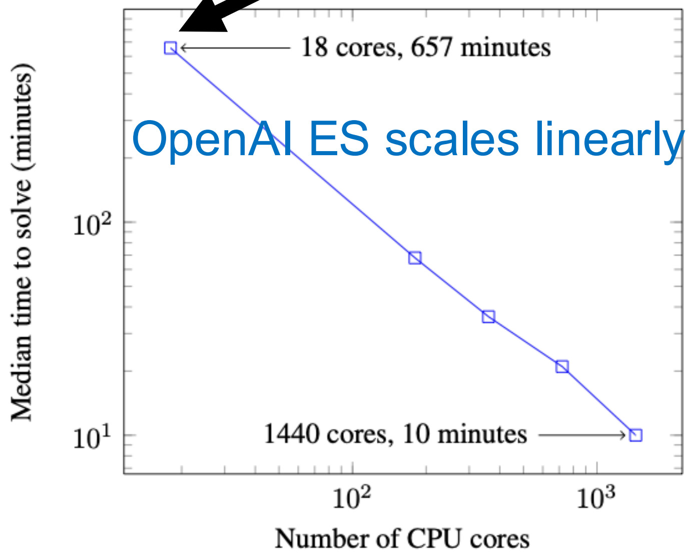
weights, derivatives, reward

In terms of wall clock time, ES benefits more from parallelization:

- In RL, CPU workers must communicate rewards, weights, derivatives after each rollout
- In ES, individuals run independently on different CPUs and communicate only fitness score (one number)



RL typical wall clock time for Mujoco humanoid

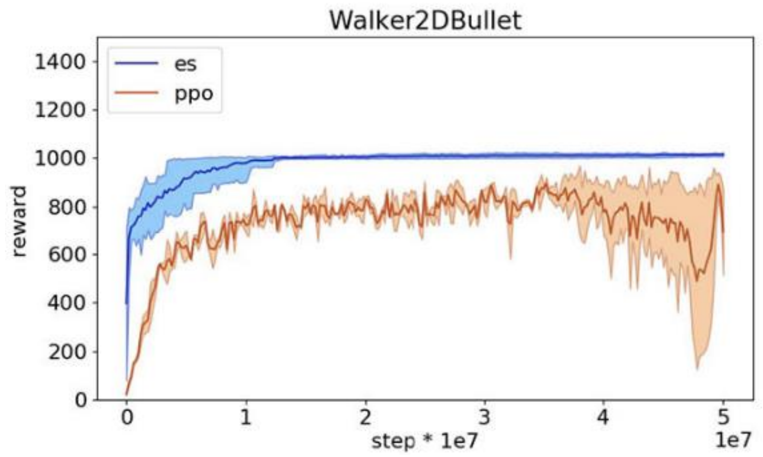


Salimans et al, (2017) Evolution Strategies as a Scalable Alternative to Reinforcement Learning

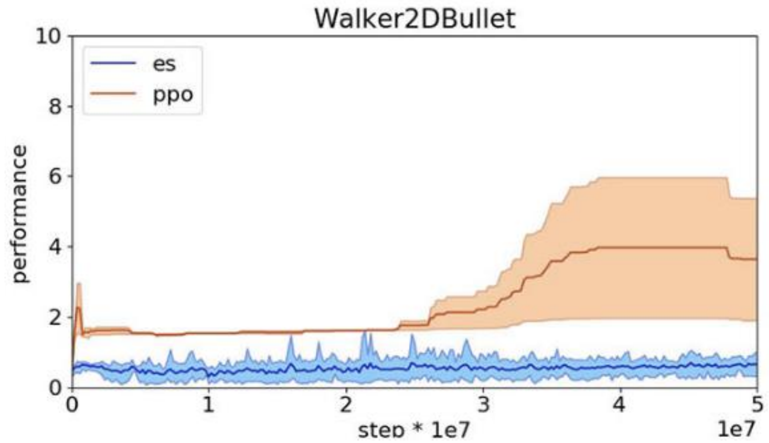
RL and ES may need different reward/fitness functions

- **RL**: Probabilistic action output requires several cost components to limit low-score actions
- **ES**: Deterministic action output can achieve comparably high score by precisley maximizing only selected components of a reward function

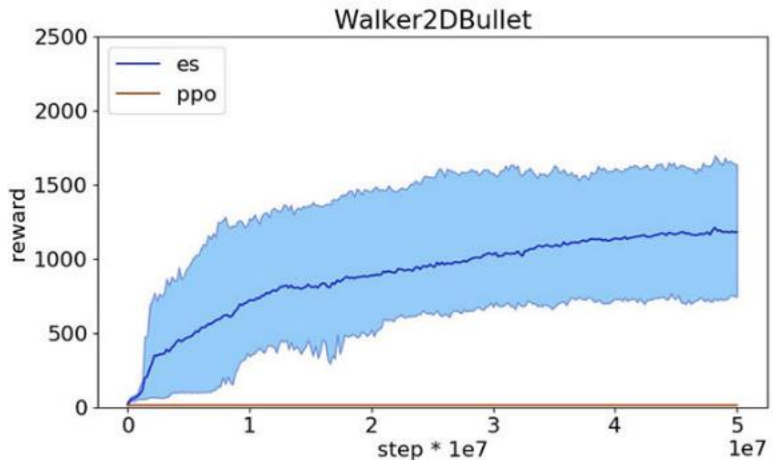
RL Reward
+ Progress
+ Stay upright
- Energy
- Stall
- Joint limit
- Not upright



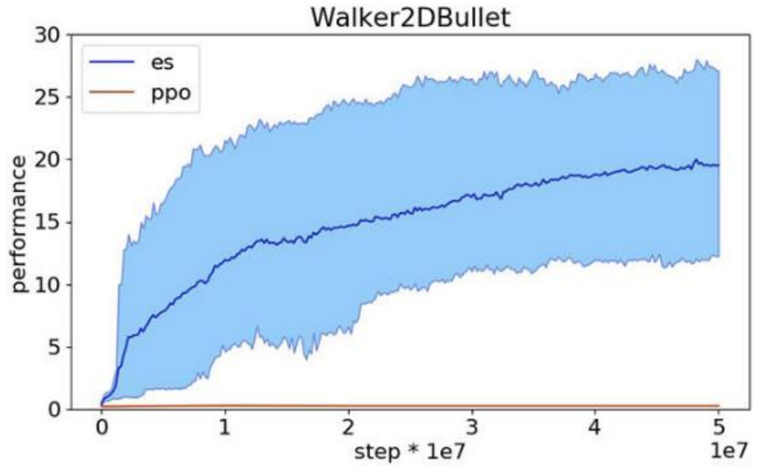
Performance
Progress (meters)



ES Reward
+ Progress



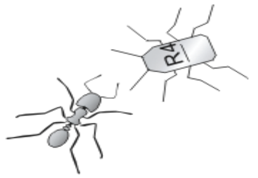
Performance
Progress (meters)



Pagliuca et al, (2020) Efficacy of Modern Neuro-Evolutionary Strategies for Continuous Control Optimization

Reinforcement Learning - Evolutionary Strategies

	Reinforcement Learning		**Evolutionary Computation**
-	Definition of Reinforcement Policy	-	Definition of Fitness Function
-	Gradient descent	+	No need of gradient
-	Lots of hyperparameters	+	Comparatively less hyperparameters
+	Better use of information over rollouts	-	Information is lost between individuals
-	Challenging in long rollouts without reward	+	Independent of rollout length
-	Operates only on weights of neural network	+	Operates on weights, learning, morphologies
-	Requires many rollouts	-	Requires many rollouts
+	Has strong mathematical foundations	-	Some algorithms are rather empirical
-	Harder to parallelize: slow (clock speed)	+	Highly parallel: fast (clock speed)



See also https://openai.com/blog/evolution-strategies/

Checkpoints

- What is the Temporal Credit Assignment problem in Reinforcement Learning
- What is the Total Return?
- What is the Q function?
- How do the Q value of a state-action pair differ from immediate reward of state-action pair?
- Given a Q-function, how do you choose the action for a given state?
- What are the outputs of a Deep Q-Learning network?
- What are the limitations of Q-learning algorithms?
- What are the outputs of a Policy learning network for discrete action space?
- What are the outputs of a Policy learning network for continuous action space?
- How do RL and ES algorithms adjust parameters to follow the gradient?
- Why do RL and ES may require different reward and fitness functions?