



Inria
INVENTEURS DU MONDE NUMÉRIQUE

Reinforcement Learning

MML 2020

Speaker: Michal Valko

Slide contributors: Nando de Freitas, Emilie Kaufmann, Hado, Diana, Bobak Shahriari, Caglar Gulcehre, David Silver, Feryal Behbahani, Ksenia Konyushkova, Marc Bellemare, Matt Hoffman, Tom Le Paine, and Tom Schaul
Special thanks: Nando, Emilie, Diana, Hado, Corentin

DeepMind

0

What is RL?



What is reinforcement learning?

- learning by **trial and error**
- learning to **act** in an unknown, stochastic environment by maximizing some **reward** signal
- Example: learning to bike without a perfect knowledge of physics



Prehistory of RL

- 1900s: observation of animal behavior (e.g. Thorndike 1911 “Law of Effect”)
- 1920s: Pavlov work on conditionnal reflexes first occurrence of “reinforcement” in animal learning
- Oak and Miller 1954: first experiments on electric brain stimuli for controlling mice behavior



History of RL

- 1950s, Shannon's machines : "Theseus", a mice finding how to get out of a maze, a chess player, a Rubik's cube solver
- 1957, Bellmann : Dynamic Programming (control of dynamical systems)
- 1961, Minsky "Towards artificial intelligence"
- 1978, Sutton : Temporal Difference Learning (artificial intelligence)
- 1989, Watkins : Q-Learning algorithm



Why we are not done yet?

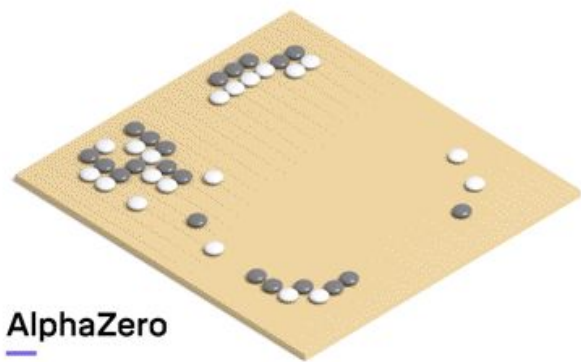
- We need to reason about
... time
- ... (long-term) consequences of actions
- ... predicting the future
- ... actively gathering experience
- ... dealing with uncertainty



Why RL
now?

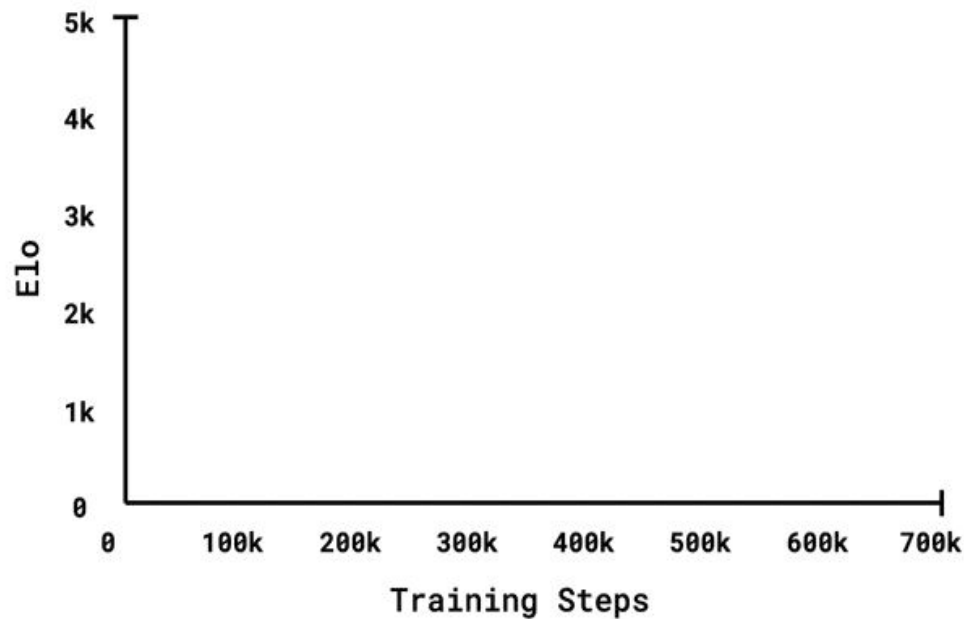


AlphaZero: Go chess shogi

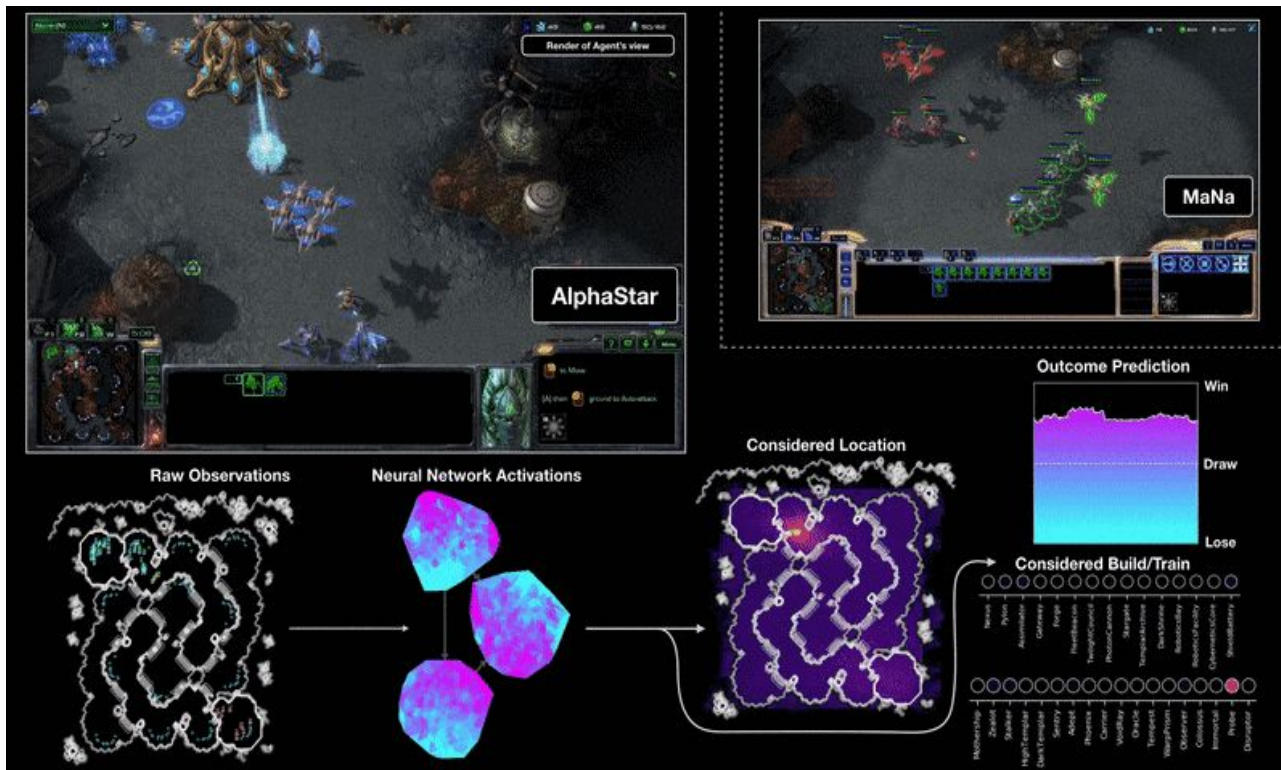


AlphaZero

[Silver et al., 2017]



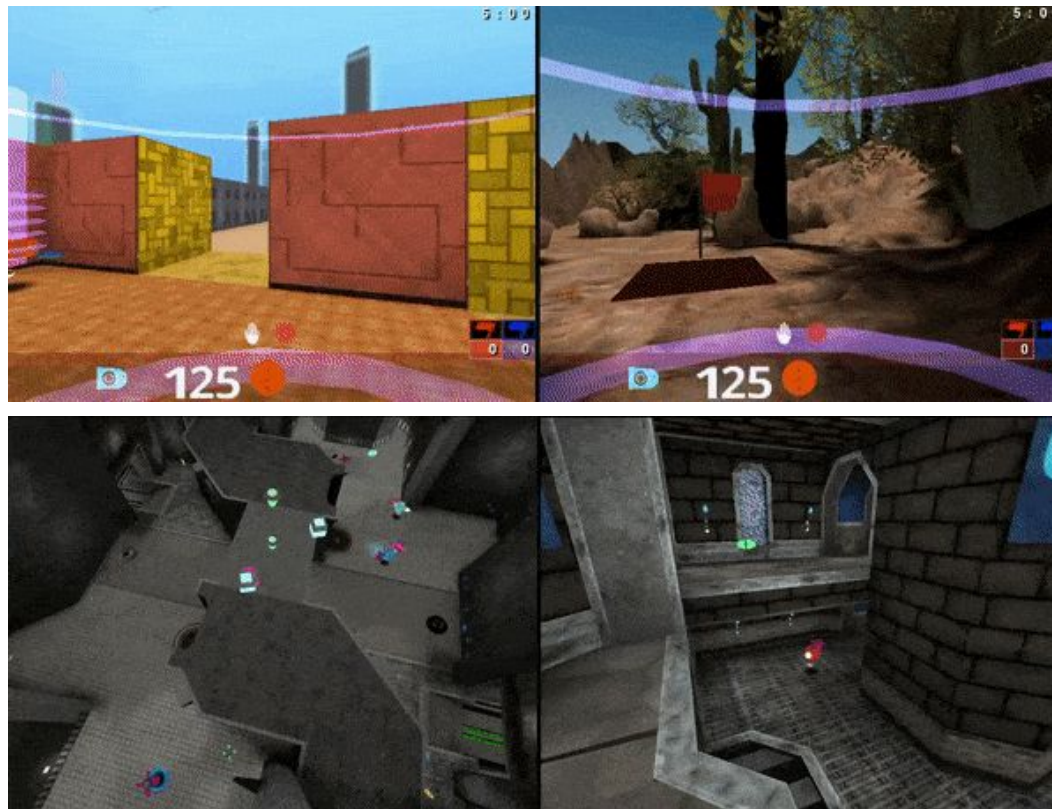
AlphaStar playing StarCraft2



[Vinyals et al., 2019]



Capture the Flag and Quake III Arena



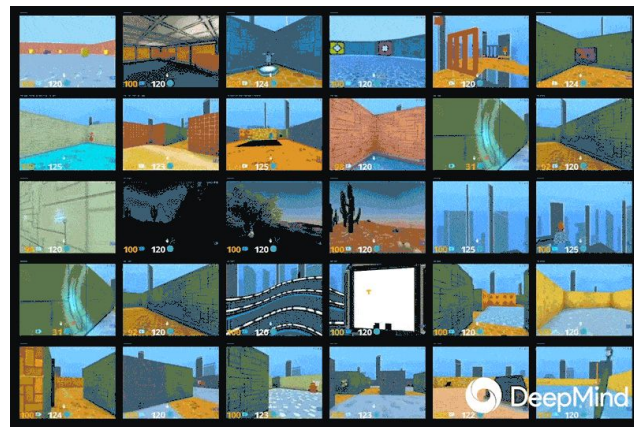
[Jaderberg et al., 2019]



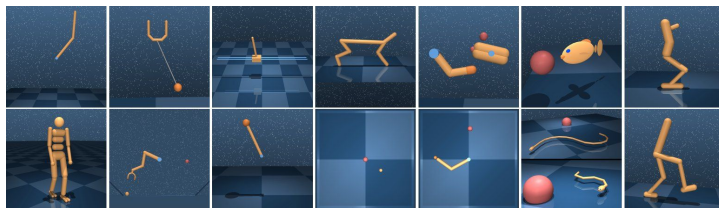
A single learning algorithm to solve them all!



Atari 57 games



DMLab 30



Control suite



Machine learning

A photograph of two chimpanzees. The chimpanzee in the foreground is looking towards the camera with a neutral expression. The chimpanzee behind it is partially visible, appearing to whisper into the ear of the first. A red circle is drawn around the top of the second chimpanzee's head. A green wireframe cube is in the bottom left corner. A green leaf is in the bottom right corner.

1. ML is about inferring knowledge from observations or experiences, subject to the physical laws of the world.
2. ML is also about using this knowledge to guide observation and hypothesis testing.
3. ML is about creating new knowledge, using the present knowledge, to solve a large diversity of novel problems.

Important to distinguish:

- Associative knowledge
- Explanatory knowledge



Magic association recipe

- Largest possible network with few general inductive biases.
- Massive curated dataset. Clear goal.
- Capitalize on the best existing communication, memory and computation infrastructure.



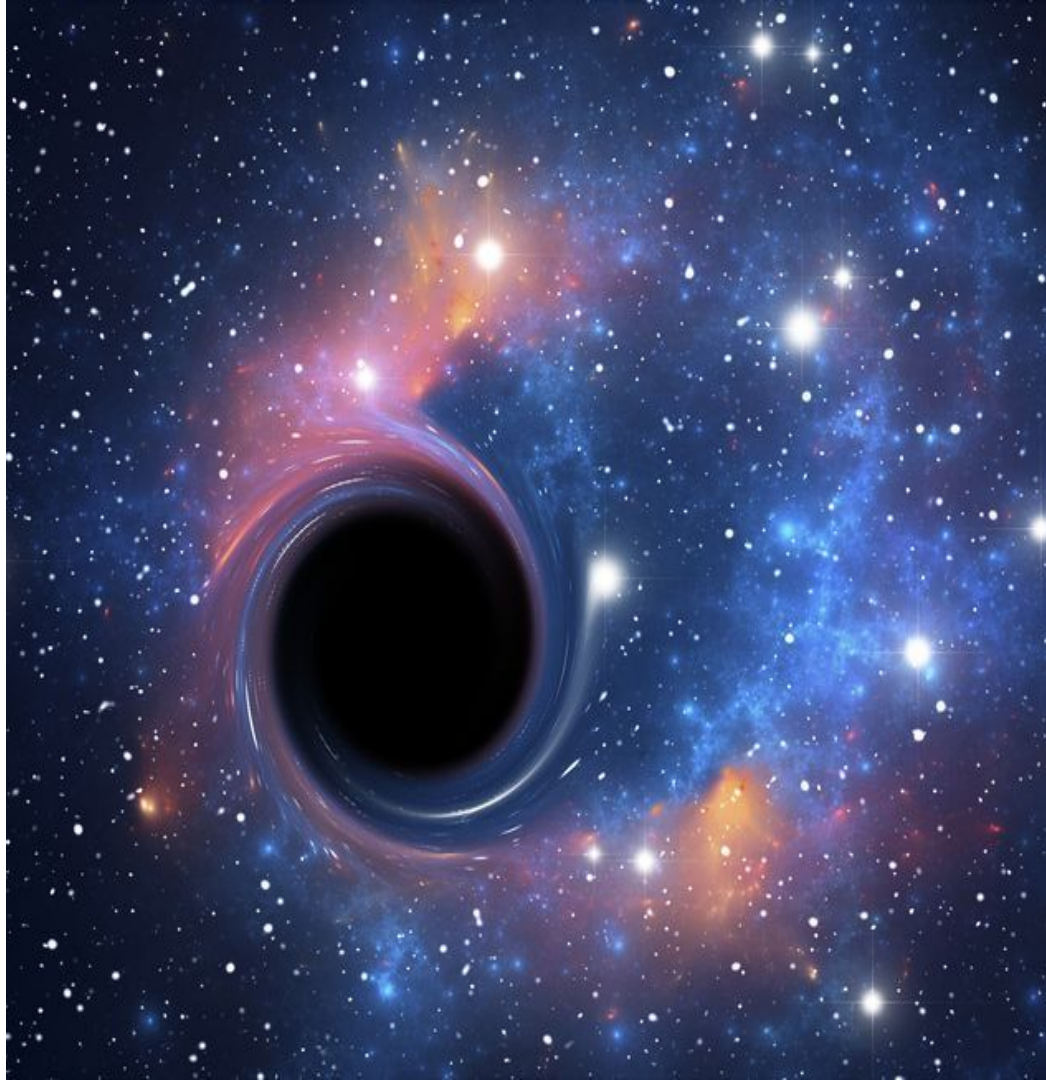
3 main ML tasks

- **Supervised Learning** Learn to make predictions, based on a large batch of data for which the target variable is observed
- **Unsupervised Learning** Find some latent structure in data (clusters, low-rank structure...)
- **Reinforcement Learning** Learn to take decisions / influence the data collection process



Outline

1. RL Concepts
2. a) MDPs: know r, p
b) MDPs: unknow r, p
3. Deep Q-networks
4. Policy gradients
5. Distributional RL
6. Exploration/Exploitation
7. Monte-Carlo Tree Search



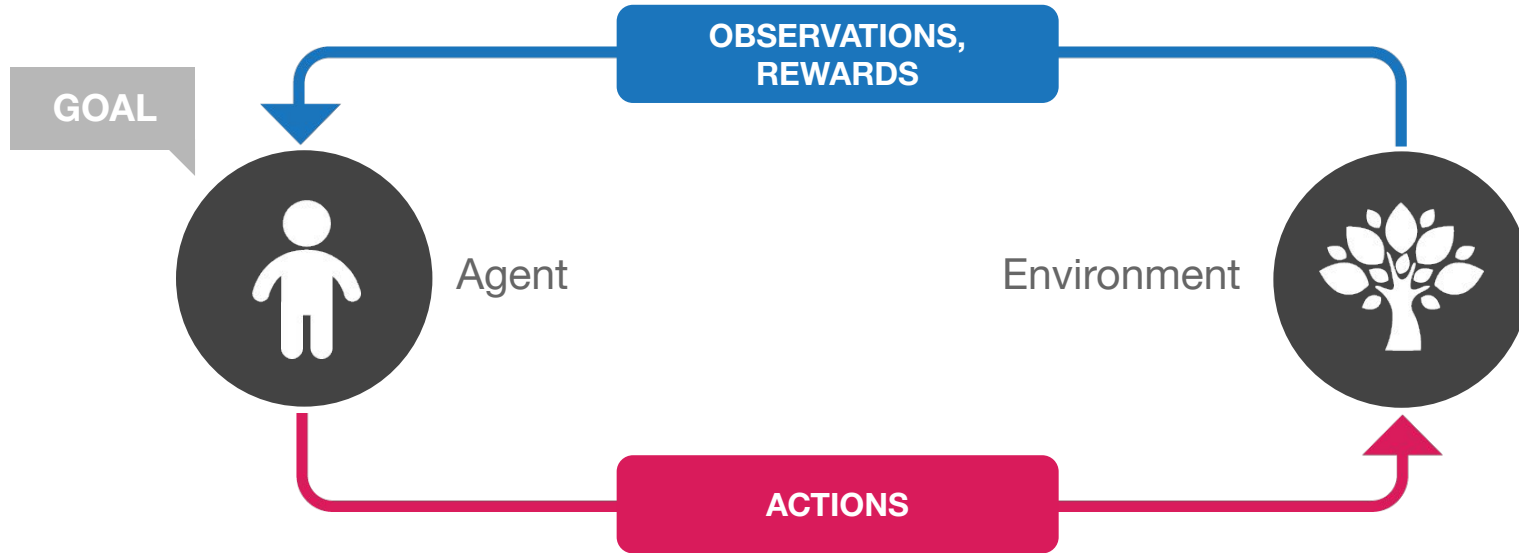
DeepMind

1

RL Concepts



Reinforcement learning

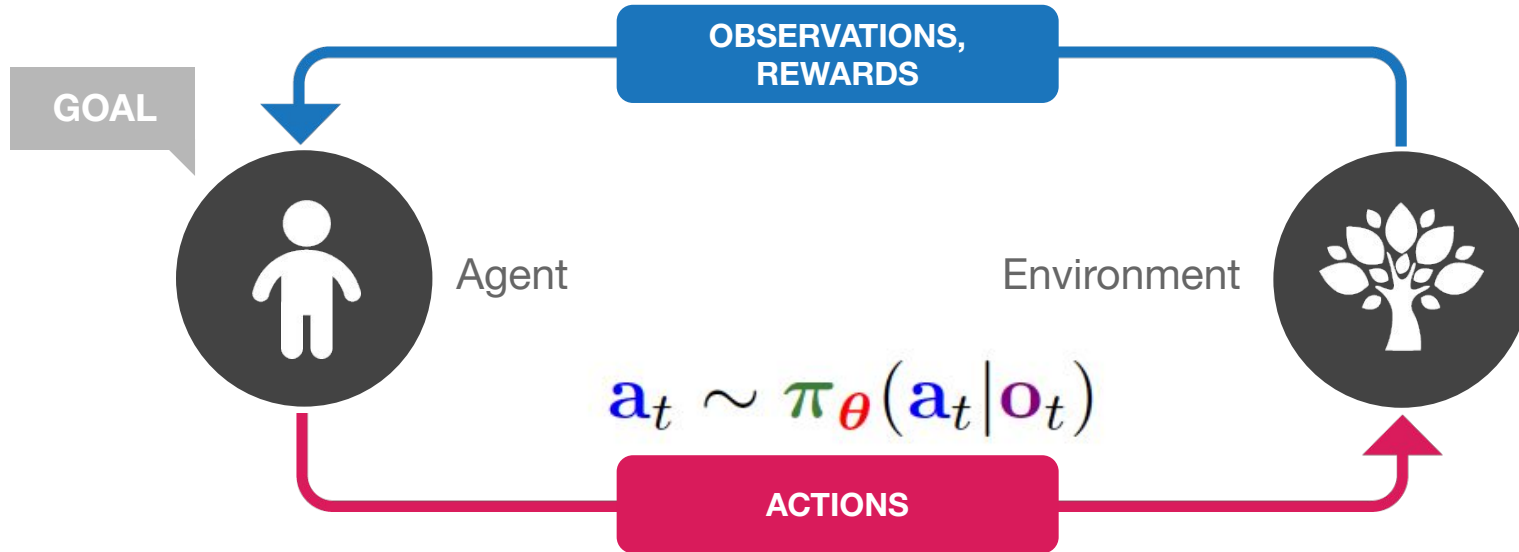


Making good decisions by learning from experience



$$\mathbf{o}_{t+1} \sim P(\mathbf{o}_{t+1} | \mathbf{a}_t, \mathbf{o}_t)$$

$$\mathbf{r}_t = \mathbf{r}(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1})$$

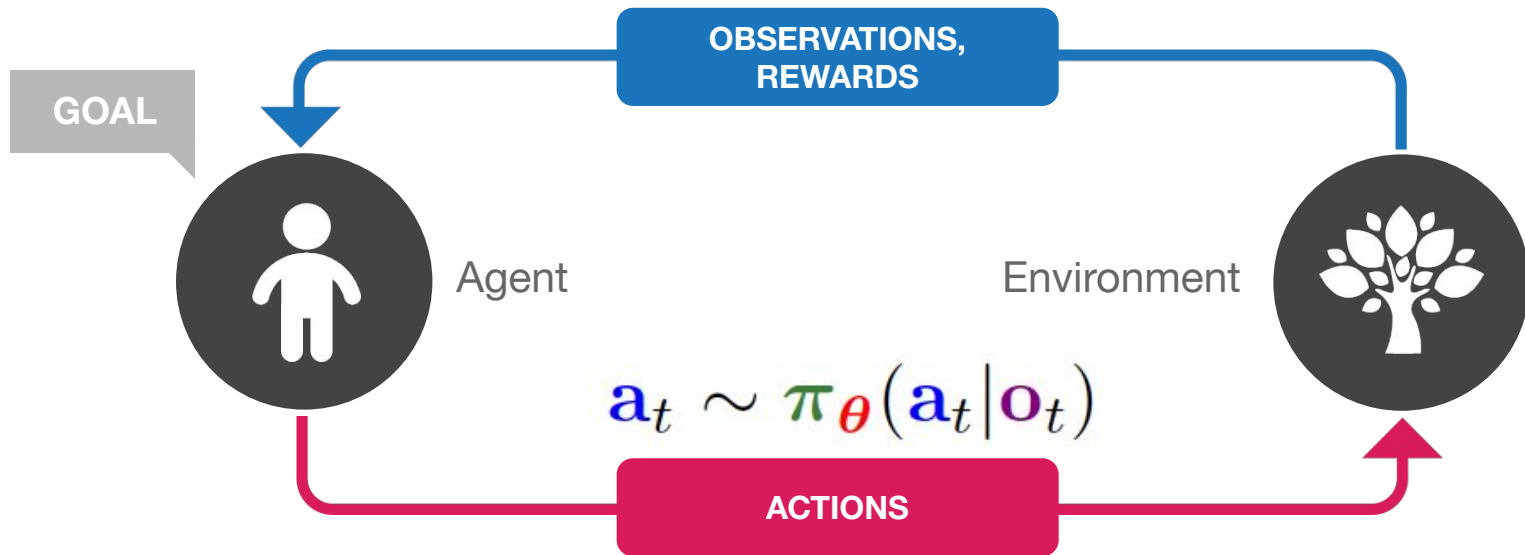


Making good decisions by learning from experience



$$\mathbf{o}_{t+1} \sim P(\mathbf{o}_{t+1} | \mathbf{a}_t, \mathbf{o}_t)$$

```
next_timestep = environment.step(action)
```



```
action = actor.policy(timestep.observation)
```

Making good decisions by learning from experience



Markov Decision Process

A Markov Decision Process (MDP) models a situation in which **repeated decisions** (= choices of actions) are made. MDP provides models for the consequence of each decisions :

- ▶ in terms of **reward**
- ▶ in terms of the evolution of the system's **state**

In each (discrete) **decision time** $t = 0, 1, 2, \dots$, a learning agent

- ▶ selects an **action** a_t based on his current **state** s_t (or possibly all the previous observations),
- ▶ gets a **reward** $r_t \in \mathbb{R}$ depending on his choice,
- ▶ transits to a **new state** s_{t+1} depending on his choice.



Markov Decision Process

A MDP is parameterized by a tuple $(\mathcal{S}, \mathcal{A}, R, P)$ where

- ▶ $\mathcal{S}$ is the **state space**
- ▶ $\mathcal{A}$ is the **action space**
- ▶ $R = (\nu_{(s,a)})_{(s,a) \in \mathcal{S} \times \mathcal{A}}$ where $\nu_{(s,a)} \in \Delta(\mathbb{R})$ is the **reward distribution** for the state-action pair (s, a)
- ▶ $P = (p(\cdot|s, a))_{(s,a) \in \mathcal{S} \times \mathcal{A}}$ where $p(\cdot|s, a) \in \Delta(\mathcal{S})$ is the **transition kernel** associated to the state-action pair (s, a)

In each (discrete) **decision time** $t = 0, 1, 2, \dots$, a learning agent

- ▶ selects an **action** a_t based on his current **state** s_t (or possibly all the previous observations),
- ▶ gets a **reward** $r_t \sim \nu_{(s_t, a_t)}$
- ▶ transits to a **new state** $s_{t+1} \sim p(\cdot|s_t, a_t)$



Markov Decision Process

A MDP is parameterized by a tuple $(\mathcal{S}, \mathcal{A}, R, P)$ where

- ▶ $\mathcal{S}$ is the **state space**
- ▶ $\mathcal{A}$ is the **action space**
- ▶ $R = (\nu_{(s,a)})_{(s,a) \in \mathcal{S} \times \mathcal{A}}$ where $\nu_{(s,a)} \in \Delta(\mathbb{R})$ is the **reward distribution** for the state-action pair (s, a)
- ▶ $P = (p(\cdot|s, a))_{(s,a) \in \mathcal{S} \times \mathcal{A}}$ where $p(\cdot|s, a) \in \Delta(\mathcal{S})$ is the **transition kernel** associated to the state-action pair (s, a)

- **The tabular case** : finite state and action spaces

$$\mathcal{S} = \{1, \dots, S\}$$

$$\mathcal{A} = \{1, \dots, A\}$$

For every $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$, $p(s'|s, a) = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$.



Markovian Dynamics

- **Reminder** : Markov chain

Definition

A Markov chain on a discrete space $\mathcal{X}$ is a stochastic process $(X_t)_{t \in \mathbb{N}}$ that satisfies the **Markov property** :

$$\mathbb{P}(X_t = x_t | X_{t-1} = x_{t-1}, \dots, X_0 = x_0) = \mathbb{P}(X_t = x_t | X_{t-1} = x_{t-1})$$

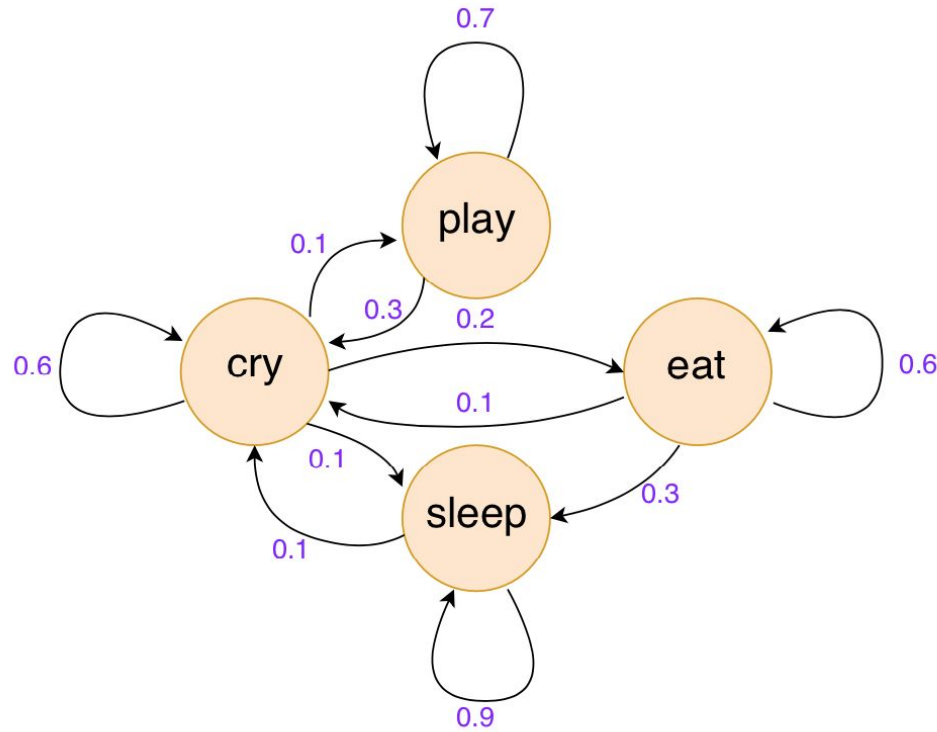
for all $t \in \mathbb{N}$ and $(x_0, \dots, x_t) \in \mathcal{X}^{t+1}$. It is homogeneous if

$$\mathbb{P}(X_t = y | X_{t-1} = x) = \mathbb{P}(X_{t-1} = y | X_{t-2} = x).$$

- An homogeneous Markov chain is characterized by its transition probabilities $p(y|x) = \mathbb{P}(X_t = y | X_{t-1} = x)$ and its initial state.
- If $\mathcal{X}$ is continuous, this definition can be extended by mean of a **transition kernel** such that $p(\cdot|x) \in \Delta(\mathcal{X})$ for all $x \in \mathcal{X}$.



Markovian Dynamics



Policy

Definition

A (Markovian) **policy** is a sequence $\pi = (\pi_t)_{t \in \mathbb{N}}$ of mappings

$$\pi_t : \mathcal{S} \rightarrow \Delta(\mathcal{A}),$$

where $\Delta(\mathcal{A})$ is the set of probability distributions over the action space.

A policy may be

Deterministic	Stochastic
$\pi_t : \mathcal{S} \rightarrow \mathcal{A}$	$\pi_t : \mathcal{S} \rightarrow \Delta(\mathcal{A})$

► **Terminology** : policy = strategy = decision rule = control



Policy

Definition

A (Markovian) **policy** is a sequence $\pi = (\pi_t)_{t \in \mathbb{N}}$ of mappings

$$\pi_t : \mathcal{S} \rightarrow \Delta(\mathcal{A}),$$

where $\Delta(\mathcal{A})$ is the set of probability distributions over the action space.

A policy may be

Stationary	Non-stationary
$\pi = (\pi, \pi, \pi, \dots)$	$\pi = (\pi_0, \pi_1, \pi_2, \dots)$

► **Terminology** : policy = strategy = decision rule = control

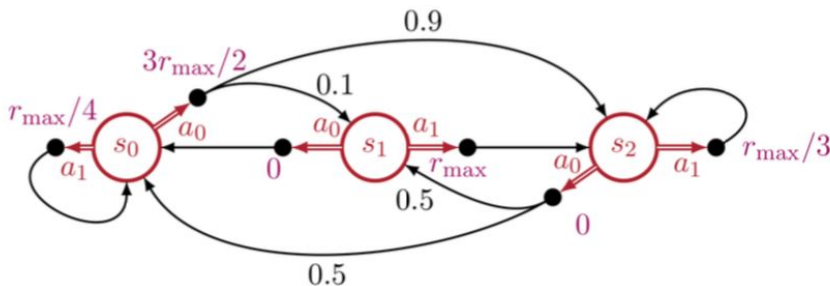


Policy

Under stationary (deterministic) policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, the random process $(s_t)_{t \in \mathbb{N}}$ is a **Markov chain**, with transition probability

$$\mathbb{P}^\pi(s_{t+1} = s' | s_t = s) = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = \pi(s)) = p(s' | s, \pi(s))$$

(can be extended to stochastic policies and continuous spaces)



→ A MDP is sometimes referred to as a **controlled Markov chain**



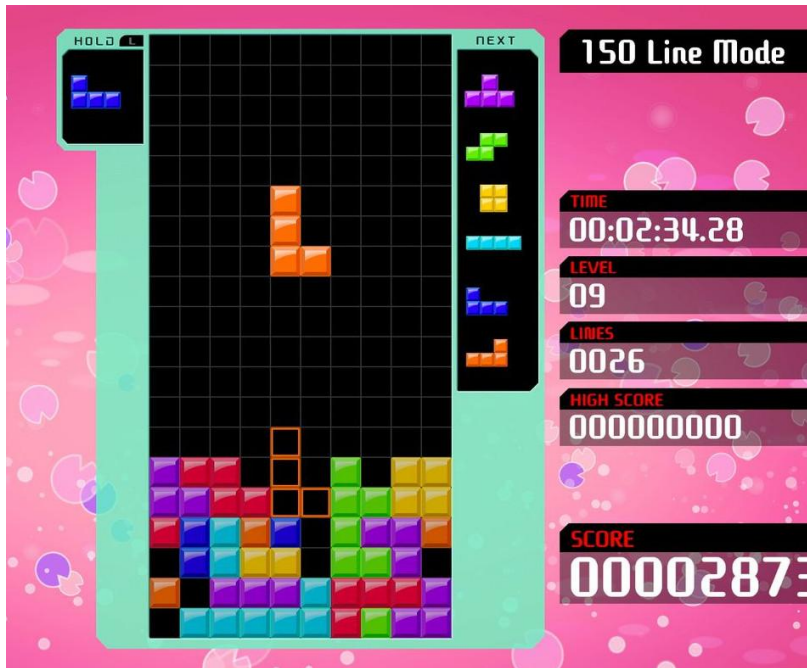
DeepMind

1a

RL examples



Tetris



State: current board and next blocks to add

Action: orientation + position of the dropped block

Reward: increment in the score/

Transition: new board + randomness in the new block

difficulty: large state space!

Making good decisions by learning from **experience**



Retail store management

You owe a bike store. During week t , the (random) demand is D_t units. On Monday morning you may choose to **command a_t additional units** : they are delivered immediately before the shop opens.

For each week :

- ▶ Maintenance cost : h per unit left in your stock
- ▶ Ordering cost : c per unit ordered + fix cost c_0 if an order is placed
- ▶ Sales profit : p per unit sold

Constraints :

- ▶ your warehouse has a maximal capacity of M bikes (any additional bike gets stolen)
- ▶ you cannot sell bikes that you don't have in stock



Retail store management

- ▶ State : number of bikes in stock on Sunday
State space : $\mathcal{S} = \{0, \dots, M\}$
- ▶ Action : number of bikes ordered at the beginning of the week
Action space : $\mathcal{A} = \{0, \dots, M\}$
- ▶ Reward = balance of the week : if you command A_t bikes,
$$r_t = -c_0 \mathbb{1}_{(a_t > 0)} - c \times a_t - h \times s_t + p \times \min(D_t, s_t + a_t, M)$$
- ▶ Transition : you end the week with

$$s_{t+1} = \max(0, \min(M, s_t + a_t) - D_t) \quad \text{bikes}$$

→ Markov Decision Process

$$r(s, a)? \quad p(\cdot | s, a)?$$



DeepMind

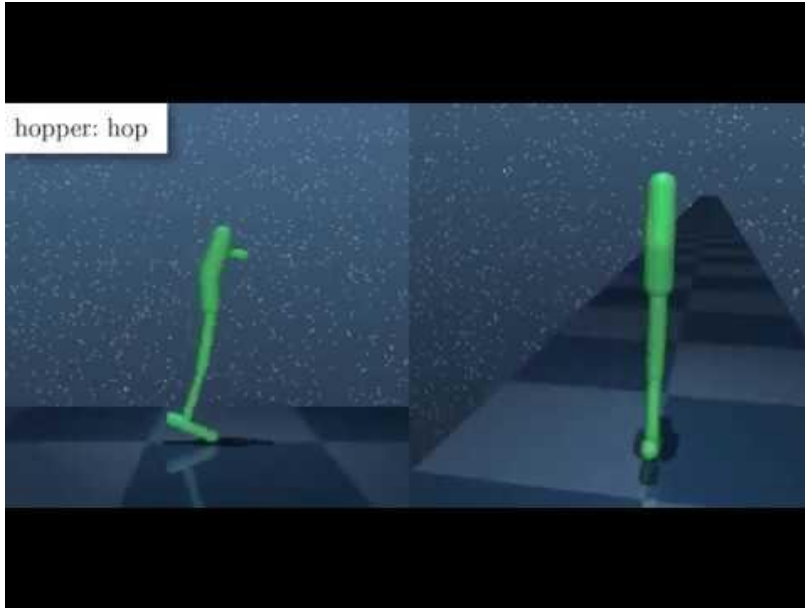
1a

1b

Modern RL examples



Environments and tasks



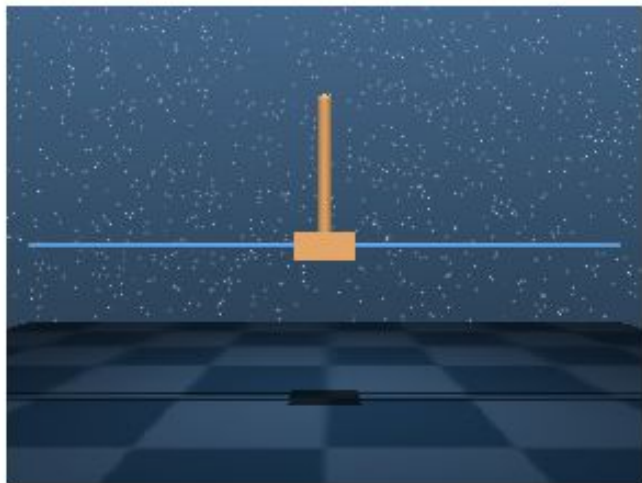
```
from dm_control import suite
```

Making good decisions by learning from **experience**



Environments and tasks

Example: The agent tries to balance the pole up by moving the cart left and right



```
from dm_control import suite
```

```
domain_name='cartpole'
```

```
task_name='balance'
```

```
environment = suite.load(domain_name, task_name)
```

Making good decisions by learning from **experience**



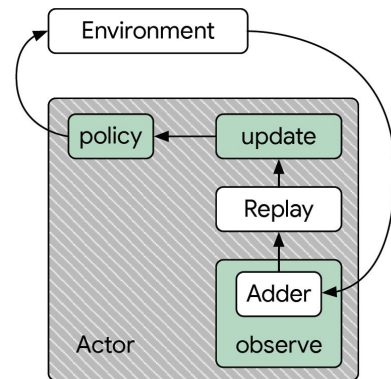
Generating episodes of experience

```
1: while True do
2:   Make initial observation  $o_0$ 
3:    $t \leftarrow 0$ ;  $e_t \leftarrow \text{False}$ 
4:   while not  $e_t$  do
5:      $a_t \leftarrow \pi(o_t)$ 
6:     Step the environment to
       obtain ( $r_t, \kappa_t, e_{t+1}, o_{t+1}$ )
7:     Observe ( $o_t, a_t, r_t, \kappa_t$ )
8:     Update the policy  $\pi$ 
9:      $t \leftarrow t + 1$ 
10:  end while
11:  Observe the final state  $o_t$ 
12: end while
```

```
# Make an initial observation.
timestep = environment.reset()

# Run an episode.
while not timestep.last():
  # Generate an action with the agent policy.
  action = actor.policy(timestep.observation)
  # Step in the environment.
  next_timestep = environment.step(action)
  # Make an observation and update the actor.
  actor.observe(timestep.observation,
               action,
               next_timestep.reward,
               next_timestep.discount)

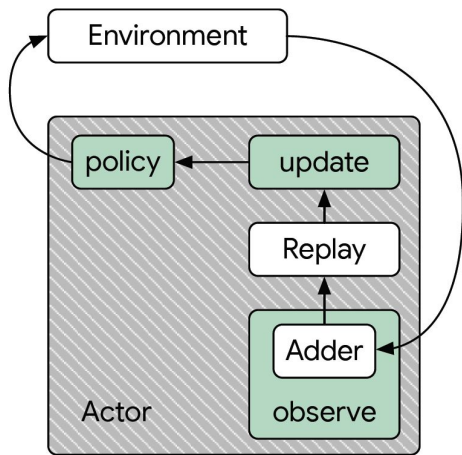
  actor.update()
  timestep = next_timestep
# Make a final observation.
actor.observe_last(timestep.observation)
```



Making good decisions by **learning from experience**



Building an actor



```
class RandomActor(core.Actor):
```

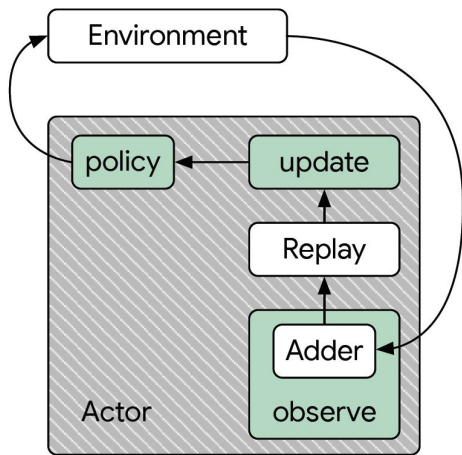
```
    def __init__(self, action_spec):  
        # Make the agent output a valid action.  
        self.scale = action_spec.maximum - action_spec.minimum  
        self.offset = action_spec.minimum  
        self.shape = action_spec.shape
```

```
    def policy(self, observation):  
        # This actor chooses actions uniformly at random.  
        raw_action = np.random.rand(*self.shape)  
        return self.scale * raw_action + self.offset
```

Making good decisions by learning from experience



Building an actor



```
def observe(self, observation, action, reward, discount):  
    """Random actor does not record experience."""  
    pass
```

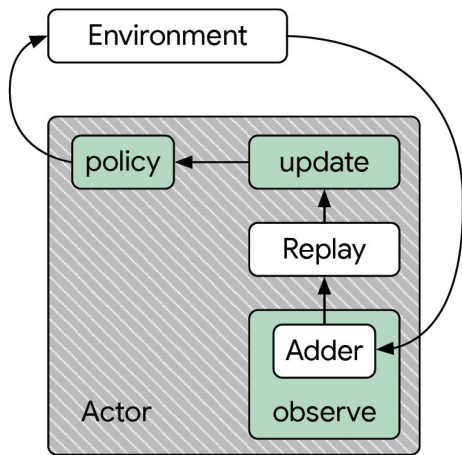
```
def observe_last(self, observation):  
    """Random actor does not record experience."""  
    pass
```

```
def update(self):  
    """Random actor does not learn from experience."""  
    pass
```

Making good decisions by learning from experience



Building an actor



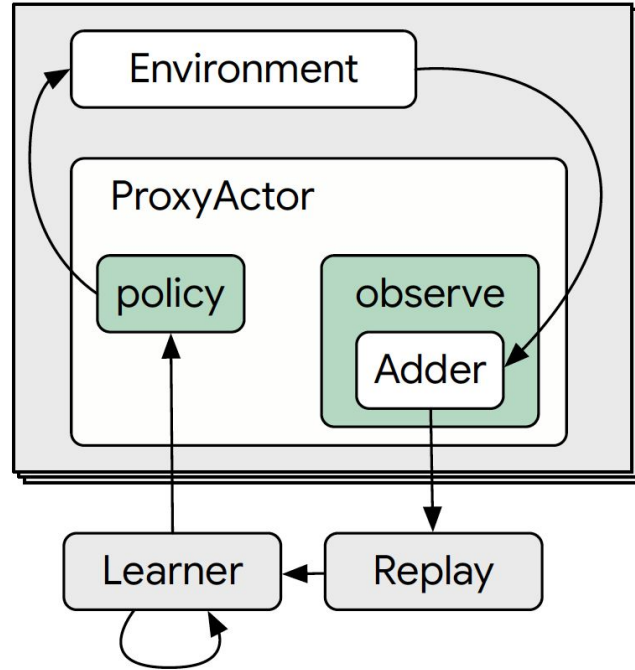
Create an instance of a random actor.

```
action_spec = environment.action_spec()  
actor = RandomActor(action_spec)
```

Making good decisions by learning from experience



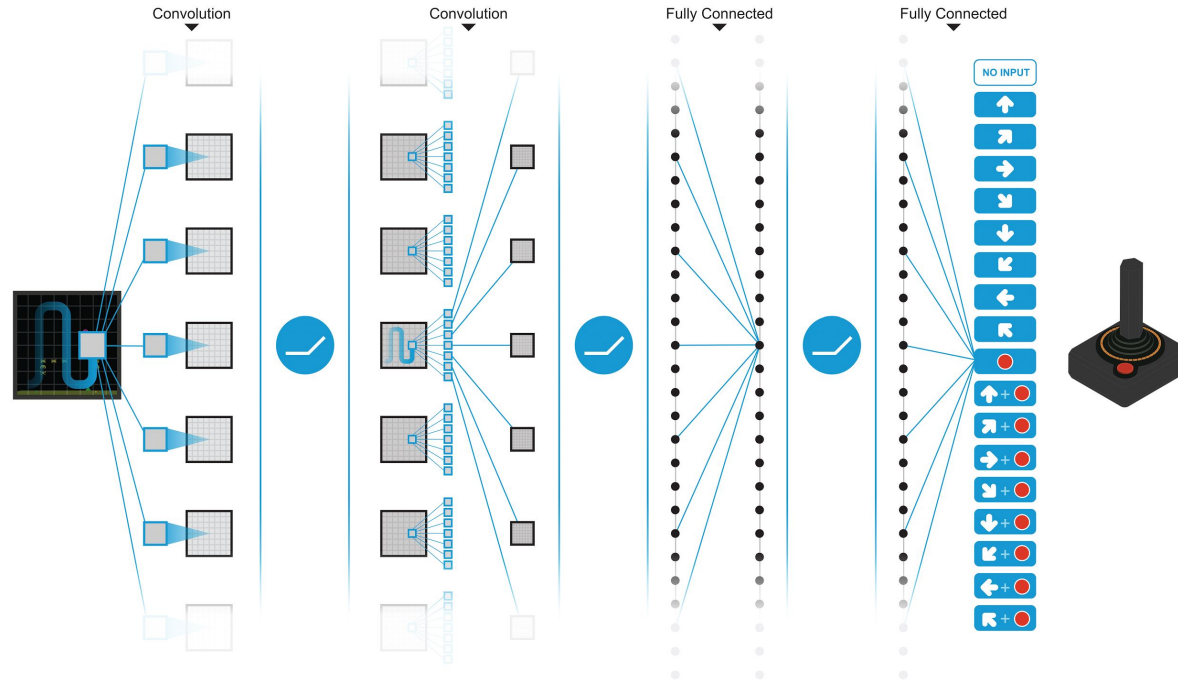
Distributed actors and centralized learner



Making good decisions by learning from experience



Deep RL uses deep neural nets as policies



Making good decisions by learning from experience



Rewards



Extrinsic

- Environmental
- Typically hardcoded



Intrinsic

- Social (eg influence, imitation, explanation, language)
- Desire to control (empowerment)
- Curiosity
- Losses learned via meta-learning
- Learning progress
- Explanation

Making **good** decisions by learning from experience



What is a good policy?

It is a policy that yield a large **value** in each state, which is always some notion of **cumulative reward**.

① Finite horizon

Given a known **horizon** $T \in \mathbb{N}^*$,

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{T-1} r_t + r_T \mid s_0 = s \right]$$

→ **When ?** In the presence of a natural notion of duration of an episode (e.g. maximal number of steps in a game)

Making good decisions by learning from experience



What is a good policy?

It is a policy that yield a large **value** in each state, which is always some notion of **cumulative reward**.

① Finite horizon

Given a known **horizon** $T \in \mathbb{N}^*$,

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{T-1} r_t + r_T \mid s_0 = s \right]$$

$$a_t \sim \pi(s_t), s_{t+1} \sim p(\cdot | s_t, a_t), r_t \sim \nu_{s_t, a_t}$$

→ **When ?** In the presence of a natural notion of duration of an episode (e.g. maximal number of steps in a game)

Making **good** decisions by learning from experience



What is a good policy?

It is a policy that yield a large **value** in each state, which is always some notion of **cumulative reward**.

① Finite horizon

Given a known **horizon** $T \in \mathbb{N}^*$,

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{T-1} r_t + r_T \mid s_0 = s \right]$$

$a_t \sim \pi(s_t), s_{t+1} \sim p(\cdot | s_t, a_t), r_t \sim \nu_{s_t, a_t}$

starting from state s

→ **When ?** In the presence of a natural notion of duration of an episode (e.g. maximal number of steps in a game)

Making **good** decisions by learning from experience



What is a good policy?

It is a policy that yield a large **value** in each state, which is always some notion of **cumulative reward**.

② Infinite time horizon with a discount parameter

Given a known **discount parameter** $\gamma \in (0, 1)$,

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s \right]$$

$a_t \sim \pi(s_t), s_{t+1} \sim p(\cdot | s_t, a_t), r_t \sim \nu_{s_t, a_t}$

starting from state s

→ **When ?** To put more weight on short-term reward / when there is a natural notion of discount

Making good decisions by learning from experience



What is a good policy?

It is a policy that yield a large **value** in each state, which is always some notion of **cumulative reward**.

③ Infinite time horizon with a terminal state

Given τ the **random time at which we first reach a terminal state**.

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{\tau} r_t \mid s_0 = s \right]$$

$$a_t \sim \pi(s_t), s_{t+1} \sim p(\cdot | s_t, a_t), r_t \sim \nu_{s_t, a_t}$$

starting from state s

→ **When ?** For tasks that have a natural notion of terminal state
(e.g. achieve some goal)

Making good decisions by learning from experience



What is a good policy?

It is a policy that yield a large **value** in each state, which is always some notion of **cumulative reward**.

④ Infinite time horizon with average reward

$$V^\pi(s) = \lim_{T \rightarrow \infty} \mathbb{E}^\pi \left[\frac{1}{T} \sum_{t=0}^{T-1} r_t \mid s_0 = s \right]$$

$$a_t \sim \pi(s_t), s_{t+1} \sim p(\cdot | s_t, a_t), r_t \sim \nu_{s_t, a_t}$$

starting from state s

→ **When?** The system should be controlled for a very long time



What is a good policy?

It is a policy that yield a large **value** in each state, which is always some notion of **cumulative reward**.

④ Infinite time horizon with average reward

$$V^\pi(s) = \lim_{T \rightarrow \infty} \mathbb{E}^\pi \left[\frac{1}{T} \sum_{t=0}^{T-1} r_t \mid s_0 = s \right]$$

$$a_t \sim \pi(s_t), s_{t+1} \sim p(\cdot | s_t, a_t), r_t \sim \nu_{s_t, a_t}$$

starting from state s

- **When ?** The system should be controlled for a very long time
- slightly harder to work with (not mentioned much in this class)

c.f. [Puterman, *Markov Decision Processes*, 1994]



Optimal policy

Given a **value function** (①,②,③ or ④), one can define the following.

Definition

The **optimal value** in state s is given by

$$V^*(s) = \max_{\pi} V^{\pi}(s).$$

Definition

An **optimal policy** π^* satisfies

$$\pi^* \in \operatorname{argmax}_{\pi} V^{\pi},$$

that is

$$\forall s \in \mathcal{S}, \pi^* \in \operatorname{argmax}_{\pi} V^{\pi}(s)$$

or

$$V^* = V^{\pi^*}.$$



Optimal policy

Properties :

- ▶ there exists an optimal policy !
(i.e. a policy maximizing the value in all states)
- ▶ there exists an optimal policy that is deterministic
- ▶ ... an even stationary (except with a finite horizon ①)

Definition

An optimal policy π^* satisfies

$$\pi^* \in \operatorname{argmax}_{\pi} V^{\pi},$$

that is

$$\forall s \in \mathcal{S}, \pi^* \in \operatorname{argmax}_{\pi} V^{\pi}(s)$$

or

$$V^* = V^{\pi^*}.$$



Back to Retail Store Management

- State : number of bikes in stock on Sunday

State space : $\mathcal{S} = \{0, \dots, M\}$

- Action : number of bikes ordered at the beginning of the week

Action space : $\mathcal{A} = \{0, \dots, M\}$

- Reward = balance of the week : if you command a_t bikes,

$$r_t = -c_0 \mathbb{1}_{(a_t > 0)} - c \times a_t - h \times s_t + p \times \min(D_t, s_t + a_t, M)$$

- Transition : you end the week with

$$s_{t+1} = \max(0, \min(M, s_t + a_t) - D_t) \quad \text{bikes}$$

Goal : From an initial stock s , maximize the sum of discounted rewards

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s \right]$$



Back to Retail Store Management

- **Uniform policy** (on reasonable orders)

$$\pi(s) \sim \mathcal{U}(\{0, \dots, M - s\})$$

- **Constant policy** : always buy m_0 machines

$$\pi(s) = \max(M - s, m_0)$$

- **Threshold policy** : whenever there are less than m_1 bikes in stock, refill it up to m_2 bikes. Otherwise, do not order.

$$\pi(s) = \mathbb{1}_{(s \leq m_1)}(m_2 - s)$$



DeepMind

2a

Solving MDPs with *known* parameters



DeepMind

Solving a MDP with a Finite Horizon



Bellman equations

Proposition

The value of a deterministic policy π satisfies the following equations :

$$V_H^\pi(s) = r_H(s)$$

$$V_h^\pi(s) = r(s, \pi_h(s)) + \sum_{s' \in \mathcal{S}} p(s'|s, \pi_h(s)) V_{h+1}^\pi(s') \quad \text{for all } h < H.$$

Proof.

$$\begin{aligned} V_h^\pi(s) &= \mathbb{E}^\pi \left[r(s_h, \pi_h(s_h)) + \sum_{t=h+1}^{H-1} r(s_t, \pi_t(s_t)) + r_H(s_H) \middle| s_h = s \right] \\ &= r(s, \pi_h(s)) + \mathbb{E}^\pi \left[\sum_{t=h+1}^{H-1} r(s_t, \pi_t(s_t)) + r_H(s_H) \middle| s_h = s, a_h = \pi_h(s) \right] \\ &= r(s, \pi_h(s)) + \sum_{s' \in \mathcal{S}} \mathbb{P}(s_{h+1} = s' | s_h = s, a_h = \pi_h(s)) \mathbb{E}^\pi \left[\sum_{t=h+1}^{H-1} r(s_t, \pi_t(s_t)) + r_H(s_H) \middle| s_{h+1} = s' \right] \\ &= r(s, \pi_h(s)) + \sum_{s' \in \mathcal{S}} p(s'|s, \pi_h(s)) V_{h+1}^\pi(s') \end{aligned}$$



Bellman equations

Proposition

The optimal values V_h^* satisfy the **Bellman equations** :

$$V_H^*(s) = r_H(s)$$

$$V_h^*(s) = \max_{a \in \mathcal{A}} \left[r(s, a) + \sum_{s' \in \mathcal{S}} p(s'|s, a) V_{h+1}^*(s') \right] \quad \text{for all } h < H.$$

Moreover, an optimal policy is given by

$$\pi_h^*(s) \in \operatorname{argmax}_{a \in \mathcal{A}} \left[r(s, a) + \sum_{s'=1}^S p(s'|s, a) V_{h+1}^*(s') \right].$$

Bellman optimality principle (1957): “An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision”



Bellman equations

$$\begin{aligned}
 V_h^*(s) &= \max_{\pi_h, \pi_{h+1}, \dots} \mathbb{E}^{\pi} \left[\sum_{t=h}^{H-1} r(s_t, a_t) + r_H(s_H) \mid s_h = s \right] \\
 &= \max_{\pi_h, \pi_{h+1}, \dots} \sum_{a \in \mathcal{A}} \pi_h(a_h = a | s_h = s) \mathbb{E}^{\pi} \left[r(s, a) + \sum_{t=h+1}^{H-1} r(s_t, a_t) + r_H(s_H) \mid s_h = s, a_h = a \right] \\
 &= \max_{\pi_h, \pi_{h+1}, \dots} \sum_{a \in \mathcal{A}} \pi_h(a_h = a | s_h = s) \left[r(s, a) + \sum_{s' \in \mathcal{S}} p(s' | s, a) \mathbb{E}^{\pi_{h+1}, \dots} \left[\sum_{t=h+1}^{H-1} r(s_t, a_t) + r_H(s_H) \mid s_{h+1} = s' \right] \right] \\
 &= \max_{a \in \mathcal{A}} \left[r(s, a) + \sum_{s' \in \mathcal{S}} p(s' | s, a) \max_{\pi_{h+1}, \dots} V_{h+1}^{\pi_{h+1}, \dots}(s') \right] \\
 &= \max_{a \in \mathcal{A}} \left[r(s, a) + \sum_{s' \in \mathcal{S}} p(s' | s, a) V_{h+1}^*(s') \right]
 \end{aligned}$$

The maximizing policy is $\pi_h^*, \pi_{h+1}^*, \dots$ with

$$V_{h+1}^{\pi_{h+1}^*, \dots} = V_{h+1}^* = \operatorname{argmax}_{\pi} V_{h+1}^{\pi}$$

and a **deterministic** mapping $\pi_h^* : \mathcal{S} \rightarrow \mathcal{A}$ given by

$$\pi_h^*(s) \in \operatorname{argmax}_{a \in \mathcal{A}} \left[r(s, a) + \sum_{s' \in \mathcal{S}} p(s' | s, a) V_{h+1}^*(s') \right].$$



Bellman equations

Proposition

The optimal values V_h^* satisfy the **Bellman equations** :

$$V_H^*(s) = r_H(s)$$

$$V_h^*(s) = \max_{a \in \mathcal{A}} \left[r(s, a) + \sum_{s' \in \mathcal{S}} p(s'|s, a) V_{h+1}^*(s') \right] \text{ for all } h < H.$$

Moreover, an optimal policy is given by

$$\pi_h^*(s) \in \operatorname{argmax}_{a \in \mathcal{A}} \left[r(s, a) + \sum_{s'=1}^S p(s'|s, a) V_{h+1}^*(s') \right].$$

- Policies are computed by **backward induction**
- known as **dynamic programming**
- done by Bellman in 50s



DeepMind

Solving a MDP with an **Infinite Horizon** and a **Discount**



Policy evaluation: Value functions

Value functions

For a policy $\pi = (\pi_0, \pi_1, \dots)$,

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi_t(s_t)) \mid s_0 = s \right]$$

and

$$V^*(s) = \max_{\pi} \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi_t(s_t)) \mid s_0 = s \right]$$



Bellman equation for a stationary policy

Proposition

Any **stationary** deterministic policy π satisfies, for all $s \in \mathcal{S}$,

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^\pi(s')$$

Proof.

$$\begin{aligned} V^\pi(s) &= \mathbb{E}^\pi \left[r(s, \pi(s)) + \sum_{t=1}^{\infty} \gamma^t r(s_t, \pi(s_t)) \middle| s_0 = s \right] \\ &= r(s, \pi(s)) + \gamma \mathbb{E}^\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, \pi(s_t)) \middle| s_0 = s, a_0 = \pi(s) \right] \\ &= r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s_1 = s' | s_0 = s, a_0 = \pi(s)) \mathbb{E}^\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, \pi(s_t)) \middle| s_1 = s' \right] \\ &= r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, \pi(s)) \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) \middle| s_0 = s' \right] \end{aligned}$$



Option 1: Solving the Bellman equations

Fix a stationary, deterministic policy π .

Proposition

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^\pi(s')$$

Introducing the vectors

$$\begin{aligned} V^\pi &= (V^\pi(s))_{s=1}^S \in \mathbb{R}^S \\ r^\pi &= (r(s, \pi(s)))_{s=1}^S \in \mathbb{R}^S \end{aligned}$$

and the matrix

$$P^\pi = \left(p(s'|s, \pi(s)) \right)_{\substack{1 \leq s \leq S \\ 1 \leq s' \leq S}} \in \mathbb{R}^{S \times S},$$

the Bellman equation rewrites

$$V^\pi = r^\pi + \gamma P^\pi V^\pi$$



Option 2: Exploiting the Bellman operator

Bellman operator

The Bellman operator associated to a policy π is defined by

$$\begin{aligned} T^\pi : \mathbb{R}^S &\longrightarrow \mathbb{R}^S \\ V &\longmapsto T^\pi(V) \end{aligned}$$

where

$$T^\pi(V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, \pi(s)) V(s')$$

The Bellman equation for policy π rewrites

$$V^\pi = T^\pi V^\pi$$

→ the vector V^π is a **fixed point** of the Bellman operator T^π



Option 2: Exploiting the Bellman operator

Definition

Let $(X, |\cdot|)$ be a normed vector space.

A mapping $f : \mathcal{X} \rightarrow \mathcal{X}$ is called L -Lipschitz is

$$\forall (x, y) \in \mathcal{X}^2, |f(x) - f(y)| \leq L|x - y|.$$

If $L < 1$, f is called a **contraction**.

Banach's fixed point theorem

Let $(\mathcal{X}, |\cdot|)$ be a *complete* normed vector space and $f : \mathcal{X} \rightarrow \mathcal{X}$ be a contraction. Then

- ▶ there exists a **unique fixed point** x^* satisfying $f(x^*) = x^*$
- ▶ for every $x_0 \in \mathcal{X}$, the sequence defined by $x_{n+1} = f(x_n)$ for all $n \geq 0$ satisfies

$$\lim_{n \rightarrow \infty} x_n = x_*$$



Option 2: Exploiting the Bellman operator

Proposition

The operator

$$\begin{aligned} T^\pi: (\mathbb{R}^S, \|\cdot\|_\infty) &\longrightarrow (\mathbb{R}^S, \|\cdot\|_\infty) \\ V &\mapsto T^\pi(V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, \pi(s)) V(s') \end{aligned}$$

is a γ -contraction.

Proof.

$$\begin{aligned} \|T^\pi(V) - T^\pi(V')\|_\infty &= \sup_{s \in \mathcal{S}} |T^\pi(V)(s) - T^\pi(V')(s)| \\ &= \sup_{s \in \mathcal{S}} |\gamma \sum_{s' \in \mathcal{S}} p(s'|s, \pi(s)) (V(s') - V'(s'))| \\ &\leq \gamma \sum_{s' \in \mathcal{S}} p(s'|s, \pi(s)) \|V - V'\|_\infty \\ &= \gamma \|V - V'\|_\infty. \end{aligned}$$



Option 2: Exploiting the Bellman operator

Proposition

The operator

$$\begin{aligned} T^\pi : (\mathbb{R}^S, \|\cdot\|_\infty) &\longrightarrow (\mathbb{R}^S, \|\cdot\|_\infty) \\ V &\mapsto T^\pi(V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, \pi(s)) V(s') \end{aligned}$$

is a γ -contraction.

Consequence :

- ▶ V^π is the unique fixed point of T^π
- ▶ V^π can be approximated by an iterative scheme :

$$V^\pi = \lim_{k \rightarrow \infty} V_k$$

where

$$\begin{cases} V_0 & \in \mathbb{R} \\ V_{k+1} & = T(V_k) \text{ for all } k \geq 0. \end{cases}$$



Policy Evaluation:

- solving linear equations (matrix inversion)
- OR iterating the Bellman operator
- Anything else?

Monte-Carlo?

provides only an approximation but doesn't
require the knowledge of p and r



Bellman equations

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

(max over ALL (Markovian) policies, possibly non-stationary and randomized)

Proposition

The **optimal values** $V^*(s)$ satisfy the **Bellman equations** :

$$V^*(s) = \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^*(s') \right]$$

Moreover, an optimal policy is given by $\pi = (\pi, \pi, \dots)$ where

$$\pi^*(s) \in \operatorname{argmax}_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^*(s') \right].$$



Values and Q-Values

Q-value of a policy

$$Q^{\pi}(s, a) = \mathbb{E}^{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right]$$

is the expected cumulative reward when performing action a in the initial state s and then following policy π .

Optimal Q-Value

$$Q^{\star}(s, a) = \max_{\pi} Q^{\pi}(s, a)$$

Properties :

- ▶ $V^{\pi}(s) = Q^{\pi}(s, \pi(s))$
- ▶ $V^{\star}(s) = Q^{\star}(s, \pi^{\star}(s))$



Greedy policy

Definition

The greedy policy with respect to a value $V \in \mathbb{R}^S$, denoted by $\pi = \text{greedy}(V)$ is defined as

$$\pi(s) \in \operatorname{argmax}_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V(s') \right]$$

Intuition : policy that “improves” a policy with value V by taking the best possible first action and then following the policy

From the Bellman equations :

$$\pi^* = \text{greedy}(V^*) \quad \text{or} \quad \pi^* = \text{greedy}(Q^*)$$



Optimal Bellman Operator

Optimal Bellman operator

The **optimal Bellman operator** (or dynamic programming operator) is

$$\begin{aligned} T^* : \mathbb{R}^S &\longrightarrow \mathbb{R}^S \\ V &\mapsto T^*(V) \end{aligned}$$

where

$$T^*(V)(s) = \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V(s') \right]$$

The optimal value function satisfies

$$V^* = T^*(V^*)$$

→ V^* is a **fixed point** of the optimal Bellman operator T^* .



Optimal Bellman Operator

Optimal Bellman operator

The **optimal Bellman operator** for Q functions is

$$\begin{aligned} T^* : \mathbb{R}^{S \times A} &\longrightarrow \mathbb{R}^{S \times A} \\ Q &\mapsto T^*(Q) \end{aligned}$$

where

$$T^*(Q)(s, a) = r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \max_b Q(s', b)$$

The optimal value function satisfies

$$Q^* = T^*(Q^*)$$

→ Q^* is a **fixed point** of the optimal Bellman operator T^* .



Optimal Bellman Operator

Properties

The optimal Bellman operator is a γ -contraction :

$$\forall V, V' \in \mathbb{R}^S, \quad \|T^*(V) - T^*(V')\|_\infty < \gamma \|V - V'\|_\infty.$$

As a consequence :

- ▶ T^* admits a unique fixed point, V^*
- ▶ for every V_0 , the sequence $V_{n+1} = T^*(V_n)$ converges to V^*

Proof. Uses that for all functions $| \max f - \max g | \leq \max |f - g|$.

$$\begin{aligned} \|T^*(U) - T^*(V)\|_\infty &= \max_{s \in S} \left| \max_{a \in \mathcal{A}} \left\{ r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) U(s') \right\} - \max_{a' \in \mathcal{A}} \left\{ r(s, a') - \gamma \sum_{s' \in S} p(s' | s, a') V(s') \right\} \right| \\ &\leq \max_{s \in S} \max_{a \in \mathcal{A}} \left| r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) U(s') - r(s, a) - \gamma \sum_{s' \in S} p(s' | s, a) V(s') \right| \\ &= \gamma \max_{s \in S} \max_{a \in \mathcal{A}} \left| \sum_{s' \in S} p(s' | s, a) (U(s') - V(s')) \right| \\ &\leq \gamma \max_{s \in S} \max_{a \in \mathcal{A}} \sum_{s' \in S} |p(s' | s, a)| \|U - V\|_\infty \leq \gamma \|U - V\|_\infty \end{aligned}$$



Optimal Bellman Operator

Properties

The optimal Bellman operator is a γ -contraction :

$$\forall V, V' \in \mathbb{R}^S, \quad \|T^*(V) - T^*(V')\|_\infty < \gamma \|V - V'\|_\infty.$$

As a consequence :

- ▶ T^* admits a unique fixed point, V^*
- ▶ for every V_0 , the sequence $V_{n+1} = T^*(V_n)$ converges to V^*

→ provides a method for approximating V^*



Value iteration

- **Idea** : Iterate the operator T^* until V doesn't change much

Algorithm 1: Value Iteration

Input : $\epsilon =$ stopping parameter

$V_0 =$ any function (e.g. $V_0 \leftarrow 0_S$)

```
1  $V \leftarrow V_0$ 
2 while  $\|V - T^*(V)\| \geq \epsilon$  do
3   |  $V \leftarrow T^*(V)$ 
4 end
```

Return: $\pi = \text{greedy}(V)$

Theorem

Value iteration converges in at most $\log \left(\frac{\|T^*(V_0) - V_0\|_\infty}{\epsilon} \right) / \log(1/\gamma)$ iterations and outputs a policy π satisfying $\|V^\pi - V^*\| \leq \frac{\gamma\epsilon}{1-\gamma}$.

Proof. Exercise (contraction property + triangular inequality)

$$\|V^* - V_k\| \leq \|V^* - T^*(V_k)\| + \|T^*(V_k) - V_k\|$$



Policy iteration

- **Idea** : alternate between **policy evaluation** and **policy improvement**

Greedy policy

Recall that $\pi = \text{greedy}(V^\pi)$ is defined as

$$\pi(s) \in \operatorname{argmax}_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V(s') \right]$$

Policy improvement theorem

For any policy π , $\pi' = \text{greedy}(V^\pi)$ improves over π : $V^{\pi'} \geq V^\pi$.

Proof. uses some monotonicity property : $U \geq V \Rightarrow T^\pi(U) \geq T^\pi(V)$

- ❶ by definition of the greedy policy, $T^{\pi'}(V^\pi) = T^*(V^\pi) \geq T^\pi(V^\pi) = V^\pi$
- ❷ the monotonicity property yields (by induction) $(T^{\pi'})^n(V^\pi) \geq V^\pi$ for all $n \in \mathbb{N}$
- ❸ using that $\lim_{n \rightarrow \infty} (T^{\pi'})^n(V^\pi) = V^{\pi'}$ concludes.



Policy iteration

- **Idea** : alternate between **policy evaluation** and **policy improvement** .

Algorithm 2: Policy Iteration

Input : $\pi_0 =$ any policy (e.g. chosen at random)

```
1  $\pi \leftarrow \pi_0$ 
2  $\pi' \leftarrow \text{NULL}$ 
3 while  $\pi \neq \pi'$  do
4    $\pi' \leftarrow \pi$ 
5   Evaluate policy  $\pi$  : compute  $V^\pi$ 
6   Improve policy  $\pi$  :  $\pi \leftarrow \text{greedy}(V^\pi)$ 
7 end
```

Return: π

Theorem

Policy iteration terminates after a **finite number of steps** and outputs the **optimal policy** π^* .



Policy iteration

Why is that ?

Policy iteration generates a sequence of policies $\pi_0, \pi_1, \dots$ such that

$$\pi_{k+1} = \text{greedy}(V^{\pi_k}).$$

By the policy improvement theorem,

$$V^{\pi_{k+1}} \geq V^{\pi_k}$$

and if $\pi_{k+1} \neq \pi_k$ there must exist s such that $V^{\pi_{k+1}}(s) > V^{\pi_k}(s)$
(otherwise $V^{\pi_k} = V^{\pi_{k+1}} = T^{\pi_{k+1}}(V^{\pi_{k+1}}) = T^*(V^{\pi_k})$ thus $\pi_k = \pi^*$)

→ as there is a finite number of possible values of V^π (finite number of policies), the sequence must be stationary at some point.



Value iteration vs. Policy iteration

Both algorithm can be implemented with Q -values.

Value Iteration

Initialize Q_0 .

At iteration k :

$$V_k(s) = \max_a Q_{k-1}(s, a) \quad (\text{apply } T^*)$$

$$Q_k(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V_k(s')$$

Output : $\pi_K(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q_K(s, a)$

Policy iteration

Initialize π_0

At iteration k :

$$Q_{k-1}(s, a) = Q^{\pi_{k-1}}(s, a) \quad (\text{policy evaluation})$$

$$\pi_k(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q_{k-1}(s, a)$$

Output : π_K



Solving a MDP with an Infinite Horizon and **No Discount** (not covered)



DeepMind

2b

First RL
algorithms

- *TD-learning*
- *Q-learning*



$r(s, a)$ and $p(s' | s, a)$ are unknown

The RL interaction protocol :

$$\mathcal{F}_t = \sigma(s_0, a_0, r_0, s_1, \dots, s_{t-1}, a_{t-1}, r_{t-1}, s_t)$$

denotes the filtration generated by the observation up to time t .

At each time t :

- ▶ select an action $a_t \sim \pi_t(s_t)$ according to some **behavior policy**

π_t is $\mathcal{F}_t$ - measurable

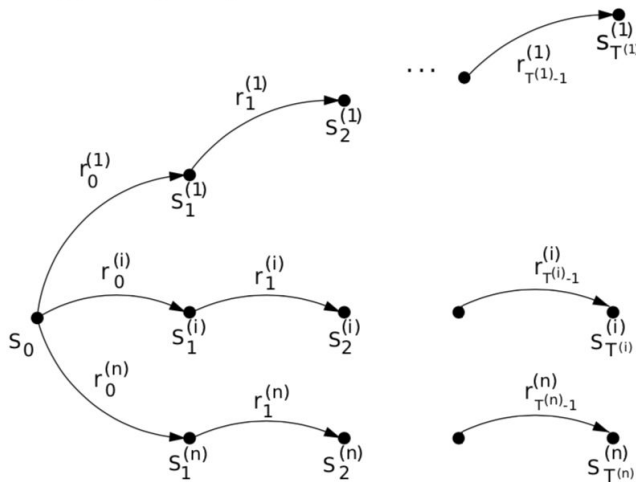
- ▶ observe the reward and next state

$$\begin{cases} r_t & \sim \nu_{(s_t, a_t)} \text{ such that } \mathbb{E}[r_t | \mathcal{F}_t] = r(s_t, a_t) \\ s_{t+1} & \sim p(\cdot | s_t, a_t) \end{cases}$$



$r(s, a)$ and $p(s' / s, a)$ are unknown

For example, starting from some state s_0 , one may observe several **trajectories** under a given policy.



One may also :

- ▶ restart in different states
- ▶ observe a single, very long, trajectory
- ▶ adaptively change the behavior policy



Monte-Carlo estimation of a mean

A naive way to estimate a value is to use its definition as an expectation :

$$V^\pi(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s \right]$$

► Given n (long enough) trajectories under π starting from $s_0^{(i)} = s$,

$$t^{(i)} = (s_0^{(i)}, r_0^{(i)}, s_1^{(i)}, r_1^{(i)}, \dots, s_{T(i)}^{(i)})$$

one can use the approximation

$$V^\pi(s) \simeq \frac{1}{n} \sum_{i=1}^n \left[\underbrace{\sum_{t=0}^{T(i)-1} \gamma^t r_t^{(i)}}_{\text{i.i.d. with mean } \simeq V^\pi(s)} \right].$$



Monte-Carlo estimation of a mean

More generally, considering Z_i that are i.i.d. with mean μ , one can define the Monte-Carlo estimator

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^n Z_i,$$

which has nice statistical properties.

Properties

- 1 Unbiased estimator : $\mathbb{E}[\hat{\mu}_n] = \mu$
- 2 Law of large number : $\lim_{n \rightarrow \infty} \hat{\mu}_n = \mu$, a.s.
- 3 Central Limit Theorem : $\frac{\sqrt{n}}{\sigma} (\hat{\mu}_n - \mu) \xrightarrow[n \rightarrow \infty]{\mathcal{L}} \mathcal{N}(0, 1)$



Monte-Carlo estimation of a mean

More generally, considering Z_i that are i.i.d. with mean μ , one can define the Monte-Carlo estimator

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^n Z_i,$$

which has nice statistical properties.

► Iterative rewriting

$$\hat{\mu}_n = \frac{n-1}{n} \hat{\mu}_{n-1} + \frac{1}{n} Z_n$$



Monte-Carlo estimation of a mean

More generally, considering Z_i that are i.i.d. with mean μ , one can define the Monte-Carlo estimator

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^n Z_i,$$

which has nice statistical properties.

► Iterative rewriting

$$\hat{\mu}_n = \hat{\mu}_{n-1} + \frac{1}{n} (Z_n - \hat{\mu}_{n-1})$$



Monte-Carlo estimation of a mean

More generally, considering Z_i that are i.i.d. with mean μ , one can define the Monte-Carlo estimator

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^n Z_i,$$

which has nice statistical properties.

► Iterative rewriting

$$\hat{\mu}_n = \hat{\mu}_{n-1} + \alpha_n (Z_n - \hat{\mu}_{n-1})$$

for the stepsize $\alpha_n = \frac{1}{n}$.

→ Could we choose other stepsizes?



Stochastic Approximation: Robbins-Monro

Goal : Find the solution to $\phi(x^*) = 0$ based on access to *noisy function evaluations*, i.e. for every x , one can observe a random value

$$Y = \phi(x) + \epsilon,$$

where ϵ has a zero mean (conditionally to previous queries).

Robbins-Monro algorithm (1951)

Given an initial x_0 , for all $n \geq 1$

- ▶ query a noisy evaluation $Y_n = \phi(x_{n-1}) + \epsilon_n$
- ▶ update $x_n = x_{n-1} + \alpha_n Y_n$

Particular case : estimate a mean μ based on i.i.d. samples Z_i

$$\phi(x) = \mu - x \quad (x^* = \mu)$$

→ with $Y_n = Z_n - \hat{\mu}_{n-1}$, the Robbins-Monro update is indeed

$$\hat{\mu}_n = \hat{\mu}_{n-1} + \alpha_n (Z_n - \hat{\mu}_{n-1})$$



Convergence of the Robbins-Monro algorithm

Theorem

Let $\phi : \mathcal{I} \subseteq \mathbb{R} \rightarrow \mathbb{R}$. Under the following assumptions

- ▶ ϕ is continuous and $\forall x \neq x^*, (x - x^*)\phi(x) < 0$
- ▶ there exists $C > 0$ such that $\mathbb{E}[Y_n | x_{n-1}] \leq C(1 + x_{n-1}^2)$.
- ▶ the stepsizes satisfy

$$\sum_{n=1}^{\infty} \alpha_n = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2 < \infty \quad (1)$$

under the Robbins-Monro algorithm, one has $x_n \xrightarrow{\text{a.s.}} x^*$.

Consequence : for the mean estimation problem, the sequence of iterates

$$\hat{\mu}_n = \hat{\mu}_{n-1} + \alpha_n(Z_n - \hat{\mu}_{n-1})$$

converges almost surely to μ for **any stepsize α_n satisfying (1)**.

→ NB : the Z_i may be dependent !



Robbins-Monro for fixed points

Goal : Find the solution to $x^* = T(x^*)$ based on access to **noisy evaluations** of $T(x)$.

Stochastic approximation for a fixed point

Given an initial x_0 , for all $n \geq 1$

- ▶ query a noisy evaluation $Z_n : \mathbb{E}[Z_n | x_{n-1}] = T(x_{n-1})$.
- ▶ update $x_n = x_{n-1} + \alpha_n (Z_n - x_{n-1})$

→ corresponds to the Robbins-Monro algorithm with

$$\phi(x) = T(x) - x \quad \text{and} \quad Y_n = Z_n - x_{n-1}.$$



DeepMind

Temporal Differences

TD(0), TD(1), TD(λ)



Temporal Differences

Given a policy π , we want to compute V^π , which satisfies

$$V^\pi = T^\pi(V^\pi)$$

where $T^\pi(V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V(s')$.

- Given a current estimate $\hat{V}$, if we generate a trajectory under π

$$s_0, r_0, s_1, r_1, \dots, s_T, r_T,$$

one can produce noisy evaluations of $T^\pi(\hat{V})(s_k)$ for all $k \in \{0, \dots, T-1\}$ using

$$Z_k = r_k + \gamma \hat{V}(s_{k+1}).$$

$$\mathbb{E}[Z_k | \hat{V}, s_0, r_0, \dots, s_k] = r(s_k, \pi(s_k)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s_k, \pi(s_k)) \hat{V}(s')$$



Temporal Differences

Given a policy π , we want to compute V^π , which satisfies

$$V^\pi = T^\pi(V^\pi)$$

where $T^\pi(V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V(s')$.

- Given a current estimate $\hat{V}$, if we generate a trajectory under π

$$s_0, r_0, s_1, r_1, \dots, s_T, r_T,$$

one can produce **noisy evaluations of $T^\pi(\hat{V})(s_k)$** for all $k \in \{0, \dots, T-1\}$ using

$$Z_k = r_k + \gamma \hat{V}(s_{k+1}).$$

$$\mathbb{E}[Z_k | \hat{V}, s_0, r_0, \dots, s_k] = T^\pi(\hat{V})(s_k)$$



Temporal Differences

Given a policy π , we want to compute V^π , which satisfies

$$V^\pi = T^\pi(V^\pi)$$

where $T^\pi(V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V(s')$.

- Given a current estimate $\hat{V}$, if we generate a trajectory under π

$$s_0, r_0, s_1, r_1, \dots, s_T, r_T,$$

one can produce **noisy evaluations** of $T^\pi(\hat{V})(s_k)$ for all $k \in \{0, \dots, T-1\}$ using

$$Z_k = r_k + \gamma \hat{V}(s_{k+1}).$$

- “Robbins-Monro” update : $\hat{V}(s_k) \leftarrow \hat{V}(s_k) + \alpha (Z_k - \hat{V}(s_k))$



Temporal Differences

The Robbins-Monro update rewrites

$$\hat{V}(s_k) \leftarrow \hat{V}(s_k) + \alpha \delta_k(\hat{V})$$

introducing the k -th temporal difference (or TD error) :

$$\delta_k(\hat{V}) := r_k + \gamma \hat{V}(s_{k+1}) - \hat{V}(s_k).$$

► Interpretation :

$$\delta_k(\hat{V}) := \underbrace{r_k + \gamma \hat{V}(s_{k+1})}_{\text{new target}} - \underbrace{\hat{V}(s_k)}_{\text{previous estimate}}$$

The value of the estimate is moved toward the value of the new target, itself built upon $\hat{V}$.

→ Bootstrapping !

Sutton, *Learning to Predict by the Method of Temporal Differences*, 1988



Algorithm: TD(0)

Input : π : policy, T : number of iterations, $(\alpha_i(s))_{i \in \mathbb{N}}$: stepsizes,
 $V_0 \in \mathbb{R}^S$: initial values, $s_0 \in \mathcal{S}$: initial state (arbitrary)

```
1  $V \leftarrow V_0, s \leftarrow s_0$ 
2  $N \leftarrow 0_S$ 
3 for  $t = 1, \dots, T$  do
4    $N(s) \leftarrow N(s) + 1$            \\ update the number of visits of state s
5    $(r, s') = \text{step}(s, \pi(s))$        \\ perform a transition under  $\pi$ 
6    $V(s) \leftarrow V(s) + \alpha_{N(s)}(s) (r + \gamma V(s') - V(s))$ 
7    $s \leftarrow s'$ 
8 end
Return:  $V$ 
```

$$(r, s') = \text{step}(s, \pi(s)) \Leftrightarrow \begin{cases} r & \sim \nu_{(s, \pi(s))} \\ s' & \sim p(\cdot | s, \pi(s)) \end{cases}$$



Algorithm: TD(0)

Input : π : policy, T : number of iterations, $(\alpha_i(s))_{i \in \mathbb{N}}$: stepsizes,
 $V_0 \in \mathbb{R}^S$: initial values, $s_0 \in \mathcal{S}$: initial state (arbitrary)

```
1  $V \leftarrow V_0, s \leftarrow s_0$ 
2  $N \leftarrow 0_S$ 
3 for  $t = 1, \dots, T$  do
4    $N(s) \leftarrow N(s) + 1$            \\ update the number of visits of state s
5    $(r, s') = \text{step}(s, \pi(s))$       \\ perform a transition under  $\pi$ 
6    $V(s) \leftarrow V(s) + \alpha_{N(s)}(s) (r + \gamma V(s') - V(s))$ 
7    $s \leftarrow s'$ 
8 end
Return:  $V$ 
```

Possible variants :

- ▶ instead of a very long trajectory, smaller trajectories with restarts
- ▶ update V for all states only at the end of each trajectory (offline)
- ▶ for each state s , update $V(s)$ at most once per trajectory (first visit)



Algorithm: TD(0)

Theorem

It the step-size (also called *learning rate*) satisfy the Robbins-Monro conditions in all state s :

$$\sum_{i=1}^{\infty} \alpha_i(s) = +\infty \quad \text{and} \quad \sum_{i=1}^{\infty} (\alpha_i(s))^2 < +\infty$$

and *all states are visited infinitely often*, then

$$\lim_{T \rightarrow \infty} \hat{V}_T = V^\pi,$$

where $\hat{V}_T$ denotes the output of TD(0) after T iterations.

► **Typical choice** : $\alpha_i(s) = \frac{1}{i^\beta}$ for $\beta \in (1/2, 1]$.

$$\hat{V}_t(s) = \hat{V}_{t-1}(s) + \frac{1}{N_t(s)^\beta} \left(r + \gamma \hat{V}_{t-1}(s') - \hat{V}_{t-1}(s) \right)$$

with $N_t(s)$ the number of visits of s up to the t -th iteration.

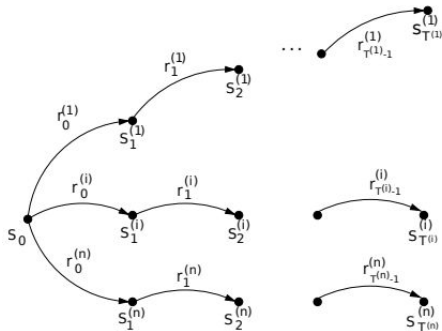


From Monte-Carlo to Temporal Differences

Incremental Monte-Carlo for the estimation of

$$V^\pi(s_0) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 \right]$$

based on n trajectories starting in s_0 :



Update after the i -th trajectory :

$$\hat{V}_i(s_0) = \hat{V}_{i-1}(s_0) + \alpha_i \left(\sum_{t=0}^{T^{(i)}-1} \gamma^t r_t^{(i)} - \hat{V}_{i-1}(s_0) \right)$$

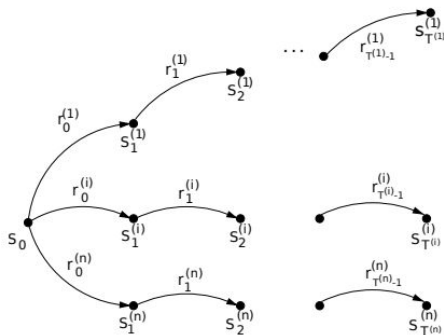


From Monte-Carlo to Temporal Differences

Incremental Monte-Carlo for the estimation of

$$V^\pi(s_0) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 \right]$$

based on n trajectories starting in s_0 :



Update after the i -th trajectory :

$$\hat{V}_i(s_0) = \hat{V}_{i-1}(s_0) + \alpha_i \left(r_0^{(i)} - \hat{V}_{i-1}(s_0) + \sum_{t=1}^{T^{(i)}-1} \gamma^t r_t^{(i)} \right)$$

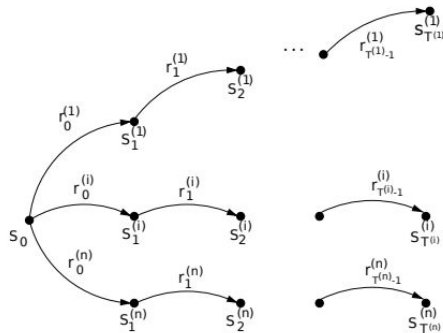


From Monte-Carlo to Temporal Differences

Incremental Monte-Carlo for the estimation of

$$V^\pi(s_0) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 \right]$$

based on n trajectories starting in s_0 :



Update after the i -th trajectory :

$$\hat{V}_i(s_0) = \hat{V}_{i-1}(s_0) + \alpha_i \left(r_0^{(i)} + \gamma \hat{V}_{i-1}(s_1^{(i)}) - \hat{V}_{i-1}(s_0) + \sum_{t=1}^{T^{(i)}} \gamma^t r_t^{(i)} - \gamma \hat{V}_{i-1}(s_1^{(i)}) \right)$$

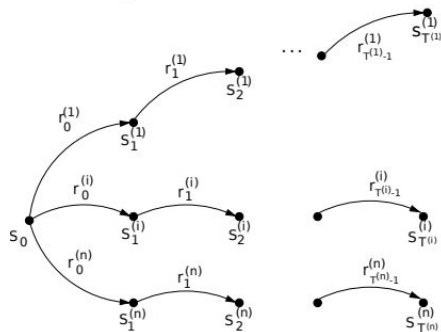


From Monte-Carlo to Temporal Differences

Incremental Monte-Carlo for the estimation of

$$V^\pi(s_0) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 \right]$$

based on n trajectories starting in s_0 :



Update after the i -th trajectory :

$$\hat{V}_i(s_0) = \hat{V}_{i-1}(s_0) + \alpha_i \left(\delta_0^{(i)}(\hat{V}_{i-1}) + \gamma \left[\sum_{t=1}^{T^{(i)}-1} \gamma^{t-1} r_t^{(i)} - \hat{V}_{i-1}(s_1^{(i)}) \right] \right)$$

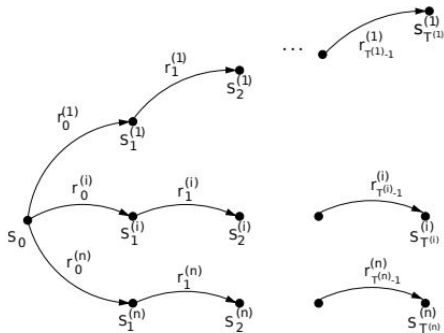


From Monte-Carlo to Temporal Differences

Incremental Monte-Carlo for the estimation of

$$V^\pi(s_0) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 \right]$$

based on n trajectories starting in s_0 :



Update after the i -th trajectory : ➔ features the **temporal differences**

$$\hat{V}_i(s_0) = \hat{V}_{i-1}(s_0) + \alpha_i \left(\sum_{t=0}^{T^{(i)}-1} \gamma^t \delta_t^{(i)} (\hat{V}_{i-1}) - \gamma^{T^{(i)}} \hat{V}_{i-1}(s_{T^{(i)}}) \right)$$



From Monte-Carlo to Temporal Differences

$$\hat{V}_i(s_0) \simeq \hat{V}_{i-1}(s_0) + \alpha_i \left(\sum_{t=0}^{T^{(i)}-1} \gamma^t \delta_t^{(i)}(\hat{V}_{i-1}) \right)$$

Limitation of naive Monte-Carlo :

- ▶ performing a full trajectory is needed before the update
- ▶ we only update the value of the initial state s_0

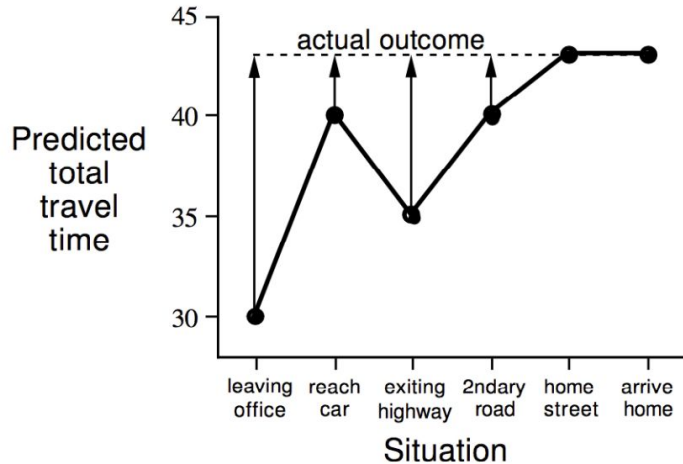
Extension :

- update the values of multiple states after each trajectory
- online updates, after each transition

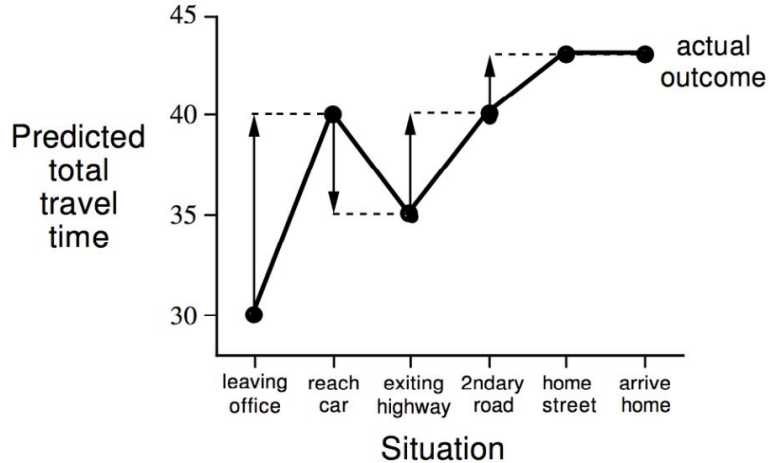


Monte-Carlo cs. Temporal Differences

Changes recommended by
Monte Carlo methods ($\alpha=1$)



Changes recommended
by TD methods ($\alpha=1$)



Every visit Monte-Carlo

Every visits Monte-Carlo (a.k.a. **TD(1)**) : after the i -th trajectory, instead of updating only $\hat{V}(s_0)$, for all $k = T^{(i)} - 1$ down to 0,

$$\hat{V}(s_k^{(i)}) \leftarrow \hat{V}(s_k^{(i)}) + \alpha_i(s_k^{(i)}) \left(\sum_{t=k}^{T^{(i)}-1} \gamma^{t-k} r_t^{(i)} - \hat{V}(s_k^{(i)}) \right)$$

Remarks :

- ▶ exact formula in the episodic setting ($s_{T^{(i)}}^{(i)}$ terminal state)
- ▶ multiple updates of states visited more than once in the trajectory
- ▶ **first visit** variant : update $s_k^{(i)}$ only is $s_k^{(i)} \notin \{s_0^{(i)}, \dots, s_{k-1}^{(i)}\}$



Every visit Monte-Carlo

Every visits Monte-Carlo (a.k.a. **TD(1)**) : after the i -th trajectory, instead of updating only $\hat{V}(s_0)$, for all $k = T^{(i)} - 1$ down to 0,

$$\hat{V}(s_k^{(i)}) \leftarrow \hat{V}(s_k^{(i)}) + \alpha_i(s_k^{(i)}) \left(\sum_{t=k}^{T^{(i)}-1} \gamma^{t-k} \delta_t^{(i)}(\hat{V}) \right)$$

Remarks :

- ▶ exact formula in the episodic setting ($s_{T^{(i)}}^{(i)}$ terminal state)
- ▶ multiple updates of states visited more than once in the trajectory
- ▶ **first visit** variant : update $s_k^{(i)}$ only is $s_k^{(i)} \notin \{s_0^{(i)}, \dots, s_{k-1}^{(i)}\}$



An online version of TD(1)

Along a trajectory following π , after the t -th transition :

- compute the temporal difference according to the current estimate

$$\delta_t = r_t + \gamma \hat{V}_{t-1}(s_{t+1}) - \hat{V}_{t-1}(s_t)$$

- update **ALL states** with this new temporal difference : for all s ,

$$\begin{cases} z_t(s) &= \mathbb{1}_{(s_t=s)} + \gamma z_{t-1}(s) \\ \hat{V}_t(s) &= \hat{V}_{t-1}(s) + \alpha_{N(s)}(s) z_t(s) \delta_t \end{cases}$$

Intuition: previous states are also responsible for the current TD error



An online version of TD(1)

Along a trajectory following π , after the t -th transition :

- compute the temporal difference according to the current estimate

$$\delta_t = r_t + \gamma \hat{V}_{t-1}(s_{t+1}) - \hat{V}_{t-1}(s_t)$$

- update **ALL states** with this new temporal difference : for all s ,

$$\begin{cases} z_t(s) &= \mathbb{1}_{(s_t=s)} + \cancel{\gamma z_{t-1}(s)} \\ \hat{V}_t(s) &= \hat{V}_{t-1}(s) + \alpha_{N(s)}(s) z_t(s) \delta_t \end{cases}$$

Intuition: previous states are also responsible for the current TD error

TD(0) only update the value in the visited state



An interpolation : TD(λ)

→ can be motivated by a stochastic approximation scheme leveraging the multi-step Bellman operator :

$$V^\pi(s) = T_m^\pi(V^\pi)(s) ,$$

with $T^{\pi,m}(V)(s) = \mathbb{E}^\pi \left[\sum_{t=0}^m \gamma^t r_t + \gamma^{m+1} V(s_{m+1}) \middle| s_0 = s \right] .$

As the “right” number of steps is not known, we define an average

$$T_\lambda^\pi = (1 - \lambda) \sum_{m=0}^{\infty} \lambda^m T^{\pi,m}$$

for $\lambda \in [0, 1)$ which still satisfies $V^\pi = T_\lambda^\pi(V^\pi)$.



An interpolation : TD(λ)

An estimate of $T_{\lambda}^{\pi}(V)(s_0)$, based on an infinitely long trajectory under π starting from s_0 is

$$Z_0 = (1 - \lambda) \sum_{m=0}^{\infty} \lambda^m \left[\sum_{t=0}^m \gamma^t r_t + \gamma^{m+1} V(s_{m+1}) \right].$$

It satisfies

$$\begin{aligned} Z_0 - V(s_0) &= (1 - \lambda) \sum_{m=0}^{\infty} \lambda^m \left[\sum_{t=0}^m \gamma^t r_t + \gamma^{m+1} V(s_{m+1}) - V(s_0) \right] \\ &= (1 - \lambda) \sum_{m=0}^{\infty} \lambda^m \left[\sum_{t=0}^m \delta_t(V) \right] = \sum_{t=0}^{\infty} \gamma^t \delta_t(V) \left[(1 - \lambda) \sum_{m=t}^{\infty} \lambda^m \right] \\ &= \sum_{t=0}^{\infty} (\gamma \lambda)^t \delta_t(V) \end{aligned}$$

Hence the (idealized) R-M update $\hat{V}(s_0) \leftarrow \hat{V}(s_0) + \alpha \sum_{t=0}^{\infty} (\gamma \lambda)^t \delta_t(\hat{V})$

→ the parameter λ is discounting the future temporal differences



The TD(λ) algorithm

An online implementation of TD(λ) idea is the following :

TD(λ)

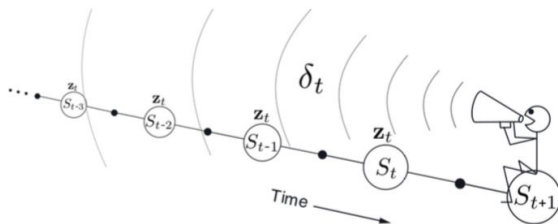
Along a trajectory following π , after the t -th transition :

- compute the temporal difference according to the current estimate

$$\delta_t = r_t + \gamma \hat{V}_{t-1}(s_{t+1}) - \hat{V}_{t-1}(s_t)$$

- update all states with this new temporal difference : for all s ,

$$\begin{cases} z_t(s) &= \mathbb{1}_{(s_t=s)} + \gamma \lambda z_{t-1}(s) \\ \hat{V}_t(s) &= \hat{V}_{t-1}(s) + \alpha_{N(s)}(s) z_t(s) \delta_t \end{cases}$$



source : [Sutton and Barto, 1998]



The TD(λ) algorithm

An online implementation of TD(λ) idea is the following :

TD(λ)

Along a trajectory following π , after the t -th transition :

- compute the temporal difference according to the current estimate

$$\delta_t = r_t + \gamma \hat{V}_{t-1}(s_{t+1}) - \hat{V}_{t-1}(s_t)$$

- update all states with this new temporal difference : for all s ,

$$\begin{cases} z_t(s) &= \mathbb{1}_{(s_t=s)} + \gamma \lambda z_{t-1}(s) \\ \hat{V}_t(s) &= \hat{V}_{t-1}(s) + \alpha_{N(s)}(s) z_t(s) \delta_t \end{cases}$$

- $\lambda = 0$ recovers the TD(0) algorithm (high bias, smaller variance)
- $\lambda = 1$ recovers the TD(1) algorithm (small bias, higher variance)





Inria
INVENTEURS DU MONDE NUMÉRIQUE

Reinforcement Learning

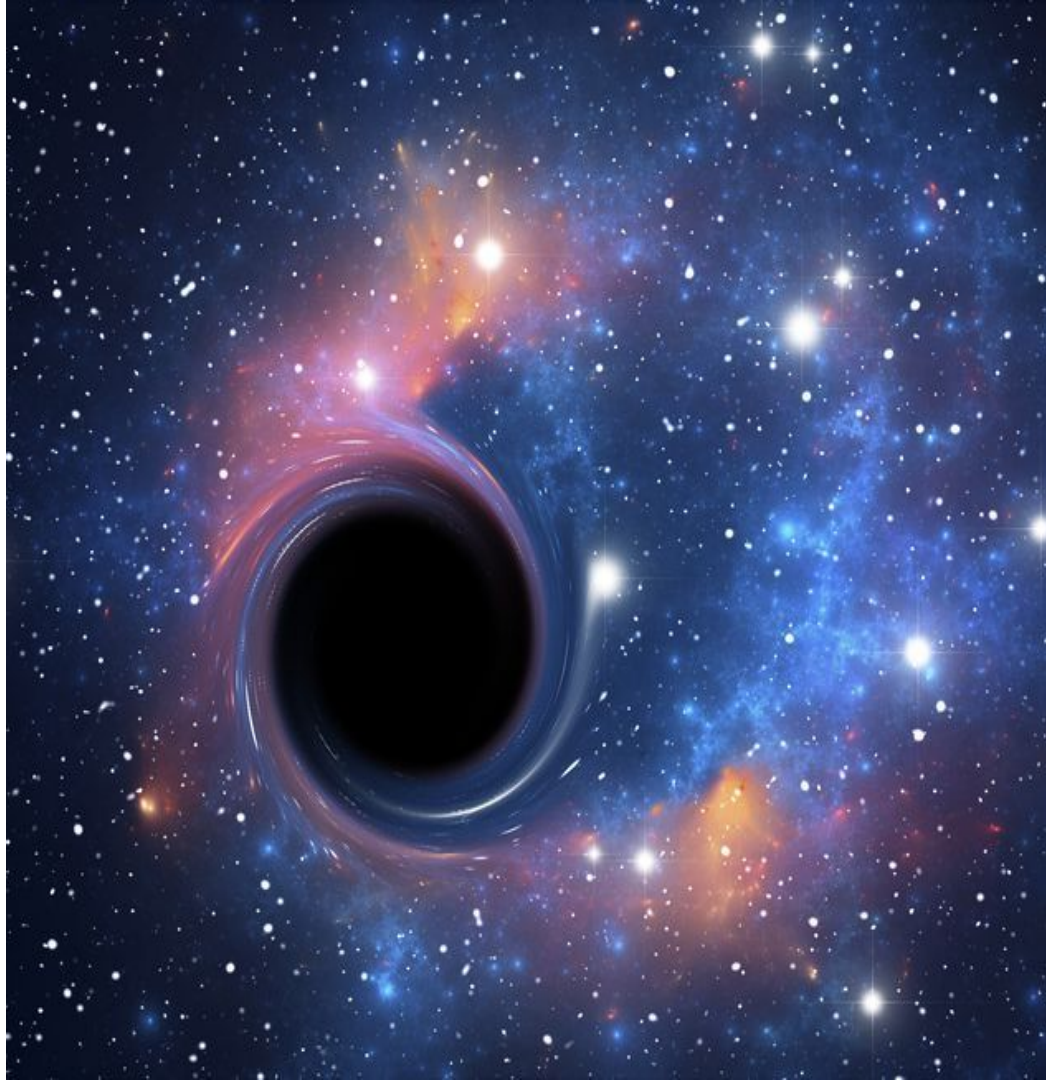
Day 2

Speaker: Michal Valko

Slide contributors: Nando de Freitas, Emilie Kaufmann, Hado, Diana, Bobak Shahriari, Caglar Gulcehre, David Silver, Feryal Behbahani, Ksenia Konyushkova, Marc Bellemare, Matt Hoffman, Tom Le Paine, and Tom Schaul
Special thanks: Nando, Emilie, Diana, Hado, Corentin

Outline

1. RL Concepts
2. a) MDPs: know r, p
b) MDPs: unknow r, p
3. Deep Q-networks
4. Policy gradients
5. Distributional RL
6. Exploration/Exploitation
7. Monte-Carlo Tree Search



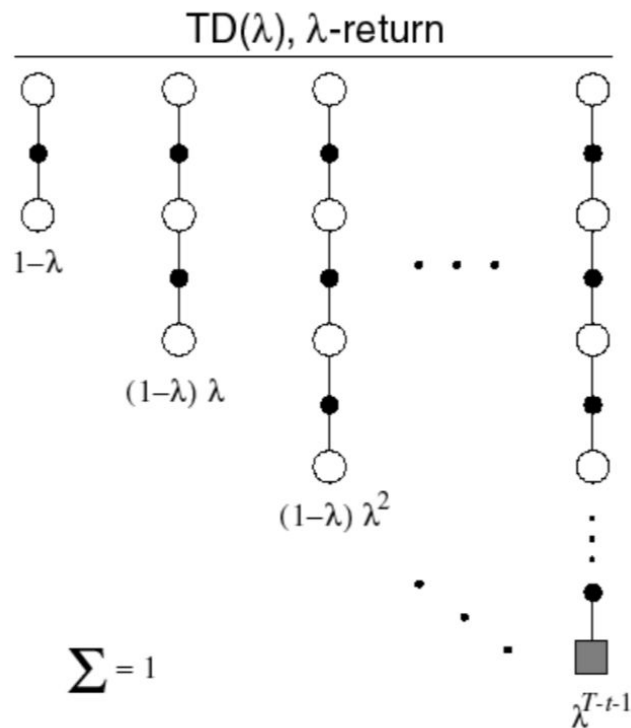
Yesterday...

- Agents, environments, rewards, transition matrix
- Policies
- Optimal policies
- Value function
- Q-function
- Objectives of RL
- Value iteration
- Policy iteration
- Monte-Carlo
- Temporal differences





The TD(λ) algorithm (intuition)



TD methods for learning the optimal policy ?

TD methods permit to approximately compute V^π for a given policy π

→ can we use them to get to π^* ?

Hope : policy evaluation is a central ingredient in **Policy Iteration**

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 = \text{greedy}(V^{\pi_0}) \rightarrow V^{\pi_1} \rightarrow \pi_2 = \text{greedy}(V^{\pi_1}) \rightarrow V^{\pi_2} \rightarrow \dots \rightarrow \pi^*$$

Limitation : the policy improvement step cannot be performed without the knowledge of the MDP parameters

$$\begin{aligned}\pi_{k+1} &= \text{greedy}(V^{\pi_k}) \\ \Leftrightarrow \pi_{k+1}(s) &= \operatorname{argmax}_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^{\pi_k}(s') \right]\end{aligned}$$

Other possibility : work directly with Q -values !



DeepMind

Q-learning



Reminder: Q-values

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^\pi(s')$$

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

Properties

- ❶ Q^* satisfies the Bellman equations

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \max_{a' \in \mathcal{A}} Q^*(s', a')$$

- ❷ $V^*(s) = Q^*(s, \pi^*(s))$
- ❸ $\pi^* = \text{greedy}(Q^*)$, i.e. $\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a)$

→ New goal : Learning Q^*



A stochastic approximation scheme for Q^* ?

- Q^* also satisfies a fixed point equation : $Q^* = T^*(Q^*)$ where

$$T^*(Q)(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \max_{a' \in \mathcal{A}} Q(s', a').$$

- Noisy evaluations of $T^*(Q)(s_k, a_k)$ along a trajectory :

$$Z_k = r_k + \gamma \max_{a' \in \mathcal{A}} Q(s_{k+1}, a')$$

satisfies $\mathbb{E}[Z_k | \mathcal{F}_k] = T^*(Q)(s_k, a_k)$.

(for *any behavior policy*)

- **Robbins-Monro update :**

$$\hat{Q}(s_k, a_k) \leftarrow \hat{Q}(s_k, a_k) + \alpha \left(r_k + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s_{k+1}, a') - \hat{Q}(s_k, a_k) \right)$$



Q-Learning

Input : T : number of iterations, $(\alpha_i(s, a))_{i \in \mathbb{N}}$: step-sizes,
 $Q_0 \in \mathbb{R}^{S \times A}$: initial Q-values, $s_0 \in \mathcal{S}$: initial state (arbitrary)
 π_t : behavior policy

```
1  $Q \leftarrow Q_0, s \leftarrow s_0$ 
2  $N \leftarrow 0_{S \times A}$ 
3 for  $t = 1, \dots, T$  do
4    $a \sim \pi_t(s)$  \\ choose an action under the behavior policy
5    $N(s, a) \leftarrow N(s, a) + 1$  \\ update the number of visits of (s, a)
6    $(r, s') = \text{step}(s, a)$  \\ perform a transition
7    $Q(s, a) \leftarrow Q(s, a) + \alpha_{N(s, a)}(s, a) (r + \gamma \max_b Q(s', b) - Q(s, a))$ 
8    $s \leftarrow s'$ 
9 end
Return:  $Q, \pi = \text{greedy}(Q)$ 
```

[Watkins, 1989]



Q-Learning

Input : T : number of iterations, $(\alpha_i(s, a))_{i \in \mathbb{N}}$: **step-sizes**,
 $Q_0 \in \mathbb{R}^{S \times A}$: initial Q-values, $s_0 \in \mathcal{S}$: initial state (arbitrary)
 π_t : **behavior policy**

```
1  $Q \leftarrow Q_0, s \leftarrow s_0$ 
2  $N \leftarrow 0_{S \times A}$ 
3 for  $t = 1, \dots, T$  do
4    $a \sim \pi_t(s)$  \ \ choose an action under the behavior policy
5    $N(s, a) \leftarrow N(s, a) + 1$  \ \ update the number of visits of (s, a)
6    $(r, s') = \text{step}(s, a)$  \ \ perform a transition
7    $Q(s, a) \leftarrow Q(s, a) + \alpha_{N(s, a)}(s, a) (r + \gamma \max_b Q(s', b) - Q(s, a))$ 
8    $s \leftarrow s'$ 
9 end
Return:  $Q, \pi = \text{greedy}(Q)$ 
```

[Watkins, 1989]



Q-Learning

Theorem

It the step-size (also called *learning rate*) satisfy the Robbins-Monro conditions in all state action pair (s, a) :

$$\sum_{i=1}^{\infty} \alpha_i(s, a) = +\infty \quad \text{and} \quad \sum_{i=1}^{\infty} (\alpha_i(s, a))^2 < +\infty$$

and *all states-action pairs are visited infinitely often* , then

$$\lim_{T \rightarrow \infty} \hat{Q}_T = Q^*,$$

where $\hat{Q}_T$ denotes the output of T iterations of Q-Learning.

→ typical step-sizes choice : $\alpha_i(s, a) = \frac{1}{i^\beta}$ with $\beta \in (1/2, 1]$.



Behavior Policy

- **Constraint** : all state-action pairs need to be visited infinitely often

$$\pi_t(s) = \mathcal{U}(\mathcal{S}) \rightarrow a_t \text{ chosen uniformly at random?}$$

- **Idea** : we care about π^* , we need to refine our estimate of Q^* in the pairs $(s, \pi^*(s))$ / we may want to maximize rewards while learning

$$\pi_t = \text{greedy}(\hat{Q}_{t-1})?$$

ϵ -greedy exploration [Sutton and Barto, 1998]

The ϵ -greedy policy performs the following :

- with probability ϵ , select $a_t \sim \mathcal{U}(\mathcal{S})$
- with probability $1 - \epsilon$, select $a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \hat{Q}_t(s_t, a)$

- tends to the greedy policy when $\epsilon \rightarrow 0$



Behavior Policy

- **Constraint** : all state-action pairs need to be visited infinitely often

$$\pi_t(s) = \mathcal{U}(S) \rightarrow a_t \text{ chosen uniformly at random?}$$

- **Idea** : we care about π^* , we need to refine our estimate of Q^* in the pairs $(s, \pi^*(s))$ / we may want to maximize rewards while learning

$$\pi_t = \text{greedy}(\hat{Q}_{t-1})?$$

Boltzmann (or softmax) exploration [Sutton and Barto, 1998]

The **softmax policy** with temperature τ is given by

$$(\pi_t(s))_a = \frac{\exp(\hat{Q}_t(s, a)/\tau)}{\sum_{a' \in \mathcal{A}} \exp(\hat{Q}_t(s, a')/\tau)}$$

and $a_t \sim \pi_t(s_t)$.

→ tends to the greedy policy when $\tau \rightarrow 0$



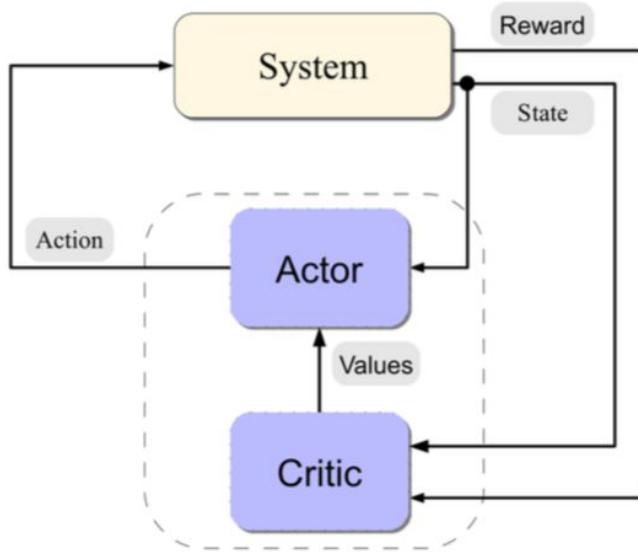
DeepMind

Actor/Critic

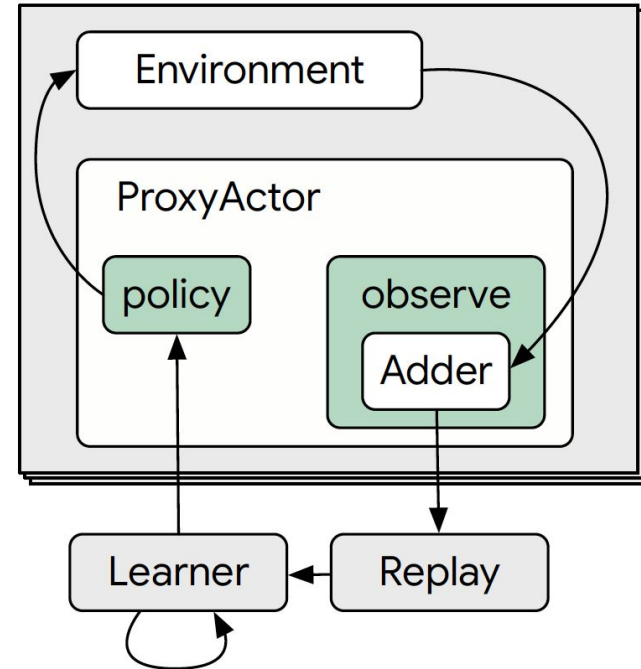


Actor/Critic architecture

- ▶ **the actor** : update its policy to improve the value given by the critic
- ▶ **the critic** : evaluates the actor's policy



source : [Szepesvari, 2010]



Actor/Critic architecture

Policy Iteration is an extreme example of an Actor/Critic architecture

- ▶ **the actor** : “acts” with $\pi = \text{greedy}(V)$ where V is the value provided by the critic
- ▶ **the critic** : computes V^π where π is the current actor’s policy



Actor/Critic architecture

Policy Iteration is an extreme example of an Actor/Critic architecture :

- ▶ **the actor** : performs **policy improvement**
- ▶ **the critic** : performs **policy evaluation**
- Actor/Critic is also referred to as **Generalized Policy Iteration**

[Sutton and Barto, 1998]

There are many algorithms of this type !



DeepMind

SARSA

a critic



SARSA algorithm

After observing the actor's recent behavior $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$, update

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha \left(r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t) \right)$$

State Action Reward State Action (SARSA) update

- ▶ if the actor is following a fixed policy π ($a_t = \pi(s_t)$), SARSA=TD(0)
- can be similarly generalized to SARSA(λ)

Remark : we will see later more sophisticated critics

- ▶ e.g. approximate Q and the greedy policy π with a neural network



An actor: Greedy + Exploration

The actor's goal : move its behavior policy towards being greedy with respect to the Q -value provided by the critic

We already saw several possibilities :

- ▶ greedy (doesn't explore enough !)
- ▶ ϵ -greedy policy
- ▶ softmax policy with temperature τ

→ actors using parametric policies → Policy Gradient

→ actors using multi-step look-ahead → Monte-Carlo Tree Search



SARSA algorithm

Input : T : number of iterations, $(\alpha_i(s, a))_{i \in \mathbb{N}}$: step-sizes,
 $Q_0 \in \mathbb{R}^{S \times A}$: initial Q-values, $s_0 \in \mathcal{S}$: initial state (arbitrary)
 $a_0 \in \mathcal{A}$: first action (arbitrary)

```
1  $Q \leftarrow Q_0, s \leftarrow s_0, a \leftarrow a_0$ 
2  $N \leftarrow 0_{S \times A}$ 
3 for  $t = 1, \dots, T$  do
4    $N(s, a) \leftarrow N(s, a) + 1$            \\ update the number of visits of (s, a)
5    $(r, s') = \text{step}(s, a)$                  \\ perform a transition
6    $a' \sim \pi_{Q,t}(s')$                    \\ choose an action with a policy parameterized by Q
7    $Q(s, a) \leftarrow Q(s, a) + \alpha_{N(s,a)}(s, a) (r + \gamma Q(s', a') - Q(s, a))$ 
8    $s \leftarrow s', a \leftarrow a'$ 
9 end
Return:  $Q, \pi = \text{greedy}(Q)$ 
```

For example,

$$\pi_{Q,t}(a|s') = (1 - \epsilon_t) \mathbb{1} \left(a = \underset{b \in \mathcal{A}}{\operatorname{argmax}} Q(s', b) \right) + \frac{\epsilon_t}{|\mathcal{A}|}$$



SARSA's guarantee

Theorem

SARSA converges to Q^* provided that

- ▶ the step-size (also called *learning rate*) satisfy the Robbins-Monro conditions in all state action pair (s, a) :

$$\sum_{i=1}^{\infty} \alpha_i(s, a) = +\infty \quad \text{and} \quad \sum_{i=1}^{\infty} (\alpha_i(s, a))^2 < +\infty$$

- ▶ all states-action pairs are visited infinitely often
- ▶ the exploration policy $\pi_{Q,t}$ becomes greedy when $t \rightarrow \infty$

→ possible choices : $\alpha_i(s, a) = \frac{1}{t^\beta}$ and $\epsilon_t = \frac{1}{t}$ for ϵ -greedy



Q-Learning versus SARSA

The update rules of the two algorithms are close but not identical :

► **Q-Learning :**

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a') - \hat{Q}(s_t, a_t) \right)$$

► **SARSA :**

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha \left(r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t) \right)$$

Both aim at learning the **target policy** $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$.

- Q-Learning converges for **any behavior policy** (exploring enough)
off-policy learning
- for SARSA the behavior policy is close to the estimated target policy
on-policy learning



DeepMind

3

Deep Q-Networks



Representing the Q-value using a neural network

- ▶ Represent value function $Q_w(x, a)$ with a neural net.
- ▶ How to train $Q_w(x, a)$? We don't have supervised values. We only know we want

$$Q_w(x, a) \approx r(x, a) + \gamma \mathbb{E}_{x'} \left[\max_{a'} Q_w(x', a') \mid x, a \right]$$

- ▶ After a transition $x_t, a_t \rightarrow x_{t+1}$,

$$\text{train } Q_w(x_t, a_t) \text{ to predict } \underbrace{r_t + \gamma \max_a Q_w(x_{t+1}, a)}_{\text{target values}}$$

- ▶ Minimize loss $\underbrace{\left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right)^2}_{\text{temporal difference } \delta_t}$.

- ▶ At the end of learning, $\mathbb{E}[\delta_t] = 0$.



DeepMind

DQN



DQN

Problems: (1) data is not iid, (2) target values change

Idea: be as close as possible to supervised learning

1. Dissociate acting from learning:
 - ▶ Interact with the environments by following behavior policy
 - ▶ Store transition samples x_t, a_t, x_{t+1}, r_t into a memory replay
 - ▶ Train by replaying iid from memory
2. Use target network fixed for a while

$$\text{loss} = \left(r_t + \gamma \max_a Q_{w_{\text{target}}}(x_{t+1}, a) - Q_w(x_t, a_t) \right)^2$$

Properties: DQN is off-policy, and uses 1-step bootstrapping.



DQN

Maintain an experience replay memory and sample $\{\mathbf{o}_j, \mathbf{a}_j, \mathbf{r}_j, \mathbf{o}'_j\}_{j=1}^N$

$$L(\boldsymbol{\theta}_i) = \mathbb{E}_{\mathbf{o}, \mathbf{a}, \mathbf{o}'} \left[\left(\mathbf{r}(\mathbf{o}, \mathbf{a}, \mathbf{o}') + \gamma \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{o}', \mathbf{a}', \boldsymbol{\theta}^*) - \mathbf{Q}(\mathbf{o}, \mathbf{a}, \boldsymbol{\theta}_i) \right)^2 \right]$$

Keep the target fixed for stability

$$\mathbf{y}_i = \mathbf{r}(\mathbf{o}, \mathbf{a}, \mathbf{o}') + \gamma \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{o}', \mathbf{a}', \boldsymbol{\theta}^*)$$

Follow the gradient to find theta

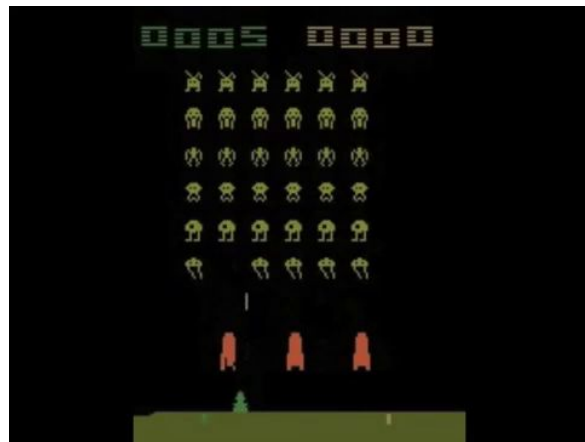
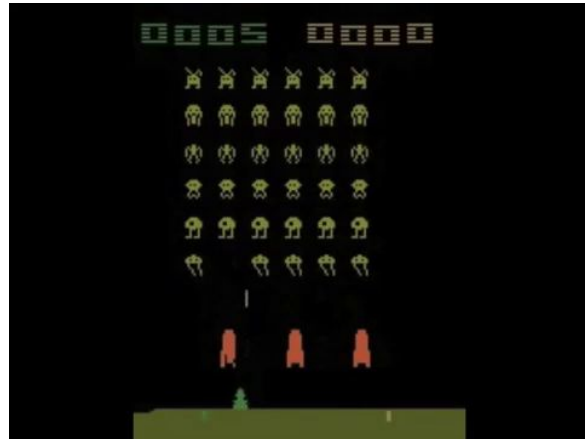
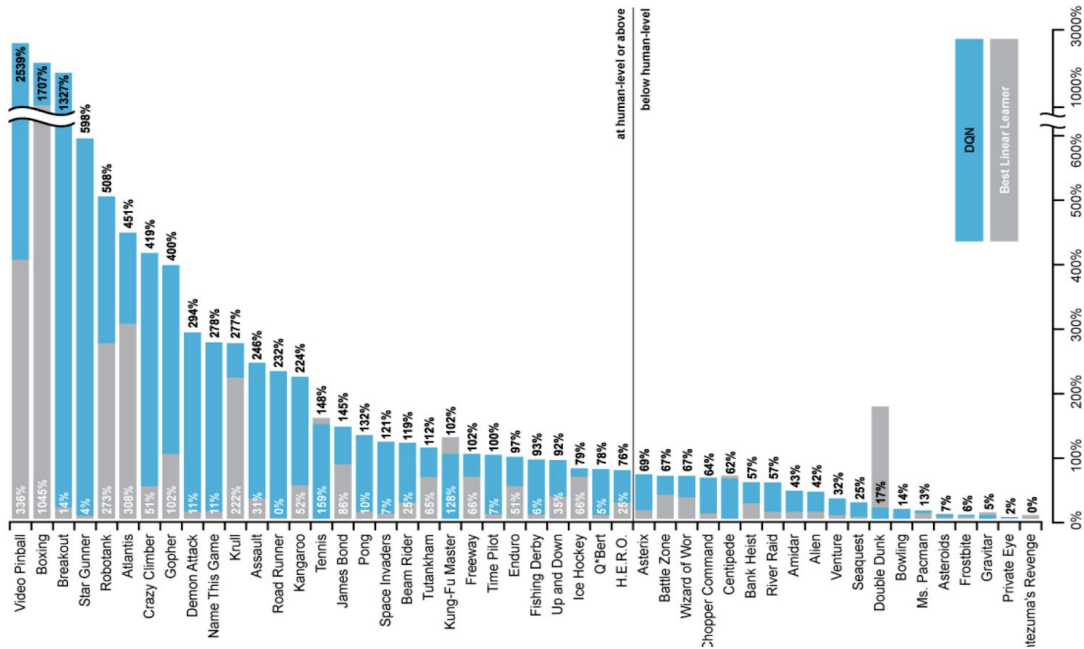
$$\nabla_{\boldsymbol{\theta}_i} L(\boldsymbol{\theta}_i) = \mathbb{E}_{\mathbf{o}, \mathbf{a}, \mathbf{o}'} \left[\left(\mathbf{r}(\mathbf{o}, \mathbf{a}, \mathbf{o}') + \gamma \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{o}', \mathbf{a}', \boldsymbol{\theta}^*) - \mathbf{Q}(\mathbf{o}, \mathbf{a}, \boldsymbol{\theta}_i) \right) \nabla_{\boldsymbol{\theta}_i} \mathbf{Q}(\mathbf{o}, \mathbf{a}, \boldsymbol{\theta}_i) \right]$$

$$\hat{\mathbf{a}} = \arg \max_{\mathbf{a}} \mathbf{Q}(\mathbf{o}, \mathbf{a}; \hat{\boldsymbol{\theta}})$$





The same algorithm learns to play 57 games



Improvements since Nature's DQN

- ▶ **Double DQN:** Remove upward bias caused by $\max_a Q(s, a, \mathbf{w})$
 - ▶ Current Q-network $\mathbf{w}$ is used to **select** actions
 - ▶ Older Q-network $\mathbf{w}_{\text{target}}$ is used to **evaluate** actions

[van Hasselt et al., 2015]

$$Q(s, a) \leftarrow r(s, a) + \gamma Q(s', \underset{a'}{\operatorname{argmax}} Q(s', a', \mathbf{w}), \mathbf{w}^-)$$

- ▶ **Prioritised replay:** Weight experience according to surprise
 - ▶ Store experience in priority queue according to DQN error

[Schaul et al., 2015]

$$|r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w})|$$

- ▶ **Duelling network:** Split Q-network into two channels
 - ▶ Action-independent **value function** $V(s, v)$
 - ▶ Action-dependent **advantage function** $A(s, a, \mathbf{w})$

[Wang et al., 2015]

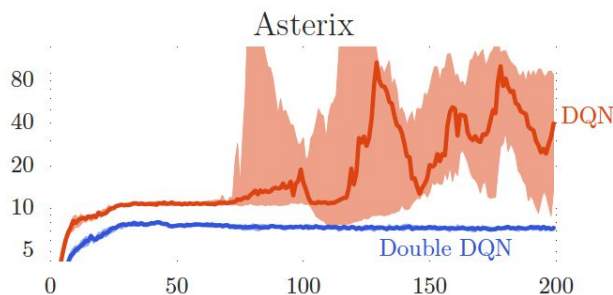
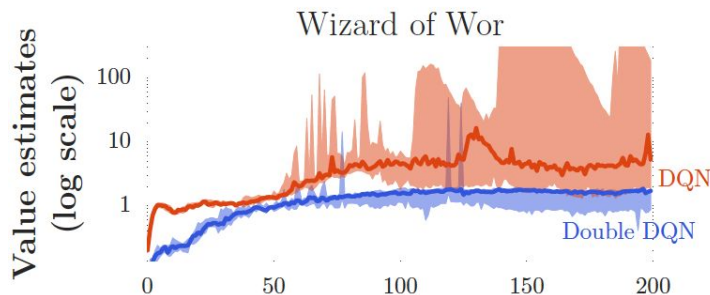
$$Q(s, a) = V(s, v) + A(s, a, \mathbf{w})$$



Double DQN

DDQN aims to reduce over-confidence in Q estimates, which results from choosing the max of Q

$$y_i = r(o, a, o') + \gamma \max_{a'} Q(o', a', \theta^*)$$



$$y_i = r(o, a, o') + \gamma Q(o', \arg \max_{a'} Q(o, a', \theta_i), \theta^*)$$




```

class DQNActor(core.Actor):
    def update(self):
        # Uses sample from replay: (o_tm1, a_tm1, r_t, d_t, o_t). Note: tm1 = t-1.

        with tf.GradientTape() as tape:
            # Get online and target Q-values.
            q_tm1 = self._q_network(o_tm1)
            q_t_value = self._target_q_network(o_t)
            q_t_selector = self._q_network(o_t)

            # Double DQN target
            best_action = tf.argmax(q_t_selector, 1, output_type=tf.int32)
            double_q_bootstrapped = indexing_ops.batched_index(q_t_value, best_action)
            target = tf.stop_gradient(r_t + d_t * double_q_bootstrapped)
            qa_tm1 = indexing_ops.batched_index(q_tm1, a_tm1)

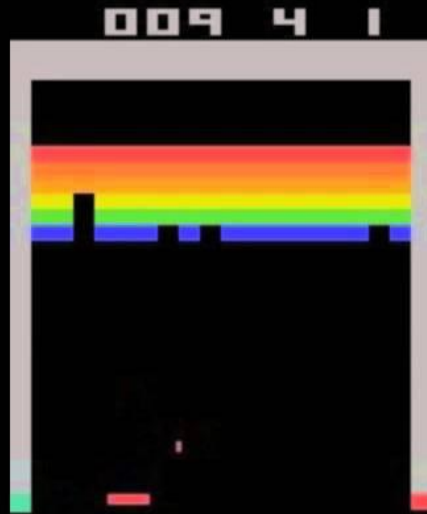
            # Temporal difference error and loss.
            td_error = target - qa_tm1
            loss = tf.reduce_mean(0.5 * tf.square(td_error), axis=0)

            # Compute gradients and update Q-network variables.
            gradients = tape.gradient(loss, self._q_network.trainable_variables)
            self.optimizer.apply(gradients, self._q_network.trainable_variables)

```



Atari with deep RL



Value functions

The state-action value function is the expected reward with the first action and observation fixed.

$$Q^{\pi}(\mathbf{o}_0, \mathbf{a}_0) = \mathbb{E}_{\tau} \left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1}) \middle| \mathbf{o}_0, \mathbf{a}_0 \right]$$

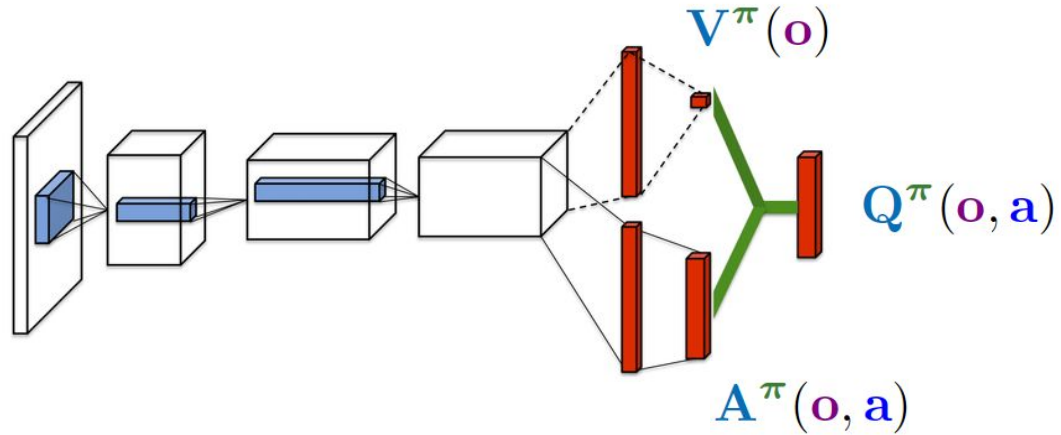
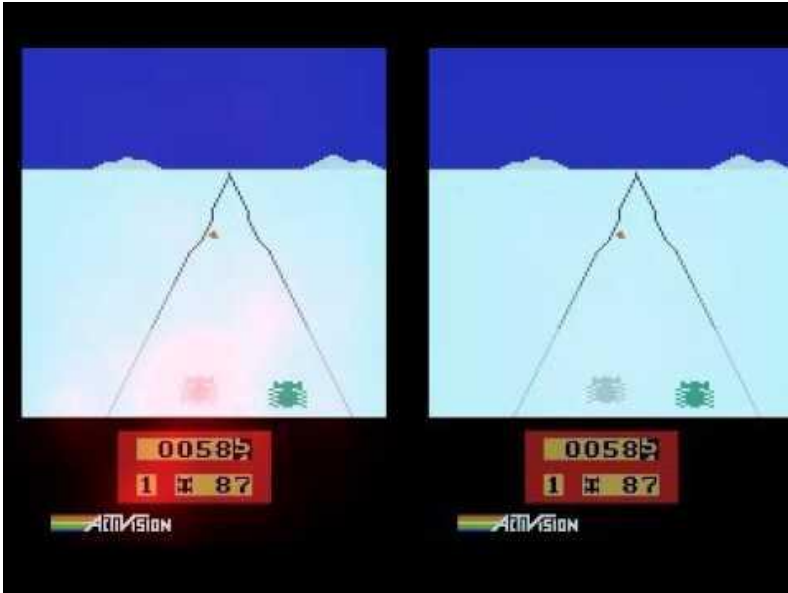
The value and advantage functions can be defined accordingly

$$V^{\pi}(\mathbf{o}) = \mathbb{E}_{\pi(\mathbf{a}|\mathbf{o})} [Q^{\pi}(\mathbf{o}, \mathbf{a})]$$

$$A^{\pi}(\mathbf{o}, \mathbf{a}) = Q^{\pi}(\mathbf{o}, \mathbf{a}) - V^{\pi}(\mathbf{o})$$



Value and advantage: Dueling networks



(Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, Nando Freitas, 2016)



Improvements since Nature's DQN

- **Persistent DQN**: Repeat same action at next state if next state is very similar to previous state. Update $Q(s, a)$

[Bellemare et al., 2015]

$$Q(s, a) \leftarrow r(s, a) + \gamma \left[\beta \max_{a'} Q(s', a') + (1 - \beta) Q(s', a) \right].$$

- **Multi-steps updates**: Propagate information over several steps:

[Hessel et al., 2017]

$$Q(s, a) \leftarrow \sum_{t=0}^{n-1} \gamma^t r_t + \gamma^n \max_{a'} Q(s_n, a').$$

Faster propagation of information but this is an **on-policy** algorithm (i.e. actions are greedy w.r.t. current Q).

- **Retrace & vtrace algorithms**: multi-steps **off-policy** learning:

[Munos et al., 2016]

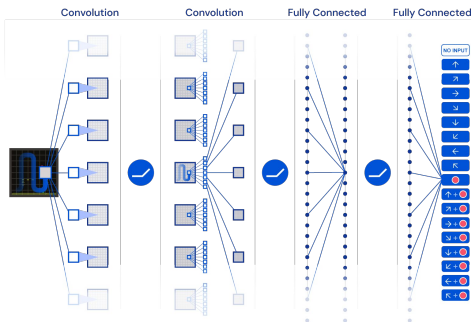
$$Q(s, a) \leftarrow \sum_{t \geq 0} \gamma^t (c_1 \dots c_t) \left(r_t + \gamma \max_{a'} Q(s_t, a') - Q(s_t, a_t) \right),$$

where $c_t = \min \left(1, \frac{\pi(a_t | x_t)}{\mu(a_t | s_t)} \right)$.

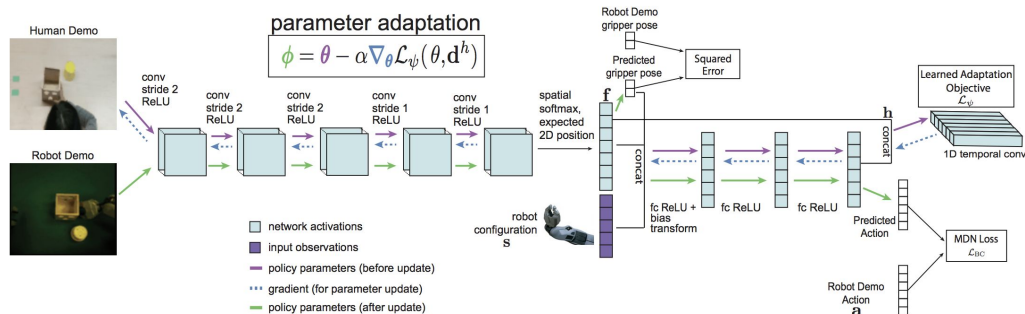


Nets got bigger in RL

DQN



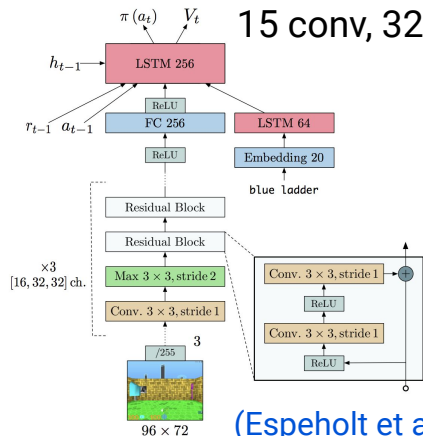
MAML third-person imitation 5 conv, 64 channels



(Yu & Finn et al 2018)

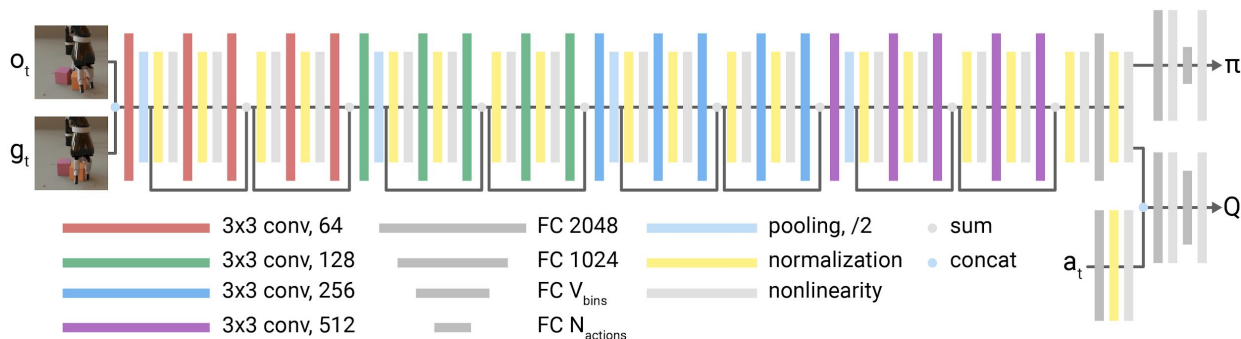
IMPALA

15 conv, 32 channels



(Espeholt et al 2018)

MetaMimic 20 conv, 512 channels



(Paine et al 2018)

DeepMind

4

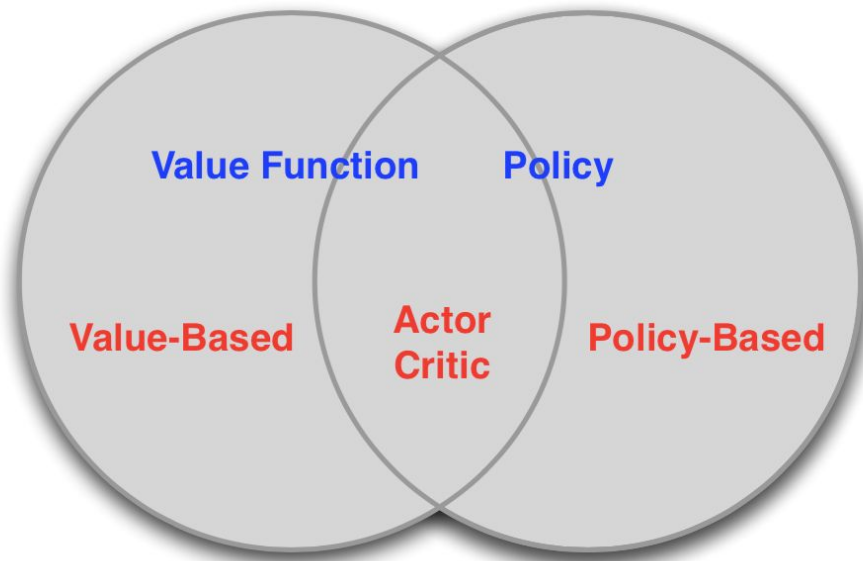
Policy gradients

*Policy-based methods following
Pontryagin's max principle*



Value-based and policy-based RL

- ▶ **Value Based**
 - ▶ Learnt values
 - ▶ Implicit policy (e.g. ϵ -greedy)
- ▶ **Policy Based**
 - ▶ No values
 - ▶ Learnt policy
- ▶ **Actor-Critic**
 - ▶ Learnt values
 - ▶ Learnt policy



Advantages and disadvantages of policy-based RL

Advantages:

- ▶ True objective
- ▶ Easy to extend to high-dimensional or continuous action spaces
- ▶ Can learn **stochastic** policies
- ▶ Sometimes policies are **simple** while values and models are **complex**
 - ▶ E.g., rich domain, but optimal policy is always “move forward”

Disadvantages:

- ▶ Susceptible to local optima (especially with non-linear FA)
- ▶ Obtained knowledge can be **specific**, does not always generalise well
- ▶ Does not extract all useful information from the data
(when used in isolation)



Policy Objective Functions

- ▶ Goal: given **policy** $\pi_\theta(s, a)$, find best **parameters** θ
- ▶ How do we measure the quality of a policy π_θ ?
- ▶ In episodic environments we can use the **start value**

$$J_1(\theta) = v_{\pi_\theta}(s_1)$$

- ▶ In continuing environments we can use the **average reward per step**

$$J_{avR}(\theta) = \mathbb{E}_{\pi_\theta} [R] = \sum_s \mu_{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \sum_r p(r | s, a) r$$

where $\mu_\pi(s) = p(S_t = s | \pi)$ is the probability of being in state s in the long run
Think of it as the ratio of time spent in s under policy π



Policy Optimization

- ▶ Policy based reinforcement learning is an **optimization** problem
- ▶ Find θ that maximises $J(\theta)$
- ▶ We will focus on **stochastic gradient ascent**, which is often quite efficient (and easy to use with deep nets)
- ▶ Some approaches do not use gradient
 - ▶ Hill climbing / simulated annealing
 - ▶ Genetic algorithms / evolutionary strategies

Policy Gradient

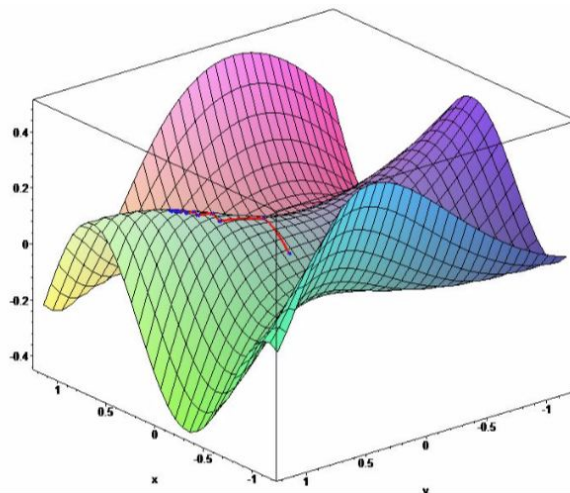
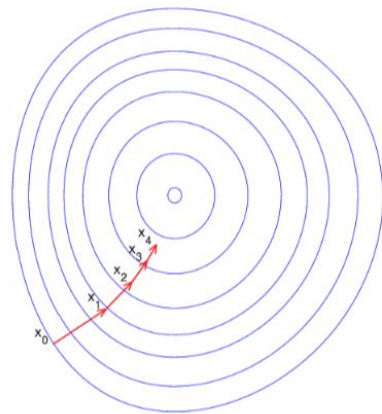
- Idea: ascent the gradient of the objective $J(\theta)$

$$\Delta\theta = \alpha \nabla_{\theta} J(\theta)$$

- Where $\nabla_{\theta} J(\theta)$ is the **policy gradient**

$$\nabla_{\theta} J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- and α is a step-size parameter
- Stochastic policies help ensure $J(\theta)$ is smooth (typically/mostly)



The score function trick

Let $r_{sa} = \mathbb{E}[R(S, A) \mid S = s, A = s]$

$$\begin{aligned}\nabla_{\theta} \mathbb{E}[R(S, A)] &= \nabla_{\theta} \sum_s \mu(s) \sum_a \pi_{\theta}(a|s) r_{sa} \\&= \sum_s \mu(s) \sum_a r_{sa} \nabla_{\theta} \pi_{\theta}(a|s) \\&= \sum_s \mu(s) \sum_a r_{sa} \pi_{\theta}(a|s) \frac{\nabla_{\theta} \pi_{\theta}(a|s)}{\pi_{\theta}(a|s)} \\&= \sum_s \mu(s) \sum_a \pi_{\theta}(a|s) r_{sa} \nabla_{\theta} \log \pi_{\theta}(a|s) \\&= \mathbb{E}[R(S, A) \nabla_{\theta} \log \pi_{\theta}(A|S)]\end{aligned}$$



Policy gradients on trajectories: derivation

- Consider trajectory $\zeta = S_0, A_0, R_1, S_1, A_1, R_1, S_2, \dots$ with return $G(\zeta)$

$$\nabla_{\theta} J_{\theta}(\pi) = \nabla_{\theta} \mathbb{E}[G(\zeta)] = \mathbb{E}[G(\zeta) \nabla_{\theta} \log p(\zeta)] \quad (\text{score function trick})$$

$$\nabla_{\theta} \log p(\zeta)$$

$$= \nabla_{\theta} \log \left[p(S_0) \pi(A_0|S_0) p(S_1|S_0, A_0) \pi(A_1|S_1) \cdots \right]$$

$$= \nabla_{\theta} \left[\log p(S_0) + \log \pi(A_0|S_0) + \log p(S_1|S_0, A_0) + \log \pi(A_1|S_1) + \cdots \right]$$

$$= \nabla_{\theta} \left[\log \pi(A_0|S_0) + \log \pi(A_1|S_1) + \cdots \right]$$

So:

$$\nabla_{\theta} J_{\theta}(\pi) = \mathbb{E} \left[G(\zeta) \nabla_{\theta} \sum_{t=0} \log \pi(A_t|S_t) \right] = \mathbb{E} \left[\left(\sum_{t=0} R_{t+1} \right) \left(\nabla_{\theta} \sum_{t=0} \log \pi(A_t|S_t) \right) \right]$$



Policy gradients on trajectories: reduce variance

- Note that, in general

$$\begin{aligned}\mathbb{E}[b\nabla_{\theta} \log \pi(A_t|S_t)] &= \mathbb{E}\left[\sum_a \pi(a|S_t) b\nabla_{\theta} \log \pi(a|S_t)\right] \\ &= \mathbb{E}\left[b\nabla_{\theta} \sum_a \pi(a|S_t)\right] \\ &= \mathbb{E}[b\nabla_{\theta} 1] \\ &= 0\end{aligned}$$

- This holds only if b does not depend on the action
(it can depend on the state)
- Implies we can subtract a **baseline** to reduce variance



Policy gradients on trajectories: reduce variance

- Consider trajectory $\zeta = S_0, A_0, R_1, S_1, A_1, R_1, S_2, \dots$ with return $G(\zeta)$

$$\nabla_{\theta} J_{\theta}(\pi) = \mathbb{E} \left[\left(\sum_{t=0} R_{t+1} \right) \left(\nabla_{\theta} \sum_{t=0} \log \pi(A_t | S_t) \right) \right]$$

but $\sum_{t=0}^k R_{t+1}$ does not depend on actions $A_{k+1}, A_{k+2}, \dots$, so

$$\begin{aligned} &= \mathbb{E} \left[\sum_{t=0} \nabla_{\theta} \log \pi(A_t | S_t) \sum_{i=0} R_{i+1} \right] && \text{(rewrite of above)} \\ &= \mathbb{E} \left[\sum_{t=0} \nabla_{\theta} \log \pi(A_t | S_t) \sum_{i=t} R_{i+1} \right] \\ &= \mathbb{E} \left[\sum_{t=0} \nabla_{\theta} \log \pi(A_t | S_t) q_{\pi}(S_t, A_t) \right] \end{aligned}$$



Policy gradients on trajectories: reduce variance

- ▶ A good baseline is $v_\pi(S_t)$

$$\nabla_\theta J_\theta(\pi) = \mathbb{E} \left[\sum_{t=0} \nabla_\theta \log \pi(A_t|S_t) (q_\pi(S_t, A_t) - v_\pi(S_t)) \right]$$

- ▶ Typically, we estimate $v_{\mathbf{w}}(s)$ explicitly, and sample

$$q_\pi(S_t, A_t) \approx G_t$$

- ▶ For instance, $G_t = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1})$



Actor-Critic

Critic Update parameters $\mathbf{w}$ of $v_{\mathbf{w}}$ by TD (e.g., one-step) or MC

Actor Update θ by policy gradient

function ADVANTAGE ACTOR CRITIC

 Initialise s, θ

for $t = 0, 1, 2, \dots$ **do**

 Sample $A_t \sim \pi_{\theta}(S_t)$

 Sample R_{t+1} and S_{t+1}

$\delta_t = R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)$

$\mathbf{w} \leftarrow \mathbf{w} + \beta \delta_t \nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t)$

$\theta \leftarrow \theta + \alpha \delta_t \nabla_{\theta} \log \pi_{\theta}(A_t | S_t)$

end for

end function

[one-step TD-error, or **advantage**]

[TD(0)]

[Policy gradient update]



Full actor-critic agent

- ▶ Actor critic includes:
 - ▶ A **representation** (e.g., LSTM): $(S_{t-1}, O_t) \mapsto S_t$
 - ▶ A **network** $v_{\mathbf{w}}: S \mapsto v$
 - ▶ A **network** $\pi_{\theta}: S \mapsto \pi$
 - ▶ A (n -step) TD **loss** on $v_{\mathbf{w}}$

$$l(\mathbf{w}) = \frac{1}{2} \left(G_t^{(n)} - v_{\mathbf{w}}(S_t) \right)^2$$

where $G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} v_{\mathbf{w}}(S_{t+n})$

- ▶ A n -step REINFORCE '**loss**' on π_{θ}

$$l(\theta) = \left[G_t^{(n)} - v_{\mathbf{w}}(S_t) \right] \log \pi_{\theta}(A_t | S_t)$$

- ▶ **Optimizers** to minimize the losses
 - ▶ Note: value and policy may share parameters (e.g., in the LSTM)
- ▶ Also known as A2C (Advantage Actor Critic)



Increasing robustness with trust regions

- ▶ One way to increase stability is to **regularise**
- ▶ A popular method is to **limit the difference between subsequent policies**
- ▶ For instance, use the Kullbeck-Leibler divergence:

$$\text{KL}(\pi_{\text{old}} \parallel \pi_{\theta}) = \mathbb{E} \left[\int \pi_{\text{old}}(a \mid S) \log \frac{\pi_{\theta}(a \mid S)}{\pi_{\text{old}}(a \mid S)} da \right] .$$

(Expectation is over states)

- ▶ A divergence is like a distance between distributions
- ▶ Then maximise $J(\theta) - \eta \text{KL}(\pi_{\text{old}} \parallel \pi_{\theta})$, for some hyperparameter η
- ▶ It can also help to use large batches

c.f. **TRPO** (Schulman et al. 2015) and **PPO** (Abbeel & Schulman 2016)



Exploration

- ▶ The entropy of a policy is

$$-\sum_s \mu(s) \sum_a \pi(a | s) \log \pi(a | s) = -\mathbb{E} [\log \pi(A_t | S_t)]$$

- ▶ Idea: add a regularisation term that pushes up entropy slightly on each step
- ▶ Encourages exploration, but does not pick fully randomly
- ▶ E.g., may increase variance in Gaussian policies
- ▶ E.g., makes softmax slightly more uniform
- ▶ Similar to KL regularisation, but now towards uniformly random policies
- ▶ Not a full solution to exploration, but works well in practice



Exploration

- ▶ The policy-gradient objective only considers improvement under current data
- ▶ Easy to get stuck in local optima — we need to **explore**
- ▶ We could use ϵ -greedy (assuming bounded action space), but that is not ideal
 - ▶ Wildly different actions may cause breakage
 - ▶ Exploration is mostly uninformed about current best guess
- ▶ Popular alternative: make sure entropy of the policy is not too low



Summary of Policy Gradient Algorithms

- ▶ The **policy gradient** has many forms

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi} [\nabla_{\theta} \log \pi_{\theta}(A|S) \textcolor{red}{G}_t] \quad \text{REINFORCE}$$

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi} [\nabla_{\theta} \log \pi_{\theta}(A|S) \textcolor{red}{(G}_t - b(S_t)\textcolor{red}{)}] \quad \text{REINFORCE}$$

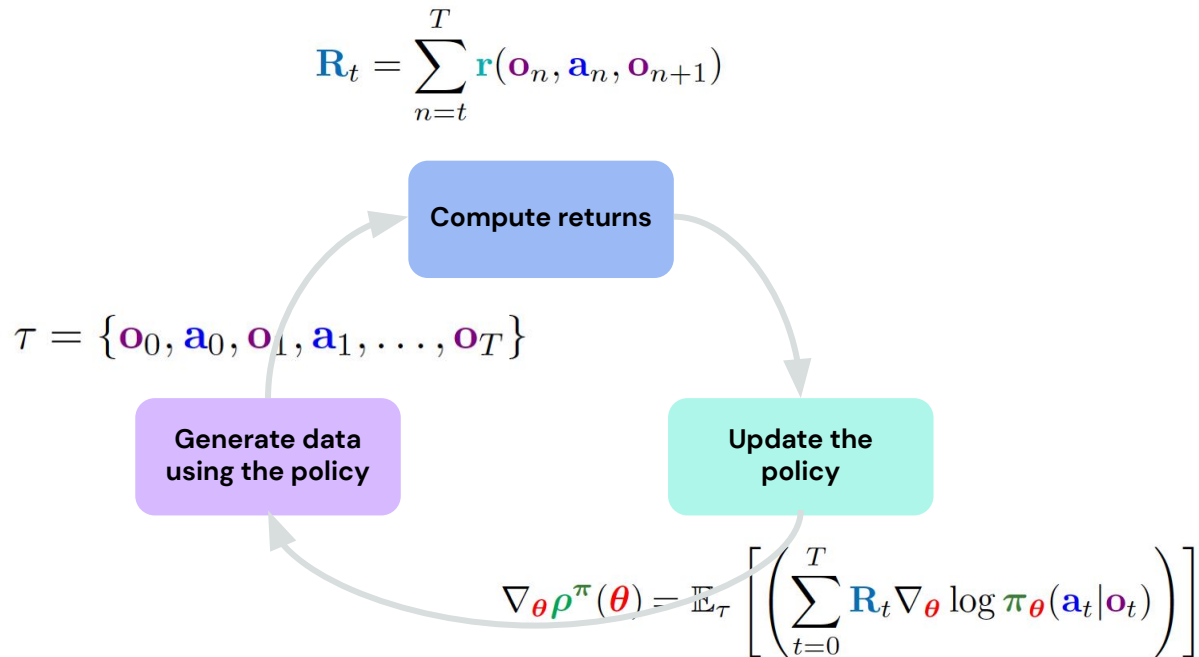
$$\nabla_{\theta} J(\theta) \approx \mathbb{E}_{\pi} \left[\nabla_{\theta} \log \pi_{\theta}(A|S) \textcolor{red}{\delta}_t^{(n)} \right] \quad \text{Advantage Actor-Critic}$$

$$\nabla_{\theta} J(\theta) \approx \nabla_{\theta} q_t(S, \pi_{\theta}(S)) \quad \text{DPG}$$

- ▶ Each leads a stochastic gradient ascent algorithm
- ▶ Critic uses **policy evaluation** (e.g. MC or TD) to estimate q_{π} or $v_{\pi}(s)$



The policy gradient



DeepMind

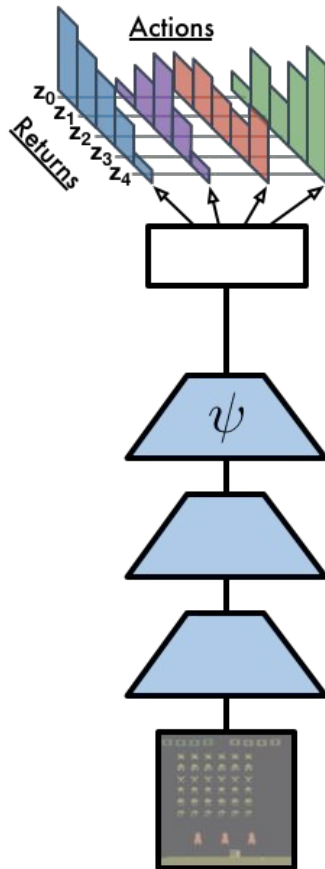
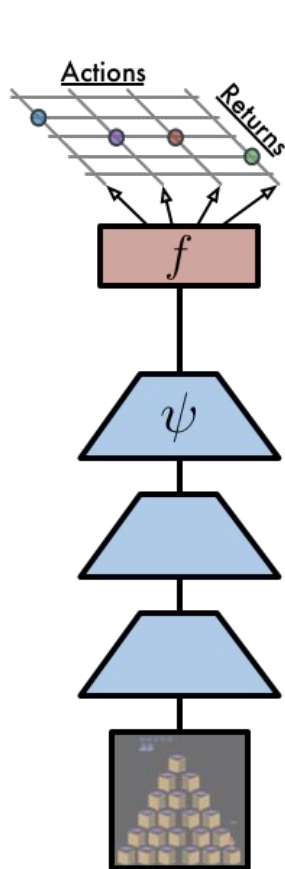
5

Distributional RL



$$Q(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta})$$

$$f_i(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta})$$



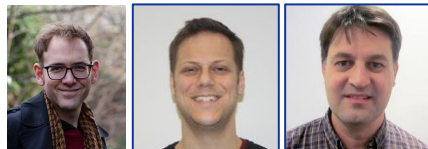
$$\mathbf{Z}(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta}) = z_i \quad w.p. \quad p_i(\mathbf{o}, \mathbf{a}) = \frac{e^{f_i(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta})}}{\sum_j e^{f_j(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta})}}$$

$$\hat{\mathbf{a}} = \arg \max_{\mathbf{a}} Q(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta})$$

$$= \arg \max_{\mathbf{a}} \mathbb{E} [\mathbf{Z}(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta})]$$

$$\approx \arg \max_{\mathbf{a}} \sum_{i=0}^{N-1} z_i p_i(\mathbf{o}, \mathbf{a})$$

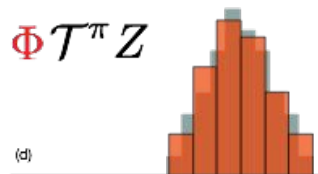
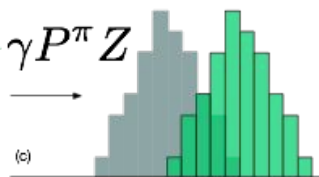
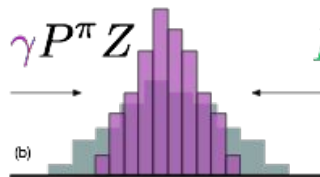
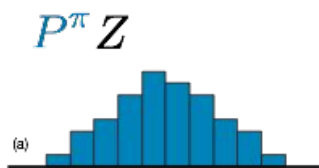
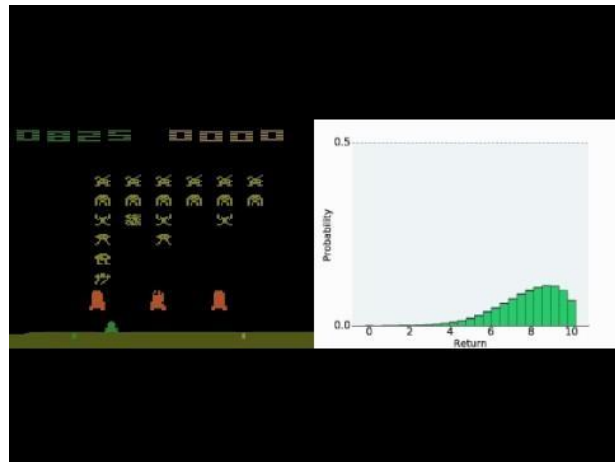




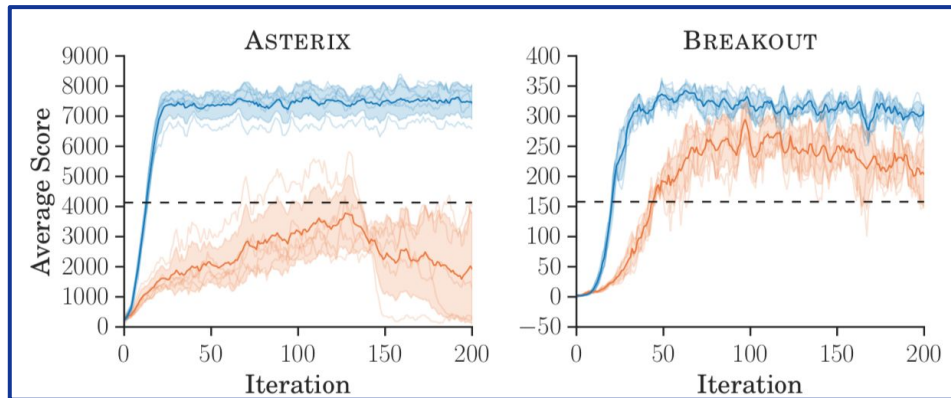
Minimize the TD distance in distribution using discrete distributions

$$\left(r(o, a, o') + \gamma \max_{a'} Q(o', a', \theta^*) - Q(o, a, \theta_i) \right)^2$$

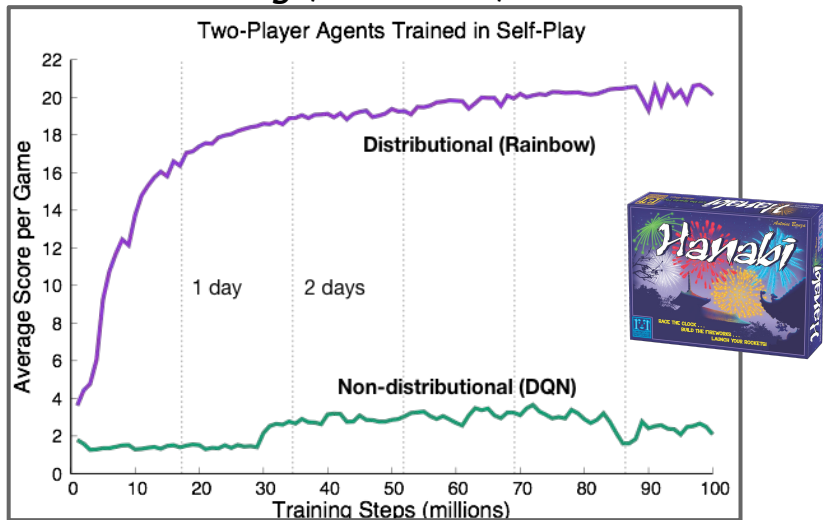
$$Dist \left(r(o, a, o') + \gamma \max_{a'} Z(o', a', \theta^*); Z(o, a, \theta_i) \right)$$



Batch deep RL, Agarwal, Schuurmans, Norouzi, 2019



Hanabi Challenge, Bard et al., 2019

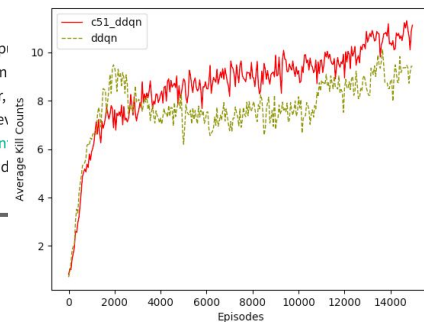


VizDoom, Felix Yu, [blog post](#)

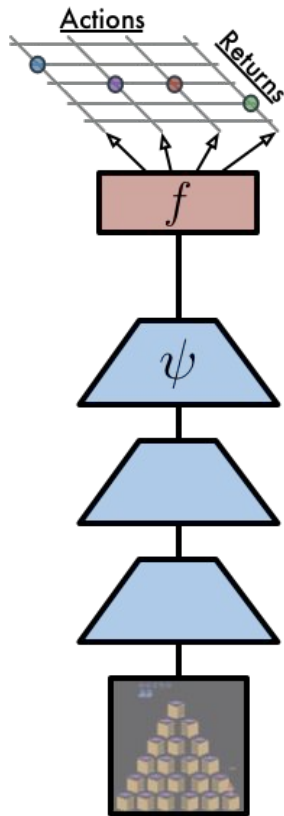
Distributional Bellman and the C51 Algorithm

OCTOBER 24, 2017

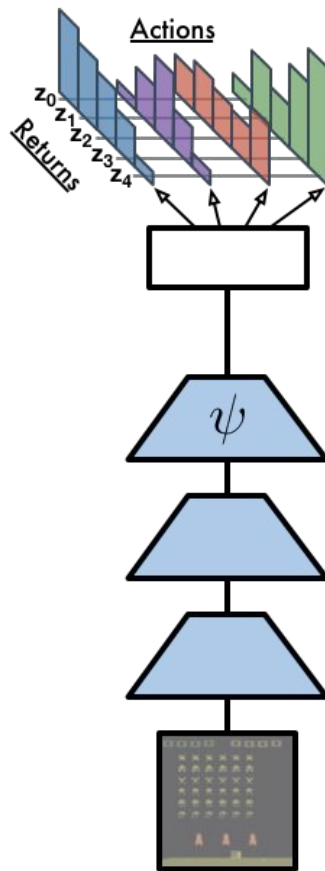
I got the chance to read this [paper](#) on Distributional Bellman p over it the first time, my impression was that it would be an im sound and the experimental results were promising. However, reinforcement learning community as I would have hoped. Ne Distributional Bellman was pretty neat, I decided to [implemen](#) hope this article can help interested readers better understand Bellman.



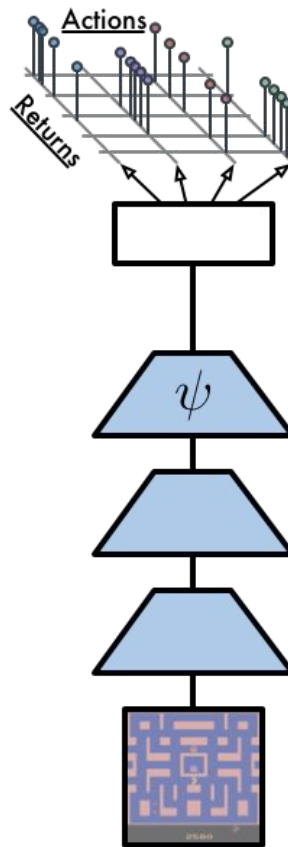
DQN



C51



QR-DQN



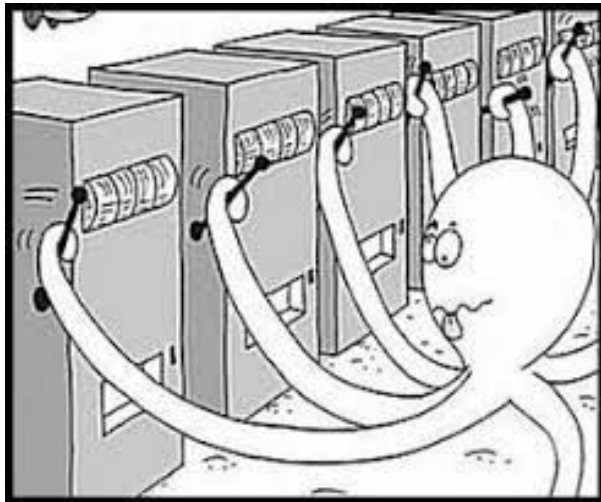
DeepMind

6

Exploration vs. Exploitation



Bandits



The Multi-Armed Bandit Setup

- ▶ rewards maximization in a stochastic bandit model
= the simplest RL problem (one state)
- ▶ bandits showcase the important exploration/exploitation dilemma
- ▶ bandit tools are useful for RL
(UCRL, PSRL, bandit-based MCTS for planning in games...)
- ▶ a rich literature to tackle many specific applications
- ▶ bandits have application beyond RL (i.e. without “reward”)



Bandits

K arms $\leftrightarrow K$ rewards streams $(X_{a,t})_{t \in \mathbb{N}}$



At round t , an agent :

- ▶ chooses an arm A_t
- ▶ receives a reward $R_t = X_{A_t,t}$

Sequential sampling strategy (**bandit algorithm**) :

$$A_{t+1} = F_t(A_1, R_1, \dots, A_t, R_t).$$

Goal : Maximize $\sum_{t=1}^T R_t$.



The Stochastic Multi-Armed Bandit Setup

K arms $\leftrightarrow K$ probability distributions : ν_a has mean μ_a



ν_1



ν_2



ν_3



ν_4



ν_5

At round t , an agent :

- ▶ chooses an arm A_t
- ▶ receives a reward $R_t = X_{A_t,t} \sim \nu_{A_t}$

Sequential sampling strategy (**bandit algorithm**) :

$$A_{t+1} = F_t(A_1, R_1, \dots, A_t, R_t).$$

Goal : Maximize $\mathbb{E} \left[\sum_{t=1}^T R_t \right]$.



Clinical trials

Historical motivation [Thompson, 1933]



$\mathcal{B}(\mu_1)$



$\mathcal{B}(\mu_2)$



$\mathcal{B}(\mu_3)$



$\mathcal{B}(\mu_4)$



$\mathcal{B}(\mu_5)$

For the t -th patient in a clinical study,

- ▶ chooses a treatment A_t
- ▶ observes a response $R_t \in \{0, 1\} : \mathbb{P}(R_t = 1 | A_t = a) = \mu_a$

Goal : maximize the expected number of patients healed



Online content optimization

Modern motivation (\$\$) [Li et al., 2010]
(recommender systems, online advertisement)



ν_1



ν_2



ν_3



ν_4



ν_5

For the t -th visitor of a website,

- ▶ recommend a movie A_t
- ▶ observe a rating $R_t \sim \nu_{A_t}$ (e.g. $R_t \in \{1, \dots, 5\}$)

Goal : maximize the sum of ratings



Regret of a bandit algorithm

Bandit instance : $\nu = (\nu_1, \nu_2, \dots, \nu_K)$, mean of arm a : $\mu_a = \mathbb{E}_{X \sim \nu_a}[X]$.

$$\mu_\star = \max_{a \in \{1, \dots, K\}} \mu_a \quad a_\star = \operatorname{argmax}_{a \in \{1, \dots, K\}} \mu_a.$$

Maximizing rewards $\leftrightarrow$ selecting $a_\star$ as much as possible
 $\leftrightarrow$ minimizing the **regret** [Robbins, 1952]

$$\mathcal{R}_\nu(\mathcal{A}, T) := \underbrace{T\mu_\star}_{\text{sum of rewards of an oracle strategy always selecting } a_\star} - \underbrace{\mathbb{E} \left[\sum_{t=1}^T R_t \right]}_{\text{sum of rewards of the strategy } \mathcal{A}}$$

What regret rate can we achieve ?

- consistency : $\frac{\mathcal{R}_\nu(\mathcal{A}, T)}{T} \rightarrow 0$
- can we be more precise ?



Regret decomposition

$N_a(t)$: number of selections of arm a in the first t rounds

$\Delta_a := \mu_\star - \mu_a$: sub-optimality gap of arm a

Regret decomposition

$$\mathcal{R}_\nu(\mathcal{A}, T) = \sum_{a=1}^K \Delta_a \mathbb{E}[N_a(T)].$$

Proof.

$$\begin{aligned} \mathcal{R}_\nu(\mathcal{A}, T) &= \mu_\star T - \mathbb{E} \left[\sum_{t=1}^T X_{A_t, t} \right] = \mu_\star T - \mathbb{E} \left[\sum_{t=1}^T \mu_{A_t} \right] \\ &= \mathbb{E} \left[\sum_{t=1}^T (\mu_\star - \mu_{A_t}) \right] \\ &= \sum_{a=1}^K \underbrace{\mu_\star - \mu_a}_{\Delta_a} \underbrace{\mathbb{E} \left[\sum_{t=1}^T \mathbb{1}(A_t = a) \right]}_{N_a(T)}. \end{aligned}$$



Regret decomposition

$N_a(t)$: number of selections of arm a in the first t rounds

$\Delta_a := \mu_\star - \mu_a$: sub-optimality gap of arm a

Regret decomposition

$$\mathcal{R}_\nu(\mathcal{A}, T) = \sum_{a=1}^K \Delta_a \mathbb{E}[N_a(T)].$$

A strategy with small regret should :

- ▶ select not too often arms for which $\Delta_a > 0$
- ▶ ... which requires to try all arms to estimate the values of the Δ_a 's

⇒ Exploration / Exploitation trade-off



Bandits

► Idea 1 :

Draw each arm T/K times

⇒ **EXPLORATION**

$$\mathcal{R}_\nu(\mathcal{A}, T) = \left(\frac{1}{K} \sum_{a: \mu_a > \mu_*} \Delta_a \right) T$$

► Idea 2 : Always trust the empirical best arm

where

$$A_{t+1} = \operatorname{argmax}_{a \in \{1, \dots, K\}} \hat{\mu}_a(t)$$
$$\hat{\mu}_a(t) = \frac{1}{N_a(t)} \sum_{s=1}^t X_{a,s} \mathbb{1}_{(A_s=a)}$$

is an estimate of the unknown mean μ_a .

⇒ **EXPLOITATION**

$$\mathcal{R}_\nu(\mathcal{A}, T) \geq (1 - \mu_1) \times \mu_2 \times (\mu_1 - \mu_2) T$$

(Bernoulli arms)



Bandits

The ϵ -greedy rule [Sutton and Barto, 1998] is the simplest way to alternate exploration and exploitation.

ϵ -greedy strategy

At round t ,

- ▶ with probability ϵ

$$A_t \sim \mathcal{U}(\{1, \dots, K\})$$

- ▶ with probability $1 - \epsilon$

$$A_t = \operatorname{argmax}_{a=1, \dots, K} \hat{\mu}_a(t).$$

→ Linear regret : $\mathcal{R}_\nu(\epsilon\text{-greedy}, T) \geq \epsilon \frac{K-1}{K} \Delta_{\min} T.$

$$\Delta_{\min} = \min_{a: \mu_a < \mu_*} \Delta_a$$



Bandits

A simple fix :

ϵ_t -greedy strategy

At round t ,

- ▶ with probability $\epsilon_t := \min\left(1, \frac{K}{d^2 t}\right)$

$$A_t \sim \mathcal{U}(\{1, \dots, K\})$$

- ▶ with probability $1 - \epsilon_t$

$$A_t = \operatorname{argmax}_{a=1, \dots, K} \hat{\mu}_a(t-1).$$

Theorem [Auer, 2002]

If $0 < d \leq \Delta_{\min}$, $\mathcal{R}_\nu(\epsilon_t\text{-greedy}, T) = O\left(\frac{K \ln(T)}{d^2}\right)$.

→ requires the knowledge of a lower bound on $\Delta_{\min}$.



Bandits

Step 1 : construct a set of statistically plausible models

- For each arm a , build a confidence interval on the mean μ_a :

$$\mathcal{I}_a(t) = [\text{LCB}_a(t), \text{UCB}_a(t)]$$

LCB = Lower Confidence Bound

UCB = Upper Confidence Bound

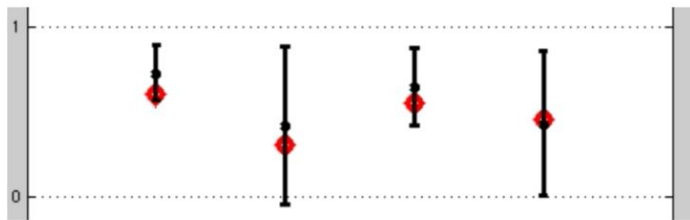


FIGURE – Confidence intervals on the means after t rounds



Bandits

Step 1 : construct a set of statistically plausible models

- For each arm a , build a confidence interval on the mean μ_a :

$$\mathcal{I}_a(t) = [\text{LCB}_a(t), \text{UCB}_a(t)]$$

LCB = Lower Confidence Bound

UCB = Upper Confidence Bound

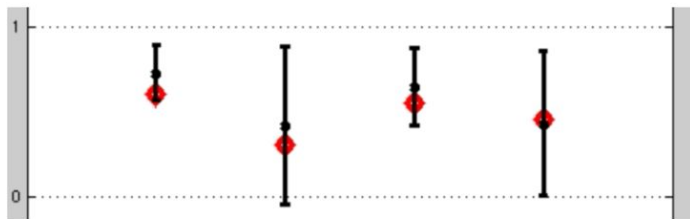


FIGURE – Confidence intervals on the means after t rounds



Bandits

Step 2 : act as if the best possible model were the true model
(*optimism in face of uncertainty*)

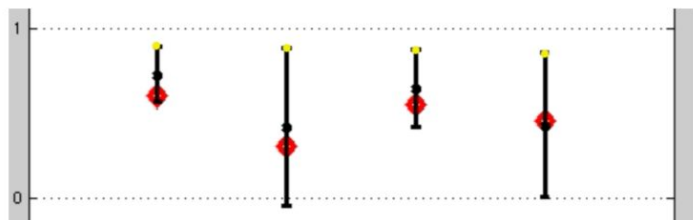


FIGURE – Confidence intervals on the means after t rounds

$$\text{Optimistic bandit model} = \arg\max_{\mu \in \mathcal{C}(t)} \max_{a=1, \dots, K} \mu_a$$

► That is, select

$$A_{t+1} = \arg\max_{a=1, \dots, K} \text{UCB}_a(t).$$



Bandits

We need $\text{UCB}_a(t)$ such that

$$\mathbb{P}(\mu_a \leq \text{UCB}_a(t)) \gtrsim 1 - t^{-1}.$$

→ tool : concentration inequalities

Example : rewards are σ^2 sub-Gaussian

Hoeffding inequality, reloaded

Z_i i.i.d. satisfying (1). For all $s \geq 1$

$$\mathbb{P}\left(\frac{Z_1 + \cdots + Z_s}{s} < \mu - x\right) \leq e^{-\frac{sx^2}{2\sigma^2}}$$



Bandits

- ▶ $N_a(t) = \sum_{s=1}^t \mathbb{1}_{(A_s=a)}$ number of selections of a after t rounds
- ▶ $\hat{\mu}_{a,s} = \frac{1}{s} \sum_{k=1}^s Y_{a,k}$ average of the first s observations from arm a
- ▶ $\hat{\mu}_a(t) = \hat{\mu}_{a,N_a(t)}$ empirical estimate of μ_a after t rounds

Hoeffding inequality + union bound

$$\mathbb{P} \left(\mu_a \leq \hat{\mu}_a(t) + \sigma \sqrt{\frac{\beta \ln(t)}{N_a(t)}} \right) \geq 1 - \frac{1}{t^{\frac{\beta}{2}-1}}$$

Proof.

$$\begin{aligned} \mathbb{P} \left(\mu_a > \hat{\mu}_a(t) + \sigma \sqrt{\frac{\beta \ln(t)}{N_a(t)}} \right) &\leq \mathbb{P} \left(\exists s \leq t : \mu_a > \hat{\mu}_{a,s} + \sigma \sqrt{\frac{\beta \ln(t)}{s}} \right) \\ &\leq \sum_{s=1}^t \mathbb{P} \left(\hat{\mu}_{a,s} < \mu_a - \sigma \sqrt{\frac{\beta \ln(t)}{s}} \right) \leq \sum_{s=1}^t \frac{1}{t^{\beta/2}} = \frac{1}{t^{\beta/2-1}}. \end{aligned}$$



Bandits

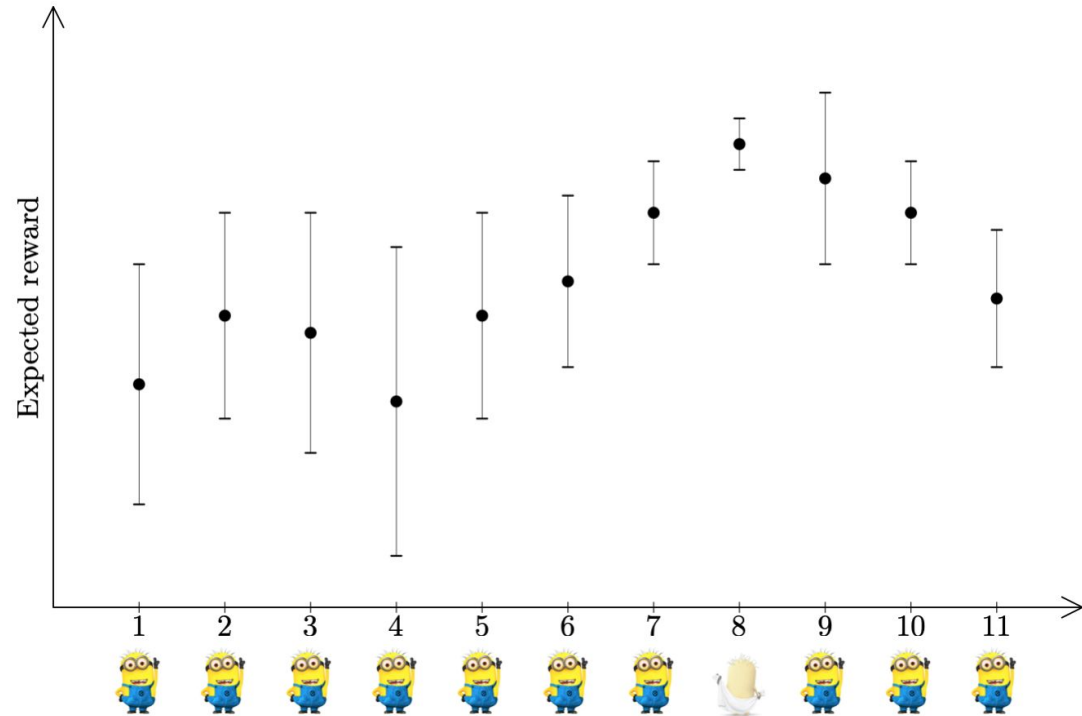
UCB(α) selects $A_{t+1} = \operatorname{argmax}_a \text{UCB}_a(t)$ where

$$\text{UCB}_a(t) = \underbrace{\hat{\mu}_a(t)}_{\text{exploitation term}} + \underbrace{\sqrt{\frac{\alpha \ln(t)}{N_a(t)}}}_{\text{exploration bonus}} .$$

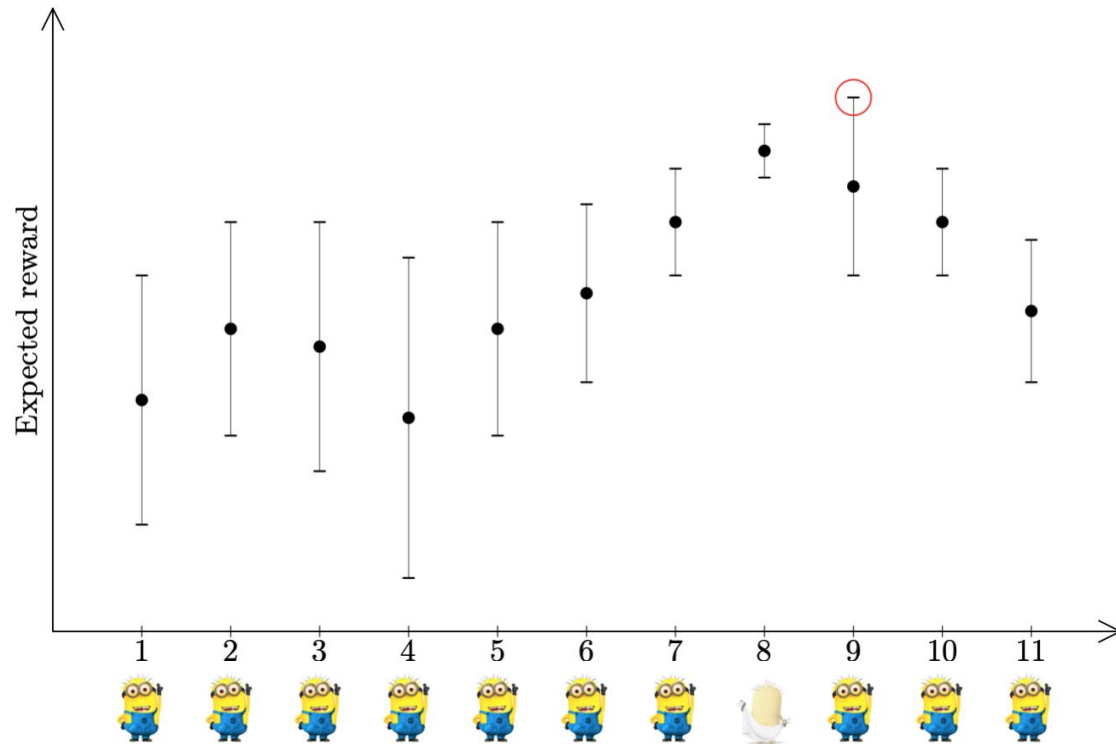
- ▶ this form of UCB was first proposed for Gaussian rewards [Katehakis and Robbins, 1995]
- ▶ popularized by [Auer, 2002] for bounded rewards : UCB1, for $\alpha = 2$
- ▶ the analysis was UCB(α) was further refined to hold for $\alpha > 1/2$ in that case [Bubeck, 2010]



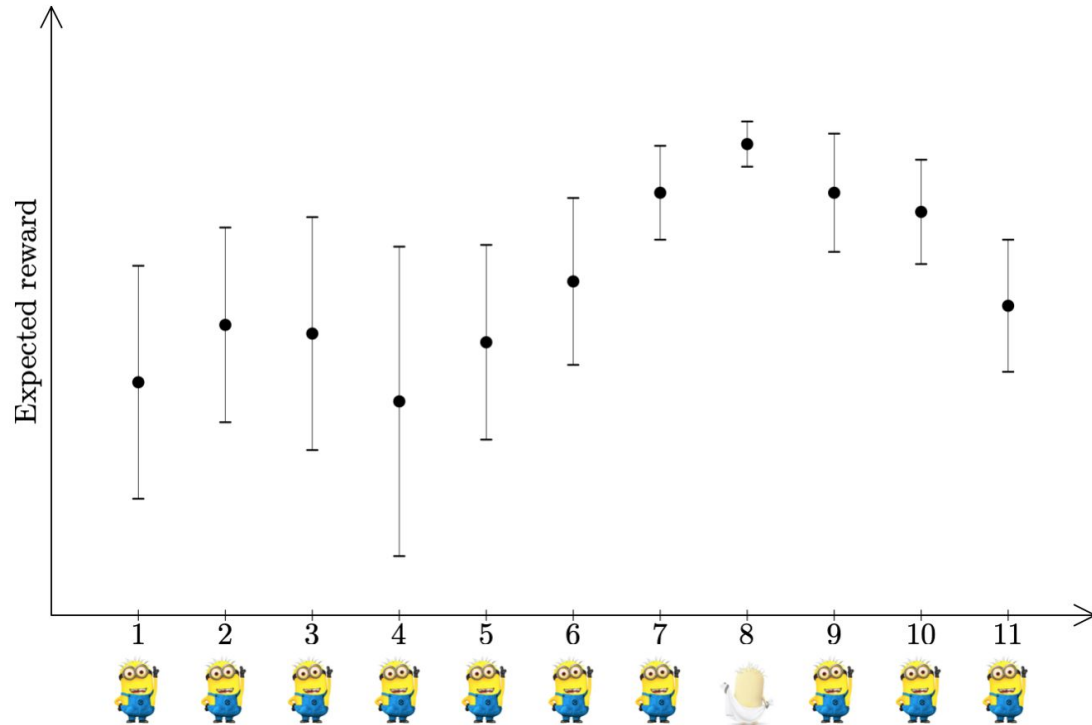
Bandits



Bandits



Bandits



Bandits

Theorem [Auer et al, 02]

UCB(α) with parameter $\alpha = 2$ satisfies

$$\mathcal{R}_\nu(\text{UCB1}, T) \leq 8 \left(\sum_{a: \mu_a < \mu_\star} \frac{1}{\Delta_a} \right) \ln(T) + \left(1 + \frac{\pi^2}{3} \right) \left(\sum_{a=1}^K \Delta_a \right).$$

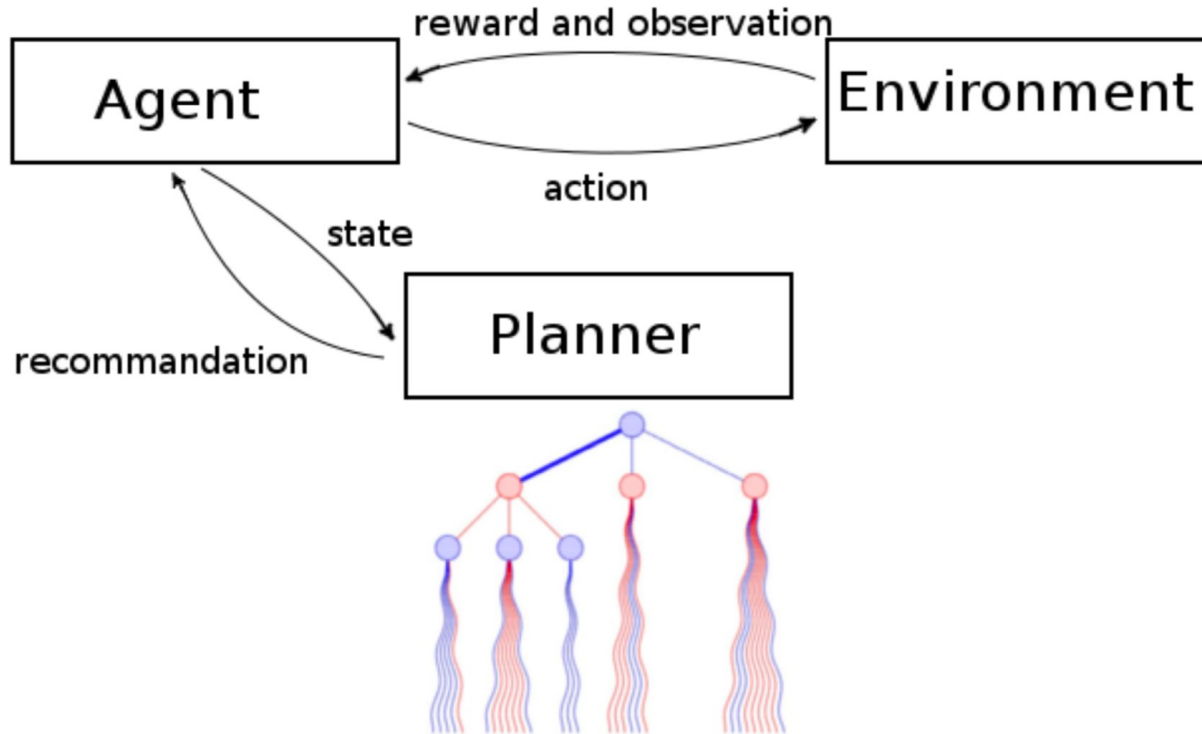


7

Monte-Carlo Tree Search For Planning in MDPs and Games

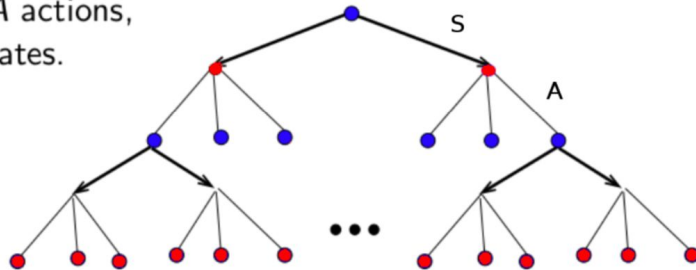


Monte-Carlo Tree Search



Monte-Carlo Tree Search

Assume A actions,
 S next-states.



A node s : Description of the environment

$\mathcal{V}[s]$: discounted sum of rewards you get if you play optimally from s .

- Maximum nodes (agent): $\mathcal{V}[s] = \max_{s' \text{ child of } s} \mathcal{V}[s']$.
- Average nodes (environment): $\mathcal{V}[s] = r(s) + \gamma \sum_{s' \text{ child of } s} p(s'|s) \mathcal{V}[s']$.

Goal: Compute the value of the root $\mathcal{V}[s_0]$.



Monte-Carlo Tree Search

MCTS is *not* an algorithm, it is a **family of methods** that use possibly random exploration to explore the tree of possible next states.

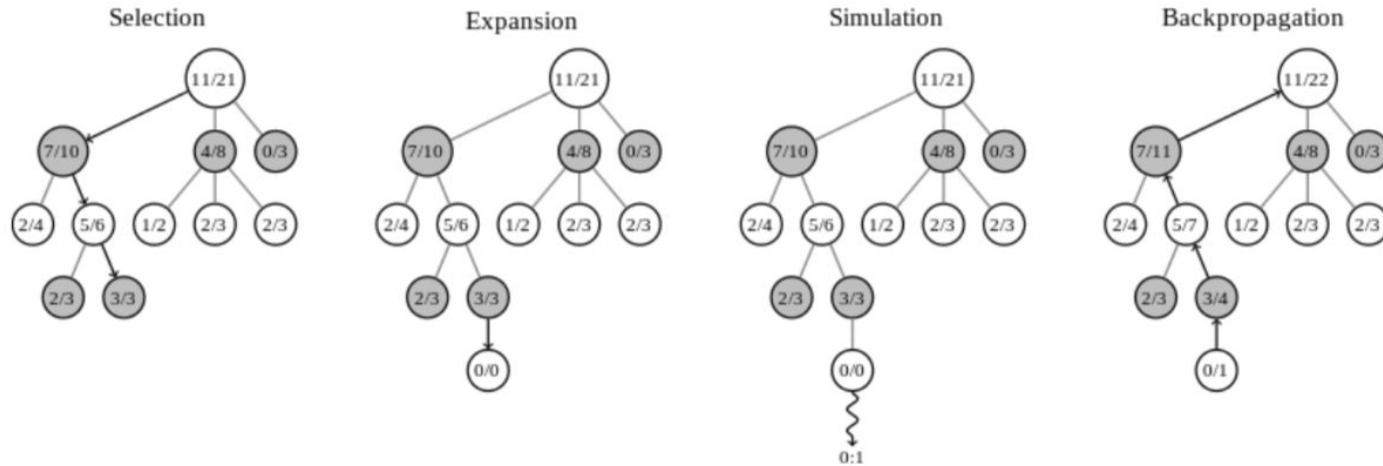


FIGURE – An generic MCTS algorithm illustrated for a game



Monte-Carlo Tree Search

Bandit-Based Monte-Carlo planning : to select a path in the tree, run a bandit algorithm each time a children (next action) needs to be selected

UCT = UCB for Trees [Kocsis and Szepesvári, 2006]

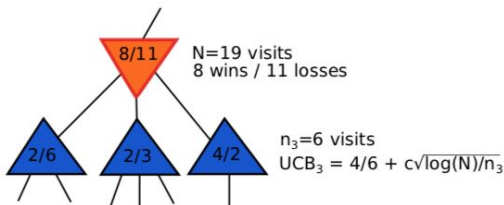
UCT in a Game Tree

In a **MAX node** s (= root player move), select an action

$$\operatorname{argmax}_{a \in \mathcal{C}(s)} \frac{S(s, a)}{N(s, a)} + c \sqrt{\frac{\ln(\sum_b N(s, b))}{N(s, a)}}$$

$N(s, a)$: number of visits of (s, a)

$S(s, a)$: number of visits of (s, a) ending with the root player winning



Monte-Carlo Tree Search

Bandit-Based Monte-Carlo planning : to select a path in the tree, run a bandit algorithm each time a children (next action) needs to be selected

UCT = **UCB for Trees** [Kocsis and Szepesvári, 2006]

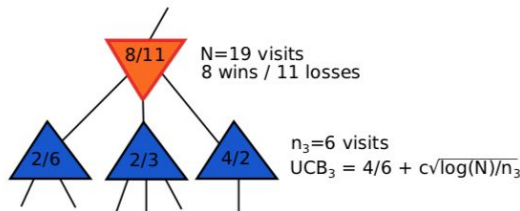
UCT in a Game Tree

In a **MIN node** s (= adversary move), select an action

$$\operatorname{argmin}_{a \in C(s)} \frac{S(s, a)}{N(s, a)} - c \sqrt{\frac{\ln(\sum_b N(s, b))}{N(s, a)}}$$

$N(s, a)$: number of visits of (s, a)

$S(s, a)$: number of visits of (s, a) ending with the root player winning



Monte-Carlo Tree Search

Bandit-Based Monte-Carlo planning : to select a path in the tree, run a bandit algorithm each time a children (next action) needs to be selected

UCT = **UCB for Trees** [Kocsis and Szepesvári, 2006]

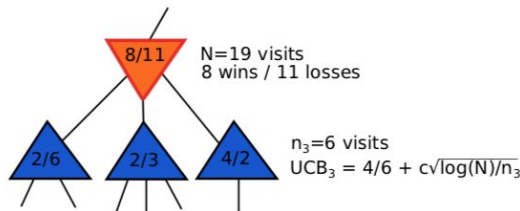
UCT in a Game Tree

In a **MIN node** s (= adversary move), select an action

$$\operatorname{argmin}_{a \in C(s)} \frac{S(s, a)}{N(s, a)} - c \sqrt{\frac{\ln(\sum_b N(s, b))}{N(s, a)}}$$

$N(s, a)$: number of visits of (s, a)

$S(s, a)$: number of visits of (s, a) ending with the root player winning



Monte-Carlo Tree Search

Bandit-Based Monte-Carlo planning : to select a path in the tree, run a bandit algorithm each time a children (next action) needs to be selected

UCT = UCB for Trees [Kocsis and Szepesvári, 2006]

UCT in a Game Tree

In a **MAX node** s (= root player move), select an action

$$\operatorname{argmax}_{a \in \mathcal{C}(s)} \frac{S(s, a)}{N(s, a)} + c \sqrt{\frac{\ln(\sum_b N(s, b))}{N(s, a)}}$$

$N(s, a)$: number of visits of (s, a)

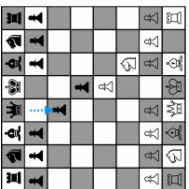
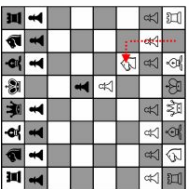
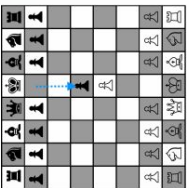
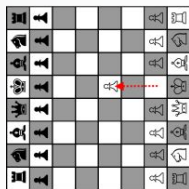
$S(s, a)$: number of visits of (s, a) ending with the root player winning

When a leaf (or some maximal depth) is reached :

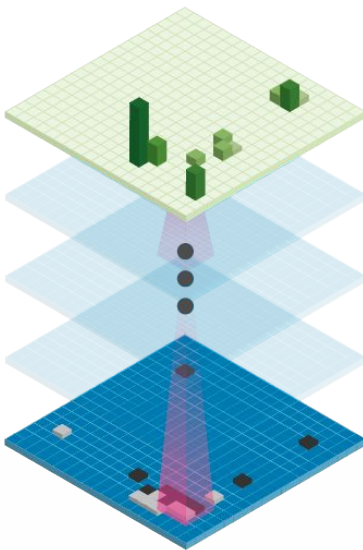
- ▶ a **playout** is performed (play the game until the end with a simple heuristic, or produce a random evaluation of the leaf position)
- ▶ the outcome of the playout (typically 1/0) is **stored in all the nodes visited in the previous trajectory**



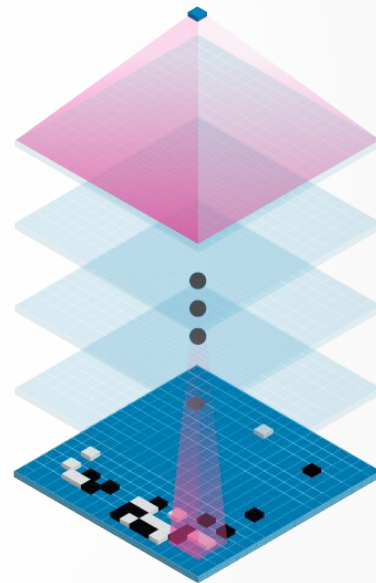
Policy and Value Nets trained by self-play



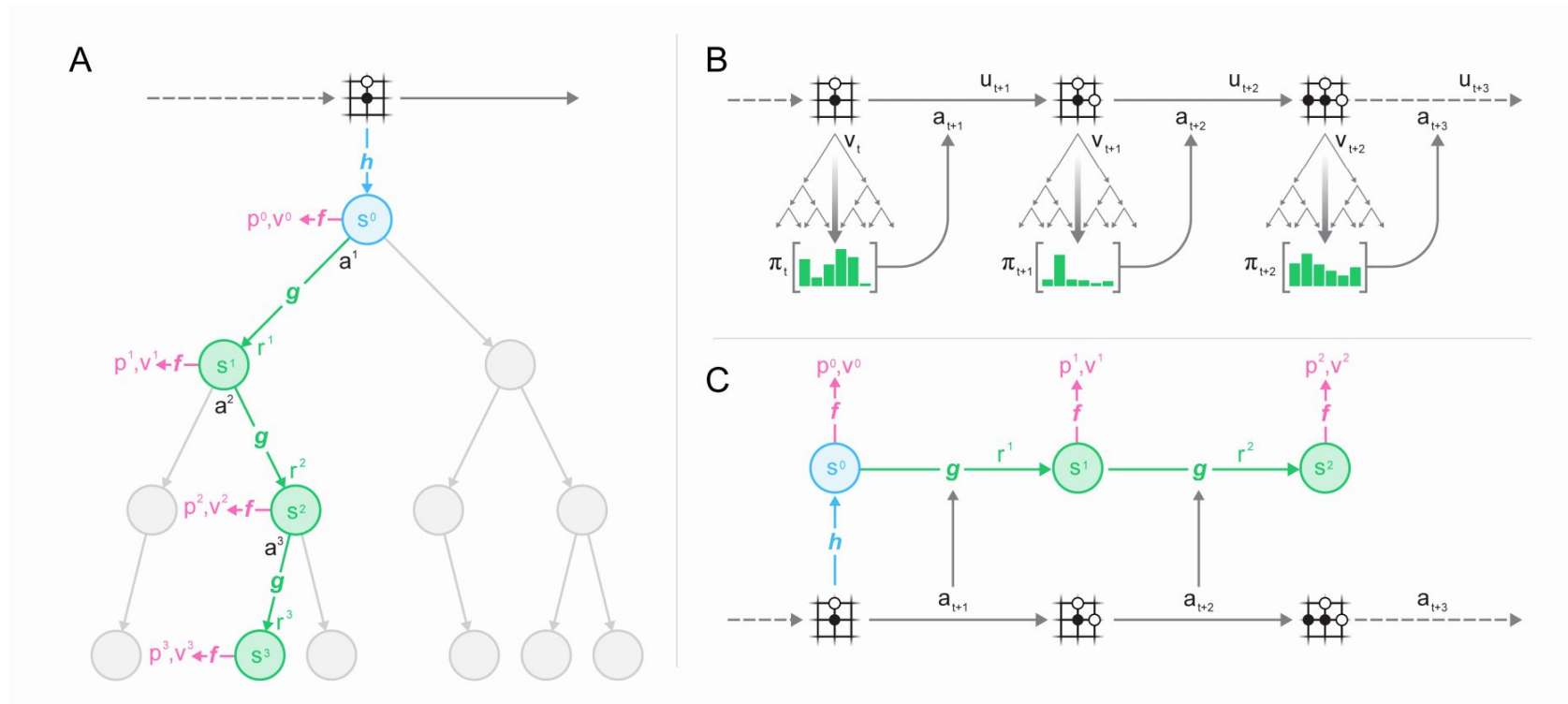
Policy Network returns
Probability distribution over moves
Selects most likely moves

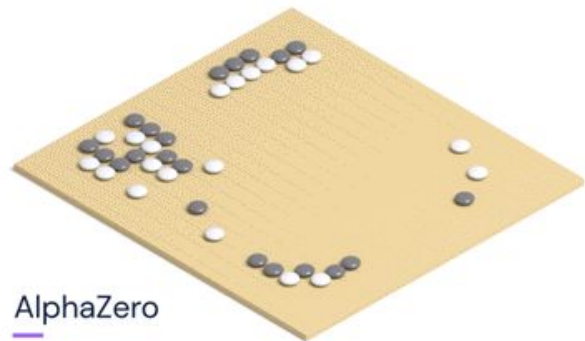


Value Network returns
Probability of winning: 0 (White) - 1 (Black)
Evaluates desirability of a position

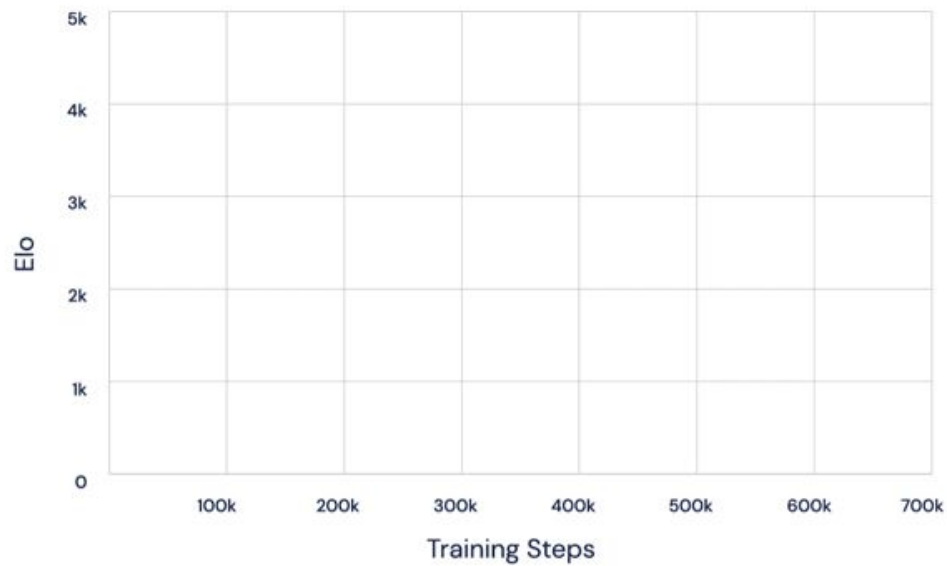


Monte-Carlo Tree Search in MuZero





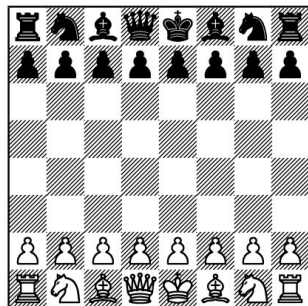
AlphaZero



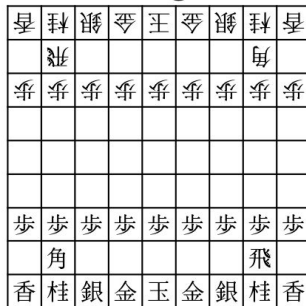
MuZero - Schrittwieser et al. 2019

<https://arxiv.org/pdf/1911.08265.pdf>

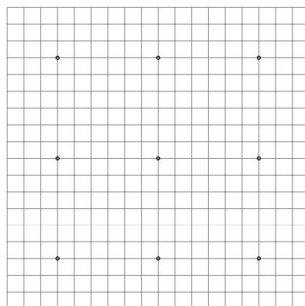
Chess



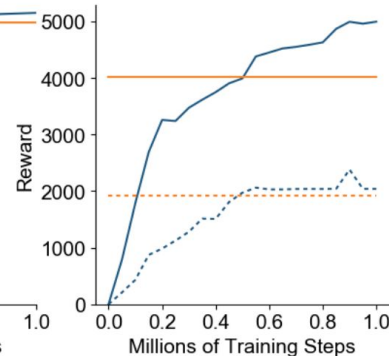
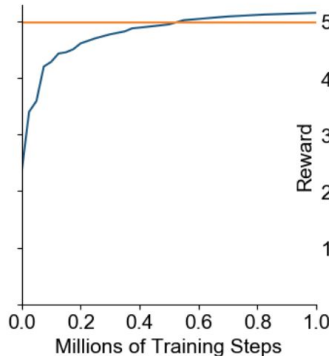
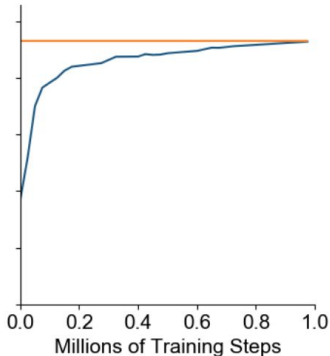
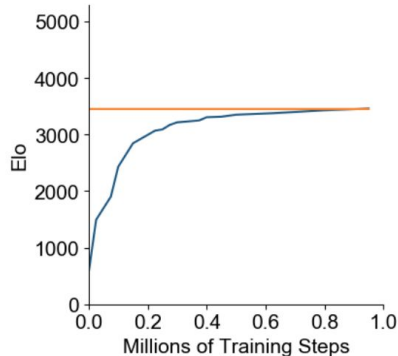
Shogi



Go



Atari



14:32

Catalyst LE



0	AlphaStar		177 / 200	945 +2015	758 +873	64	113	940	2 1
			SUPPLY	MINERALS	GAS	WORKERS	ARMY	APM	
0	LiquidTLO		147 / 172	335 +1595	442 +1030	61	86	1377	2 2
			SUPPLY	MINERALS	GAS	WORKERS	ARMY	APM	

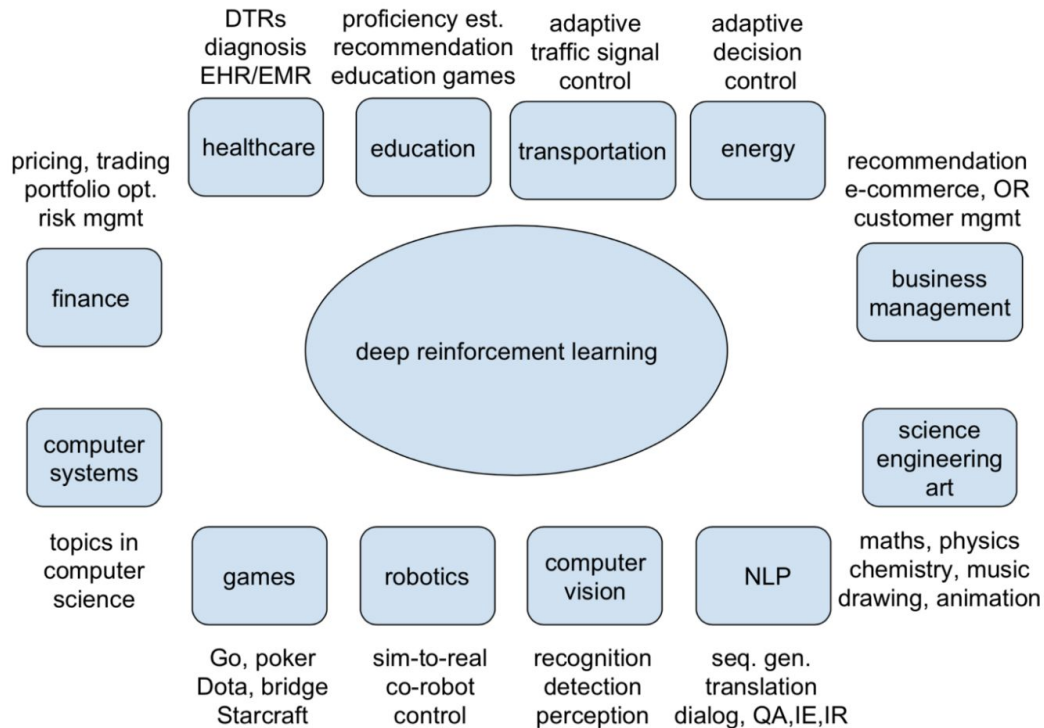


DeepMind

Challenges and open questions



Why RL? For any sequence of decisions



DeepMind AI Reduces Google Data Centre Cooling Bill by 40%



Challenges and open questions

- Reproducibility
- Long horizon, time
- Abstraction, space-time and universality
- Policy evaluation and exploration in growing batch RL
- Distributional RL
- Prioritized replay and self-imitation
- Safety and risk



References

- Wang et al., 2016 [Dueling Network Architectures for Deep Reinforcement Learning](#)
- Modayil et al., 2011 [Multi-timescale Nexting in a Reinforcement Learning Robot](#)
- Sutton et al., 2000 [Policy Gradient Methods for Reinforcement Learning with Function Approximation](#)
- Mnih et al., 2013 [Playing Atari with Deep Reinforcement Learning](#)
- Espeholt et al., 2018 [IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures](#)
- Yu & Finn et al 2018 [One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning](#)
- Paine et al., 2018 [One-Shot High-Fidelity Imitation: Training Large-Scale Deep Nets with RL](#)
- Bellemare, Dabney, Munos, 2017 [A Distributional Perspective on Reinforcement Learning](#)
- Agrawal et al., 2019 [Learning to Generalize from Sparse and Underspecified Rewards](#)
- Bard et al., 2019 [The Hanabi Challenge: A New Frontier for AI Research](#)
- Dabney et al., 2018 [Implicit Quantile Networks for Distributional Reinforcement Learning](#)
- Lillicrap et al., 2015 [Continuous control with deep reinforcement learning](#)
- Barth-Maron et al., 2018 [Distributed Distributional Deterministic Policy Gradients](#)
- Ho & Ermon, 2016 [Generative Adversarial Imitation Learning](#)
- Goodfellow et al., 2014 [Generative Adversarial Networks](#)
- Zolna et al., 2019 [Task-Relevant Adversarial Imitation Learning](#)
- OpenAI et al., 2019 [Solving Rubik's Cube with a Robot Hand](#)
- Paine et al., 2019 [Making Efficient Use of Demonstrations to Solve Hard Exploration Problems](#)
- Pierrot et al., 2019 [Learning Compositional Neural Programs with Recursive Tree Search and Planning](#)
- Cabi et al., 2019 [A Framework for Data-Driven Robotics](#)
- Zhu et al., 2018 [Reinforcement and imitation learning for diverse visuomotor skills](#)

