香港中文大學
The Chinese University of Hong Kong

Introduction to Reinformance Learning

Hongsheng Li
Assistant Professor
Department of Electronic Engineering
The Chinese University of Hong Kong

Characteristics of Reinforcement Learning

- What makes reinforcement learning different from other machine learning paradigms?
  - There is no supervisor, only a reward signal
  - Feedback is delayed, not instantaneous
  - Time really matters (sequential, non i.i.d data)
  - Agent’s actions affect the subsequent data it receives

Examples of Reinforcement Learning

- Fly stunt manoeuvres in a helicopter
- Manage an investment portfolio
- Control a power station
- Make a humanoid robot walk
- Play many different Atari games better than humans

Rewards

- A reward r_t is a scalar feedback signal
- Indicates how well agent is doing at step t
- The agent's job is to maximize cumulative reward
- Reinforcement learning is based on the reward hypothesis

- All goals can be described by the maximization of expected cumulative reward

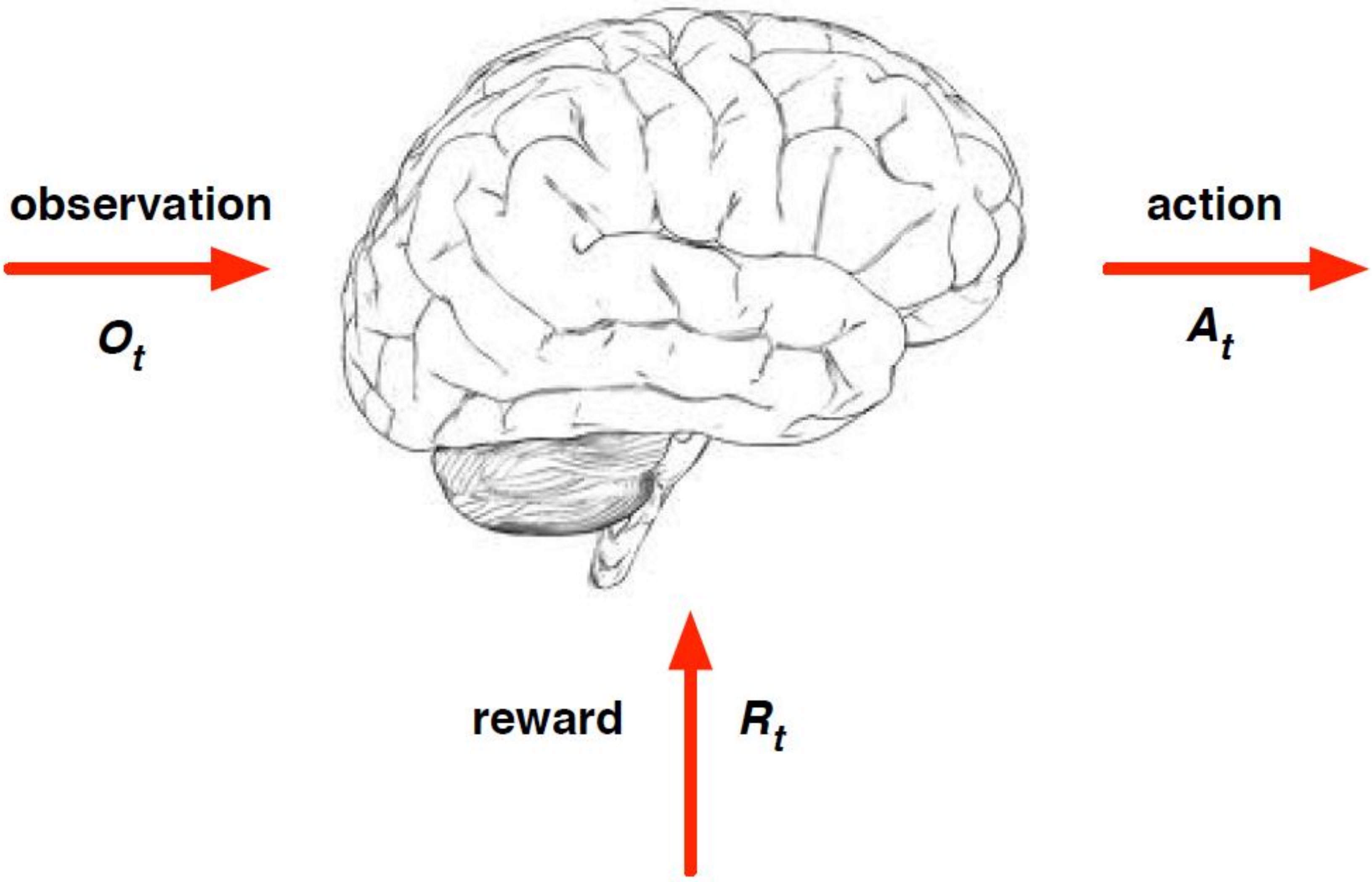
Examples of Rewards

- Manage an investment portfolio
  - +ve reward for each $ in bank
- Control a power station
  - +ve reward for producing power
  - −ve reward for exceeding safety thresholds
- Make a humanoid robot walk
  - +ve reward for forward motion
  - −ve reward for falling over
- Play many different Atari games better than humans
  - +/−ve reward for increasing/decreasing score

Sequential Decision Making

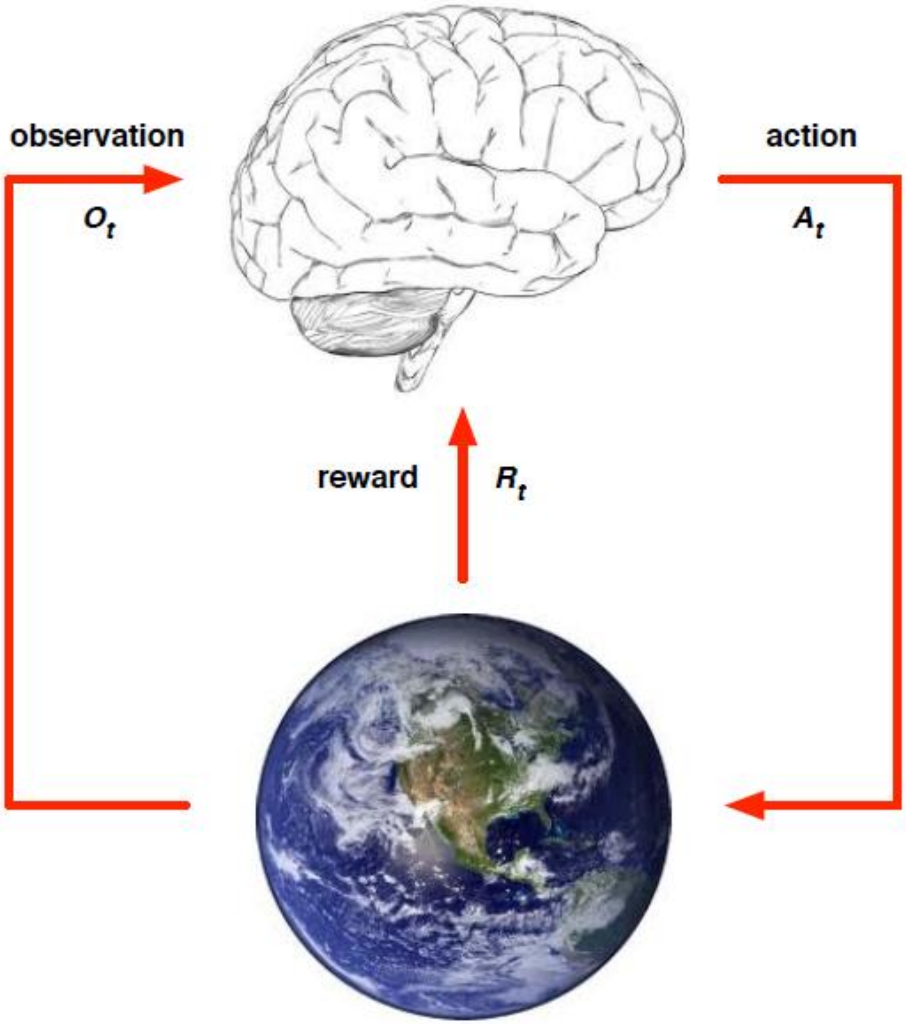
- Goal: select actions to maximize total future reward
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward
- Examples
  - A financial investment (may take months to mature)
  - Refuelling a helicopter (might prevent a crash in several hours)
  - Blocking opponent moves (might help winning chances many moves from now)

Agent and Environment



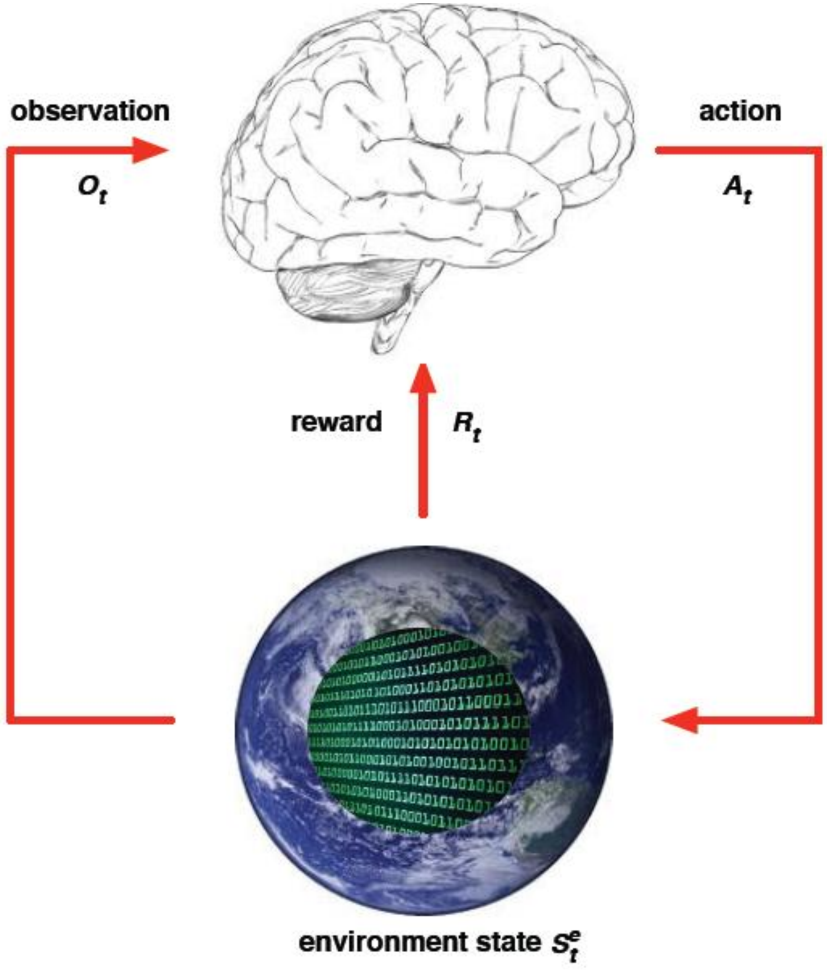
Agent and Environment

- At each step t the agent:
  - Executes action a_t
  - Receives observation o_t
  - Receives scalar reward r_t
- The environment:
  - Receives action a_t
  - Emits observation o_{t+1}
  - Emits scalar reward r_{t+1}
- t increments at env. step



Environment State

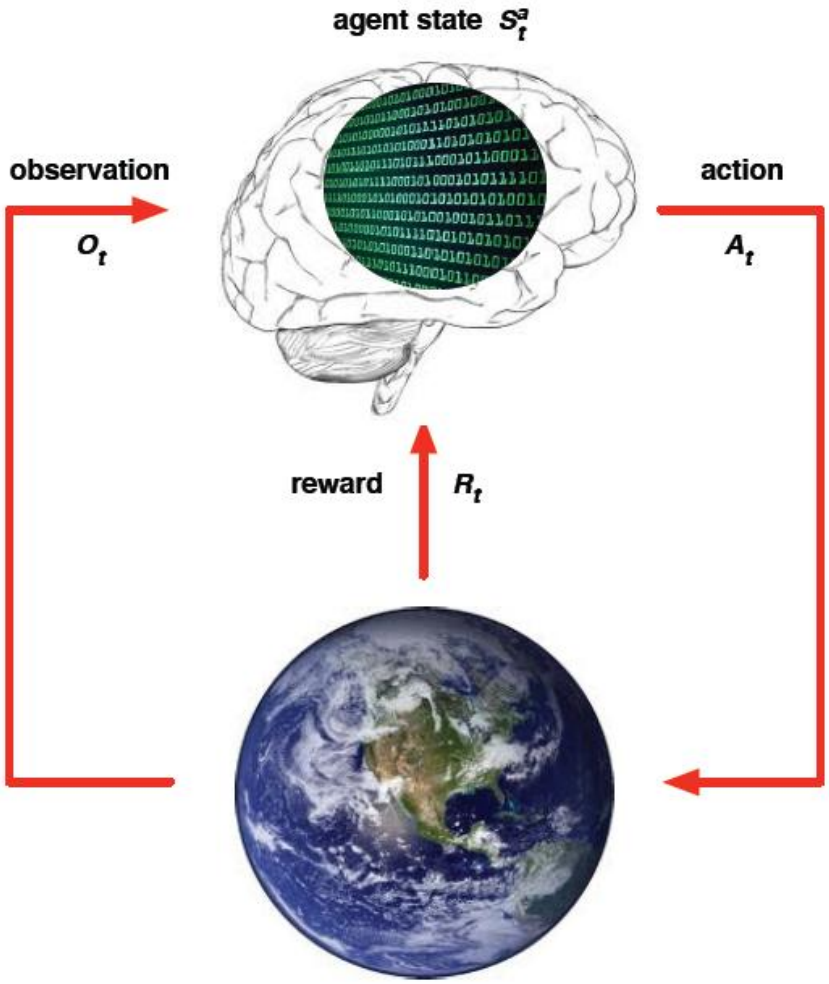
- The environment state S_t^e is the environment's private representation
- i.e. whatever data the environment uses to pick the next observation/reward
- The environment state is not usually visible to the agent
- Even if S_t^e is visible, it may contain irrelevant information



香港中文大學 The Chinese University of Hong Kong

Agent State

- The agent state S_t^a is the agent's internal representation
- i.e. whatever information the agent uses to pick the next action
- i.e. it is the information used by reinforcement learning algorithms



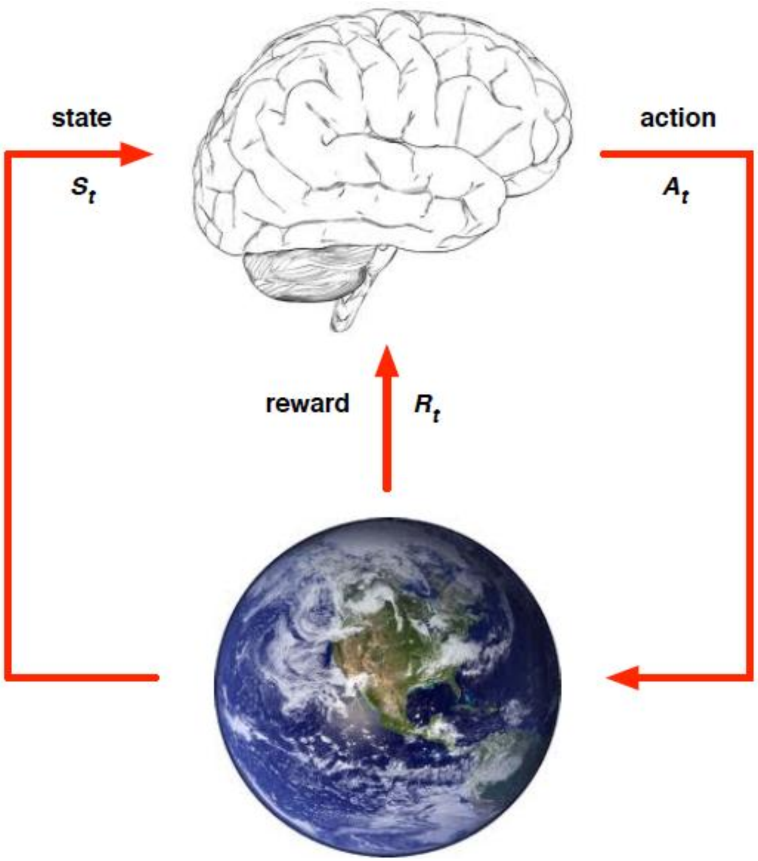
Fully Observable Environments

- Full observability: agent directly observes environment state

$$o_t = s_t^a = s_t^e = s_t$$

- Agent state = environment state = information state
- Reward is estimated by a reward function

$$r_t = r(s_t, a_t)$$

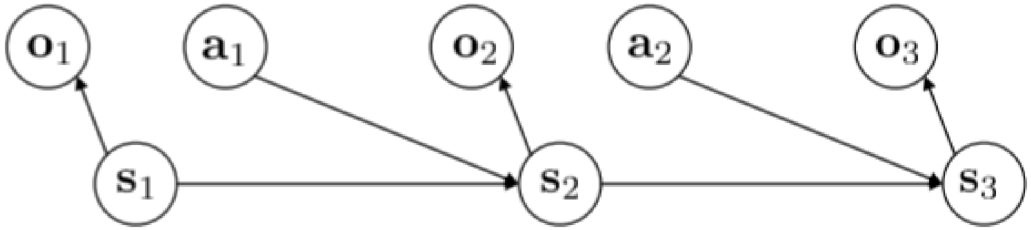


Information State

- An information state (a.k.a. Markov state) contains all useful information from the previous time steps
- A state s_t is Markov if and only if

$$p(s_{t+1}|s_t, a_t) = p(s_{t+1}|s_t, a_t, \cdots, s_1, a_1)$$

- The future is independent of the past given the present
- i.e. The state is a sufficient statistic of the future
- The environment state is Markov
- Formally, this is a Markov decision process (MDP)



Major Components of an RL Agent

- An RL agent may include one or more of these components:
  - **Policy**: agent's behaviour function
  - **Value function**: how good is each state and/or action
  - **Model**: agent's representation of the environment

Policy

- A policy is the agent's behaviour
- It is a mapping function from state to action, e.g.
- Deterministic policy: $a = \pi_\theta(s)$
- Stochastic policy: $\pi_\theta(a|s) = p(a_t|s_t)$

Value Function and Q-function

- Value function is a prediction of future reward from current state following the current policy $\pi_\theta(s)$

$$V^\pi(\mathbf{s}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t\right]$$

- We also define the Q-function as the future reward from state and taking action $\mathbf{a}_t$

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t\right]$$

- The value function can therefore be reformulated as

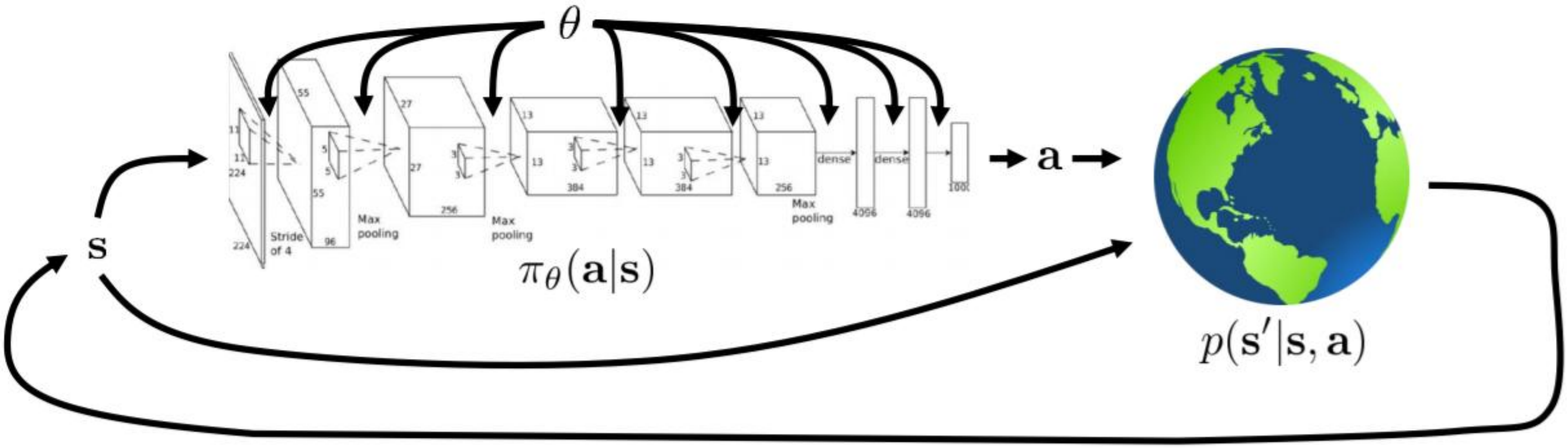
$$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$$

Model

- A model predicts what the environment will do next
- $\mathcal{P}$ predicts the next state
- $\mathcal{R}$ predicts the next (immediate) reward, e.g.

$$\mathcal{P}^a = p(s_{t+1} | s_t, a_t)$$
$$\mathcal{R}^a = E[r_{t+1} | s_t, a_t]$$

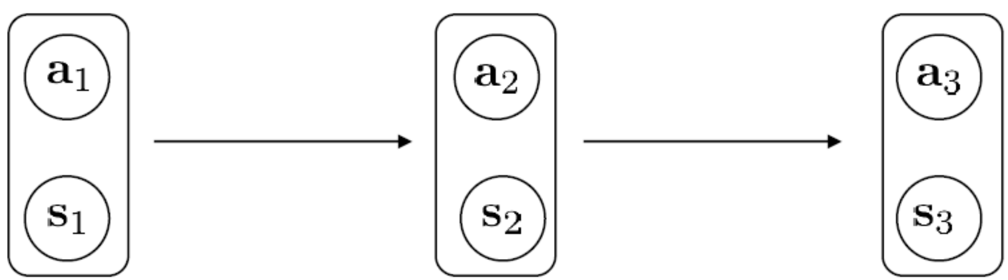
The Goal of Reinforcement Learning



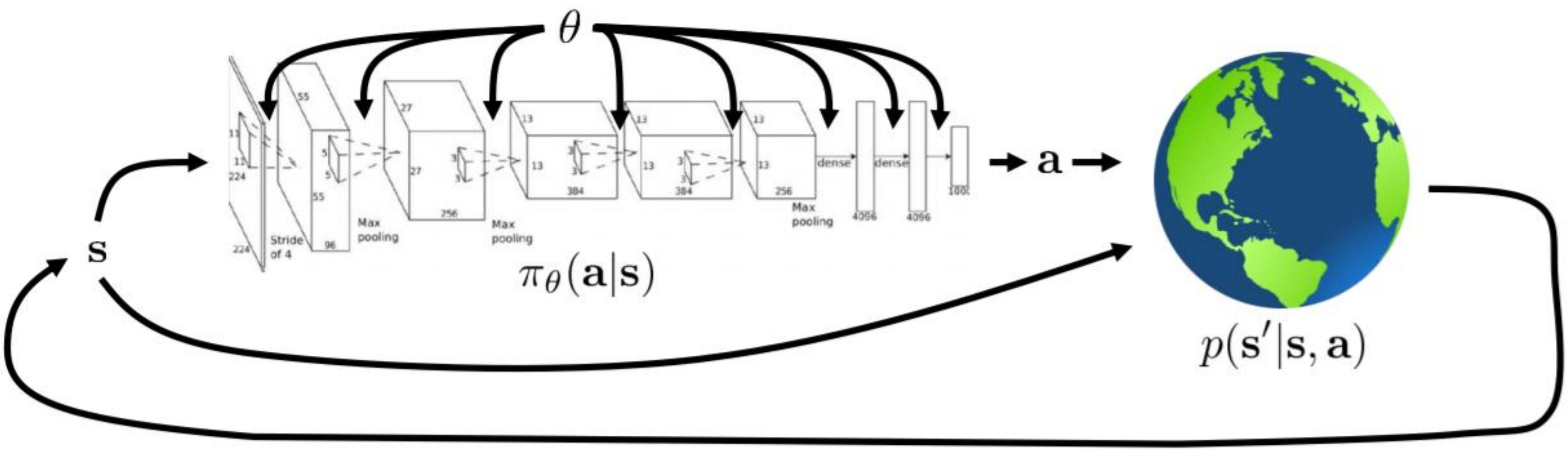
Joint probability of states and actions

$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\underbrace{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}_{\text{Markov chain on }(\mathbf{s},\mathbf{a})}$$

$$p((\mathbf{s}_{t+1},\mathbf{a}_{t+1})|(\mathbf{s}_t,\mathbf{a}_t)) =$$
$$p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)\pi_\theta(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})$$



The Goal of Reinforcement Learning



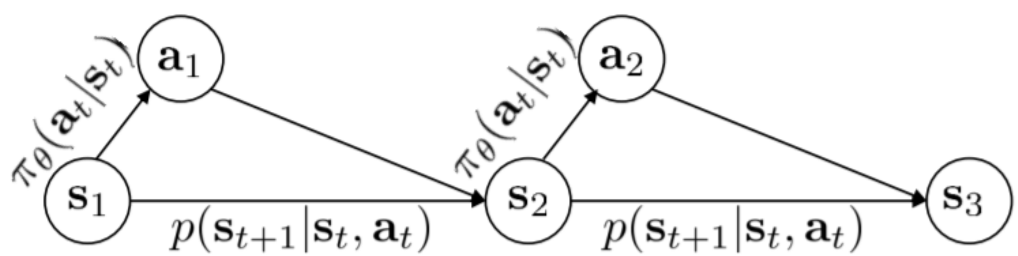
Joint probability of states and actions

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \underbrace{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}_{\text{Markov chain on } (\mathbf{s}, \mathbf{a})}$$

Final Objective:

Maximizing the expected cumulative rewards

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$



Finite and Infinite Horizon Case

- If the overall time step *T* is finite, the objective can be defined as

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]$$

$$= \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)]$$

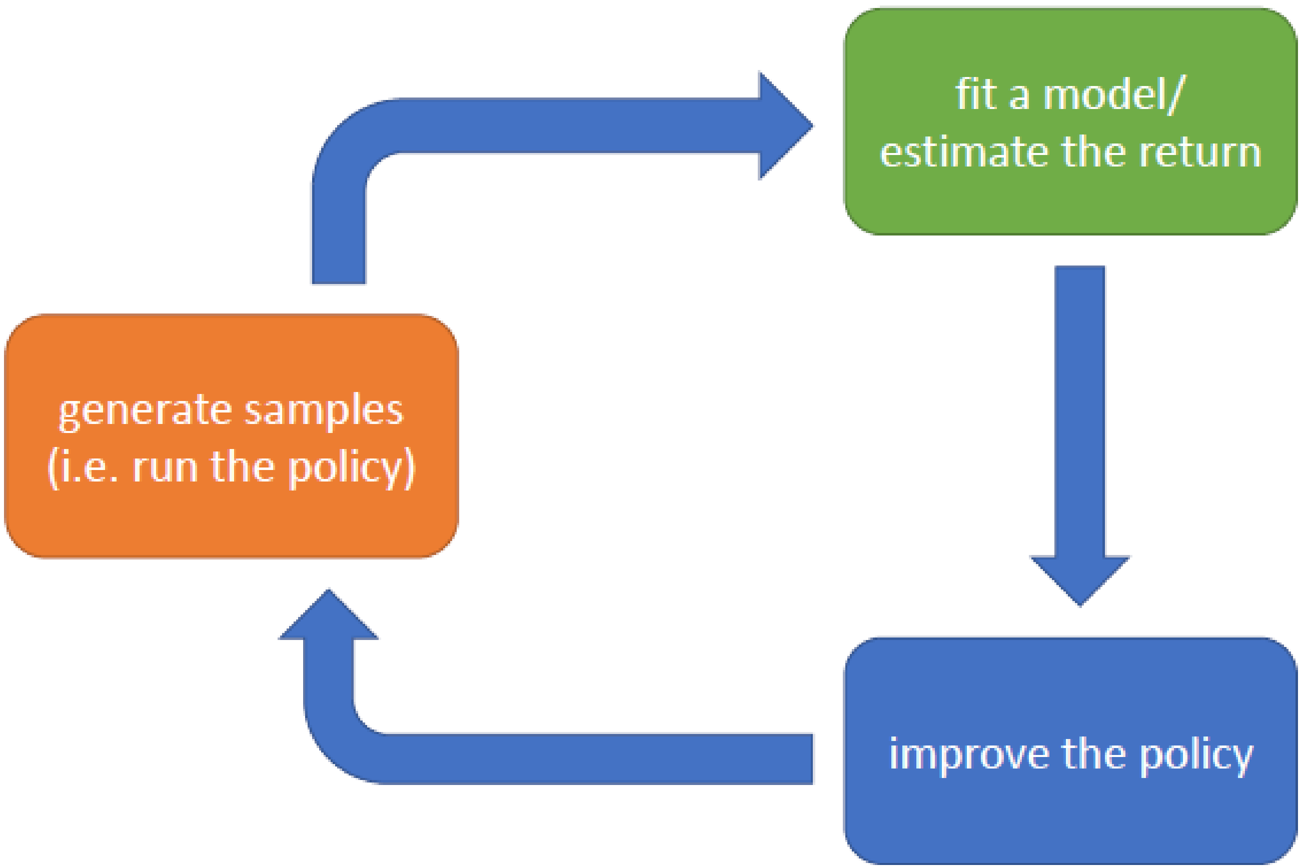
- For the infinite time steps,

$$\theta^\star = \arg\max_\theta \frac{1}{T}\sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)] \rightarrow E_{(\mathbf{s},\mathbf{a})\sim p_\theta(\mathbf{s},\mathbf{a})}[r(\mathbf{s},\mathbf{a})]$$

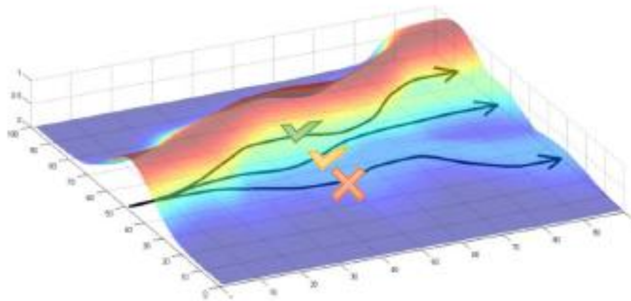
$$\text{(in the limit as } T\rightarrow\infty)$$

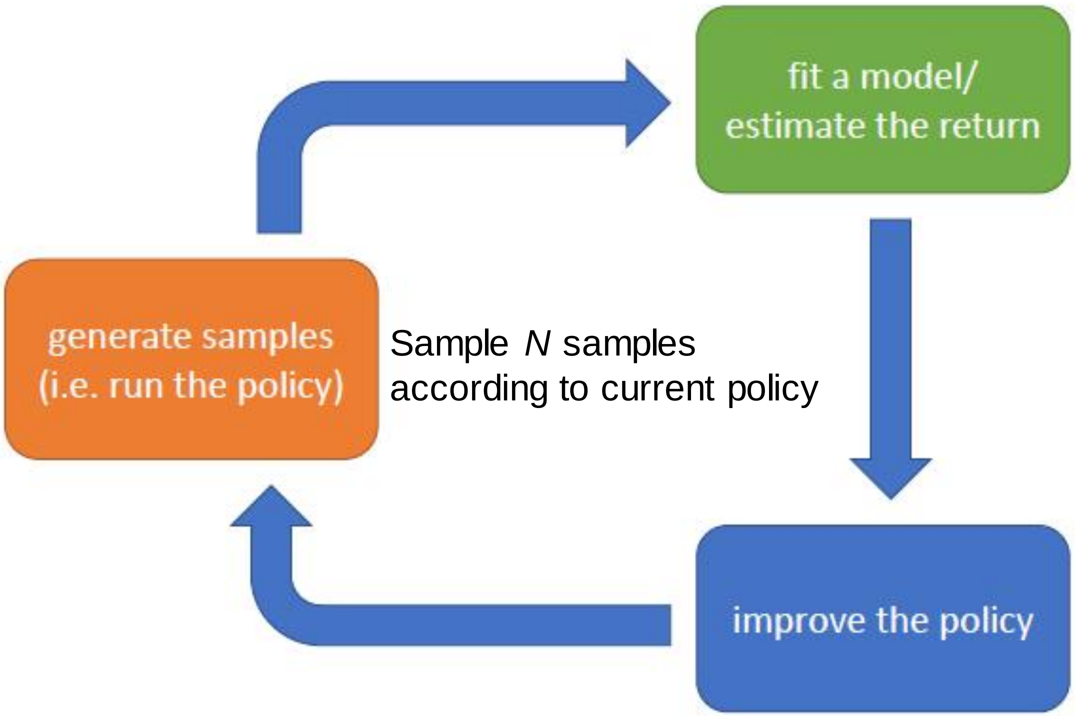
- In reinforcement learning, we almost always care about expectation

Reinformance Learning Algorithm



RL: A Simple Example

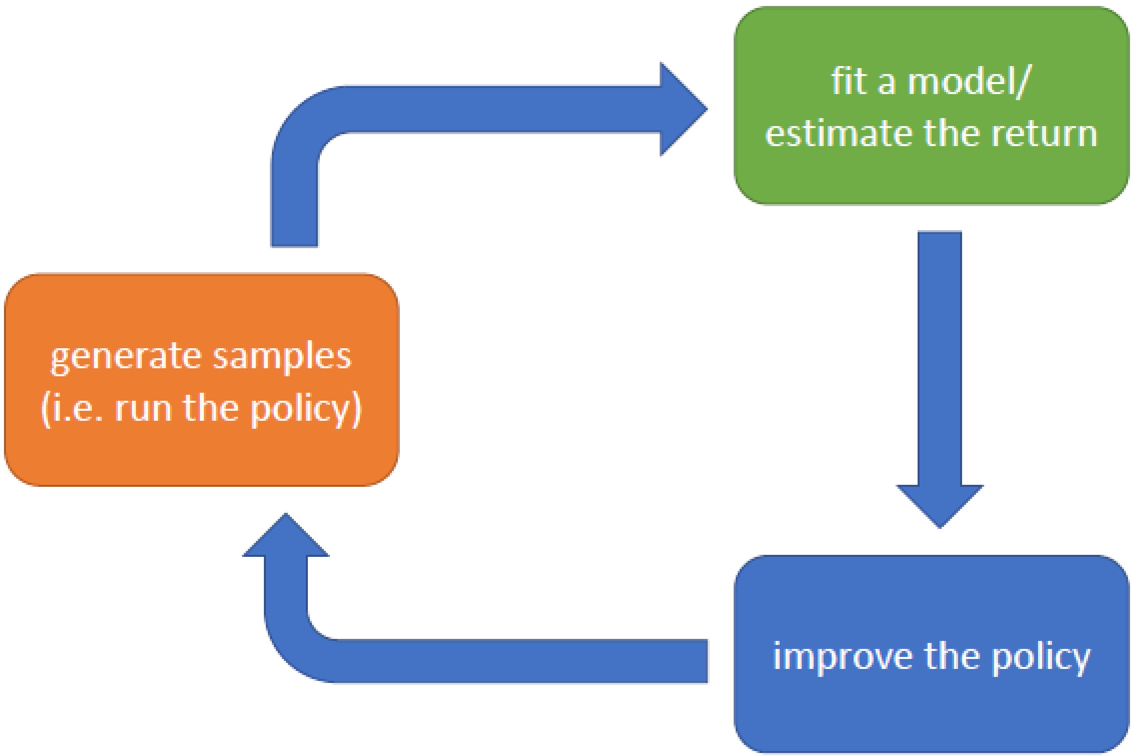




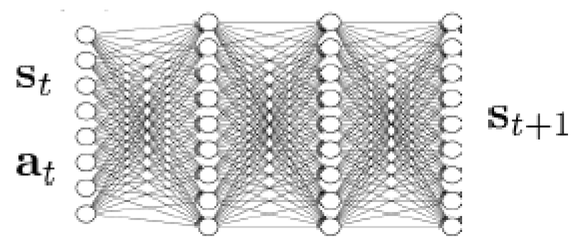
$$J(\theta) = E_\pi\left[\sum_t r_t\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_t r_t^i$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Simple RL with Deep Neural Networks



learn f_ϕ such that $\mathbf{s}_{t+1} \approx f_\phi(\mathbf{s}_t, \mathbf{a}_t)$



backprop through f_ϕ and r to train $\pi_\theta(\mathbf{s}_t) = \mathbf{a}_t$

Which Parts are Expensive

$$J(\theta) = E_\pi\left[\sum_t r_t\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_t r_t^i$$

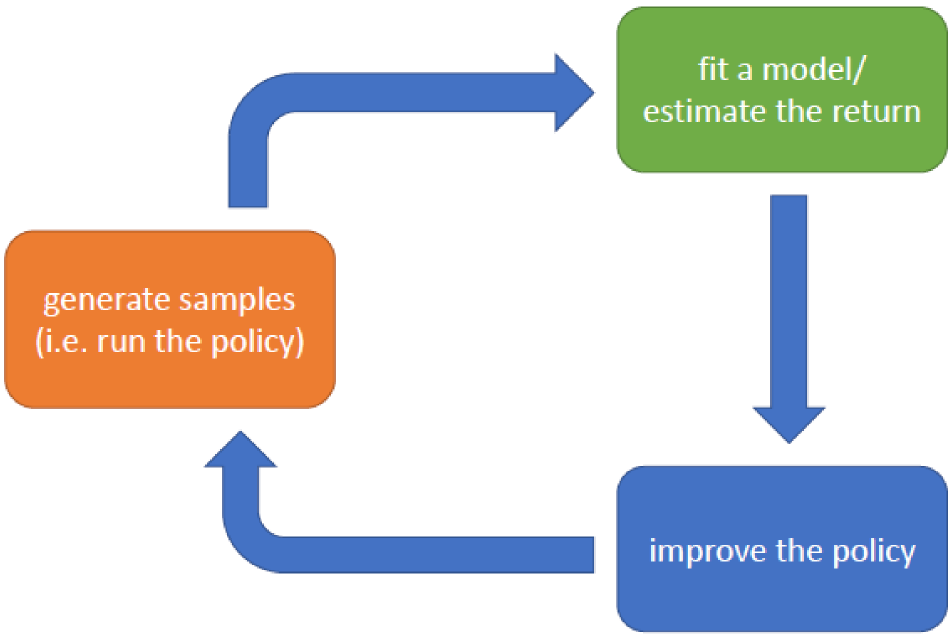
trivial, fast

learn $\mathbf{s}_{t+1} \approx f_\phi(\mathbf{s}_t, \mathbf{a}_t)$

expensive

real robot/car/power grid/whatever:
1x real time, until we invent time travel

MuJoCo simulator:
up to 10000x real time

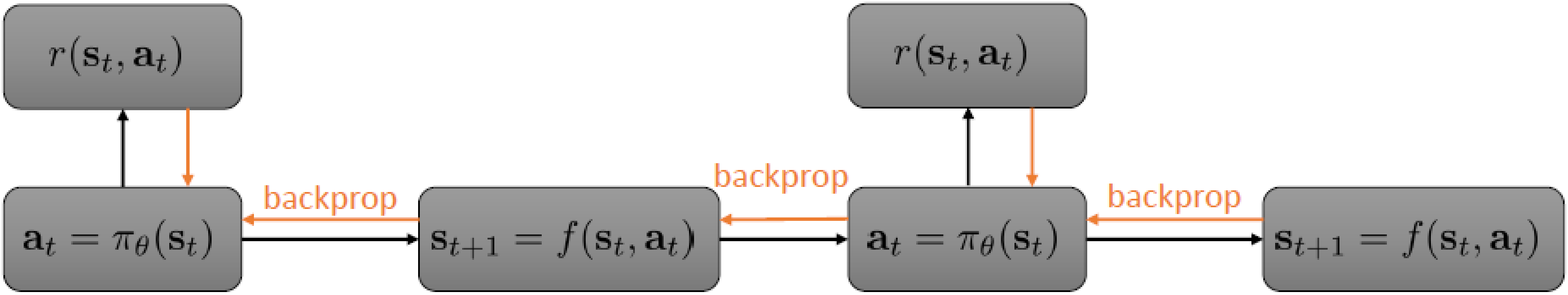


$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

backprop through f_ϕ and r to train $\pi_\theta(\mathbf{s}_t) = \mathbf{a}_t$

Why Not Enough?

- Only handles deterministic dynamics
- Only handles deterministic policies
- Only continuous states and actions
- Very difficult optimization problem



Stochastic System

- If we have policy and we know the Q-function, then we can improve the policy

$$\text{set } \pi'(\mathbf{a}|\mathbf{s}) = 1 \text{ if } \mathbf{a} = \arg\max_{\mathbf{a}} Q^{\pi}(\mathbf{s}, \mathbf{a})$$

- Compute gradient to increase probability of good actions ***a***

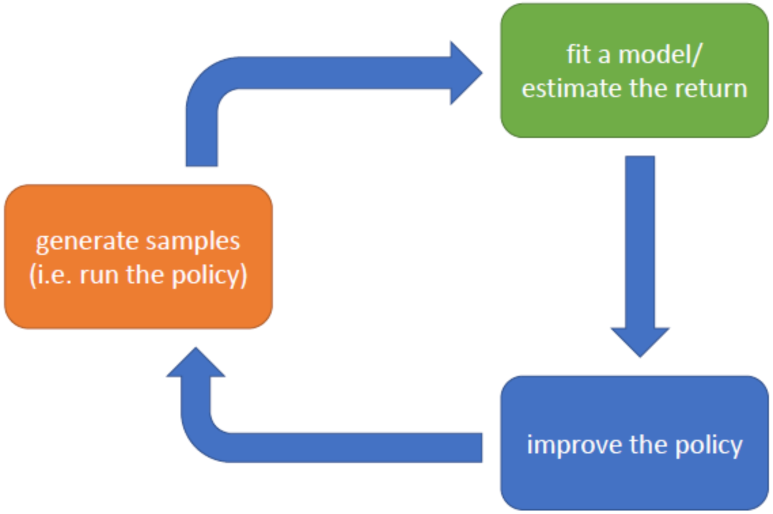
$$\text{if } Q^{\pi}(\mathbf{s}, \mathbf{a}) > V^{\pi}(\mathbf{s}), \text{ then } \mathbf{a} \text{ is } \textit{better than average}$$

$$\text{modify } \pi(\mathbf{a}|\mathbf{s}) \text{ to increase probability of } \mathbf{a} \text{ if } Q^{\pi}(\mathbf{s}, \mathbf{a}) > V^{\pi}(\mathbf{s})$$

- Recall that *V* is the expecation of *Q* over all actions

Review of Reinforcement Learning

- Definitions
  - Markov Decision Process
- RL objective
  - Maximize expected reward
- Structure of RL algorithms
  - Sample generation
  - Fitting a model/estimating return
  - Policy improvement
- Value functions and Q-functions

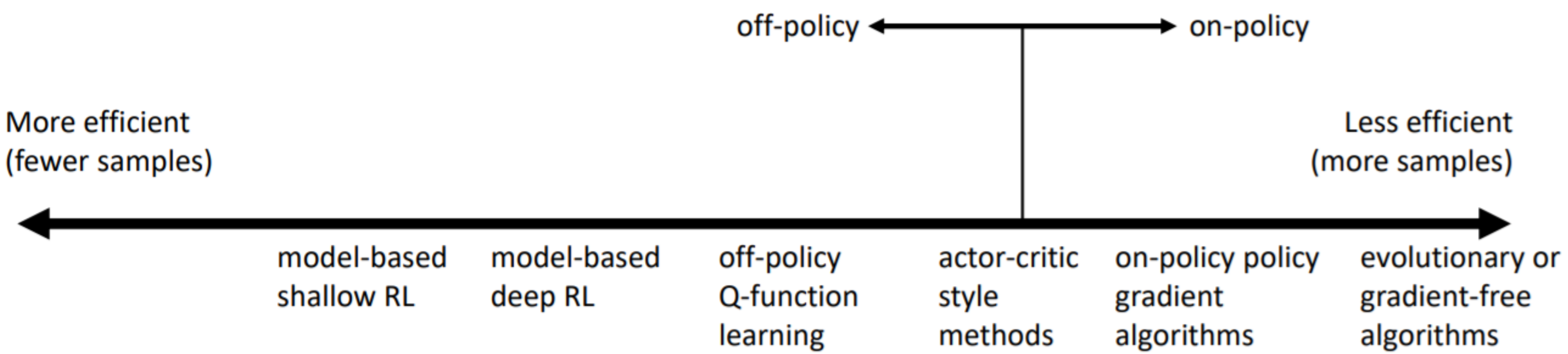


Categorizing of RL Algorithms

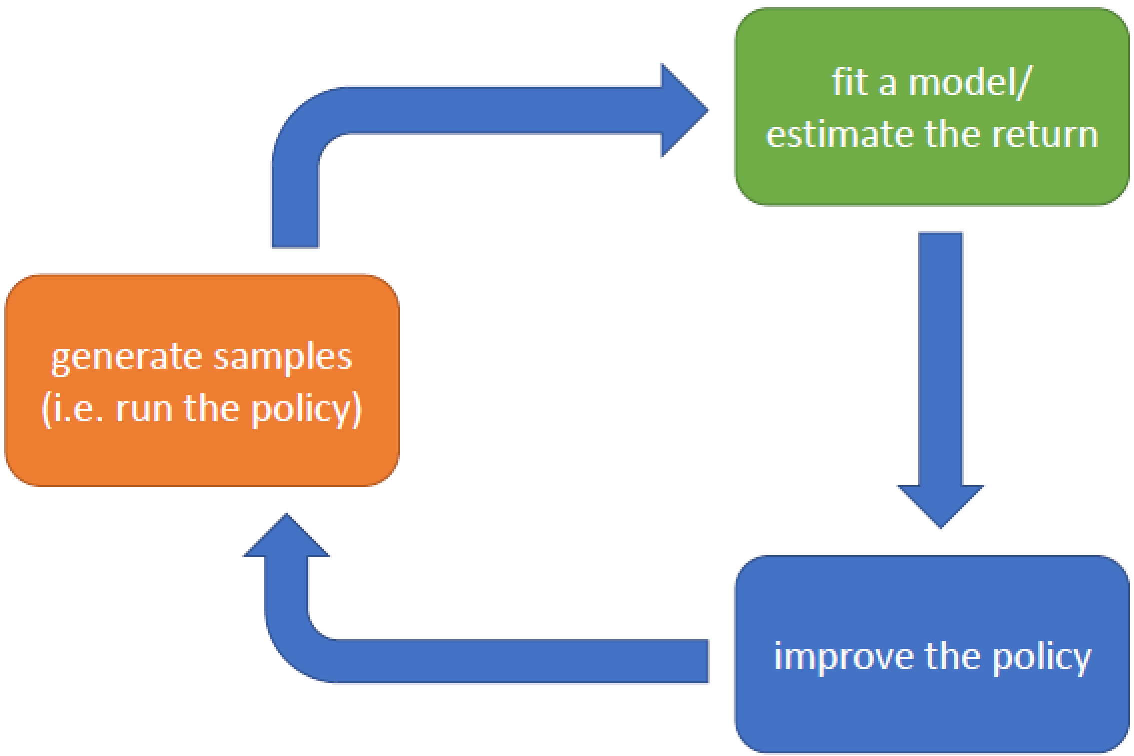
$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]$$

- Policy gradients: directly differentiate the above objective
- Value-based: estimate value function *V* or Q-function of the optimal policy (no explicit policy)
- Actor-critic: estimate value function or Q-function of the current policy, use it to improve policy
- Model-based RL: estimate the transition model
  - Use it for planning (no explicit policy)
  - Use it to improve a policy

Sampling Efficiency for Different Algorithms



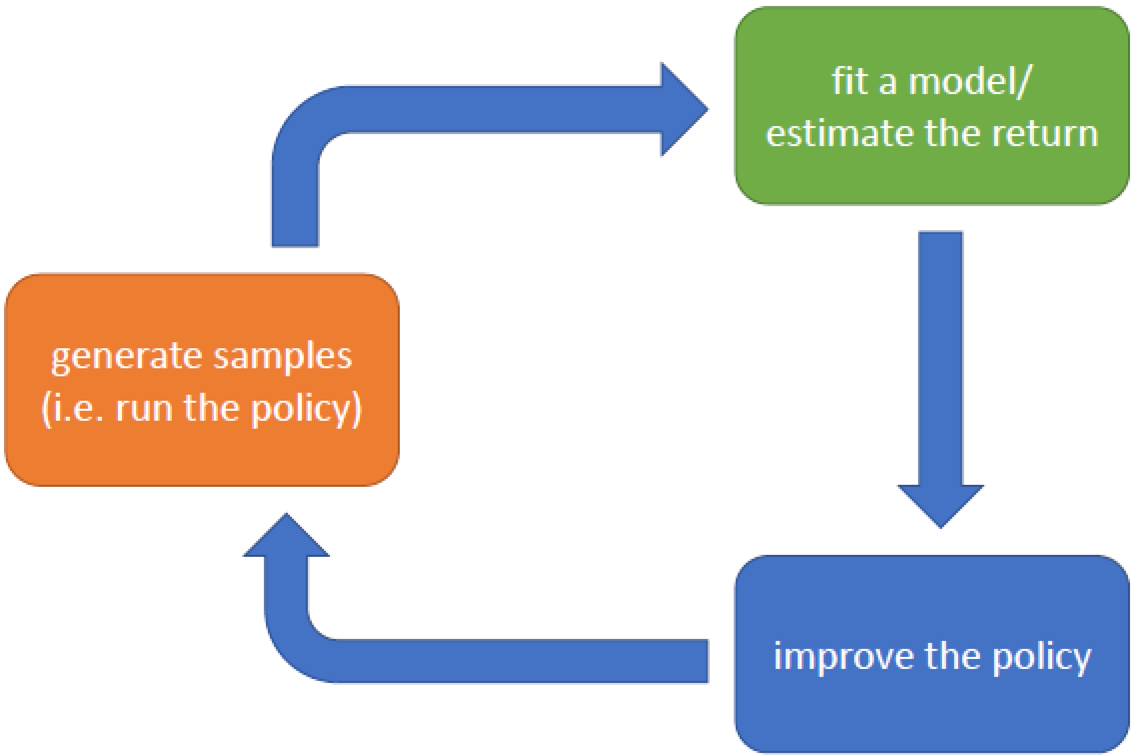
Value-based RL Algorithms



fit $V(\mathbf{s})$ or $Q(\mathbf{s}, \mathbf{a})$

set $\pi(\mathbf{s}) = \arg\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$

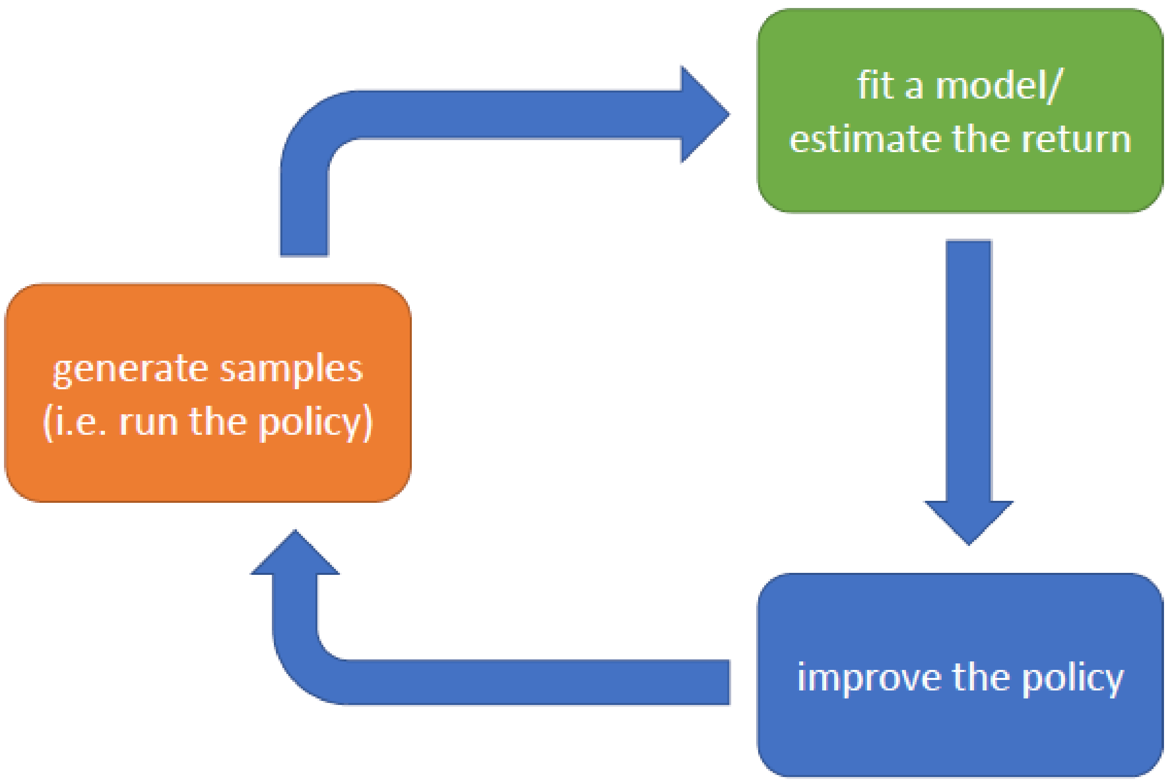
Policy Gradients



evaluate returns
$R_\tau = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

$\theta \leftarrow \theta + \alpha \nabla_\theta E[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)]$

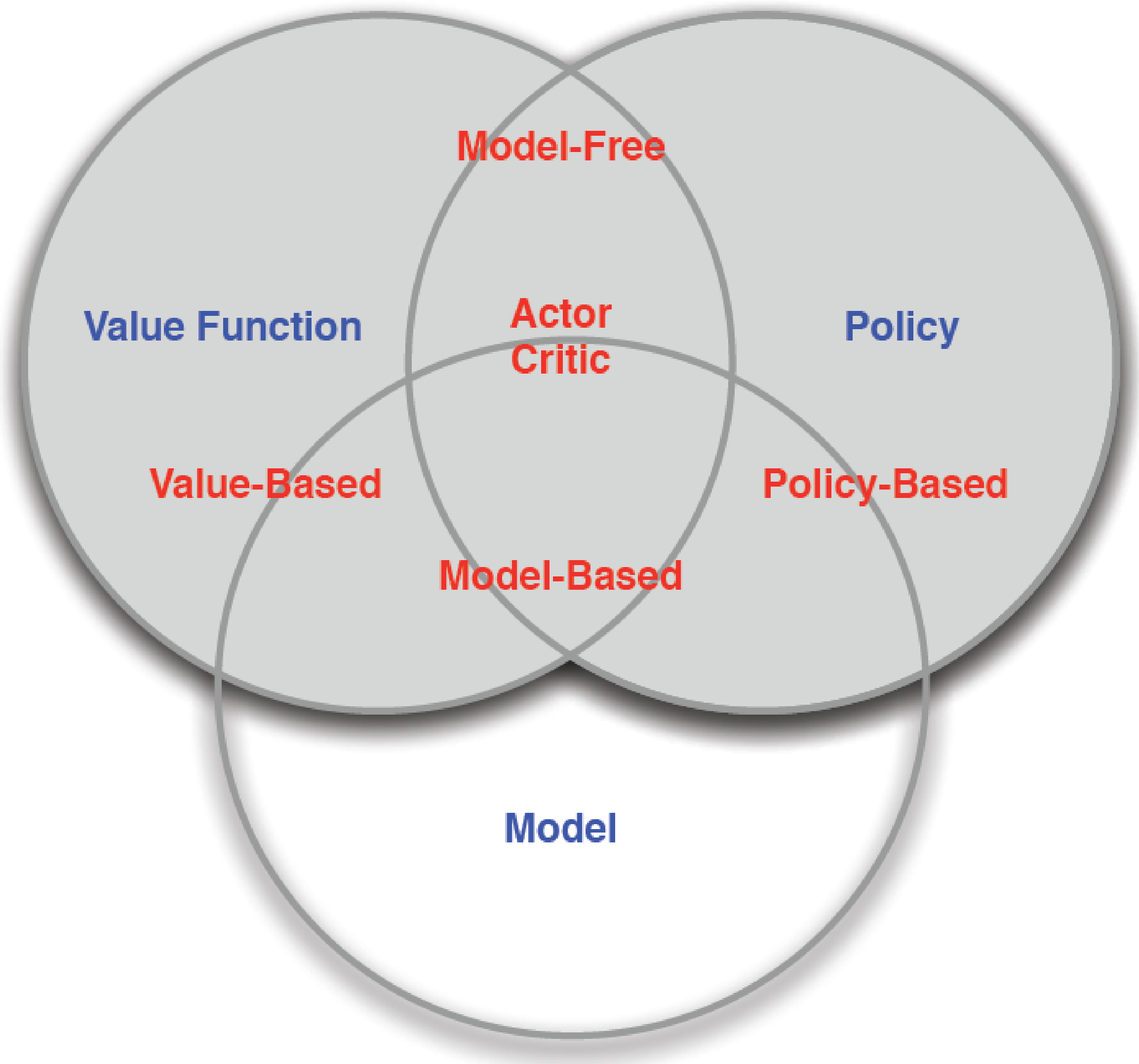
Actor-critic: value functions + policy gradients



fit $V(\mathbf{s})$ or $Q(\mathbf{s}, \mathbf{a})$

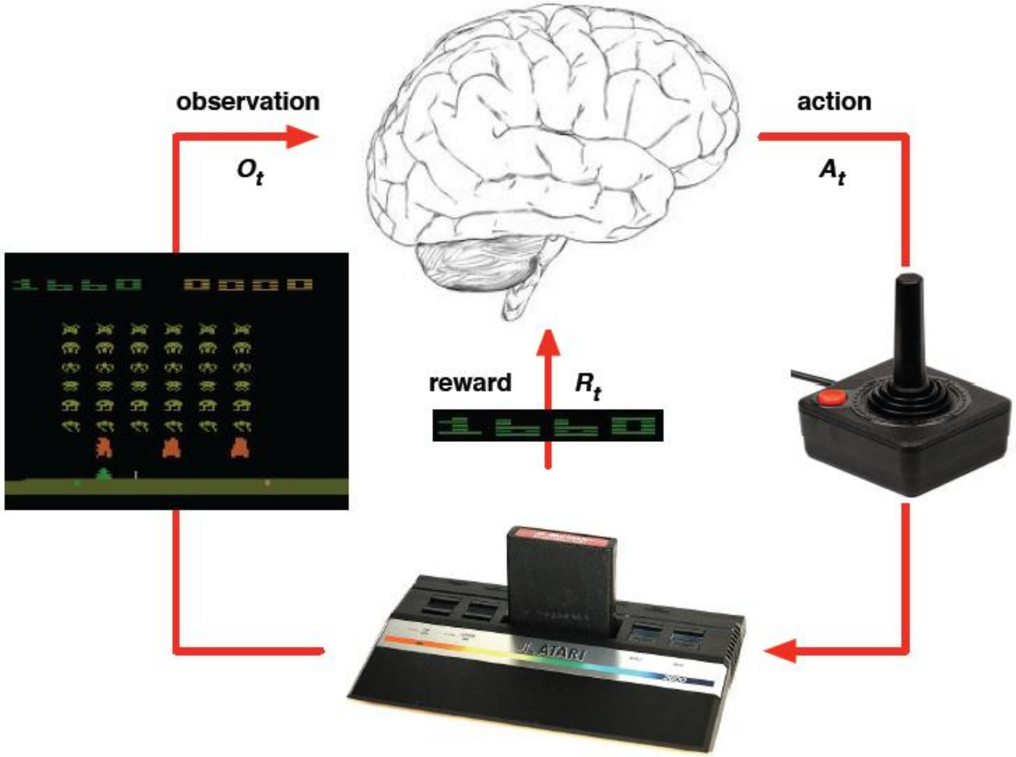
$\theta \leftarrow \theta + \alpha \nabla_\theta E[Q(\mathbf{s}_t, \mathbf{a}_t)]$

Categorizing RL agents

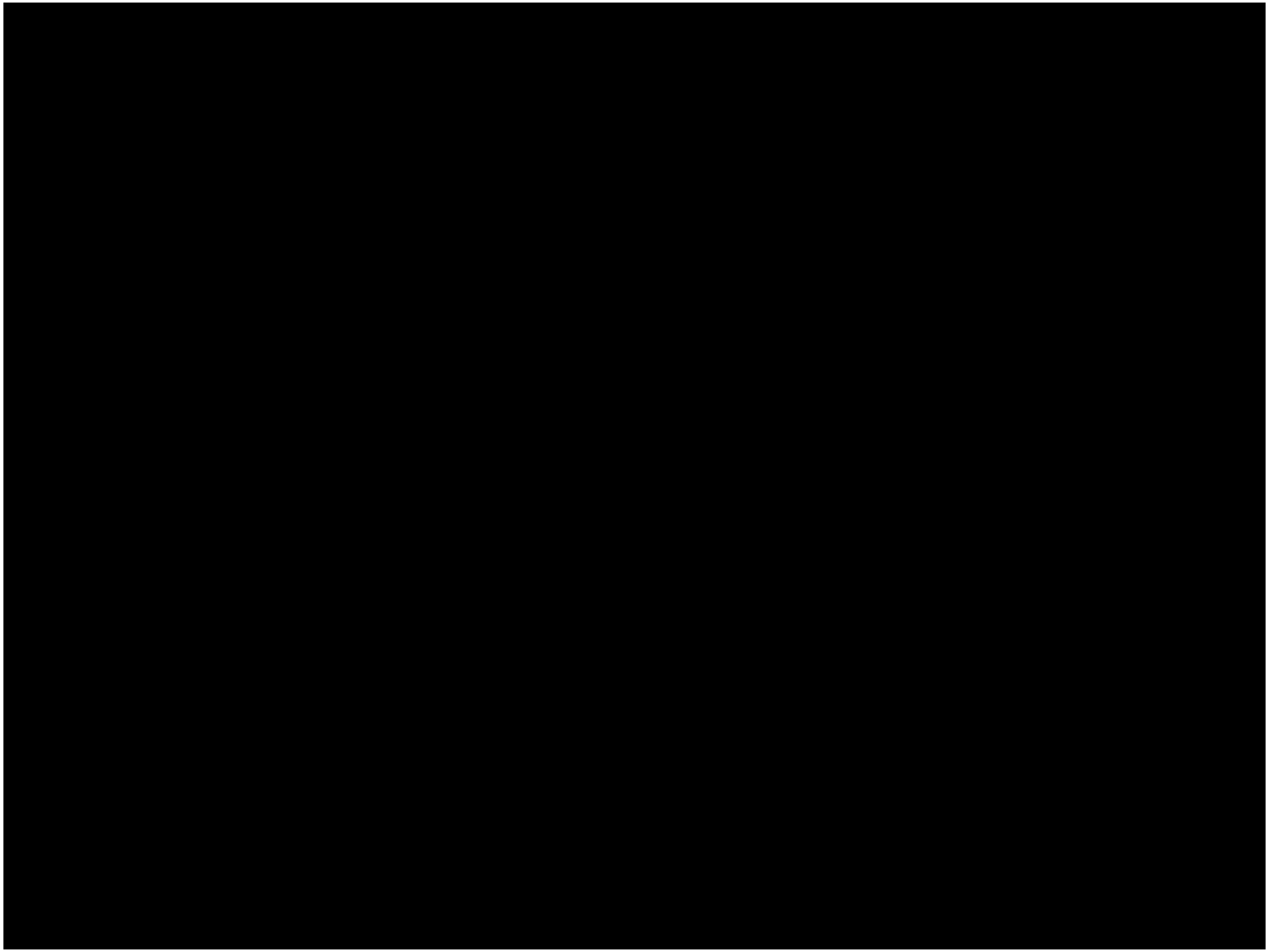


Atari with Q-functions

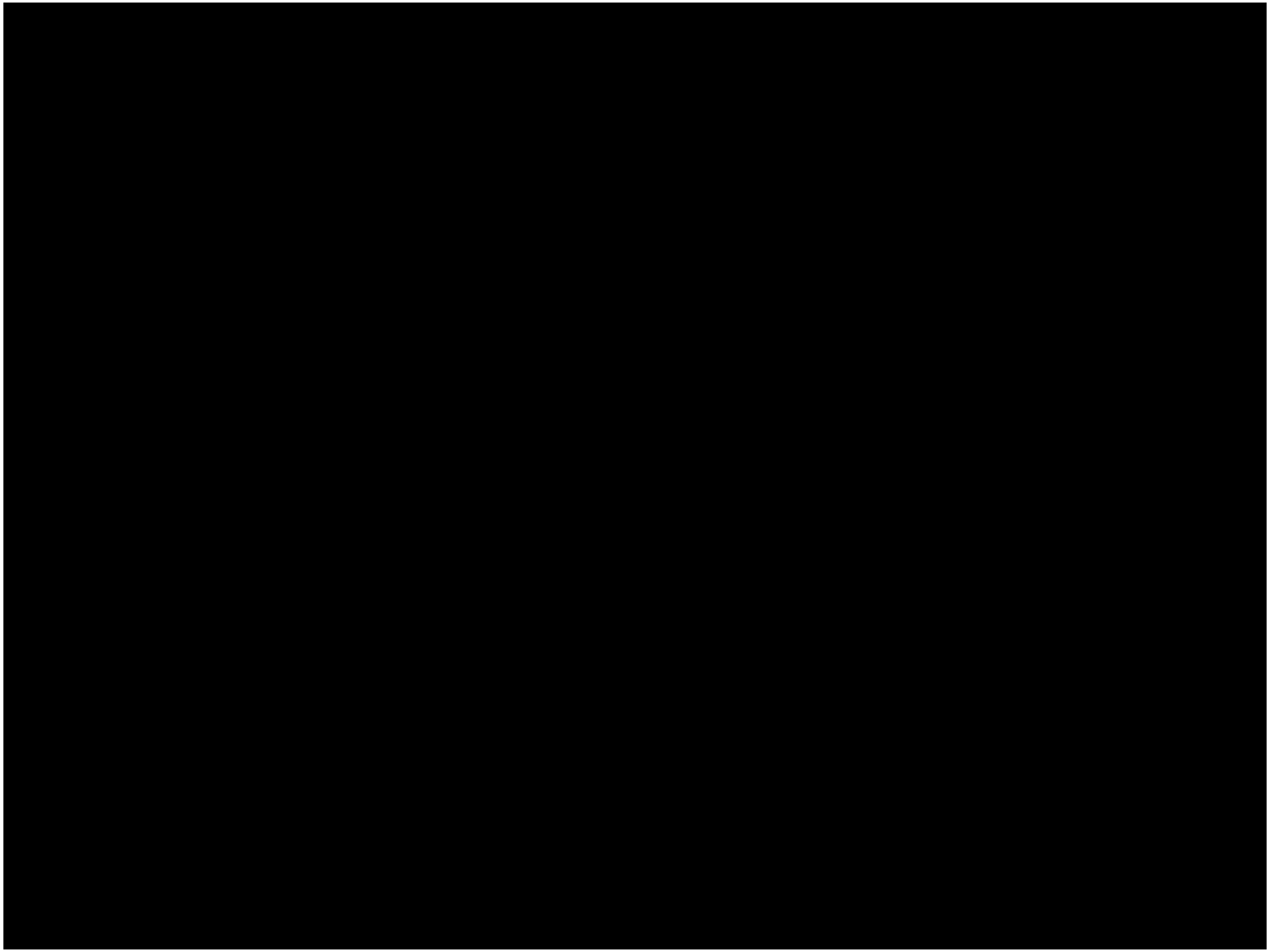
- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores



Example



Example



Policy Gradient Introduction

- The objective of reinforcement learning to maximize the expected cumulative rewards

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]$$

- Infinite horizon case

$$\theta^\star = \arg\max_\theta E_{(\mathbf{s},\mathbf{a}) \sim p_\theta(\mathbf{s},\mathbf{a})}[r(\mathbf{s}, \mathbf{a})]$$

- Finite horizon case

$$\theta^\star = \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t) \sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)]$$

Approximating the Expectation

- The $p_\theta(\tau)$ is a joint probability distribution over all possible states and actions at all time steps, which is intractable to evaluate
- We sample from the current policy π_θ and obtain *N* sequences of states and actions

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

sum over samples from π_θ

- Recall that

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T} \underbrace{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}_{\text{Markov chain on } (\mathbf{s}, \mathbf{a})}$$

Direct Policy Differentiation

- Rewrite the objective function

The probability distribution

$$J(\theta) = E_{\tau\sim p_\theta(\tau)}\left[\underline{\sum_t r(\mathbf{s}_t,\mathbf{a}_t)}\right] = E_{\tau\sim p_\theta(\tau)}\,\underline{[r(\tau)]} = \int \pi_\theta(\tau) r(\tau)d\tau$$

where
$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\underbrace{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}_{\text{Markov chain on }(\mathbf{s},\mathbf{a})}$$

- The gradient of the objective function w.r.t. the model parameters can be formulated as

$$\nabla_\theta J(\theta) = \int \underline{\nabla_\theta \pi_\theta(\tau)} r(\tau)d\tau = \int \underline{\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)} r(\tau)d\tau = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

where
$$\underline{\nabla_\theta \pi_\theta(\tau)} = \pi_\theta(\tau)\frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \underline{\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)}$$

Direct Policy Differentiation

- The gradient of expected total reward w.r.t parameters is therefore

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

- For the joint probability

$$\pi_\theta(\tau) = \pi_\theta(s_1, a_1, \cdots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

$$\log \pi_\theta(\tau) = \log p(\mathbf{s}_1) + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\nabla_\theta \left[\cancel{\log p(\mathbf{s}_1)} + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \cancel{\log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \right]$$

Direct Policy Differentiation

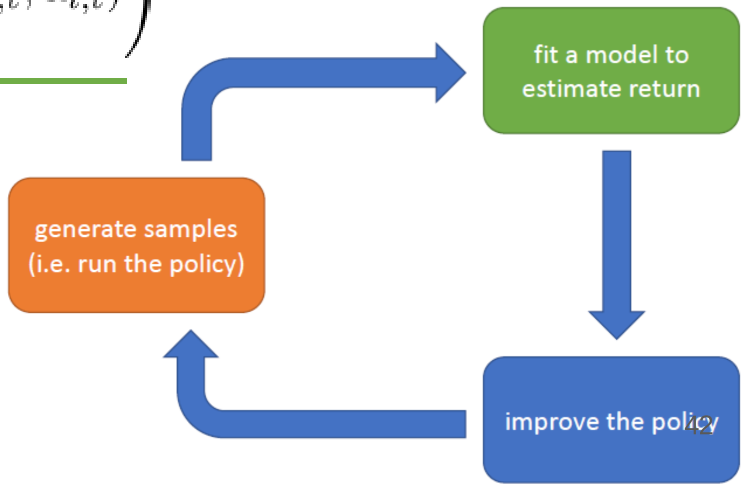
- The gradient is formulated as

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

- The calculation of expectation over $\pi_\theta(\tau)$ is untractable
- We can approximate it by sampling from the current policy

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$



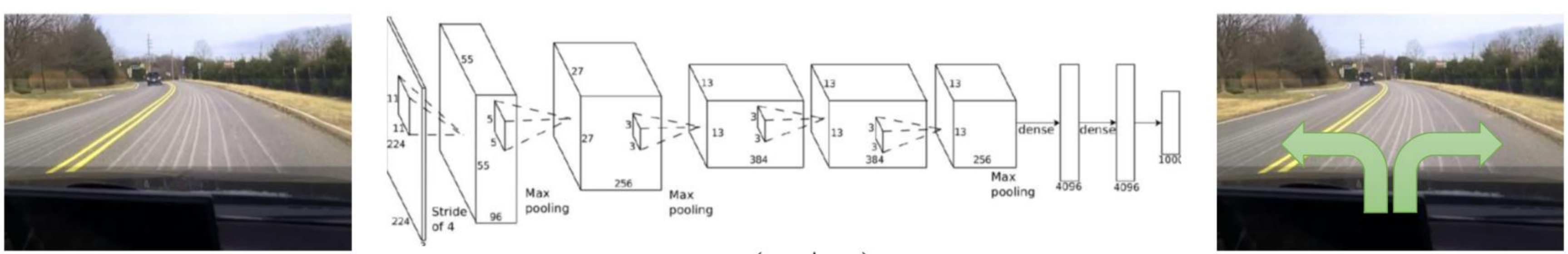
The REINFORCE Algorithm

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})\right)$$

what is this?



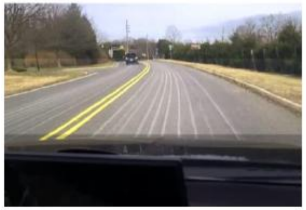
$\mathbf{s}_t$ $\qquad \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ $\qquad \mathbf{a}_t$

Comparison to Maximum Likelihood

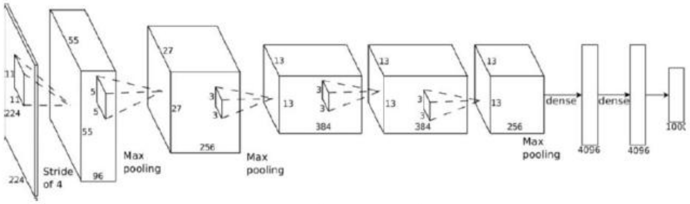
- Very similar to maximum likelihood estimation but use total reward instead of ground-truth label to supervise

$$\text{policy gradient:} \quad \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

$$\text{maximum likelihood:} \quad \nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right)$$

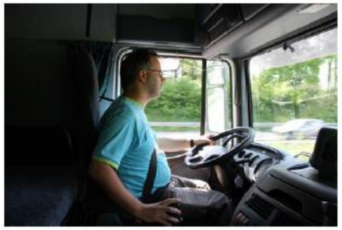


$\mathbf{s}_t$

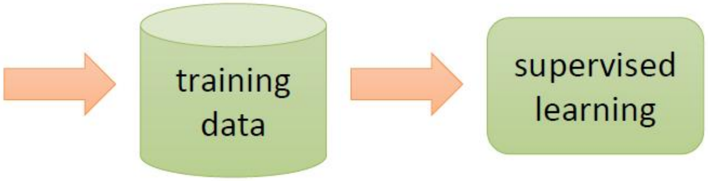


$\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$

$\mathbf{a}_t$



$\mathbf{s}_t$
$\mathbf{a}_t$



$\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$

Example: Gaussian Policies

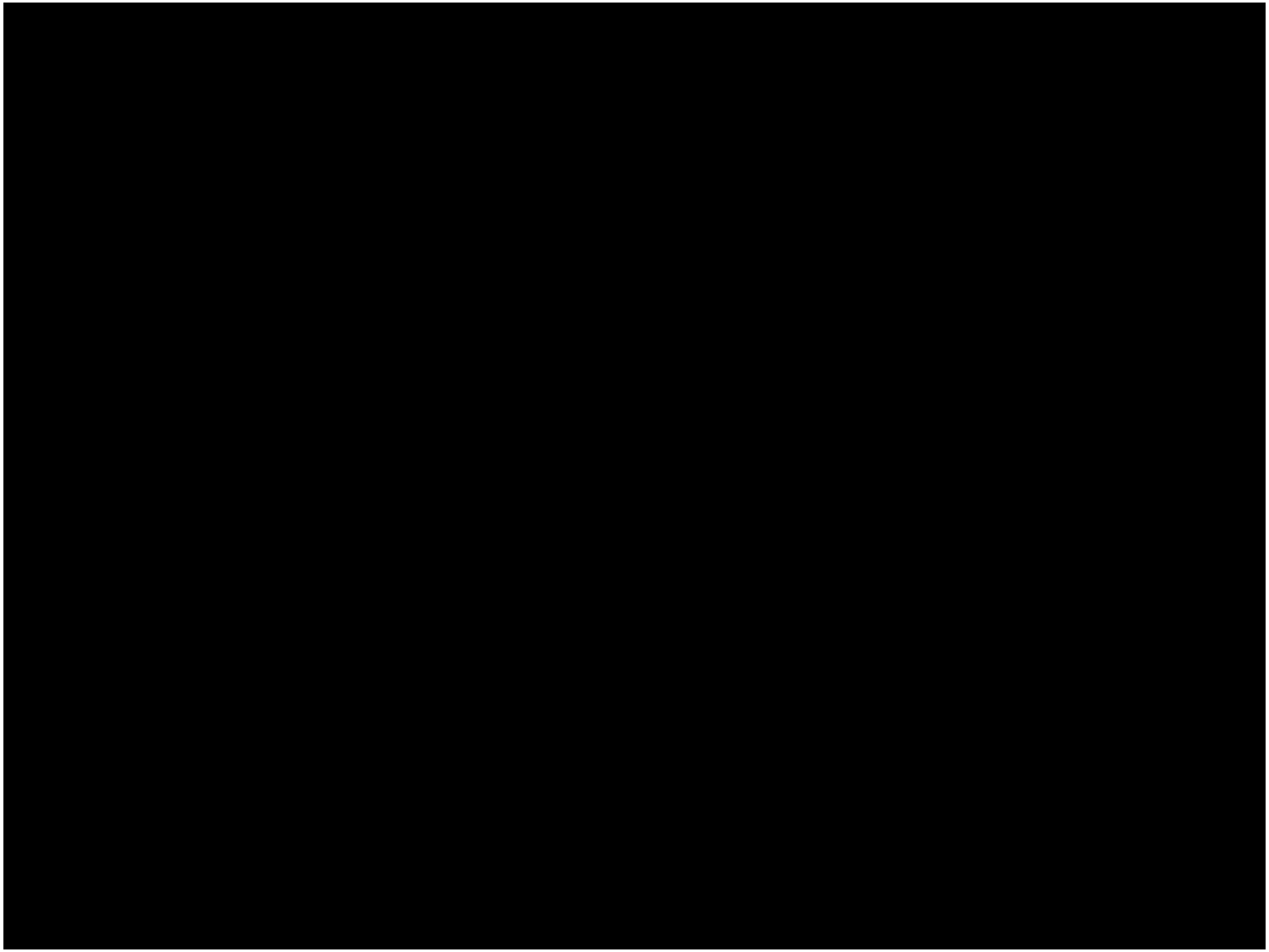
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

example: $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}(f_{\text{neural network}}(\mathbf{s}_t); \Sigma)$

$$\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\|f(\mathbf{s}_t) - \mathbf{a}_t\|_\Sigma^2 + \text{const}$$

$$\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\Sigma^{-1}(f(\mathbf{s}_t) - \mathbf{a}_t)\frac{df}{d\theta}$$

Example: Robot Learn to Walk



Deep Q-Learning

- Value function maps each state $s_t = s$ as the expected total reward in the future

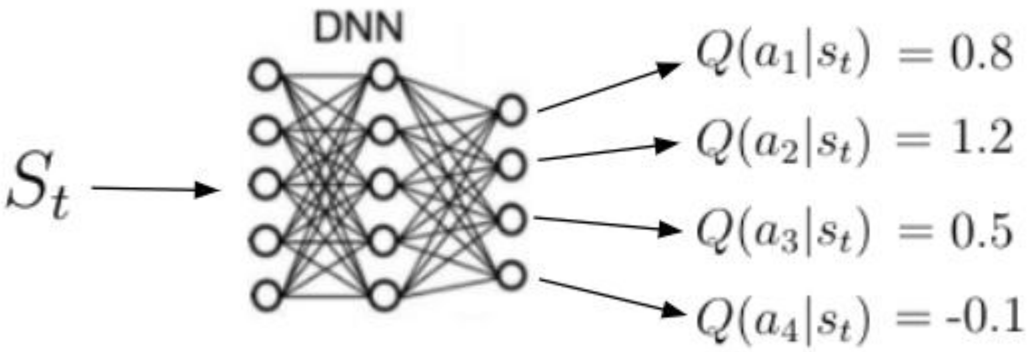
$$
\begin{aligned}
V^\pi(s_t) &= E_{\pi_\theta}\left[r_{t+1} + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s\right] \\
&= E_{\pi_\theta}\left[r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s\right] \\
&= E_{\pi_\theta}[g_t | s_t = s]
\end{aligned}
$$

- Q-function (action-value function) is the expected total reward if taking action ***a*** at current state ***s***

$$
\begin{aligned}
Q^\pi(s_t, a_t) &= E_{\pi_\theta}\left[r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a\right] \\
&= E_{\pi_\theta}\left[g_t | s_t = s, a_t = a\right]
\end{aligned}
$$

Q-functions

- The calculation of $Q(s, a)$ can be achieved by a neural network
- Given a state ***s***, it outputs the expected total future reward of which action to take given the current state ***s***



Q-Learning

- If we assume the current Q-function is correct, then the estimated total reward at the next time step should follow the previous equation

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha(\underbrace{r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})}_{\text{Learning Target}} - \underbrace{Q(s_t, a_t)}_{\text{Q-function}})$$

- Algorithm
  - Calculate $Q(s_t, a_t)$
  - Go to the next state s_{t+1}, take an action a_{t+1} that follows ε-greedy strategy and calculate the value $Q(s_{t+1}, a_{t+1})$
  - Calculate the learning target
  - Update previous $Q(s_t, a_t)$ with learning rate α

ε-greedy strategy: $\mu(a|s) = \begin{cases} \text{random action }, & \text{if } p \le \epsilon \\ \arg\max_a Q(s,a), & \text{otherwise} \end{cases}$

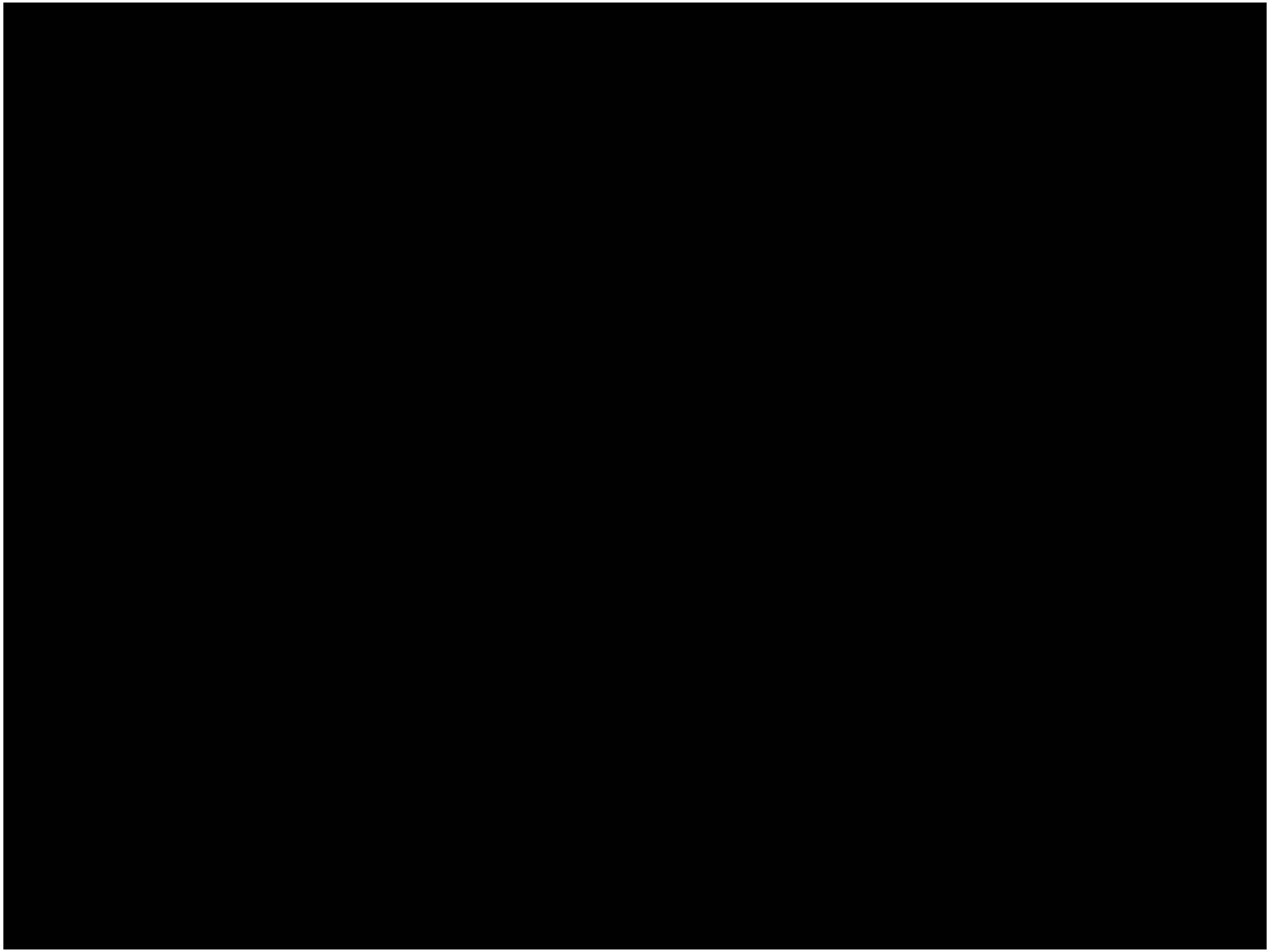
p is a random number in [0,1]

Deep Q-Learning

- Algorithm: using off-policy sample generation

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise state s_t
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(s_t, a; \theta)$
        Execute action a_t and observe reward r_t and state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
        Set $s_{t+1} = s_t$
        Sample random minibatch of transitions (s_t, a_t, r_t, s_{t+1}) from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } s_{t+1} \\ r_j + \gamma \max_{a'} Q(s_{t+1}, a'; \theta) & \text{for non-terminal } s_{t+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(s_t, a_j; \theta))^2$
    end for
end for

Example



Conclusions

- Reinformance learning is a powerful tool to train deep neural network with time-delayed reward signals
- It can be generally viewed as a trial-and-error approach to obtain the optimal networks
- We only briefly introduced value-based and policy-gradients-based methods. There are much more to explore along this direction!