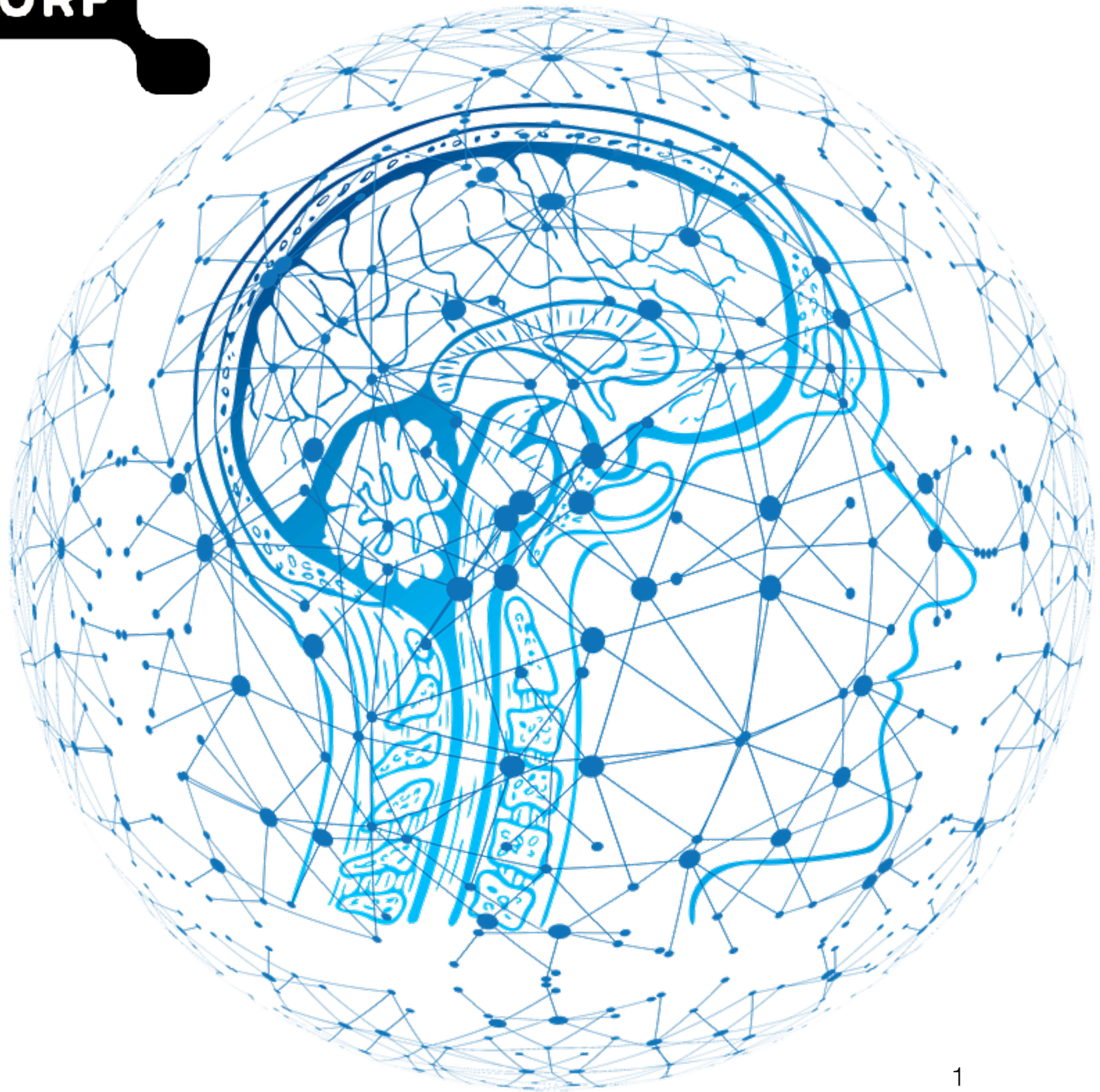
SURF



Introduction to Deep Learning

Reinforcement Learning

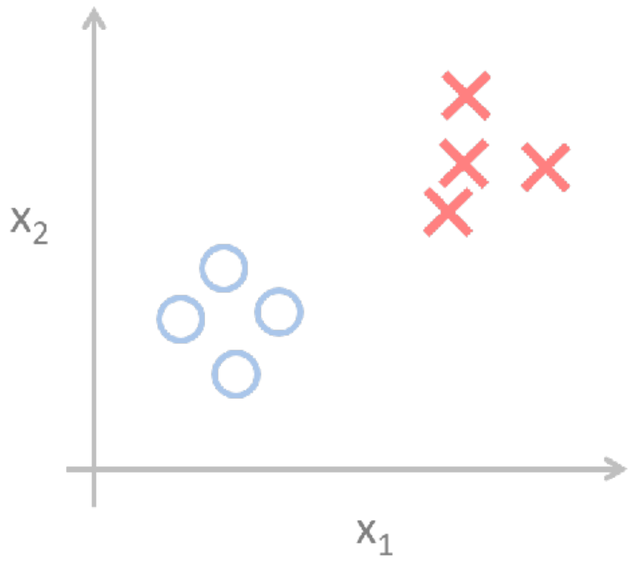
Maxwell Cai

maxwellcai.com

Categories of machine learning

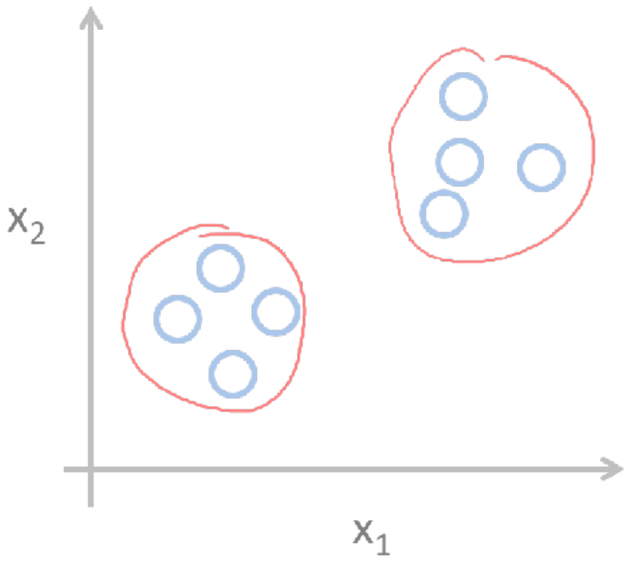
Supervised

Learn from the labels



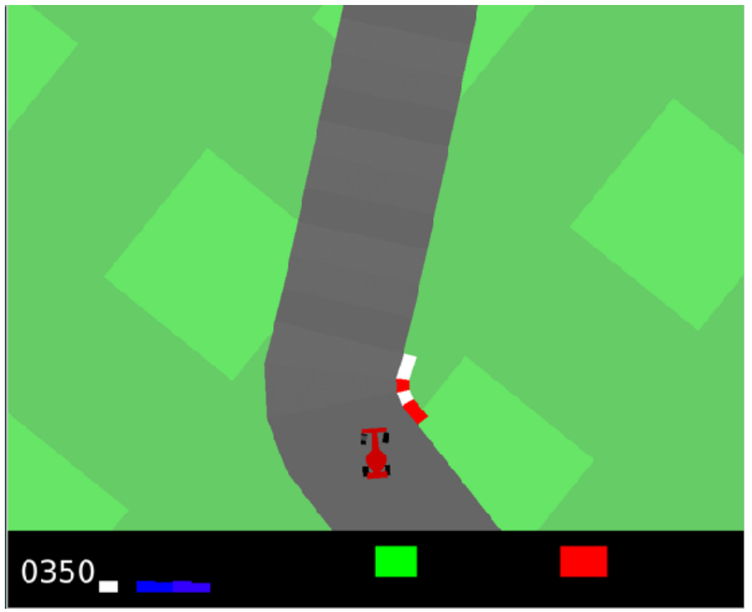
Unsupervised

Detect patterns in the data

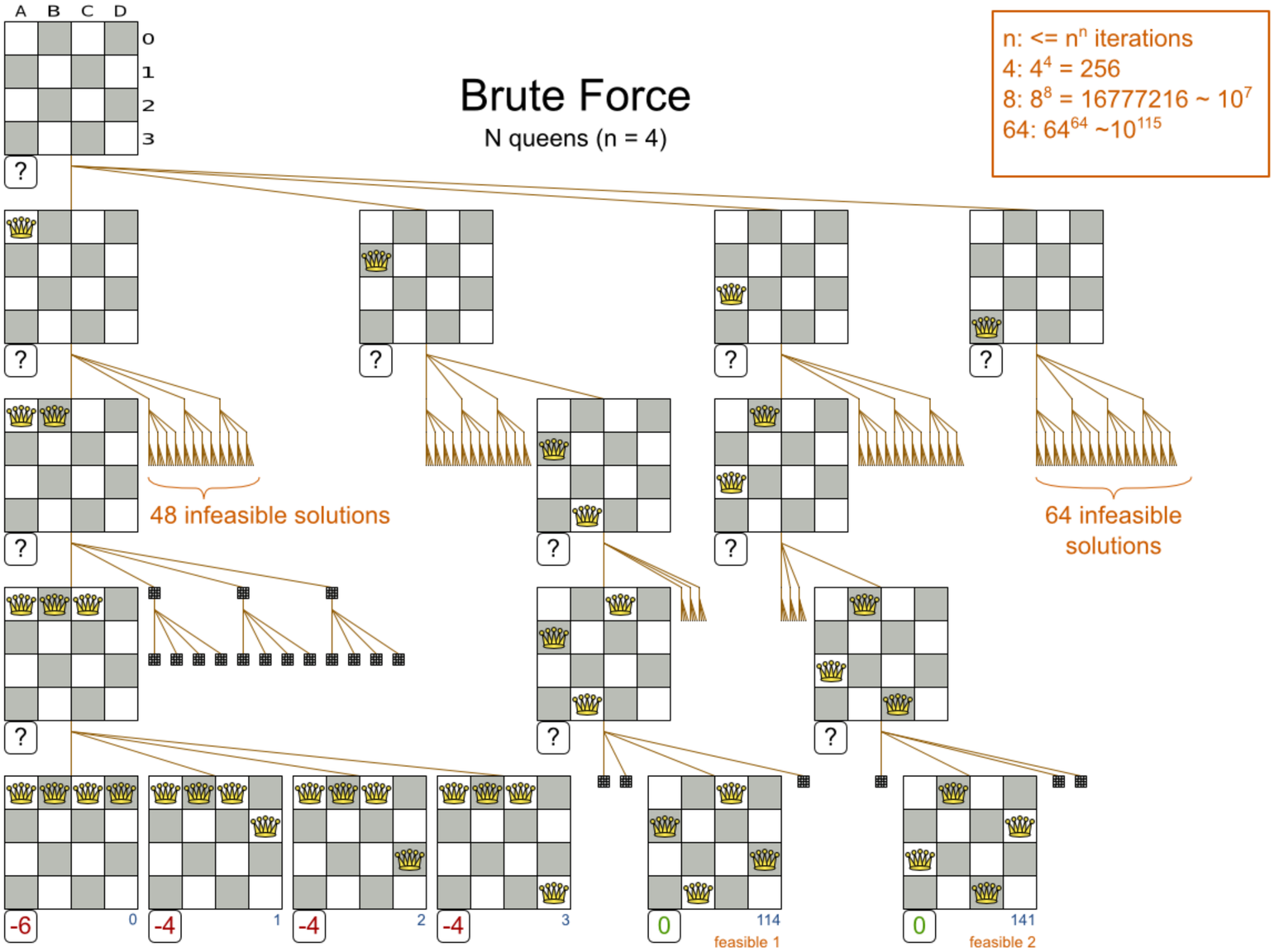


Reinforcement

Learn from mistakes



From Brute Force to AI



IBM's Deep Blue

Think about AlphaGo Zero

Are recurrent neural networks (RNNs) useful in this kind of applications?

- Utilise information in the **past** steps (RNNs can *probably* help)
- Make decision for the **near future** (RNNs can *probably* help)
- Make **sequences of decisions** for the **far future** that maximise gains (RNNs can't help)
- Deal with an **infinitely large** solution space (RNN can't help)

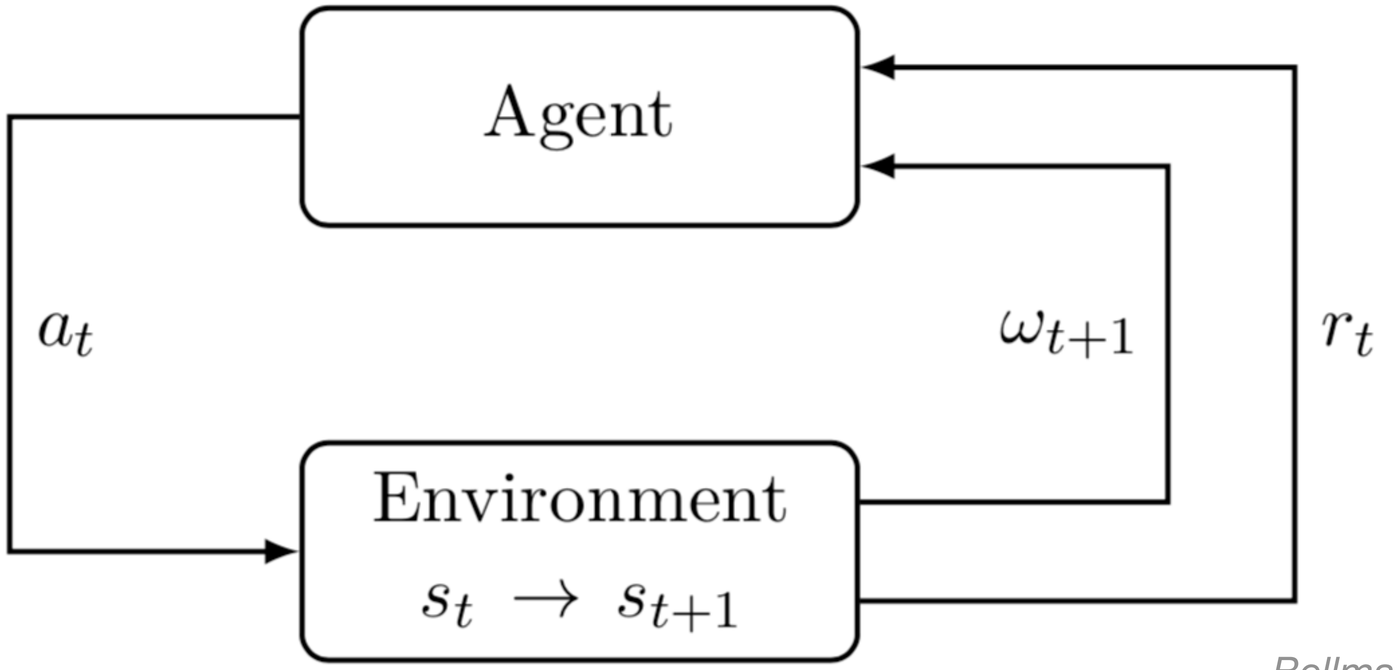
Reinforcement Learning

… is an area of machine learning that deals with **sequential decision-making**.

… is a task of learning how agents ought to take **sequences of actions** in an environment in order to maximise **cumulative** rewards.

… learns a good behaviour through its experience.

… forms a balance between the exploration/exploitation dilemma



State $s_t \in \mathcal{S}$

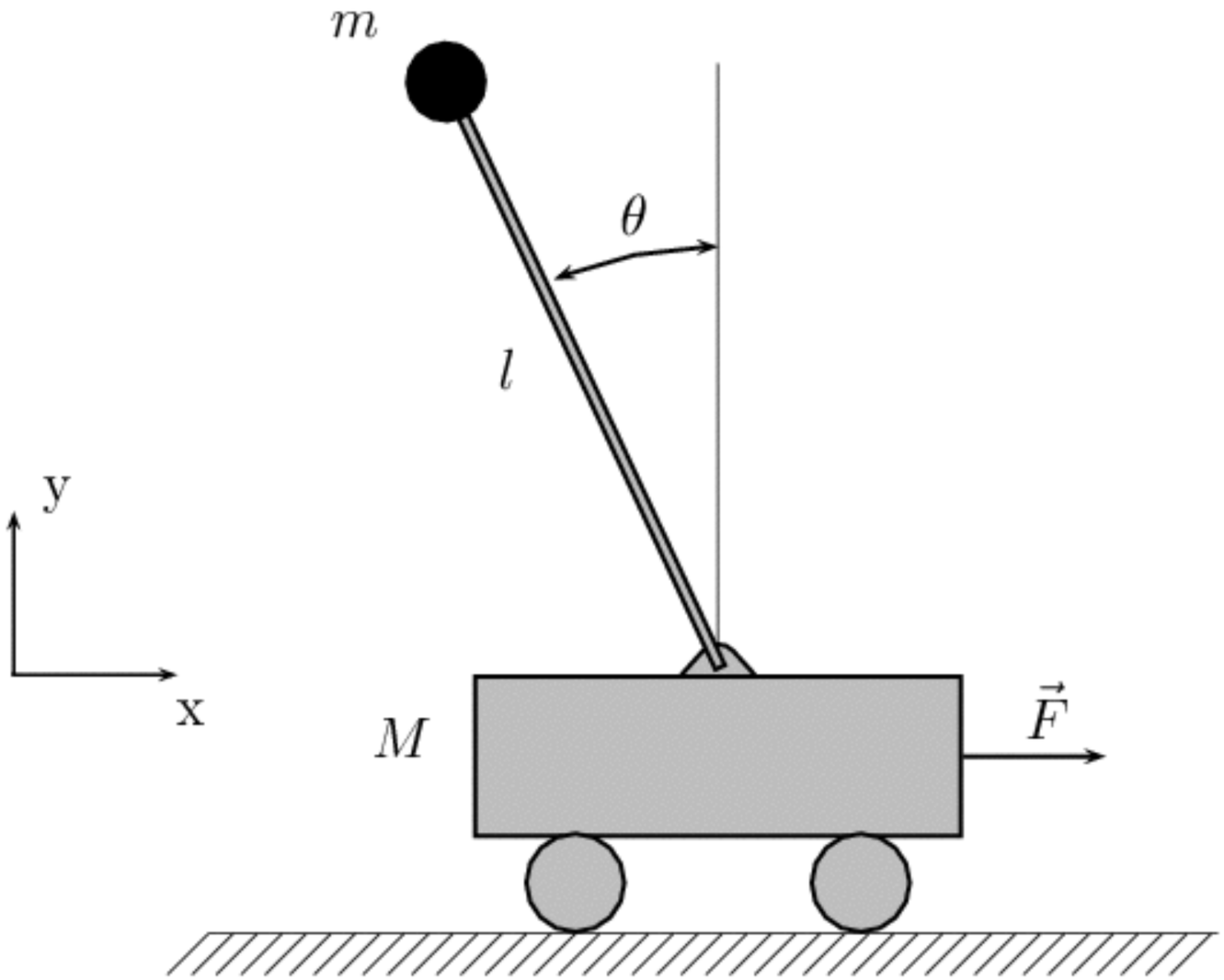
Action $a_t \in \mathcal{A}$

Observation $\omega_t \in \Omega$

Bellman (1957); François-Lavet et al. (2018), arXiv:1811.12560

Agent & Environment

RL agent learns to balance a cart pole



Observation

Type: Box(4)

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -41.8°	~ 41.8°
3	Pole Velocity At Tip	-Inf	Inf

Actions

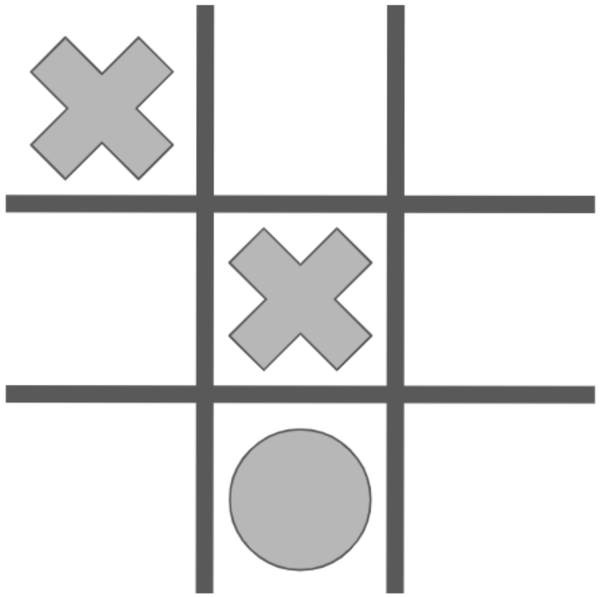
Type: Discrete(2)

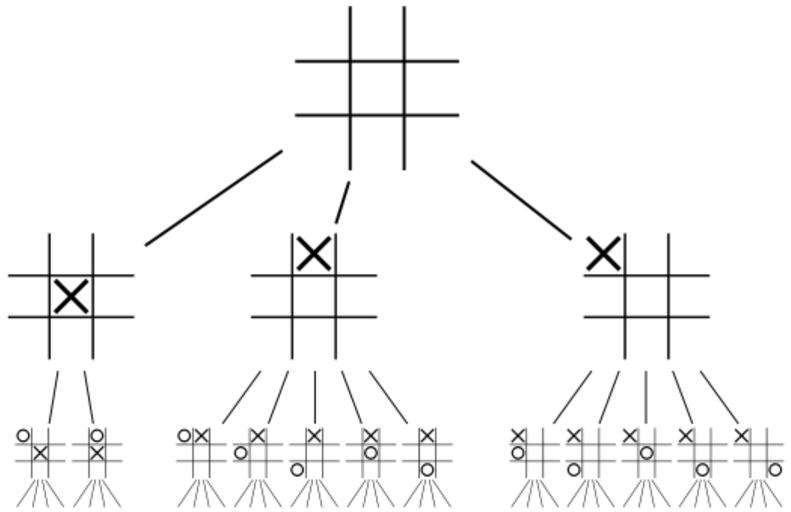
Num	Action
0	Push cart to the left
1	Push cart to the right

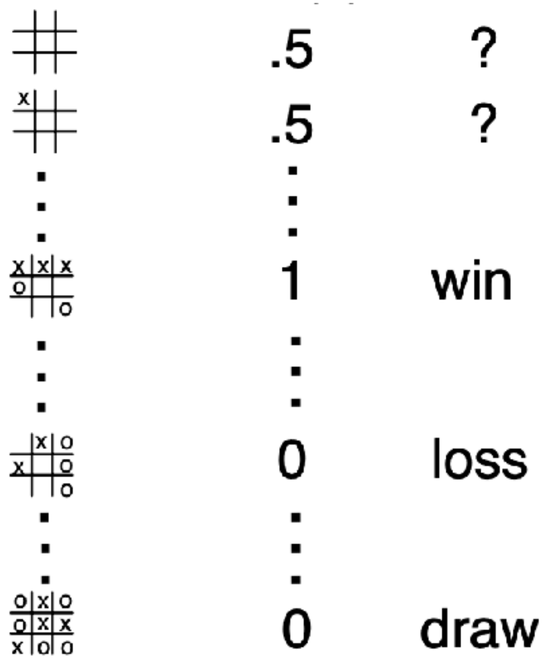
Image source / *movie source*

How do we formulate the process of sequential decision making?

Playing Tic-Tac-Toe







Strategy to move to the next step:

- Look ahead one step
- Take an action from one of the following strategies:
  - Randomly
  - Choose the action corresponding to the largest value (greedy)

The Q-learning Algorithm

The problem is that, we do **not** know the right strategy (i.e., **policy**) to take actions *a priori* …
And sometimes not even the environment is known to us.
We have to **explore** the environment…

Game Board:

Current state (*s*): **0 0 0 / 0 1 0**

Q Table: $\gamma = 0.95$

	0 0 0 / 1 0 0	0 0 0 / 0 1 0	0 0 0 / 0 0 1	1 0 0 / 0 0 0	0 1 0 / 0 0 0	0 0 1 / 0 0 0
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.4	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Image source: Shaked Zychlinski/Medium

The Q-learning Algorithm

In order to decide the best strategy given the current state s_t and action a_t, we can design a **table** Q to **query** the **future** expectation.

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Initial Q-table

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	...	...	...	...	...	...	...
	327	0	0	0	0	0	0
	...	...	...	...	...	...	...
	499	0	0	0	0	0	0

Updated Q-table

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	...	...	...	...	...	...	...
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	...	...	...	...	...	...	...
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Source: Wikipedia

Deep-Q Network (DQN)

Mnih et al. (2013); Image: Ankit Choudhary

State

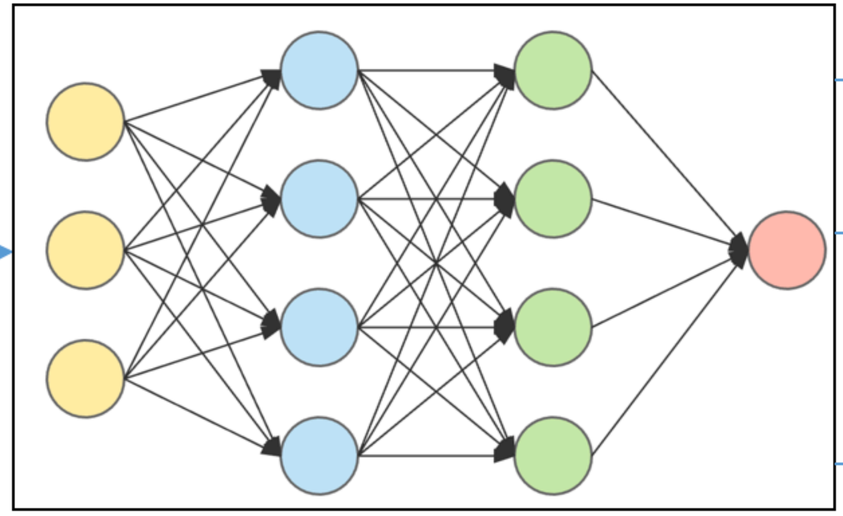
Action

Q Table	
State-Action	Value
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

Q-Value

The Q-table can quickly become too large.
How about replacing it with a neural network?

State



Q-Value Action 1

Q-Value Action 2

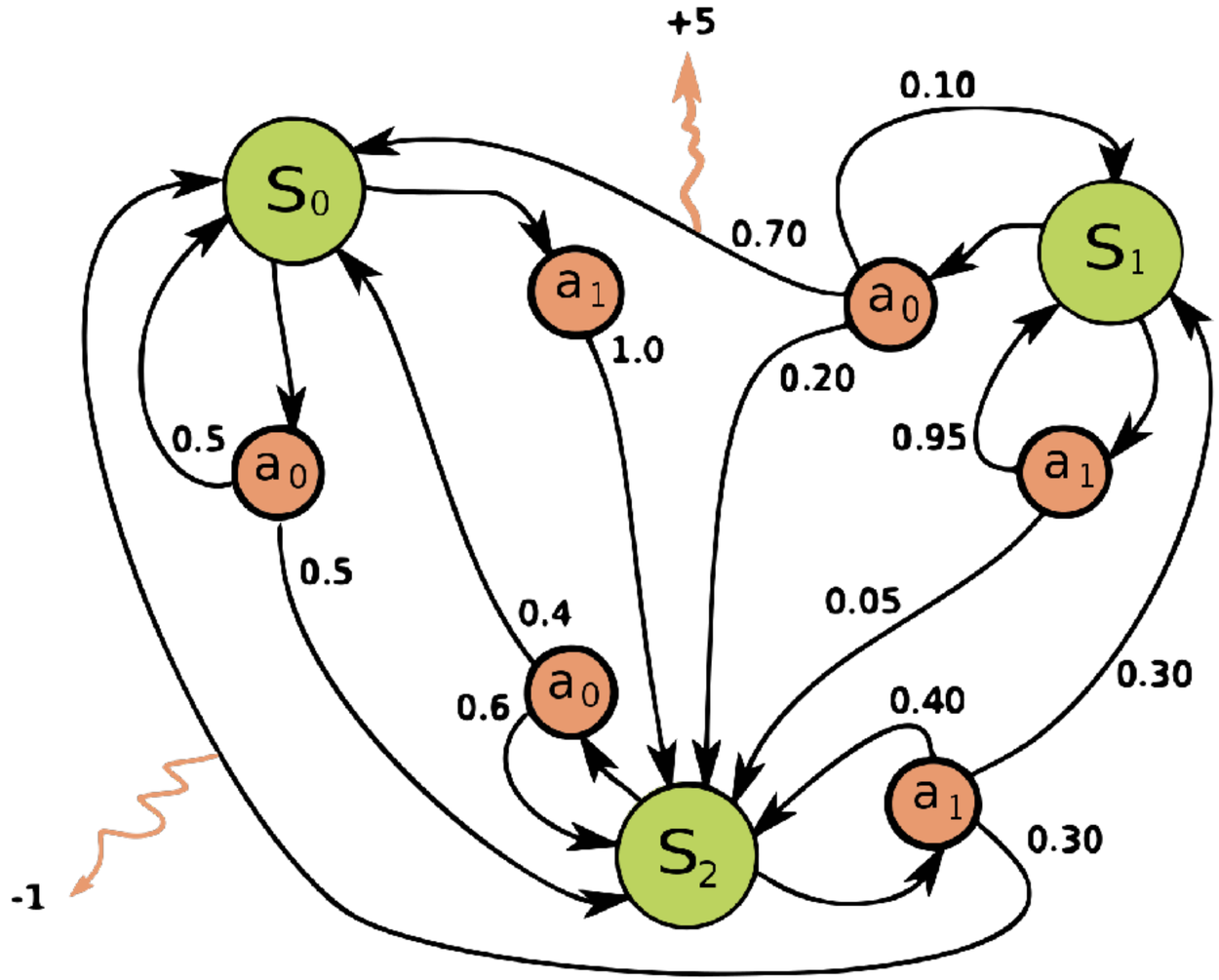
-

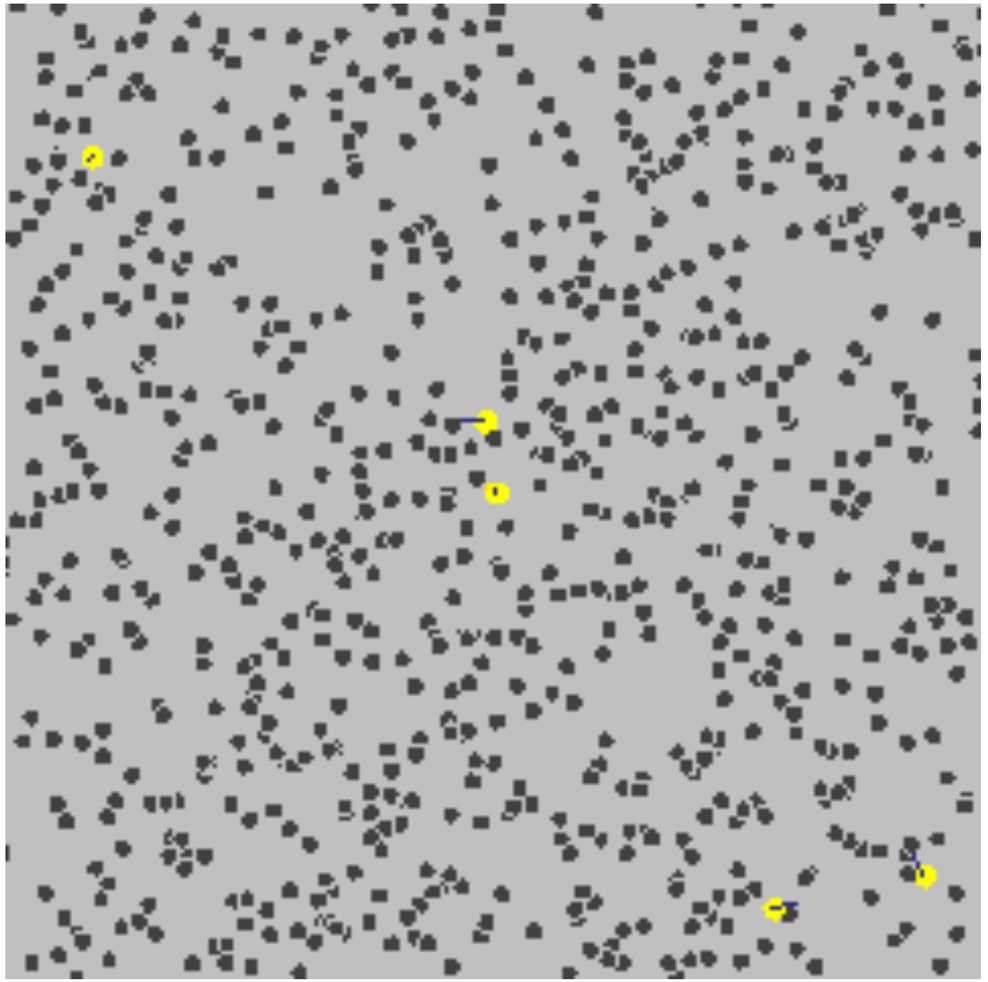
-

Q-Value Action N

Markov Decision Process

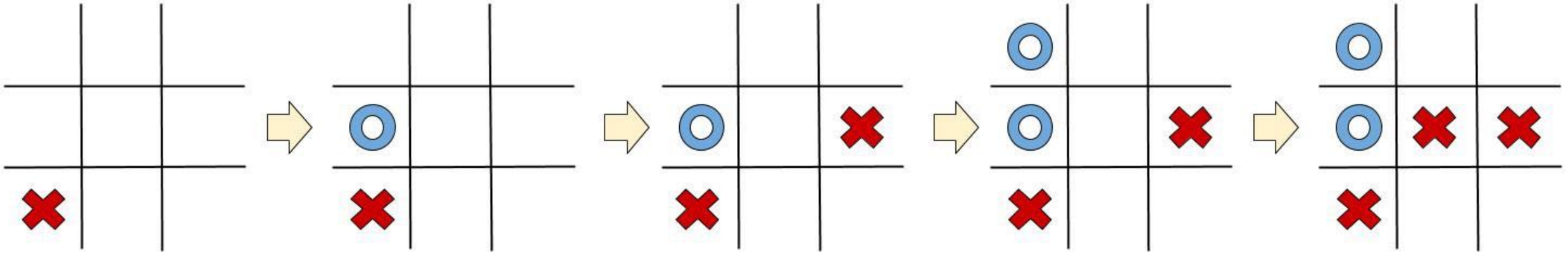
The **evolution** of the **environment** is usually modelled as a **Markov Decision Process (MDP)**.





Brownian motion has Markov property (memoryless). *Image source: Wikipedia*

Trajectories of an MDP



In MDP, different trajectories may lead to different cumulative rewards, but the objective is to optimise **from the current state onwards**.

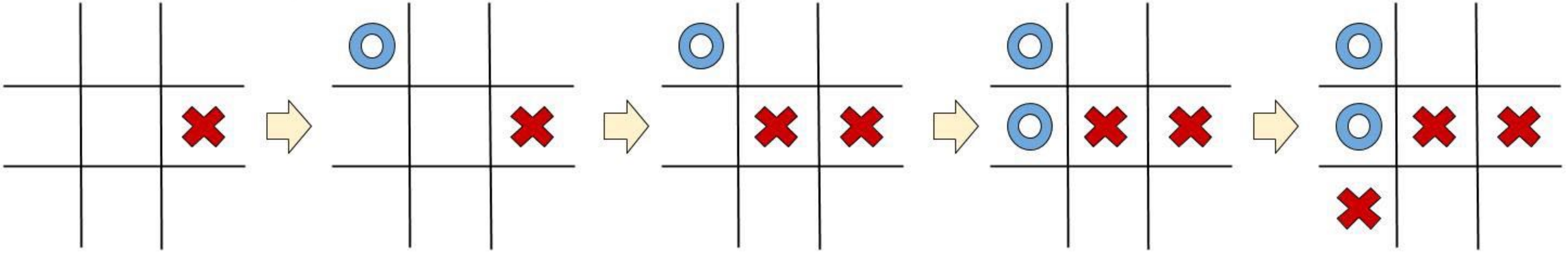


Image source: Shaked Zychlinski/Medium

Core Problem

… is to maximise the **cumulative** reward in an **episode**.

Bellman's Equation $$Q^{\pi}(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a')$$

Episodical Value Function

$$V^{\pi}(s) = R_t = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \dots] = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \,|\, s_t = s, \pi]$$

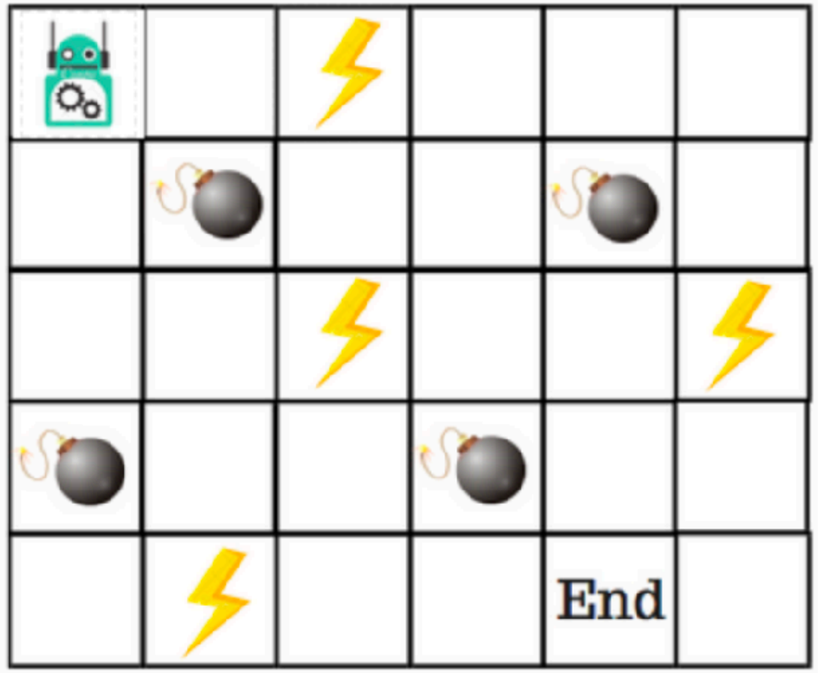
Take *action* into account:

$$Q^{\pi}(s,a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \,|\, s_t = s, a_t = a, \pi]$$

Greedy: $$\pi^*(s) = \arg\max_{a} Q(s,a)$$

Advantage function: $$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

(the advantage of taking action a in state s)



How does an agent explore the environment?

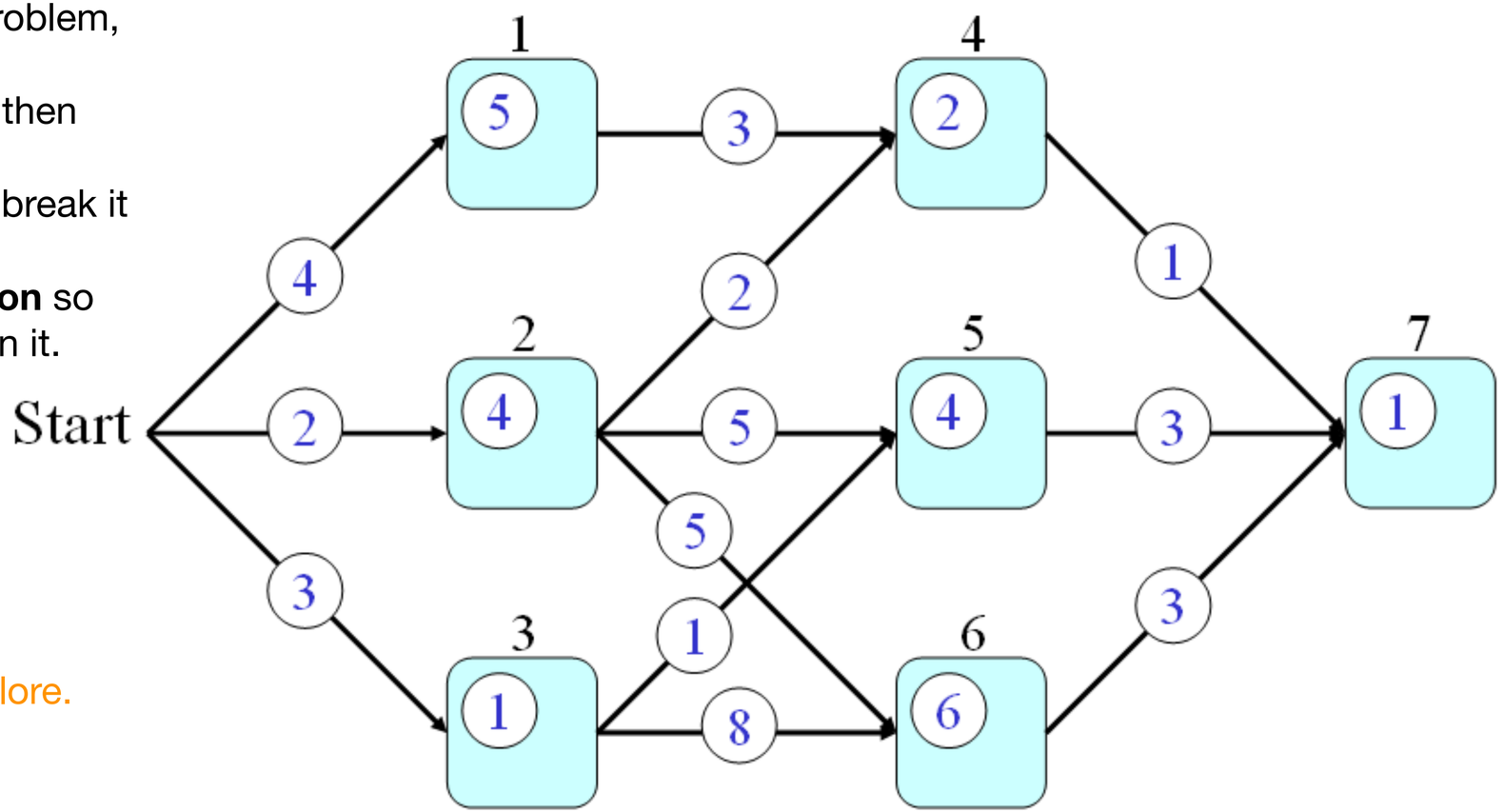
Dynamic Programming

If the underlying environment is known ***a priori***, we could potentially solve the problem using DP.

Cannot solve a big problem?

- The big problem can be defined as a smaller problem, plus a **trivial** step.
- If the smaller problem can be solved optimally, then the original problem can be solved;
- If the small problem is still too big, **recursively** break it into even smaller ones, until directly solvable.
- If an optimal solution is found, **store the solution** so that the bigger problem can be solved based on it.

$$F(0) = a$$
$$F(n+1) = f(F(n))$$

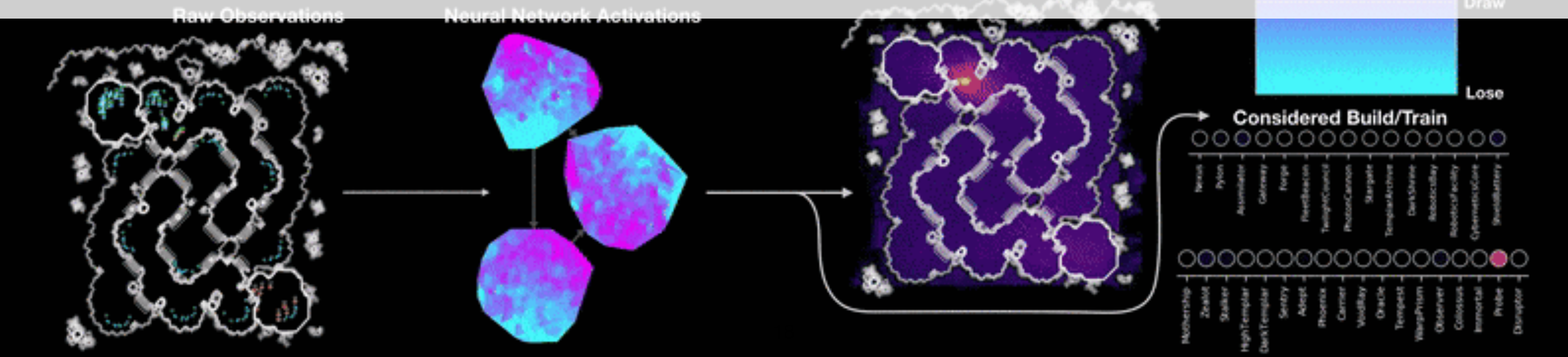


In DP, the environment is **known**, no need to explore.
Directly find the optimal solution.

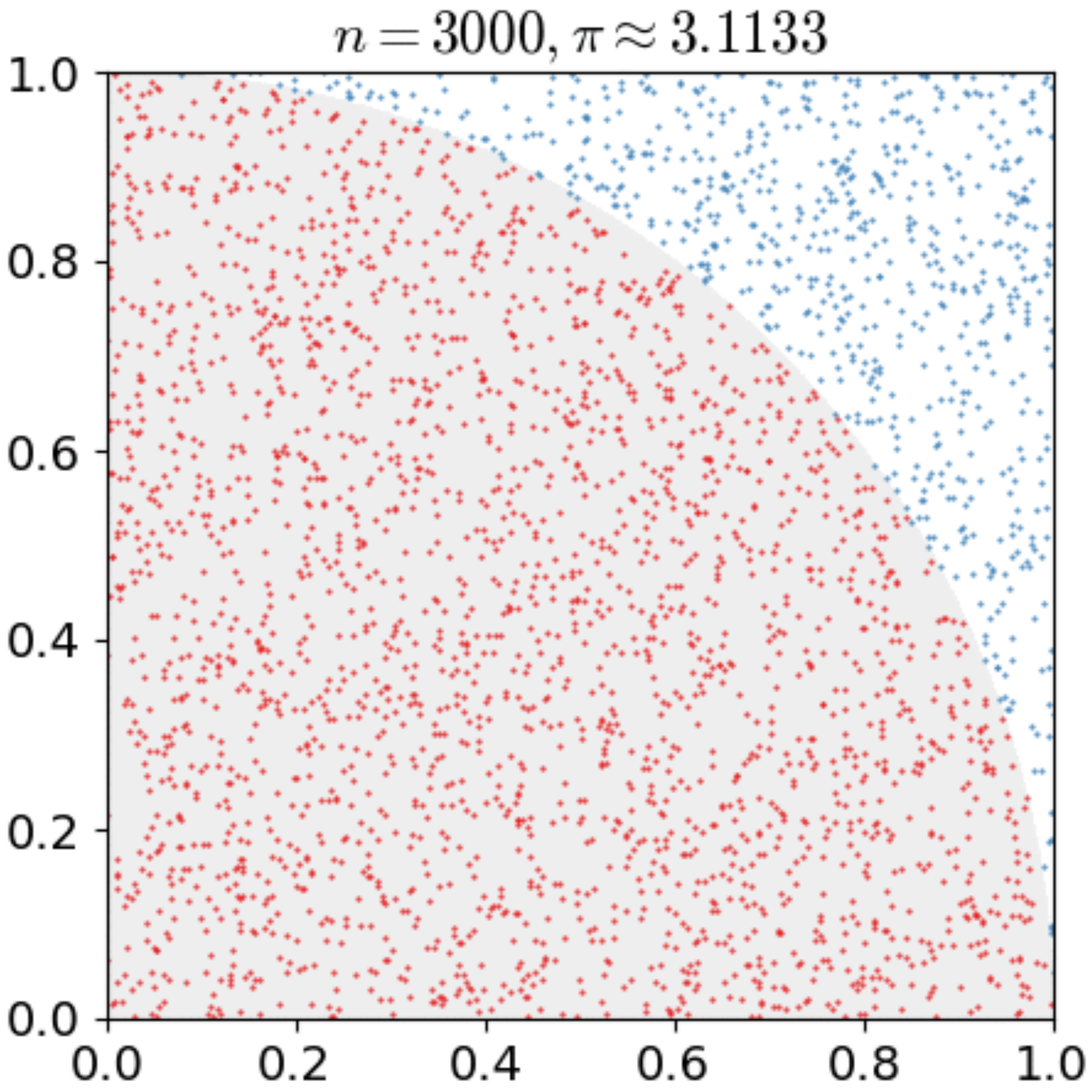
"Those who cannot remember the past are condemned to repeat it." — Dynamic Programming

Playing StarCraft with DP?

No-no.



Monte-Carlo Search



- Play a bunch of games; record the experience.
- Collect the rewards **at the end of the episode** and calculate the maximum expected reward.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

If the environment is **unknown**, we will have to explore it by **interacting** with it in **many different ways.** The more ways we try, the more clear we understand the environment.

Temporal Difference Learning

Monte Carlo $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$

Requires a **complete** episode

Final reward

Sometimes, it is too expensive or even impossible to finish a full episode.

TD $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$

Reward for next time step

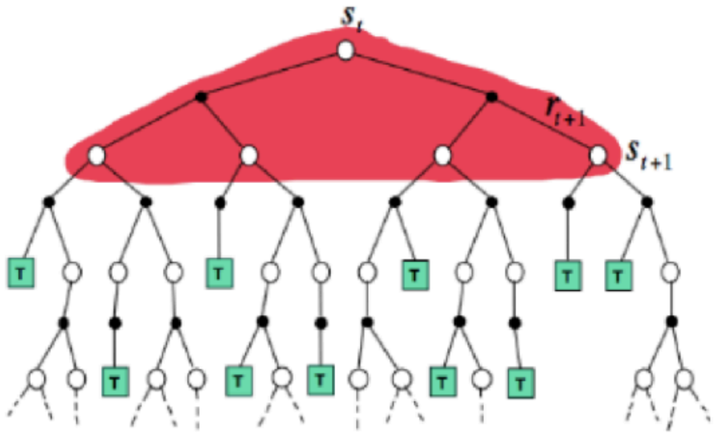
Learns from an **incomplete** episode through **bootstrapping**

Comparison of Value-based methods

Environment known

Dynamic Programming

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1}) \right]$$

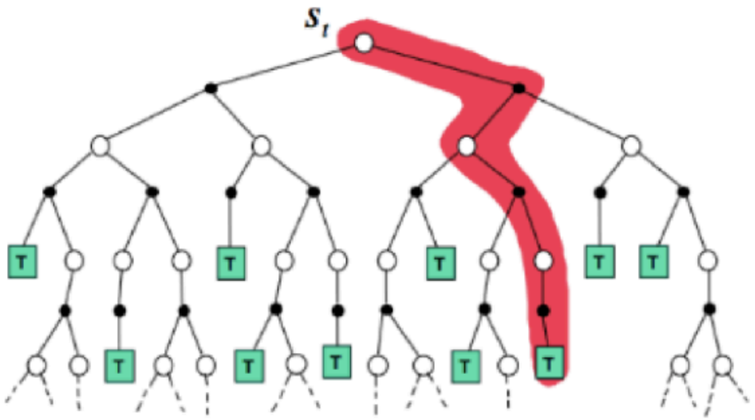


Optimal substructure

Environment unknown
Exploration cheap

Monte-Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$

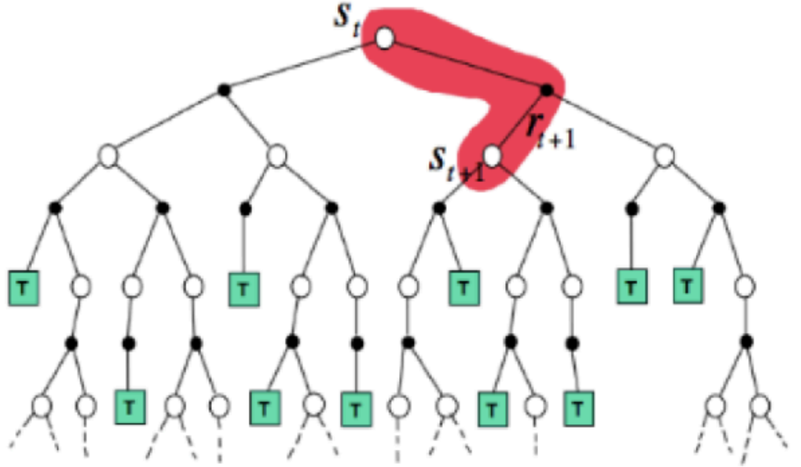


Sample a few complete trajectories

Environment unknown
Exploration expensive

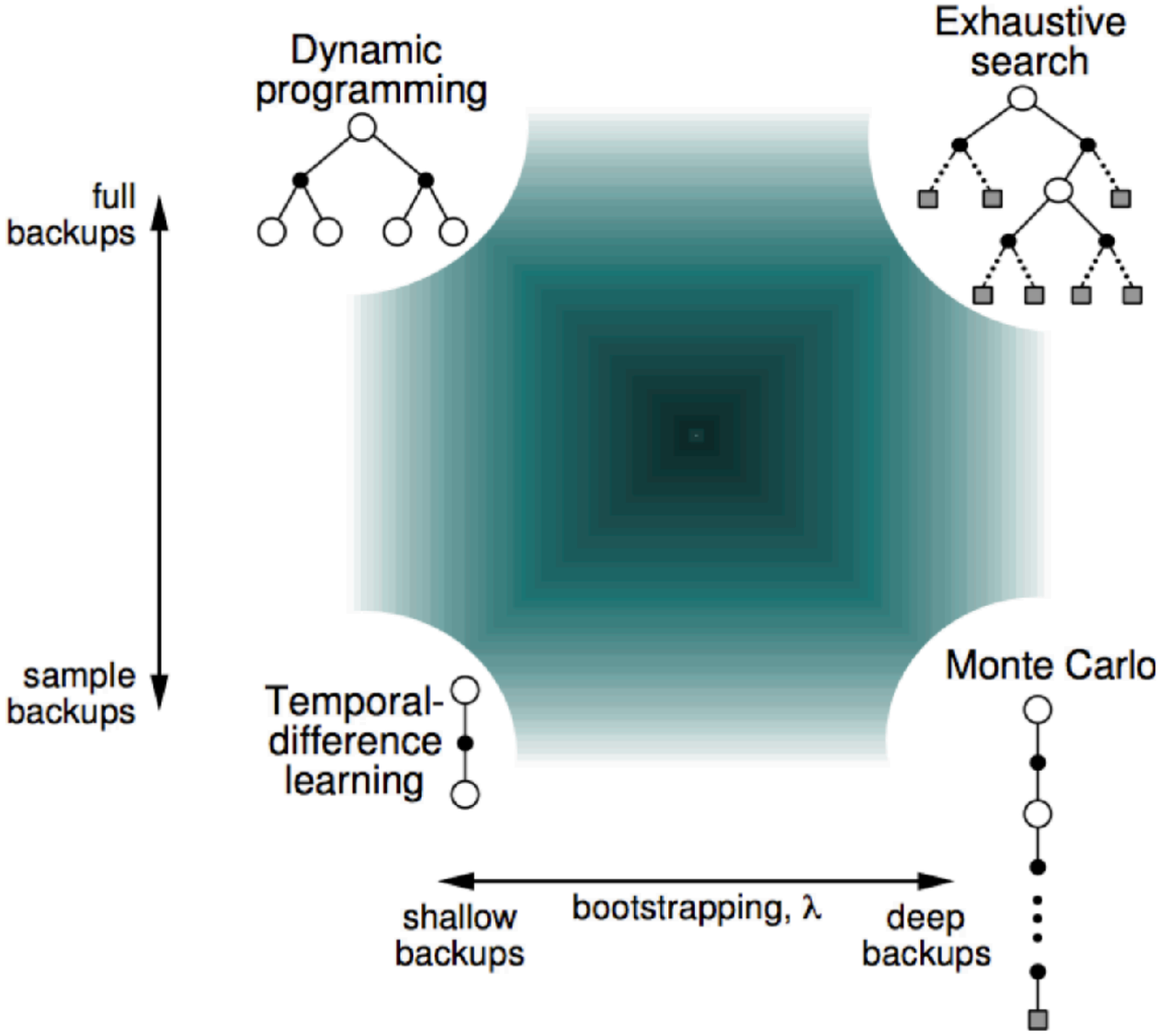
Temporal-Difference

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$



Sample a few steps

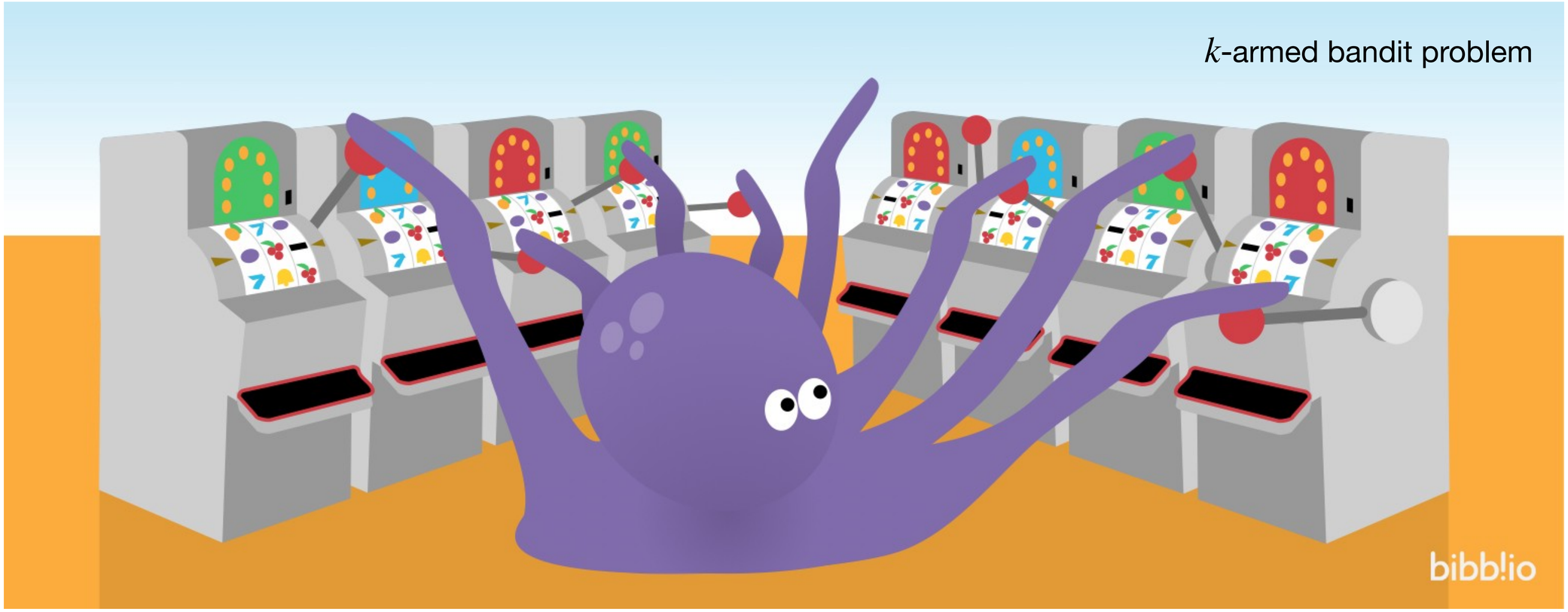
Image source: David Silver


Dynamic programming
Exhaustive search
full backups
sample backups
Temporal-difference learning
Monte Carlo
shallow backups
bootstrapping, λ
deep backups

Exploration vs. Exploitation

The exploration-exploitation dilemma

Through **initial exploration**, we obtain a policy (not necessarily optimal), which is at least better than naive guesses. Should we use this existing sub-optimal policy to solve the problem (some rewards guaranteed), or **keep exploring** until we get an even better policy that leads to better results (not guaranteed)?



Greedy vs. ϵ-greedy

We likely want to find a **tradeoff** between exploration and exploitation.

$p \sim [0,1]$

$p \geq \epsilon$

$p < \epsilon$

Random number generator

Exploitation
Keep using the current (sub-optimal) policy

Exploration
Try other options and see if any luck to get better results

Pros: very simple to implement;
Cons: how do we select a proper ϵ?

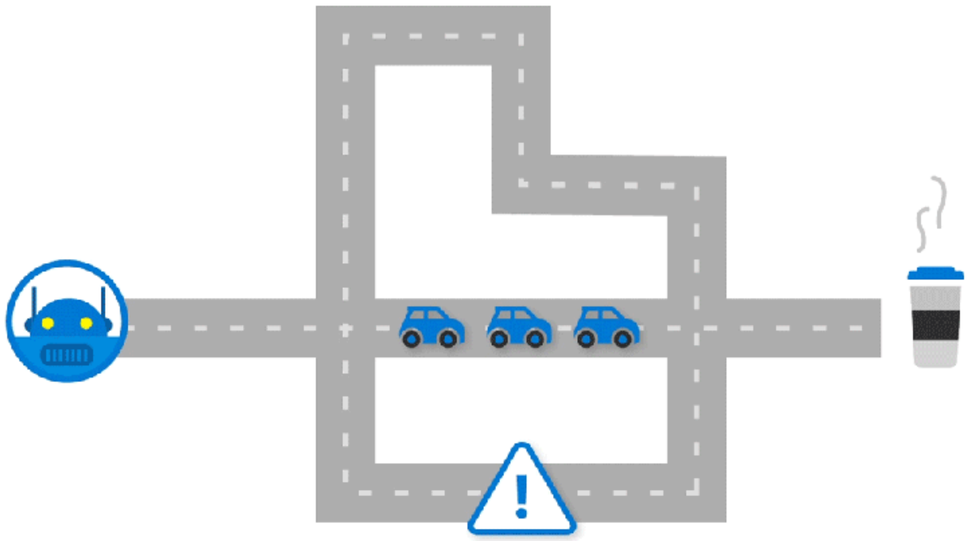
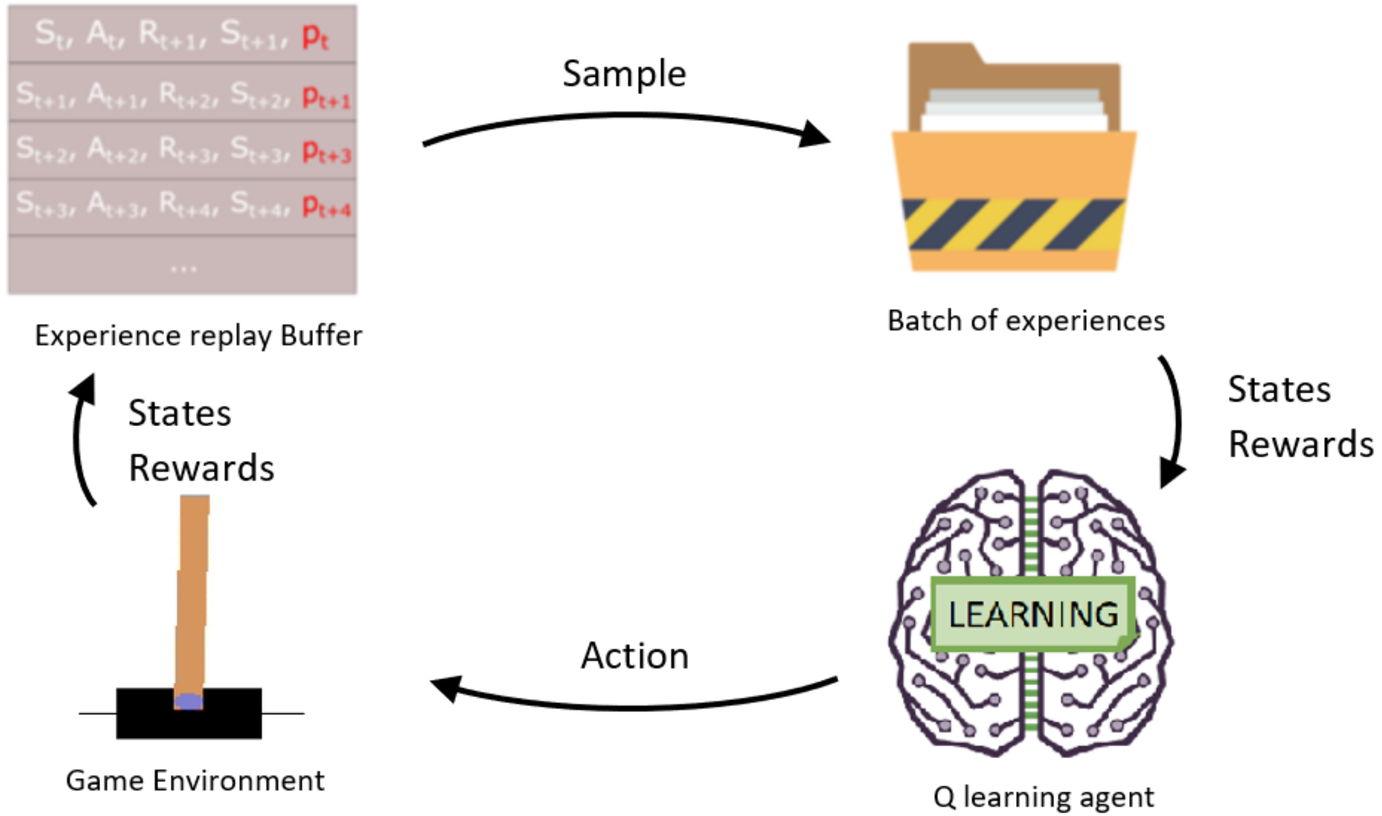


Image source: Microsoft Research

Stabilise the Training with Replay Buffer

There are **good** actions and **bad** actions.
It is **harmful** to update the network with **bad** actions.
SGD requires data to be independent and identically distributed.

Instead of using the **latest** experience to update the Q-network, we **accumulate** the experience in a **replay buffer** (aka experience buffer), and update the network by **randomly sampling** the replay buffer.



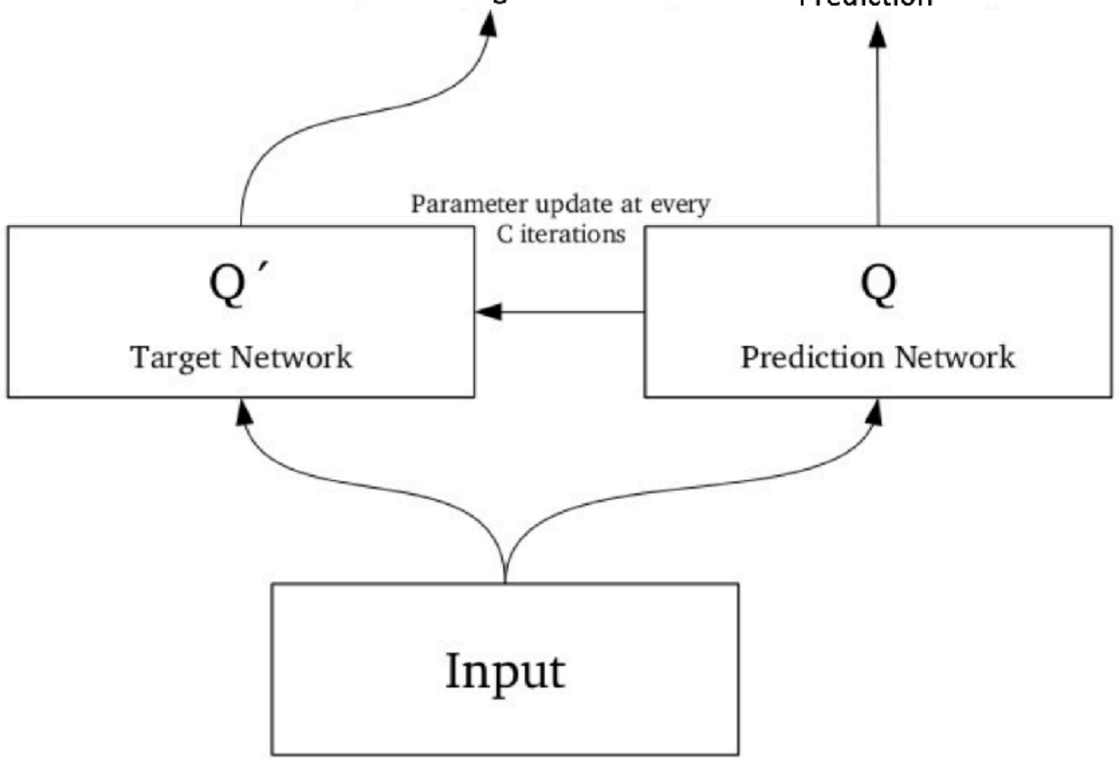
Stabilise the Training with a Target Network

Alternatively, we can deter the update of our Q-network.

$$\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_i^-)-Q(s,a;\theta_i)\right)^2\right]$$

Target

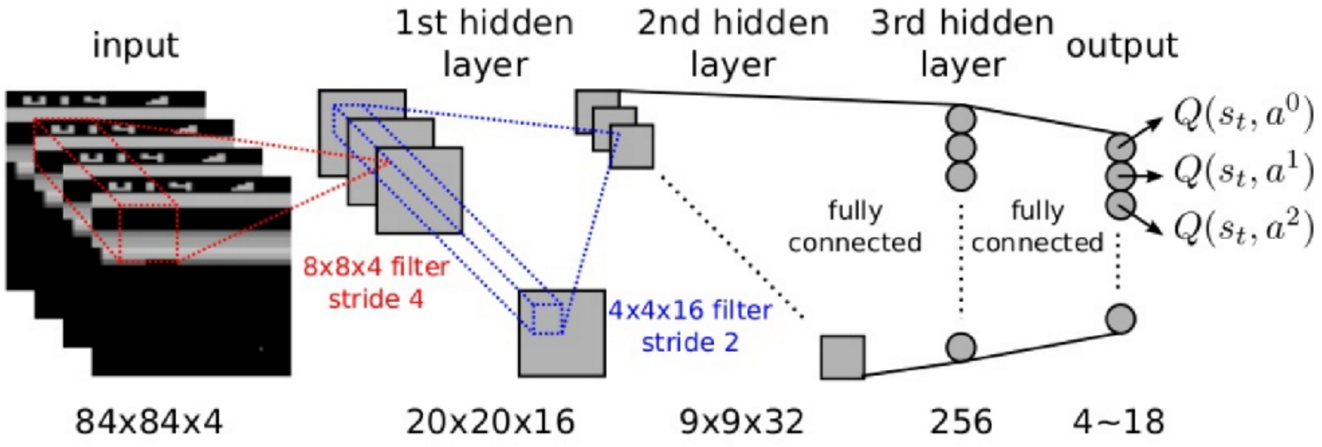
Prediction



Intuition: I have an idea of how to play this well, I'm going to try it out for a bit until I find something better.

Playing Atari with a DQN

With a CNN+DQN, an agent accepts high-dimensional visual inputs and play the Atari 2600 game at a superhuman level!



Credit: leonardoaraujosantos.gitbook.io; Mnih et al. (2013); DeepMind

What if the problem is too hard/expensive to explore?

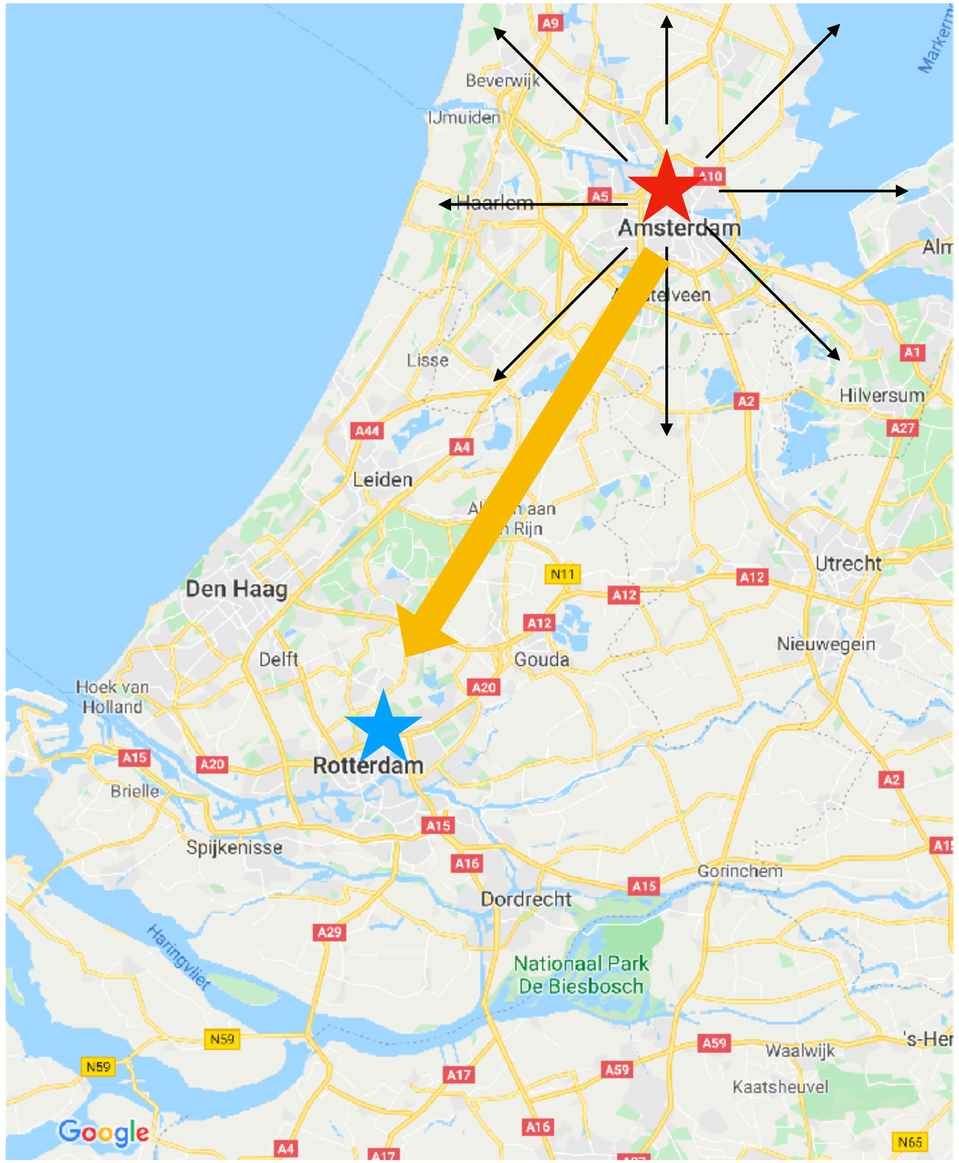
Then don’t explore it. Try to discover some general guidelines (policy).

Policy-based RL

Example: Traveling from Amsterdam to Rotterdam
Constraint: no prior knowledge, no maps/GPS.

Solution 1 (value-based RL):
1. Travel a small distance along all possible directions;
2. Evaluate the distance to Rotterdam.
3. Repeat (1,2) until Rotterdam is reached.

Solution 2 (policy-based RL):
1. Ask someone around
2. The person being asked gives a policy “go southwest for 80 km”
3. Execute the policy
4. Check if Rotterdam is reached. If not, repeat (1,2,3).



Policy-based RL

Constructing the policy from the value function can sometimes be very expensive. Any shortcut?

Optimise the policy **directly** (without consulting the value function): **policy-based RL**

Deterministic & Stochastic Policies

Deterministic Policy: state-action mapping

$$a = \pi(s)$$

```
action = policy(state)
```

Stochastic Policy: probability distribution of state-action pairs

$$\mathbb{P}[A_t = a \,|\, S_t = s] = \pi(a \,|\, s)$$

```
proba(action) = policy(state)
```

Takes uncertainties into account

Rock-paper-scissors

Formulation of Policy Gradients

To quickly find the optimal policy, we follow its **gradients**. But *how*?

Recall that

$$\nabla J(\tau) \equiv \frac{\nabla J(\tau)}{J(\tau)} J(\tau) = J(\tau) \nabla \log[J(\tau)]$$

Objective

$$J(\tau) = \mathbb{E}[r(t)] = \int \underbrace{\pi(\tau)}_{\text{Probability}} r(\tau) d\tau$$

Trajectory $\tau = (s_1, a_1, s_2, a_2, \ldots, s_t, a_t)$

Take the grad on both sides

$$\nabla J(\tau) = \nabla \int \pi(\tau) r(\tau) d\tau = \int \nabla \pi(\tau) r(\tau) d\tau = \int \underbrace{\pi(\tau)}_{\text{Probability}} \nabla \log \pi(\tau) r(\tau) d\tau = \mathbb{E}[\nabla \log \pi(\tau) r(\tau)]$$

We can calculate the policy grad by **sampling** trajectories and calculate their **expectations**!

Take the log On both sides

$$\pi(\tau) = \pi(s_1, a_1, s_2, a_2, \ldots, s_t, a_t) = p(s_1) \prod_{t=1}^{T} \pi(a_t \mid s_t) p(s_{t+1} \mid s_t, a_t)$$

$$\log \pi(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi(a_t \mid s_t) + \log p(s_{t+1} \mid s_t, a_t)$$

Policy Gradients (cont.)

$$\log \pi(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi(a_t \mid s_t) + \log p(s_{t+1} \mid s_t, a_t)$$

Take the grad

$$\nabla \log \pi(\tau) = \nabla [\log p(s_1) + \sum_{t=1}^{T} \log \pi(a_t \mid s_t) + \log p(s_{t+1} \mid s_t, a_t)] = \nabla [\sum_{t=1}^{T} \pi(a_t \mid s_t)]$$

Starting point

Ending point

Recall that

$$\nabla J(\tau) = \mathbb{E}[\nabla \log \pi(\tau) r(\tau)]$$

max. log likelihood (measuring how likely the trajectory τ is following the current policy; similar to KL divergence)

Therefore

$$\nabla J(\tau) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla \log \pi(a_{i,t} \mid s_{i,t}) \right) \left(\sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right)$$

Cumulative reward

Policy update

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

Gradient ascent!

The REINFORCE algorithm

… is a **Monte-Carlo Policy-Gradient** method

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t,\boldsymbol{\theta})$

The policy is updated **after** each episode.
Good policy can be learned from a **large sample** of episodes.

Williams, R. J. (1992); Sutton & Barto (2017)

REINFORCE with Baseline

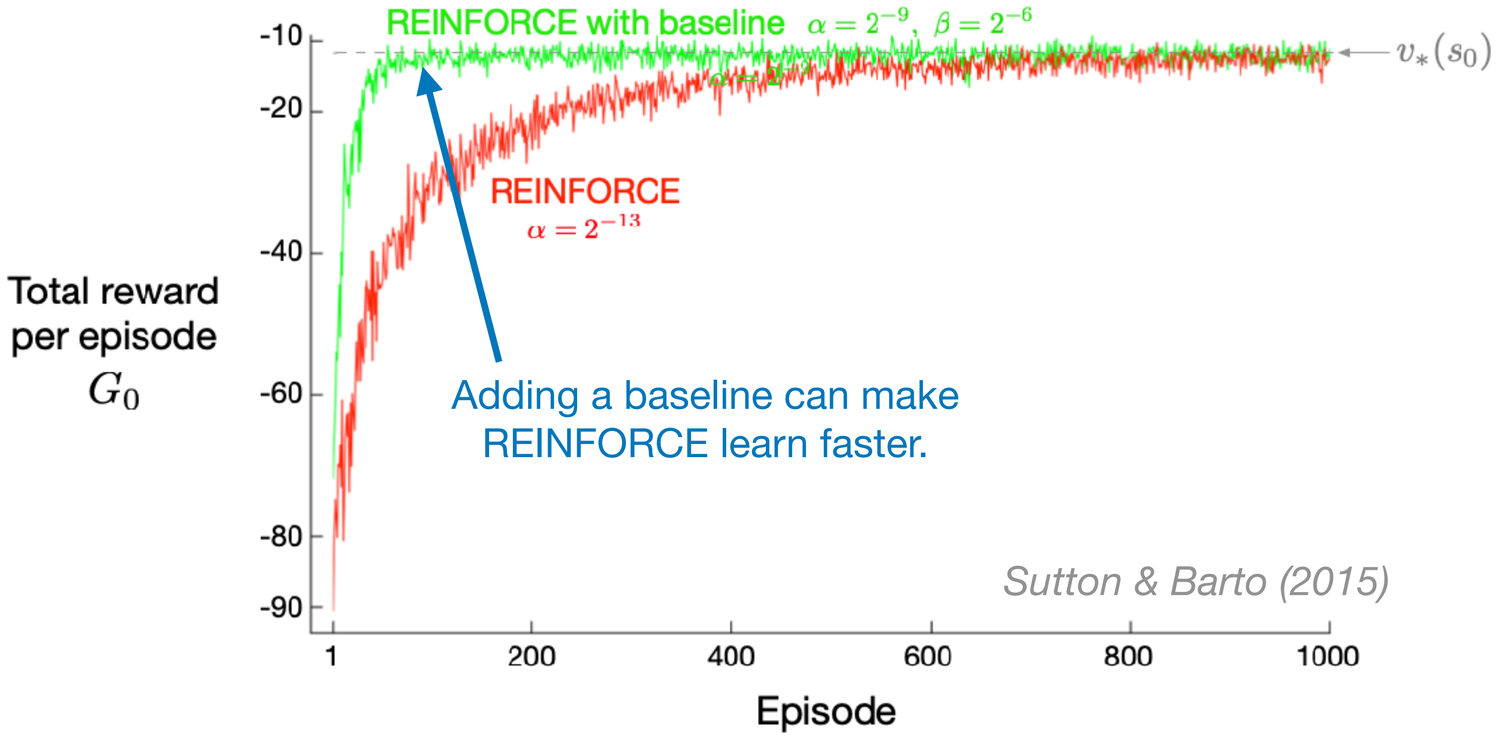
Monte-Carlo based approaches typically suffer from **high variance**
The REINFORCE algorithm is no exception, because the policy is updated by taking **random** samples.

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla \log_\pi \left(a_t \,|\, s_t, \theta\right) \quad \Rightarrow \quad \theta_{t+1} = \theta_t + \alpha [G_t - b(s_t)] \nabla \log_\pi \left(a_t \,|\, s_t, \theta\right)$$

Common practice: **normalise** the return (i.e., **whitening**)

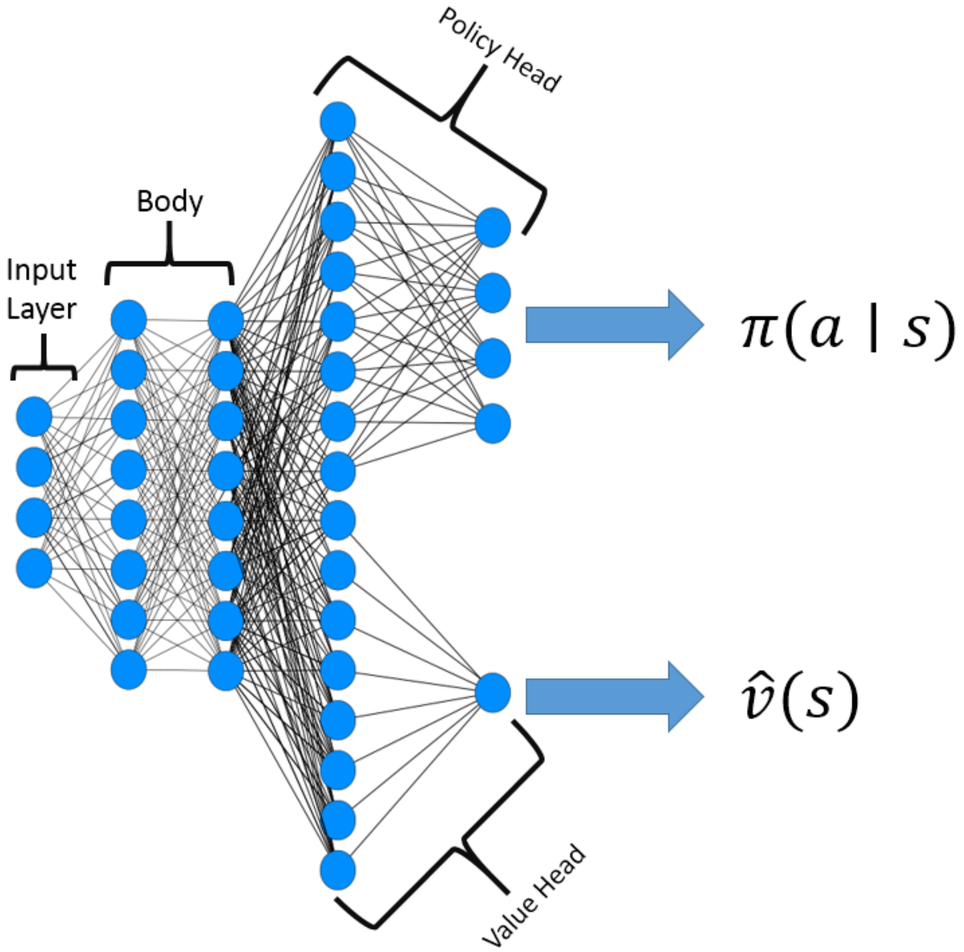
$$G_t^* = \frac{G_t - \bar{G}}{\sigma_G}$$

$$\theta_{t+1} = \theta_t + \alpha G_t^* \nabla \log_\pi \left(a_t \,|\, s_t, \theta\right)$$



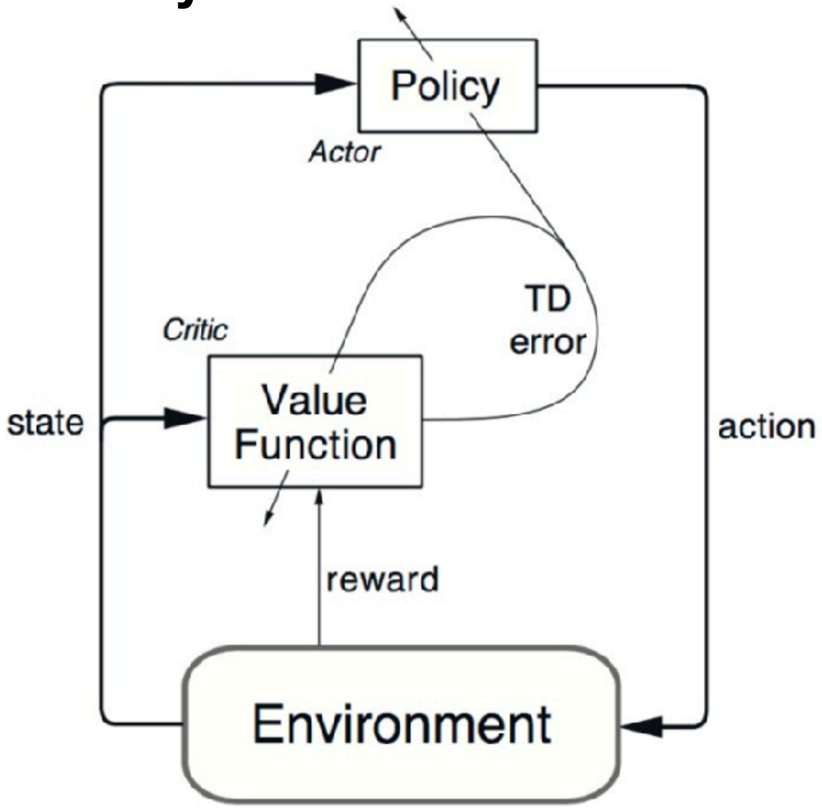
Actor-Critic Methods

If we are learning a **policy**, why not learn a **value** function **simultaneously**?
Can we use the value function to **guide** the update of the policy?
If so, we can update our policy **per-step** instead of **per-episode**.



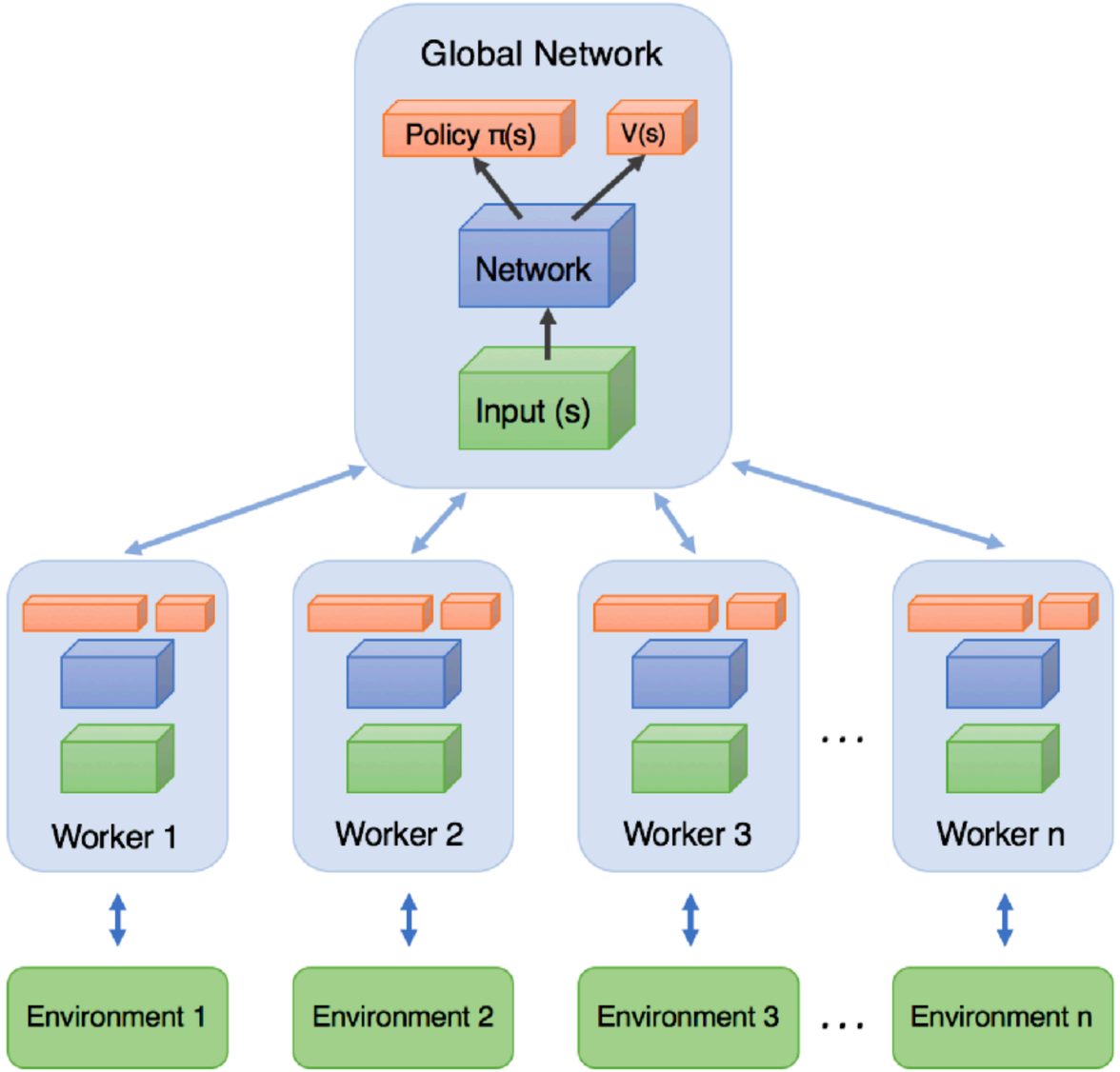
$\pi(a \mid s)$

$\hat{v}(s)$



$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, G_t\right] \quad \text{REINFORCE}$$
$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; Q^w(s,a)\right] \quad \text{Q Actor-Critic}$$
$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; A^w(s,a)\right] \quad \text{Advantage Actor-Critic}$$
$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; \delta\right] \quad \text{TD Actor-Critic}$$

Parallel Policy Updating



Asynchronous Actor-Critic Agents (A3C)

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared θ, θ^-, *and counter* $T = 0$.
Initialize thread step counter $t \leftarrow 0$
Initialize target network weights $\theta^- \leftarrow \theta$
Initialize network gradients $d\theta \leftarrow 0$
Get initial state s
repeat
  Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$
  Receive new state s' and reward r

$$y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$$

  Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
  $s = s'$
  $T \leftarrow T + 1$ and $t \leftarrow t + 1$
  if $T \mod I_{target} == 0$ **then**
    Update the target network $\theta^- \leftarrow \theta$
  end if
  if $t \mod I_{AsyncUpdate} == 0$ or s is terminal **then**
    Perform asynchronous update of θ using $d\theta$.
    Clear gradients $d\theta \leftarrow 0$.
  end if
until $T > T_{max}$

Mnih et al. (2016); Medium (Emergent // Future)

Model-based & Model-free Policy

Example: Work-home commute

Task: Find the best way to go home Friday afternoon

Objective: Avoid traffic jams / minimise travel time

Solution 1: Model-based

- Based on prior experience, build a map
- Mark roads with traffic jams at the rush hours
- Avoid them by finding alternative routing strategies

Solution 2: Model-free

- Based on prior experience, build a list of action sequences

$$
\begin{aligned}
&[(s_{11}, a_{11}), (s_{12}, a_{12}), \ldots, (s_{1t}, a_{1t}) \rightarrow \quad R_1] \\
&[(s_{21}, a_{21}), (s_{22}, a_{22}), \ldots, (s_{2t}, a_{2t}) \rightarrow \quad R_2] \\
&\qquad\qquad\qquad\qquad\qquad\qquad \ldots \\
&[(s_{n1}, a_{n1}), (s_{n2}, a_{n2}), \ldots, (s_{nt}, a_{nt}) \rightarrow \quad R_n]
\end{aligned}
$$

- Find the sequence that minimise the traffic jam or travel time

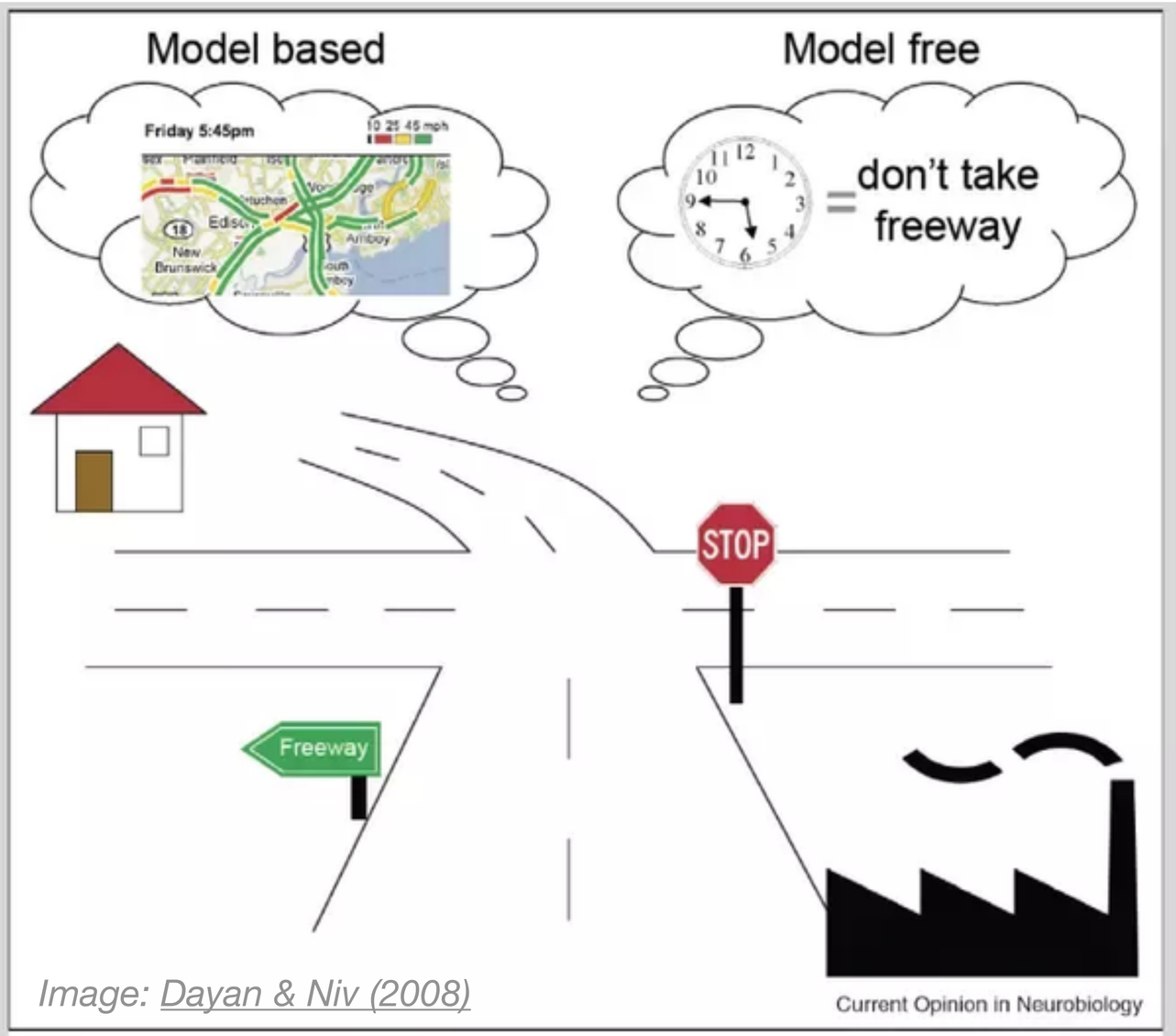


Figure 1: Two ways to choose which route to take when traveling home from work on friday evening.

What if the action space is continuous?

There are infinitely amount of actions…

Gaussian Policy Parameterisation for Continuous Actions

Instead of outputting a specific action, outputting a **distribution** of actions.
But how? **Parameterisation**.

Common practice: use the Gaussian PDF

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

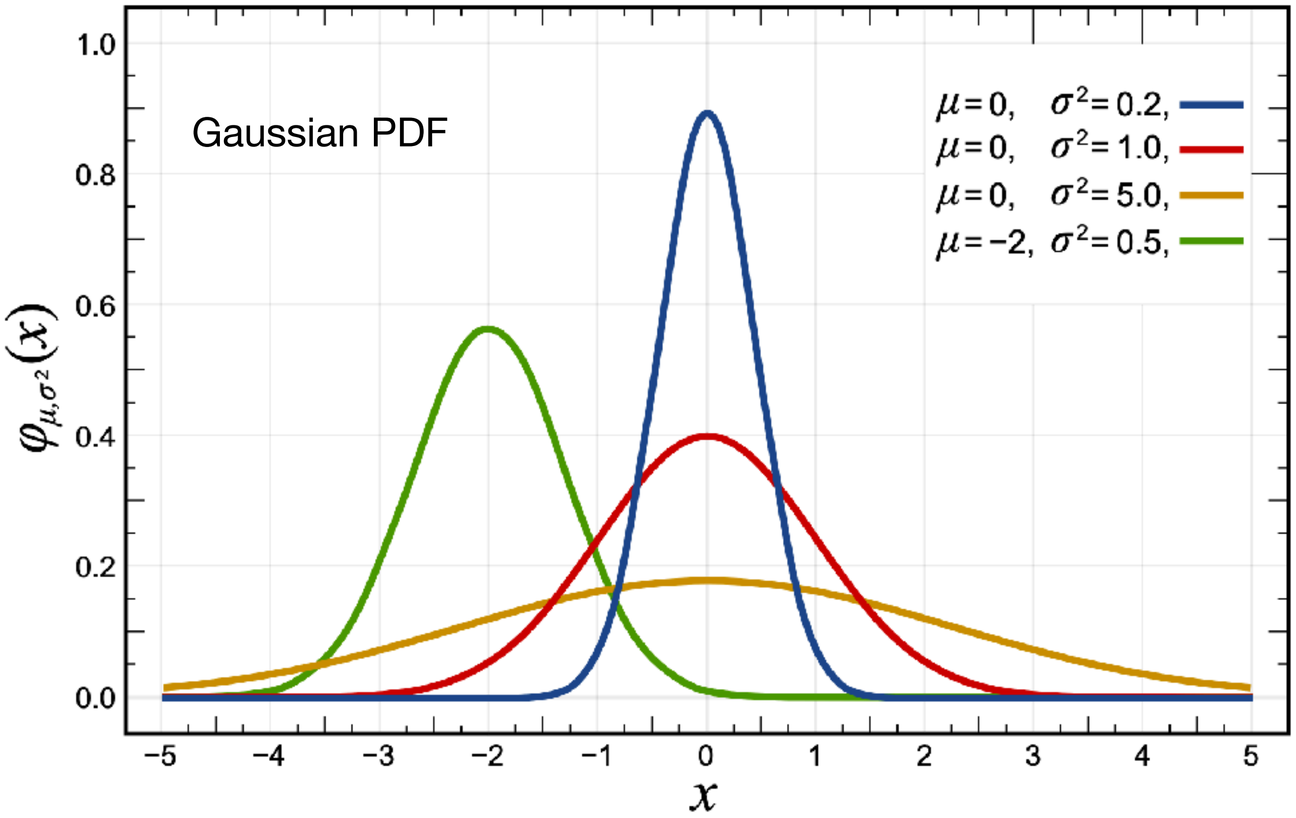
Parameterise μ and σ so that they are **state-dependent**:

$$\pi(a \mid s, \theta) = \frac{1}{\sigma(s,\theta)\sqrt{2\pi}} \exp\left(-\frac{[a-\mu(s,\theta)]^2}{2\sigma(s,\theta)^2}\right)$$

$\mu : \mathcal{S} \times \mathbb{R}^{d'} \to \mathbb{R}$

$\sigma : \mathcal{S} \times \mathbb{R}^{d'} \to \mathbb{R}^+$

$\theta = [\theta_\mu, \theta_\sigma]^\mathbb{T}$



Deterministic Policy Gradient

Challenges of using PG to deal with continuous action spaces:

- The policy $\pi_\theta(a \mid s) = \mathbb{P}[a \mid s; \theta]$ is **stochastic,** because it is dealing with a **continuous** action space.
- It can be **expensive** to derive the **gradients** because it samples both s and a, which typically requires more samples. Any cheaper solution?
- Furthermore, the actor-critic policy gradient is **on-policy**, which results in **unstable** learning.

$$J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \pi_\theta(s, a) r(s, a) \mathrm{d}a \mathrm{d}s = \mathbb{E}_{s \sim \rho_\pi, a \sim \pi_\theta}\left[r(s, a)\right] = \mathbb{E}_{s \sim \rho_\pi, a \sim \pi_\theta}\left[r(s, \mu_\theta(s))\right]$$

Let's say that we are only interested in the **best action**, and hence **deterministic**:

$$a = \mu_\theta(s)$$

which can be considered as a **special case** of the stochastic policy policy $\pi_{\mu_\theta, \sigma}$ with $\sigma = 0$.

Silver et al. (2014); Lil'Log

Deterministic Policy Gradients

Notations:

$Q^\pi(s,a)$: the value of (s,a) pair when policy π is followed: $Q^\pi(s,a) = \mathbb{E}_{a\sim\pi}[G_t \mid S_t = s, A_t = a]$

$\rho_0(s)$: initial distribution over states;

$\rho^\mu(s \to s', k)$: starting from state s, the visitation probability density at state s' after following k steps under policy μ.

$\rho^\mu(s')$: discounted state distribution: $\rho^\mu(s') = \int_{\mathcal{S}} \sum_{k=1}^{\infty} \gamma^{k-1} \rho_0(s) \rho^\mu(s \to s', k) ds$

Objective function: $J(\theta) = \int_{\mathcal{S}} \rho^\mu(s) Q(s, \mu_\theta(s)) ds$. Taking the derivative on both side:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \int_{\mathcal{S}} \rho^\mu(s) \nabla_a Q^\mu(s,a) \nabla_\theta \mu_\theta(s) \big|_{a=\mu_\theta(s)} ds \\
&= \mathbb{E}_{s\sim\rho^\mu}[\nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)} \nabla_\theta \mu_\theta(s)]
\end{aligned}$$

First Q w.r.t. a to choose the best action.

Then use the best action to optimise the deterministic policy μ w.r.t. θ

Silver et al. (2014); Lil'Log

Putting Everything Together: DDPG

Replay buffer + DQN + Actor-Critic + policy gradient + policy parameterisation = DDPG

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Lillicrap et al. (2015)

“Standing on the shoulders of giants.”

Gym

A toolkit for developing and comparing RL algorithms.

openai / gym

Watch 1k | Star 22.9k | Fork 6.5k

Code | Issues 180 | Pull requests 32 | Actions | Projects | Wiki | Security | Insights

master | 100 branches | 38 tags | Go to file | Add file | Code

justinkterry Miscellaneous Toy Text fixes (#2082) ... c4d0af3 28 days ago 1,237 commits

.github	add stale bot config	15 months ago
bin	fix mujoco-related build failure	2 years ago
docs	updated Gridworld: A simple 2D grid environment (#2073)	2 months ago
examples	Clean some docstrings (#1854)	8 months ago
gym	Miscellaneous Toy Text fixes (#2082)	28 days ago
scripts	remove six and __future__ imports (#1840)	8 months ago
tests	Respect the order of keys in a Dict's observation space when flatteni...	12 months ago
vendor	Switch to Docker for tests (#285)	4 years ago
.dockerignore	Switch to Docker for tests (#285)	4 years ago
.gitignore	Fix autodetect dtype warnings (#1234)	2 years ago
.travis.yml	Remove Python 3.5 support, travis and setup.py maintenance (#2084)	last month
CODE_OF_CONDUCT.rst	Initial release. Hello world :).	5 years ago
CONTRIBUTING.md	CONTRIBUTING.md (#1969)	6 months ago
LICENSE.md	Update Docs: HTTP -> HTTPS (#813)	3 years ago
README.rst	0.17.3 release and notes	2 months ago
py.Dockerfile	Remove Python 3.5 support, travis and setup.py maintenance (#2084)	last month
setup.py	Remove Python 3.5 support, travis and setup.py maintenance (#2084)	last month

About

A toolkit for developing and comparing reinforcement learning algorithms.

gym.openai.com/

Readme

View license

Releases 38

Fixed fetch/slide Latest
on May 29, 2019

+ 37 releases

Packages

No packages published

Used by 15.6k

+ 15,597

Contributors 261

https://github.com/openai/gym

Stable Baselines

hill-a / stable-baselines

forked from openai/baselines

Watch 71 | Star 2.7k | Fork 3.8k

Code | Issues 116 | Pull requests 11 | Actions | Projects 1 | Wiki | Security | Insights

master | 5 branches | 24 tags | Go to file | Add file | Code

This branch is 708 commits ahead, 221 commits behind openai:master. | Pull request | Compare

mily20001 and araffin Make EvalCallback work for recurrent policies (#1017) ... b3f414f on Oct 12 | 834 commits

.github	Fix `check_env`, `Monitor.close` and add Makefile (#673)	10 months ago
data	added tensorboard to A2C	2 years ago
docs	Make EvalCallback work for recurrent policies (#1017)	2 months ago
scripts	Update documentation (#848)	7 months ago
stable_baselines	Make EvalCallback work for recurrent policies (#1017)	2 months ago
tests	Make EvalCallback work for recurrent policies (#1017)	2 months ago
.coveragerc	Fixes (GAIL, A2C and BC) + Add Pretraining (#206)	2 years ago
.dockerignore	Type check with pytype (#565)	13 months ago
.gitignore	Refactor Tests + Add Helpers (#508)	13 months ago
.readthedocs.yml	Update documentation (#848)	7 months ago
.travis.yml	Release 2.10.0 (#737)	9 months ago

About

A fork of OpenAI Baselines, implementations of reinforcement learning algorithms

stable-baselines.readthedocs.io/

reinforcement-learning-algorithms | reinforcement-learning | machine-learning | gym | openai | baselines | toolbox | python | data-science

Readme

MIT License

Releases 24

Bug fixes release Latest
on Aug 5

+ 23 releases

https://github.com/hill-a/stable-baselines

RLlib

Algorithm	Frameworks	Discrete Actions	Continuous Actions	Multi-Agent	Model Support
A2C, A3C	tf + torch	Yes +parametric	Yes	Yes	+RNN, +LSTM auto-wrapping, +Transformer, +autoreg
ARS	tf + torch	Yes	Yes	No	
BC	tf + torch	Yes +parametric	Yes	Yes	+RNN
ES	tf + torch	Yes	Yes	No	
DDPG, TD3	tf + torch	No	Yes	Yes	
APEX-DDPG	tf + torch	No	Yes	Yes	
Dreamer	torch	No	Yes	No	+RNN
DQN, Rainbow	tf + torch	Yes +parametric	No	Yes	
APEX-DQN	tf + torch	Yes +parametric	No	Yes	
IMPALA	tf + torch	Yes +parametric	Yes	Yes	+RNN, +LSTM auto-wrapping, +Transformer, +autoreg
MAML	tf + torch	No	Yes	No	
MARWIL	tf + torch	Yes +parametric	Yes	Yes	+RNN
MBMPO	torch	No	Yes	No	
PG	tf + torch	Yes +parametric	Yes	Yes	+RNN, +LSTM auto-wrapping, +Transformer, +autoreg
PPO, APPO	tf + torch	Yes +parametric	Yes	Yes	+RNN, +LSTM auto-wrapping, +Transformer, +autoreg
SAC	tf + torch	Yes	Yes	Yes	
LinUCB, LinTS	torch	Yes +parametric	No	Yes	
AlphaZero	torch	Yes +parametric	No	No	

https://docs.ray.io/en/master/rllib.html

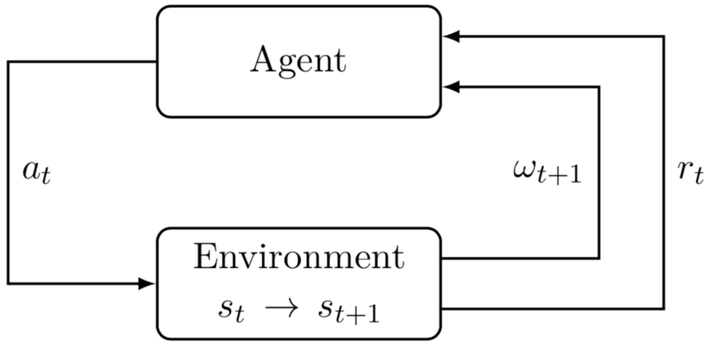
Boeing 747 Cockpit

Our world is complicated. Reinforcement learning, albeit being extremely promising, still has a long way to go.

Take Home Messages

Reinforcement learning

… is the area of machine learning that deals with **sequential decision-making**;
… is a task that **optimises** the **behaviour** of the agent when interacting with a given **environment**;
… aims to find a **balance** between **exploration** and **exploitation**;
… models the environment as a **Markov decision process**;
… stores the experience in lookup tables (Q-tables) or as policies.



Q-learning

… uses **value functions** to measure the **value** of an action a given state s;
… stores the values in a lookup table $Q(s, a)$;
… infers/approximates the optimal policy π^* according to $Q(s, a)$;
… is a model-free, deterministic algorithm.

Policy gradients

… directly optimise the **policy** by sampling (instead of evaluating) the value of a trajectory;
… model desirable **actions** by learning a **probability distribution**;
… are applicable to a wider range of problem and generally cheaper to train (comparing to Q-learning);
… sometimes difficult to reach convergence.

Reading Materials

If you are interested to learn more:

https://pathmind.com/wiki/deep-reinforcement-learning

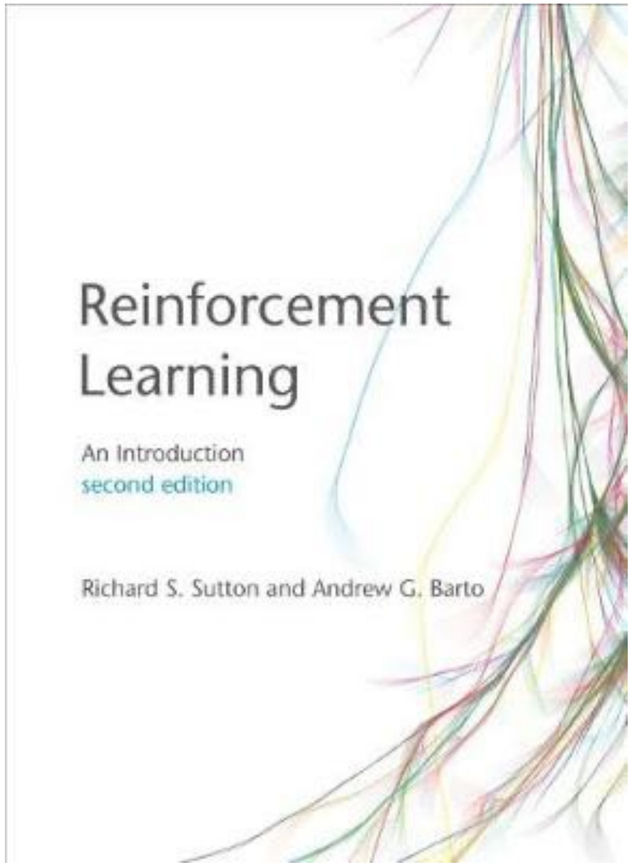
https://flyyufelix.github.io/2017/10/12/dqn-vs-pg.html

https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12

https://www.davidsilver.uk/teaching/

https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

https://github.com/dennybritz/reinforcement-learning



Sutton & Barto (2015)