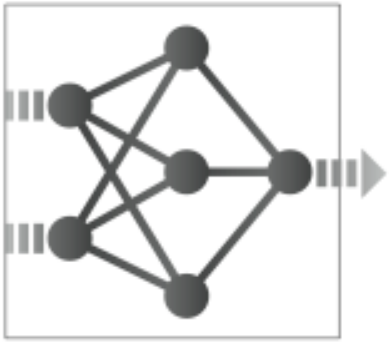
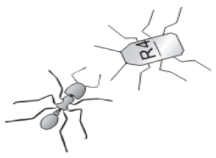
Reinforcement Learning with Neural Networks





What you will learn in this class

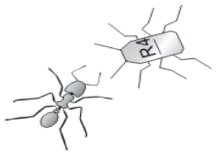
- The Reinforcement Learning Framework
- Reward and Total Return
- The state-action value function (Q function)
- Value Learning
  - Deep Q Learning
- Policy Learning
  - Policy Gradient Learning

Reinforcement learning

Input: *state* (sensory information, position, energy, e.g.), *action* (forward, rotate, turn, e.g.)

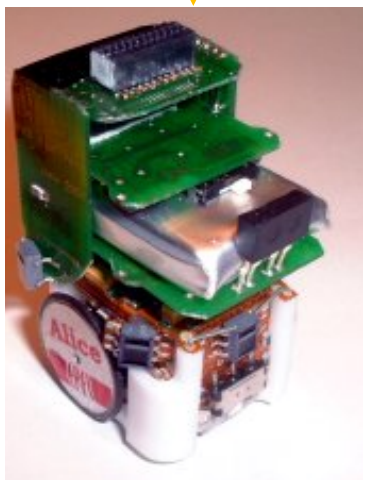
Reward: *r* (collected dirt, e.g.)

Goal: learn *behavior* (policy) that maximizes the total future rewards



Reinforcement learning framework

AGENT



State s_{t+1}

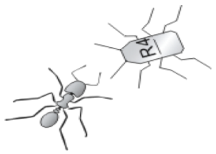
Reward r_t
can be positive, negative ,or absent

ENVIRONMENT

Action a_t

The agent wants to choose actions (the *policy*) that maximize the total future reward (the *Total Return*)

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



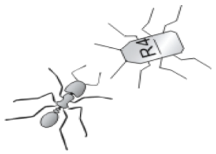
Reward discount and rollouts

Should all rewards, present and future, have the same weight?
The *discount* factor γ is used to give more importance to present rewards than to remote future rewards

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

Rollout: the finite number of steps *n* during which the agent interacts with the environment

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



The Q Function

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

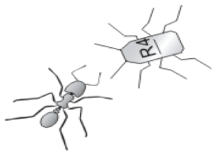
The total return R_t is the discounted sum of all future rewards

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

The Q function describes the *expected* total return that an agent in state s can receive by performing a certain action a. It can also be seen as a look-up table that the agent gradually builds through 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 captured 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

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

Reinforcement Learning Methods

VALUE LEARNING

Find

$$Q(s, a)$$

and pick best action

$$a = \underset{a}{\text{argmax}} Q(s, a)$$

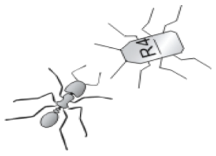
POLICY LEARNING

Directly find

$$\pi(s)$$

and sample action

$$a \sim \pi(s)$$



Deep Q Networks

State s

Turn left

Action a

neural network

$Q(s, a)$

Input Agent Output

State s

neural network

$Q(s, a_1)$

$Q(s, a_2)$

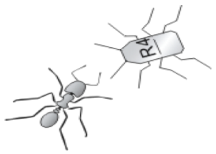
$Q(s, a_3)$

$Q(s, a_n)$

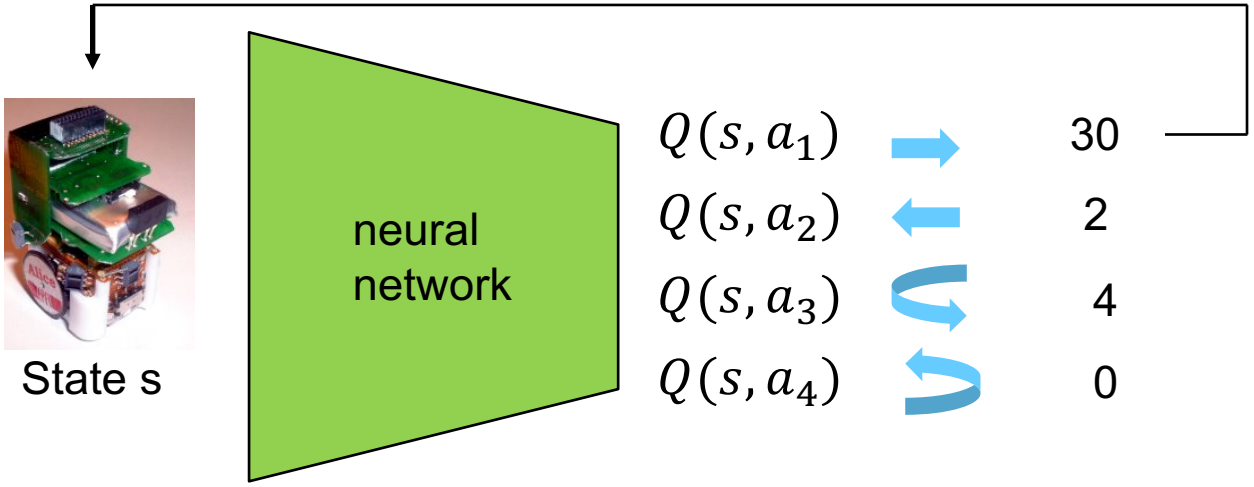
Input Agent Output

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



target prediction

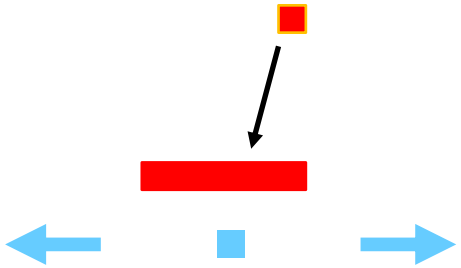
$$\text{Q-loss} = \mathbb{E}\left[\left\|\left(r + \gamma \max_{a'} Q(s', a')\right) - Q(s, a)\right\|^2\right]$$

Use back-propagation of error to adapt network weights

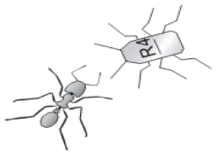
DQN learning to play Atari Breakout game

State = screen image

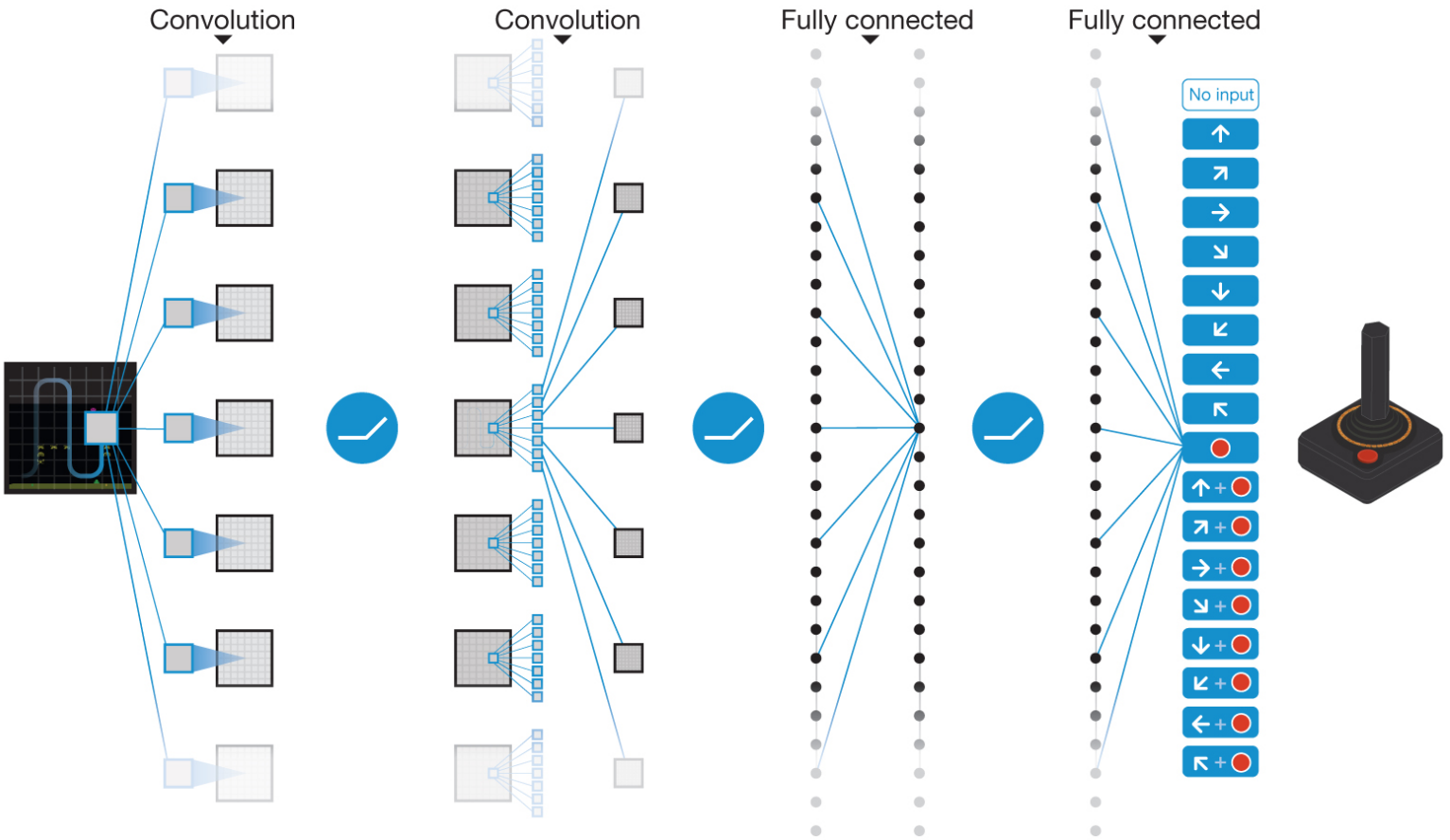
Paddle actions = left, stay, right

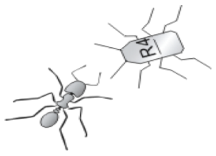


https://www.youtube.com/watch?v=V1eYniJ0Rnk

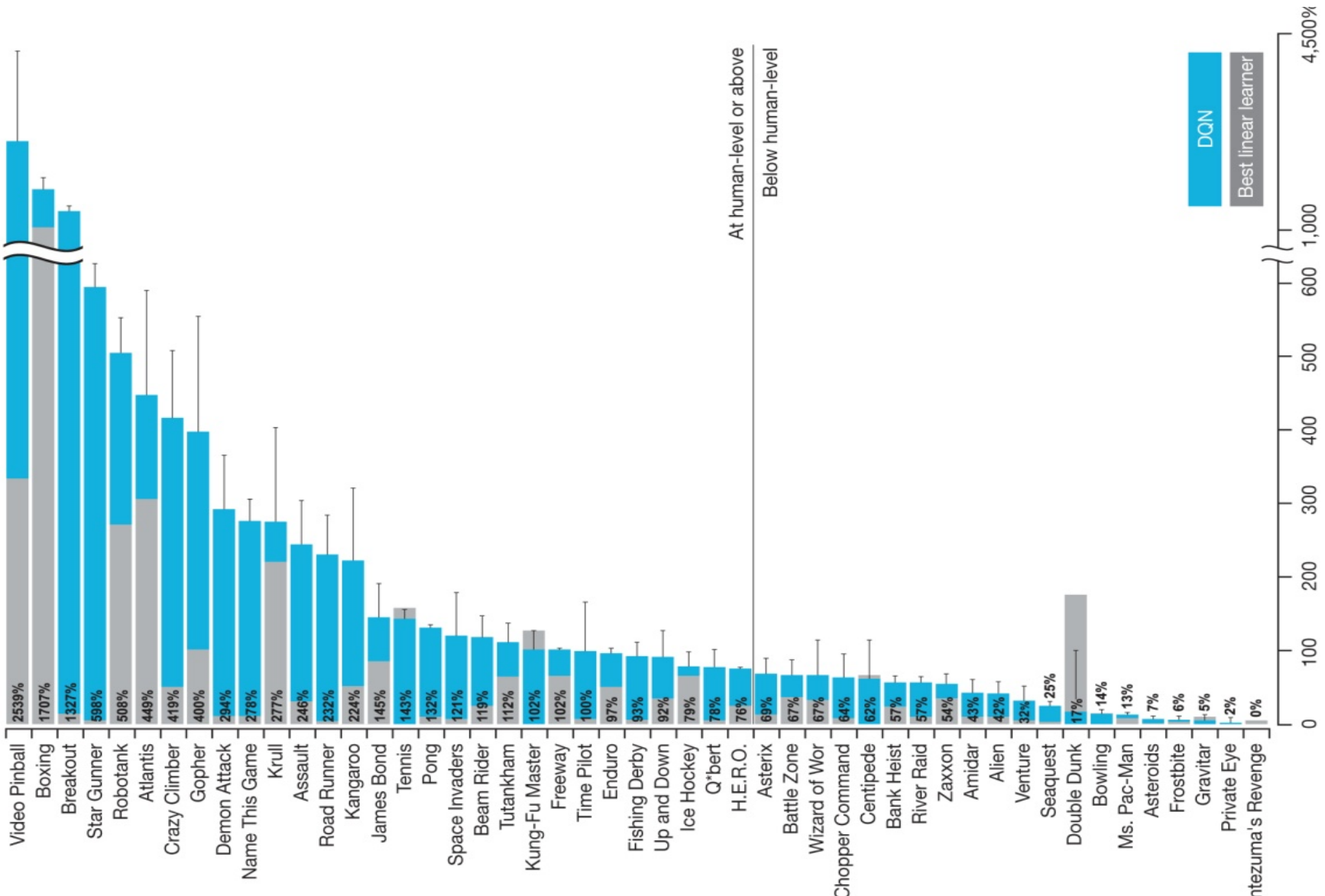


DQN playing Atari games





V Mnih *et al. Nature* **518**, 529-533 (2015) doi:10.1038/nature14236



V Mnih *et al. Nature* **518**, 529-533 (2015) doi:10.1038/nature14236

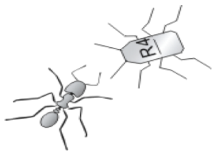
Q learning: strengths and limitations

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

BUT

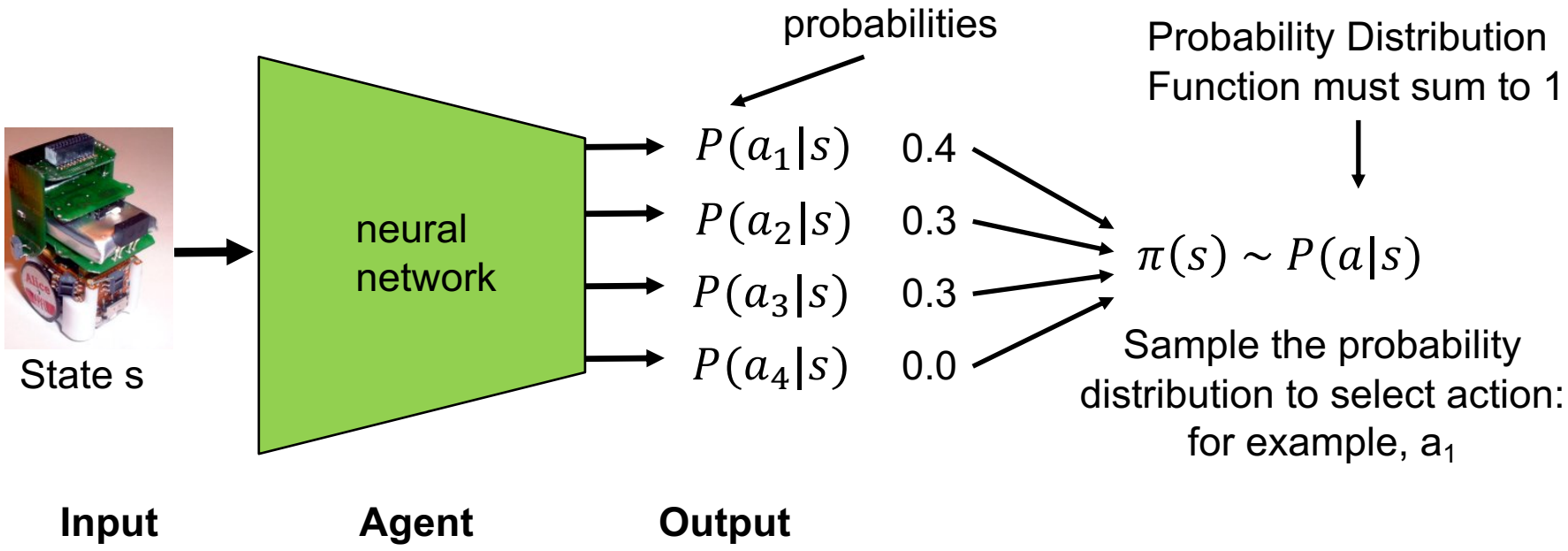
It requires a discrete action space (turn left, go forward, stay, etc.)

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



Policy learning

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



Policy learning

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

State s

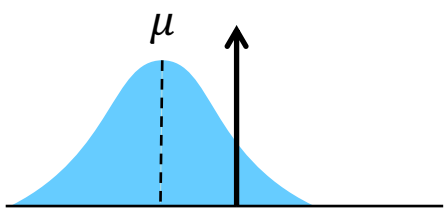
neural network

$\mu = -0.8$

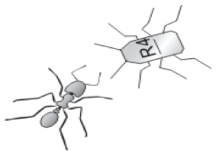
$\sigma^2 = 0.5$

Input **Agent** **Output**

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

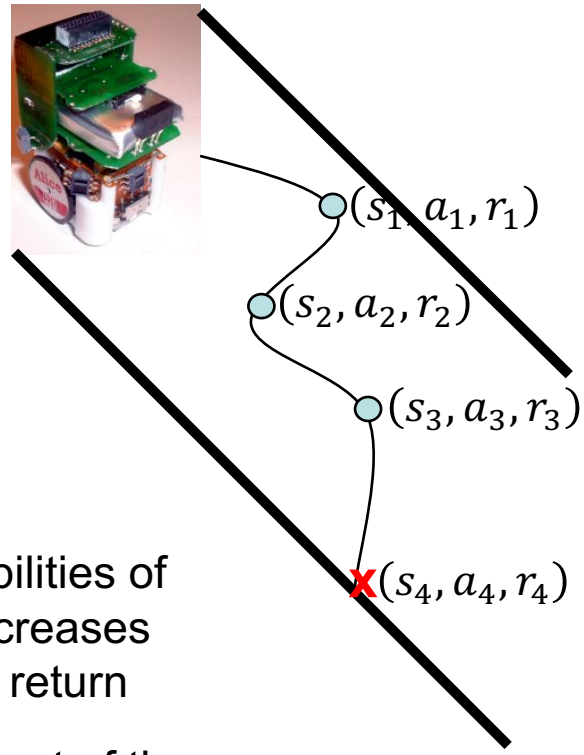


steering angle



Training on policy gradient

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



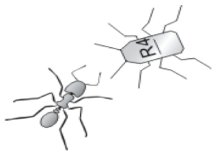
$$loss = -\log P(a_t|s_t)\, R_t$$

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

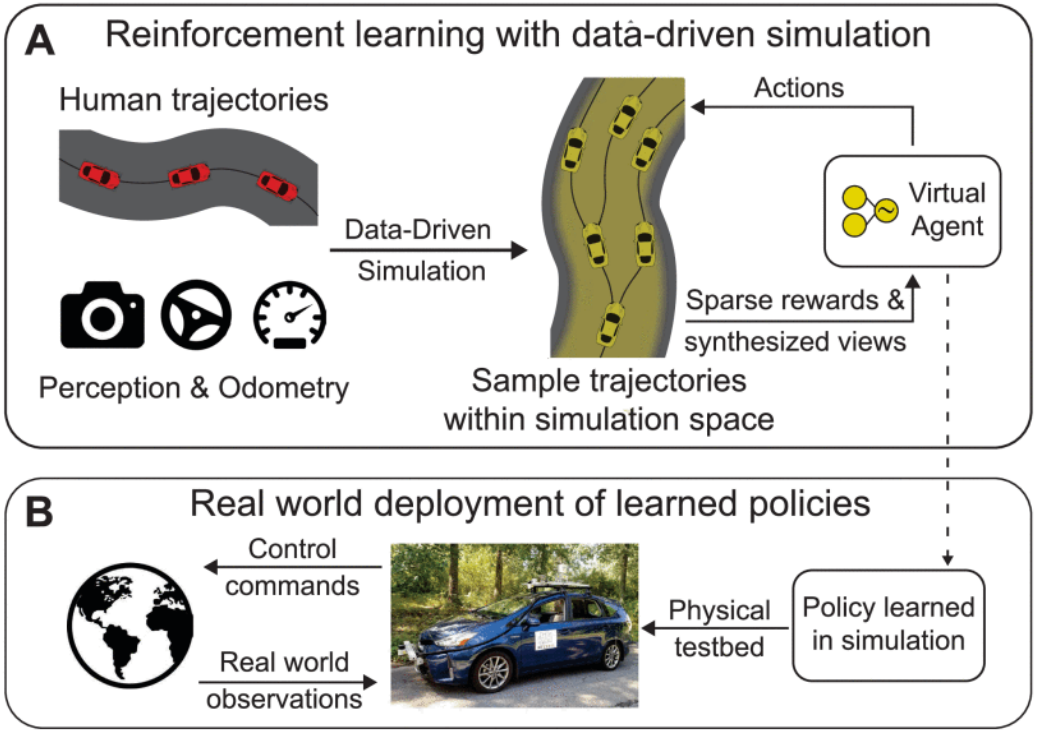
$$\Delta w = -\nabla loss$$
$$\Delta w = \nabla \log P(a_t|s_t)\, R_t$$

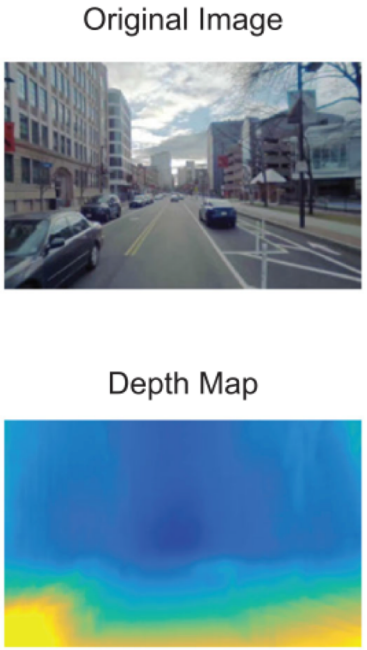
The weight change is the gradient ascent of the loss function with respect to the agent's weights

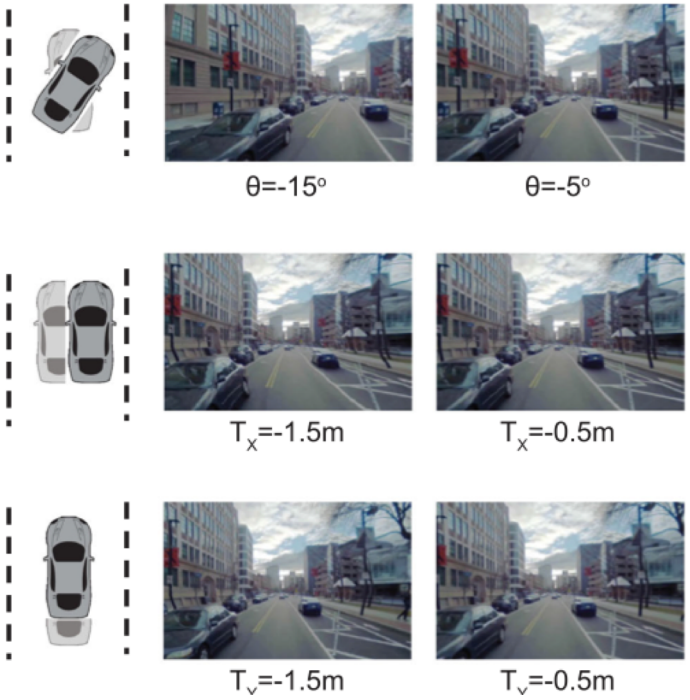
An alternative method that does not use gradient ascent is evolutionary computation



Autonomous driving by Policy Gradient Learning







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