SAISON 22/23

Formation

Introduction au Deep Learning

18

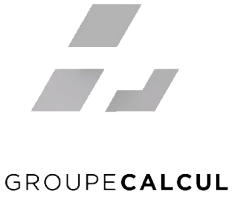
Deep Reinforcement Learning
Tactiques et stratégies

FIDLE

You can also subscribe to :

http://fidle.cnrs.fr/listeinfo
Fidle information list

https://listes.services.cnrs.fr/wws/info/devlog
List of ESR* « Software developers » group



https://listes.math.cnrs.fr/wws/info/calcul
List of ESR* « Calcul » group

(*) ESR is Enseignement Supérieur et Recherche, french universities and public academic research organizations

Ressources

https://fidle.cnrs.fr

Powered by CNRS CRIC, and UGA DGDSI
of Grenoble, Thanks !

Course materials (pdf)

Practical work environment*

Corrected notebooks

Videos (YouTube)

(*) Procedure via Docket or pip
Remember to get the latest version !

Program

20 Séquences
du 17 novembre
au 14 mai 2023

1 ✓
History, Fundamental Concepts
Basic Regression DNN
Basic Classification DNN

2 3 ✓
High Dimensionnal Data
CNN

4 ✓
Demystify **mathematics** for neural networks.

5 ✓
Training strategies Evaluation
Sparse data (text) Embedding

6 ✓
Sequences data
RNN

7 ✓
PyTorch
A small detour with **PyTorch**.

8 ✓
«Attention is All You Need»
Transformers

9 ✓ New !
Graph Neural Network
GNN

10 ✓
Autoencoder networks
AE

11 ✓
Variational Antoencoder
VAE

12 JDLS 2023 — JDLS 12 mai
Project session
«My project in 180 s»

13 ✓
Generative Adversarial Networks
GAN

14 ✓ New !
Diffusion Model
Text to image

15 ✓
AI, Law, Society and **Ethics**

16 ✓ New !
Model and training optimization
Resource efficiency

17 ✓
Jean-Zay
GPU acceleration

18
Deep Reinforcement Learning
RL

19 New !
Physics-Informed Neural Networks
PINNS

20
JDLS 2023
Deep Learning for Science !

FIDLE

SAISON 22/23

Roadmap

18

Deep Reinforcement Learning

RL

18.1 **About RL**

➔ RL Introduction

➔ Tabular RL : Sarsa & Q-learning

➔ Example : Cliff Walking

18.2 **From RL to Deep RL**

➔ Advanced topicss in RL

➔ DQNs

18.3 **Policy Gradient**

➔ On-policy Policy Gradient

➔ Off-policy Policy Gradient

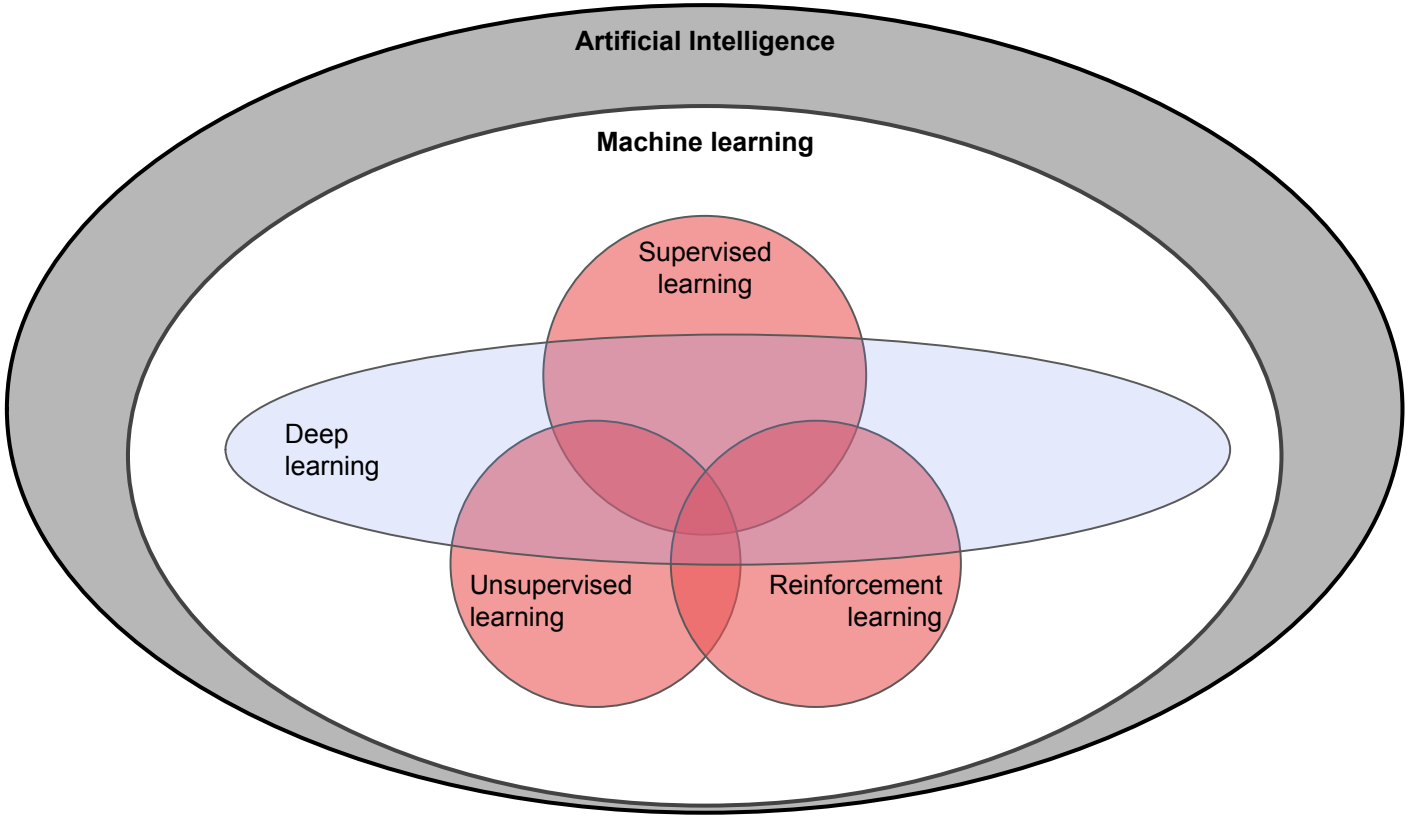
➔ Example : Car pole

18.4 **Success & Perspectives**

About RL

FIDLE

RL areas of AI



RL - History

1930's
Trial and Error learning
Psychology, animal learning

1950's
Optimal control
Value function, dynamic programming

1950's
Temporal Difference
Animal learning, Game theory

Richard Bellman
(1920-1984)

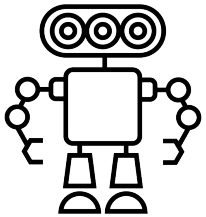
Marvin Minsky
(1927-2016)

18.1

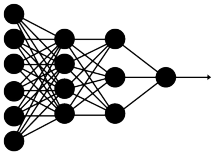
1980's late
Modern Reinforcement Learning (Q- learning, Sarsa)

1996
Neurodynamic programming

Robotic



Game Theory



Neural Network

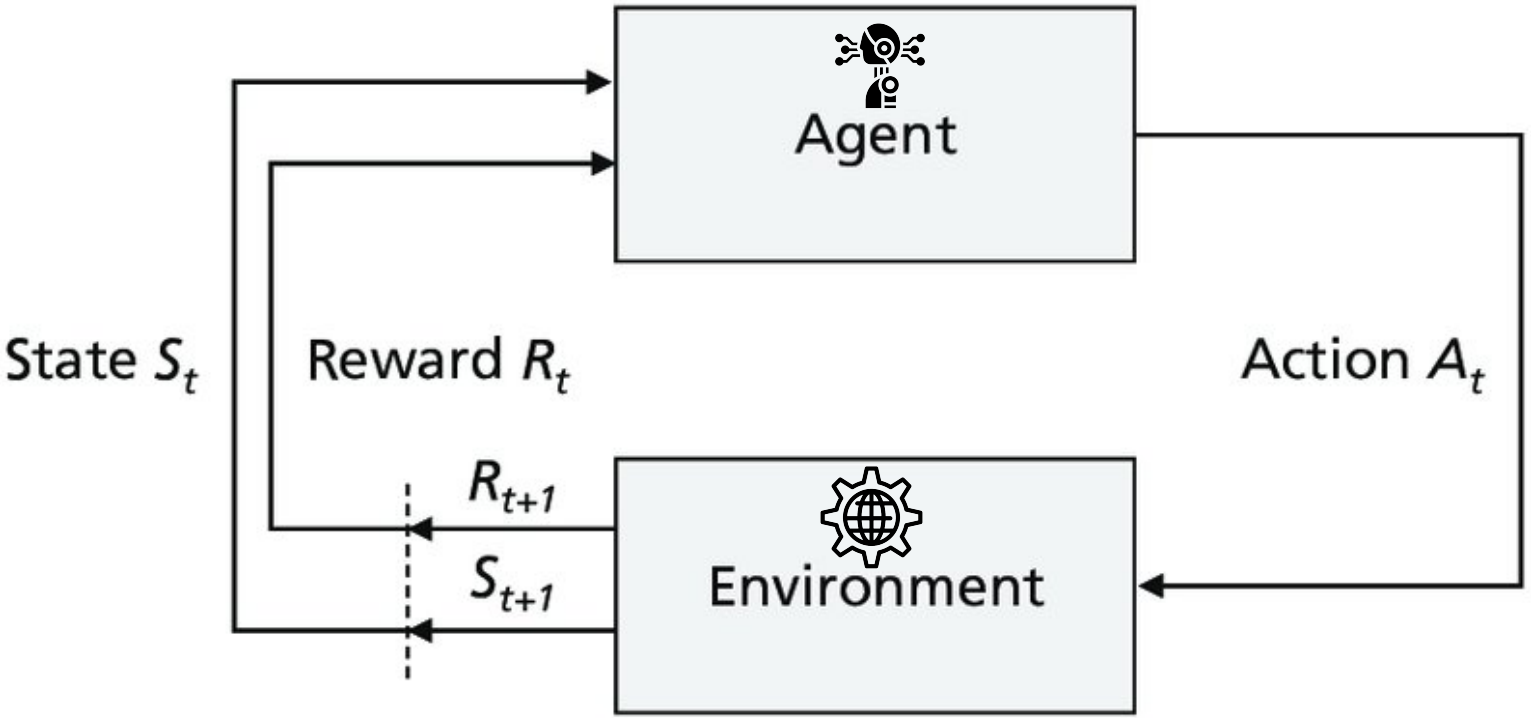
18.2 18.3

2013
DQN Playing Atari with Deep Reinforcement Learning

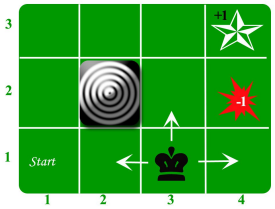
DEEPMIND

OpenAI

RL - Actions/Policies learning

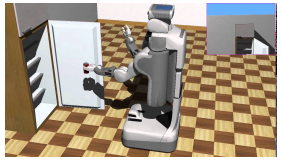


RL - Environments



Grid world

Video Game

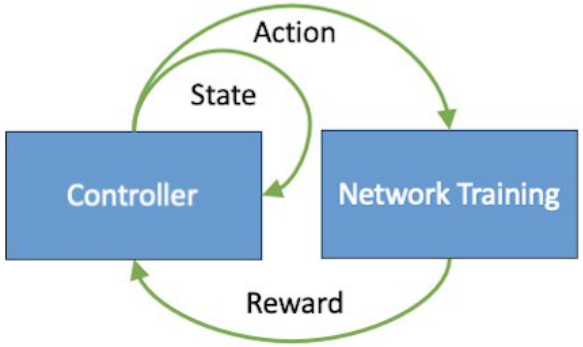


Simulator

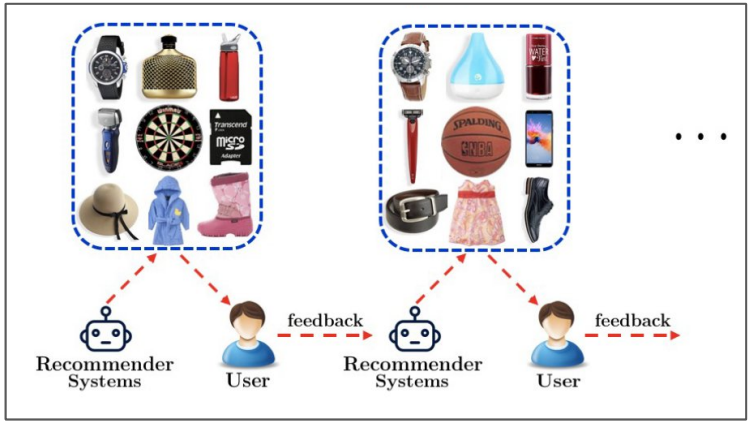
Reality

Deterministic	Fully Observable	Episodic	Static	Discrete Control	Mono-Agent
Stochastic	**Partially Observable**	**Sequential**	**Dynamic**	**Continuous Control**	**Multi-Agents Adversarial Collaborative**

RL - Applications



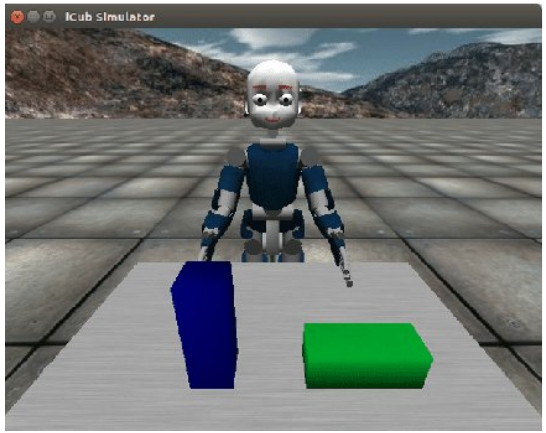
AutoML

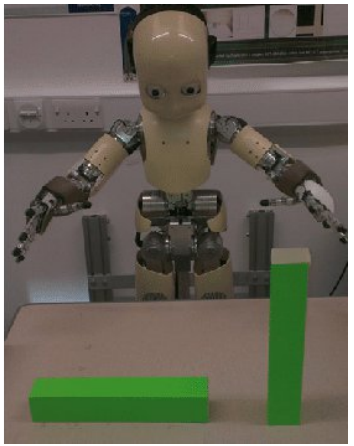


RL - Limits of Simulation

Need simulators to train model

"From robotic simulator to real world" biases

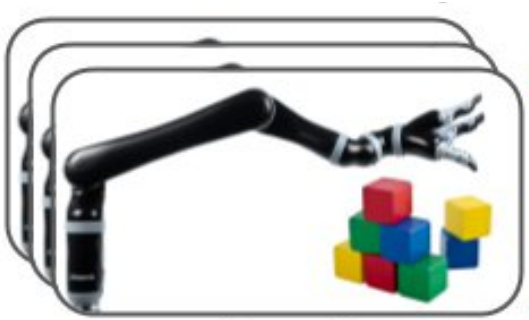




RL vs Supervised Learning

(Self-)Supervised learning

Dataset of experience



- Hard to collect a representative labeled dataset
- Hard to deal with dynamic environment

vs

Reinforcement learning

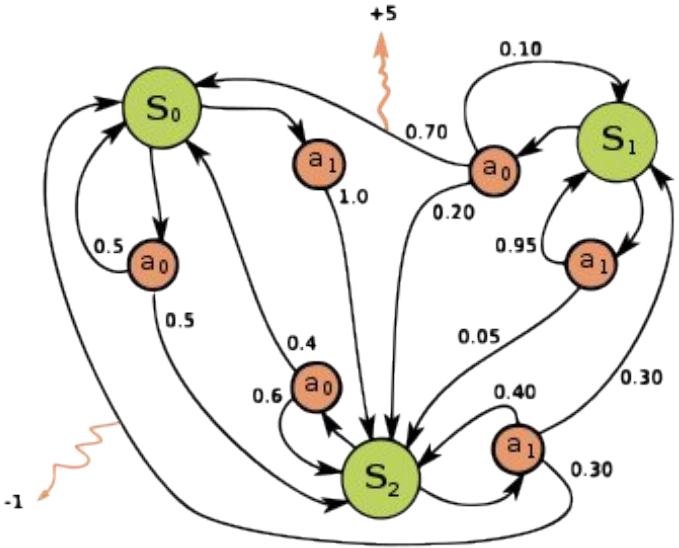
Lifelong learning

- Slow live training
- Reward function hard to design

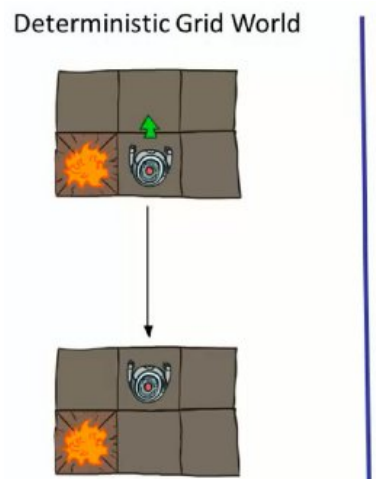
Questions Break

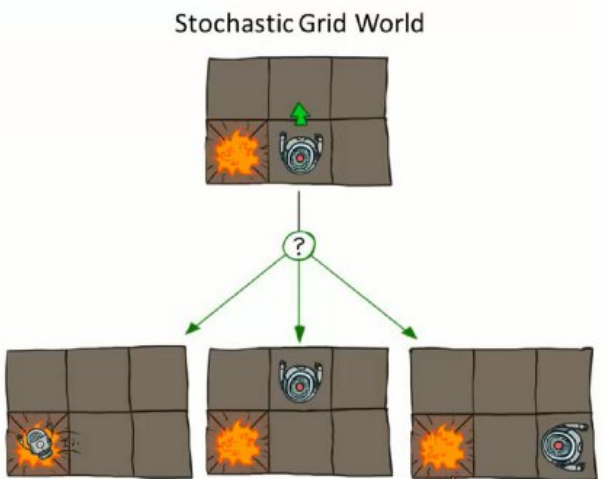
Optimal Control - MDP & Grid World

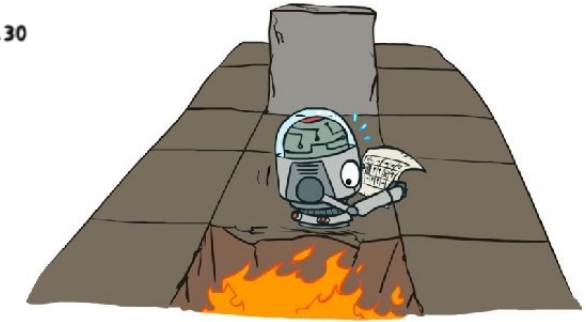
MDP: Markov Decision Process

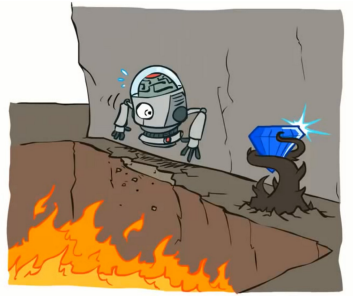


Markov chain



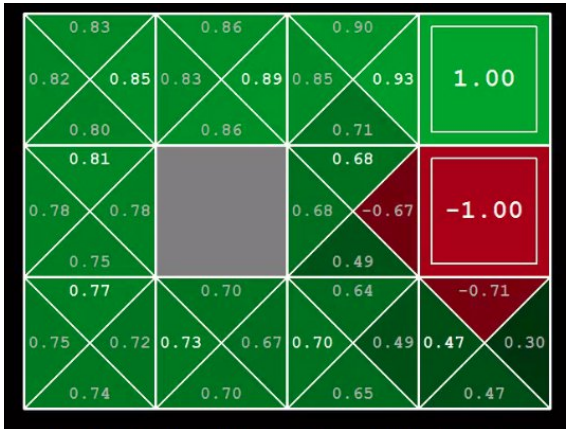






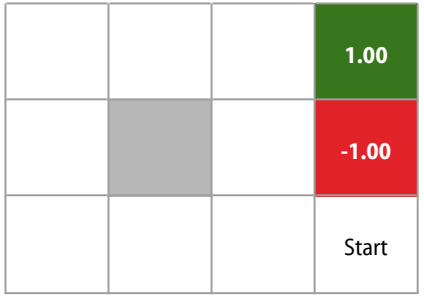
Optimal Control - Bellman Equations

Q(s,a) : Action-State Value function



Q table

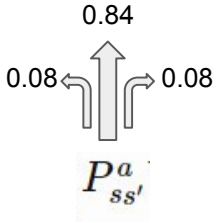
Grid world



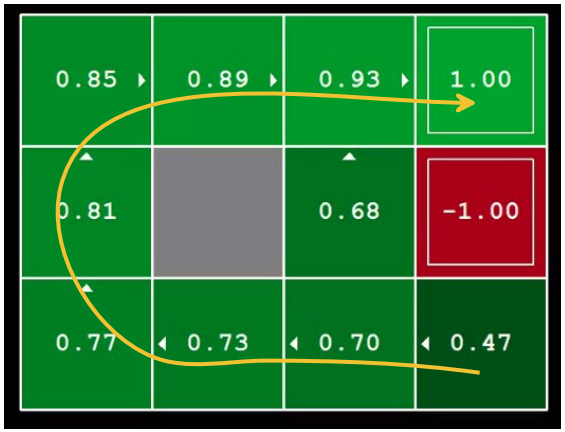
R(s, a) : Reward

s′ : next state

$\gamma = 0.96$
discount rate



V(s) : State Value function



V table

$$Q_*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} \max_{a' \in \mathcal{A}} Q_*(s',a')$$

$$\max_a Q(s,a) = V(s)$$

$$V_*(s) = \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} V_*(s') \right)$$

Optimal Control - Discount rate

Which discount rate you apply for yourself ?

day by day:

- 0.1
- 0.5
- 0.9

Reward trajectory

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots$$

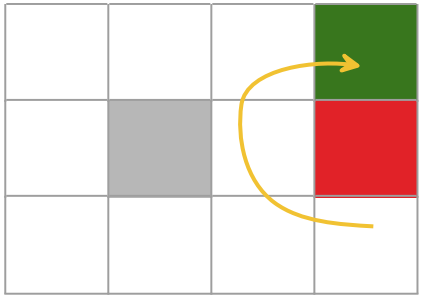
Return: cumulative reward

Gamma: discount rate

Trajectory (read Tau)
Sequence of states and actions

$$R(\tau) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

$\gamma = 0.90$



Optimal Control - Dynamic Programming

0.	0.	0.	1.00
0.		0.	-1.00
0.	0.	0.	0.

1. Initialisation

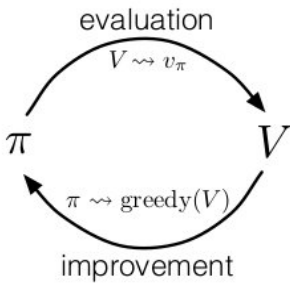
$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

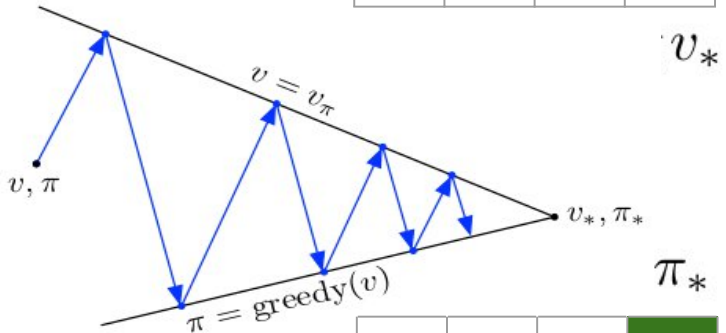
↓	↑	→	1.00
→		↓	-1.00
↓	→	←	↓

2. Policy Evaluation

Loop:
$\Delta \leftarrow 0$
Loop for each $s \in \mathcal{S}$:
$v \leftarrow V(s)$
$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number determining

0.85	0.89	0.93	1.00
0.81		0.68	-1.00
0.77	0.73	0.70	0.47





3. Policy Improvement

If policy stable, then stop and return

V ≈ v* and π ≈ π*; else go to 2

→	→	→	1.00
↑		↑	-1.00
↑	←	←	←

Optimal Control - Dynamic Programming

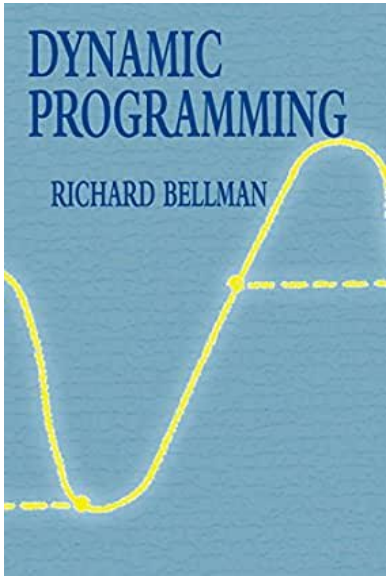
Dynamic Programming is **not** Machine learning and have very limited applications because it need :

- Finite MDP : Static, Episodic
- Perfect known Model : **Fully Observable**

Is computational expensive if environment is large

⬇

Then, Let's go back to the Reinforce Learning world : like an embodied agent !!



"... breaks problem into simpler steps at different points in time."

Monte Carlo / Temporal Difference

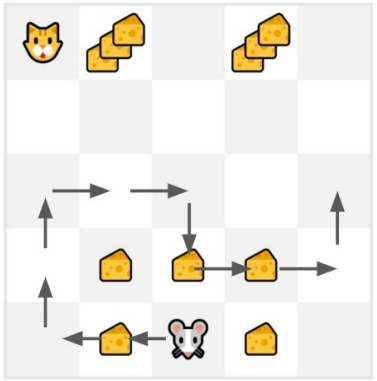
- **Monte Carlo**

G_t: reward trajectory until the **game over**

$$= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

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

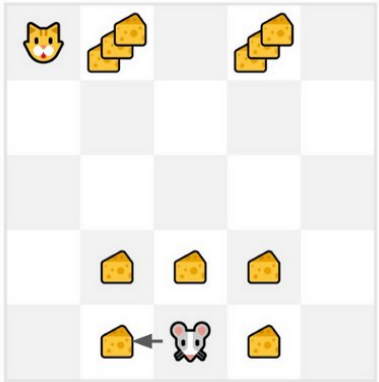
- New value of state t
- Former estimation of value of state t (= Expected return starting at that state)
- Learning Rate
- Return at timestep t
- Former estimation of value of state t (= Expected return starting at that state)



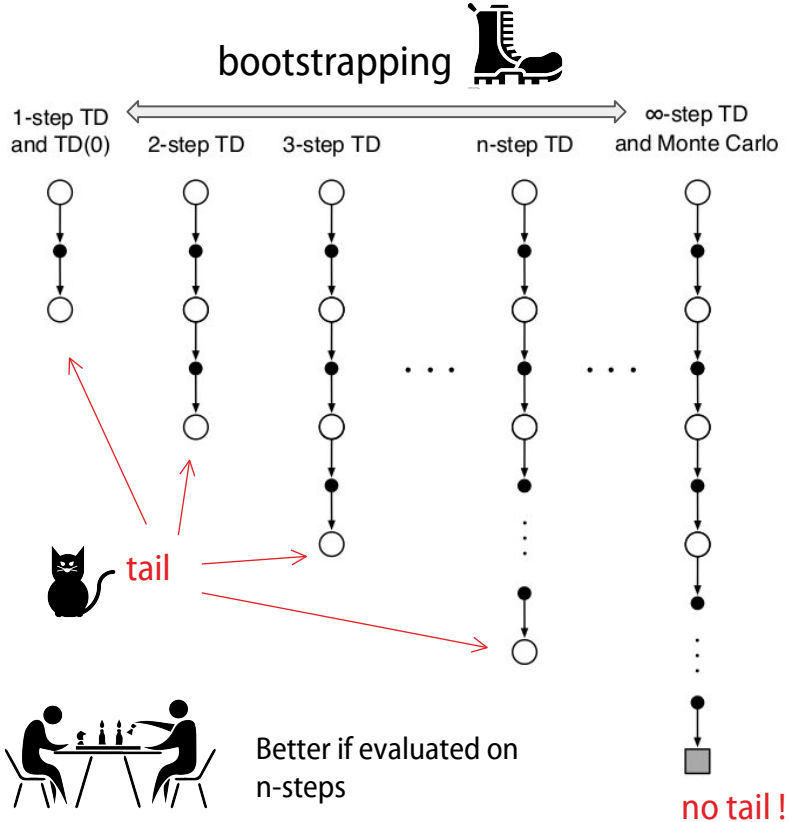
- **Temporal Difference**

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

- New value of state t
- Former estimation of value of state t
- Learning Rate
- Reward
- Discounted value of next state
- TD Target



Temporal Difference(n)

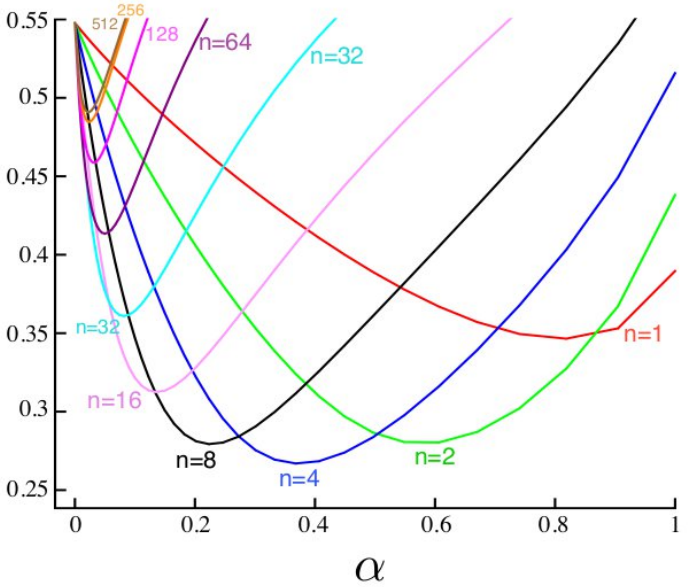


tail

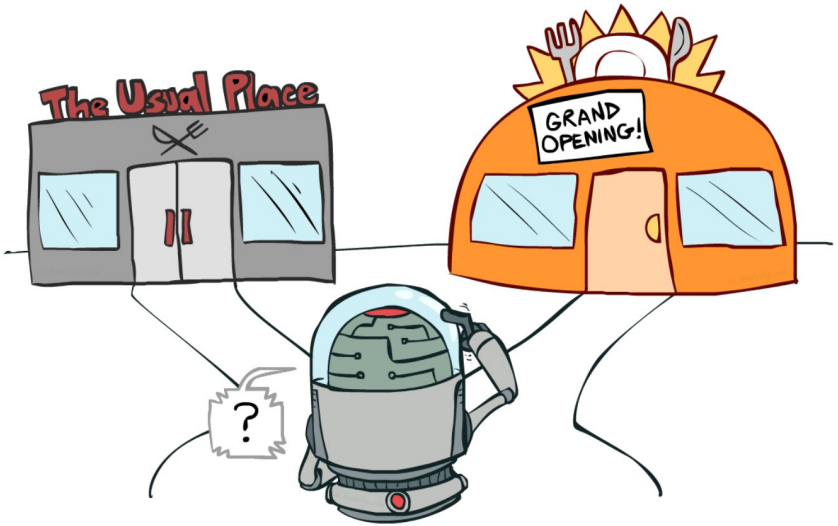
$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

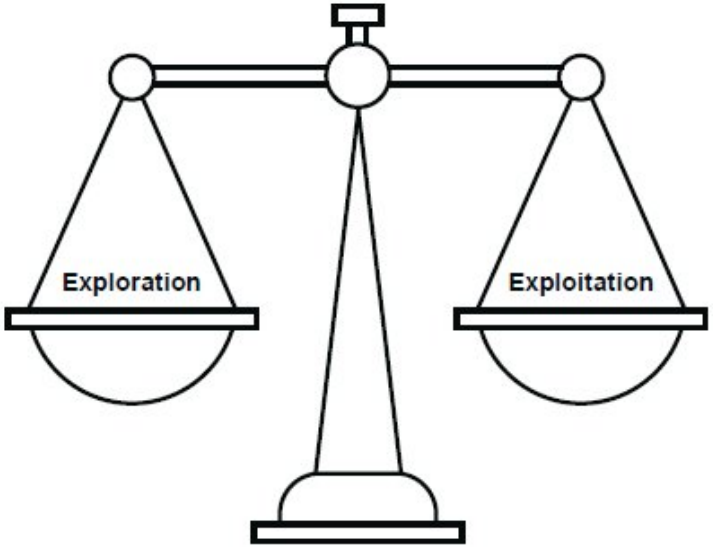
n-step TD Target

Average RMS error over 19 states and first 10 episodes



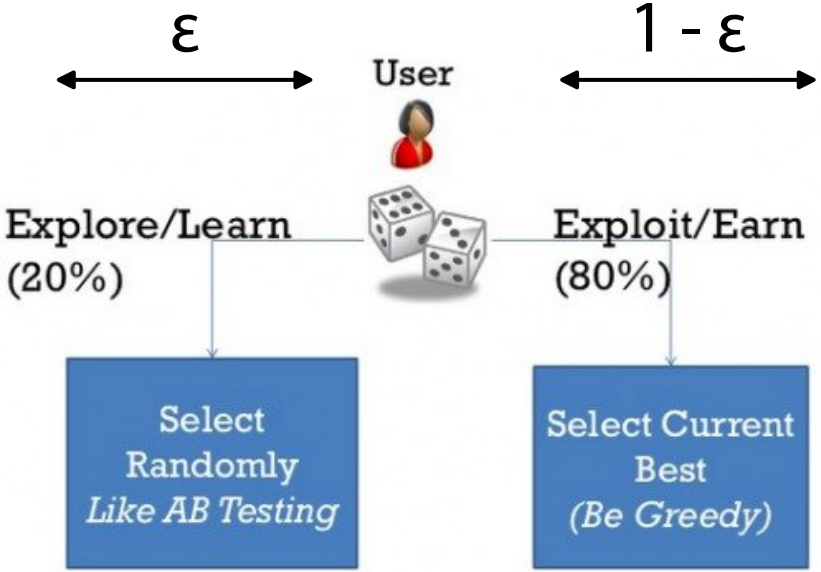
Trial and Error Learning





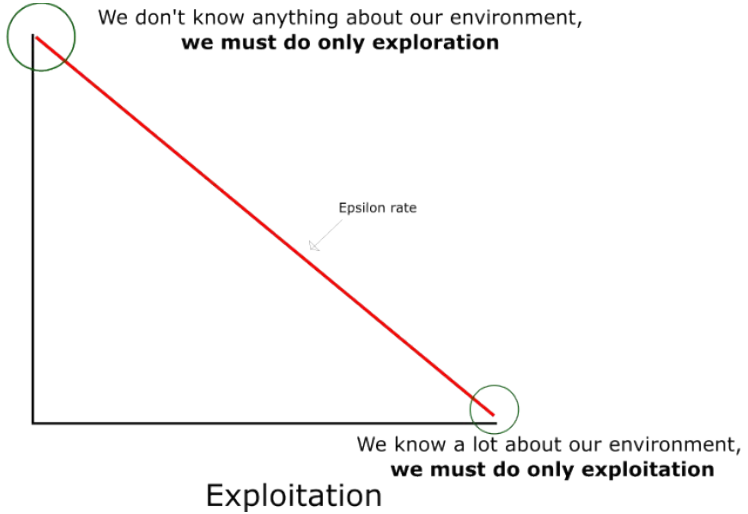
Exploration / Exploitation Trade-off

Trial and Error Learning ε-greedy



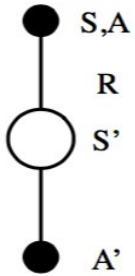
Exploration

Decrease epsilon in time !!

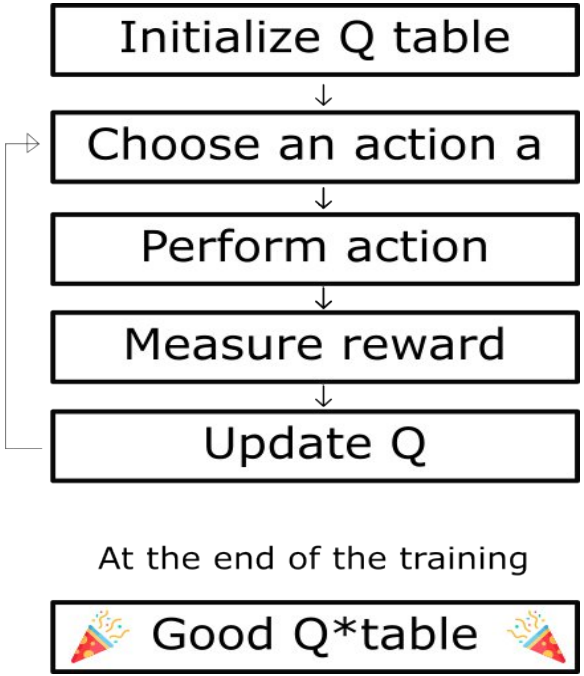


Exploitation

SARSA - On-policy



2 x $\mathcal{E}$-greedy !

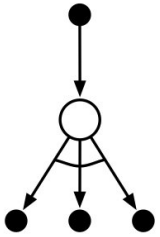


Choose A from S using policy derived from Q (e.g., ε-greedy)

Choose A' from S' using policy derived from Q (e.g., ε-greedy)
$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$
$S \leftarrow S'$; $A \leftarrow A'$;

Q-learning - Off-policy

1 x $\mathcal{E}$-greedy !



Q-learning

Initialize Q table

↓

Choose an action a

↓

Perform action

↓

Measure reward

↓

Update Q

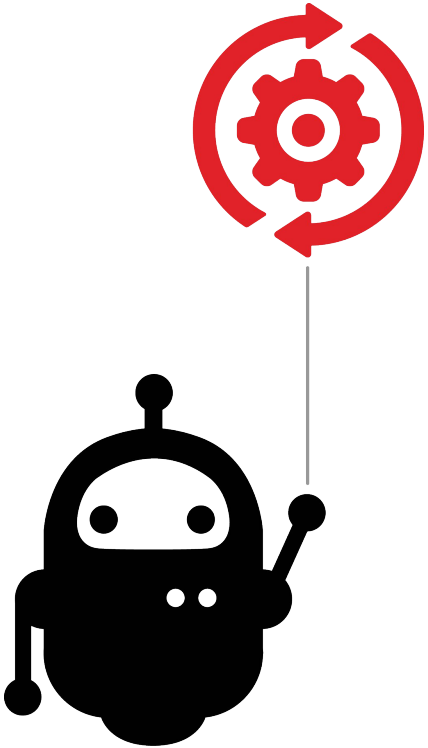
Choose A from S using policy derived from Q (e.g., ε-greedy)

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$$
$$S \leftarrow S'$$

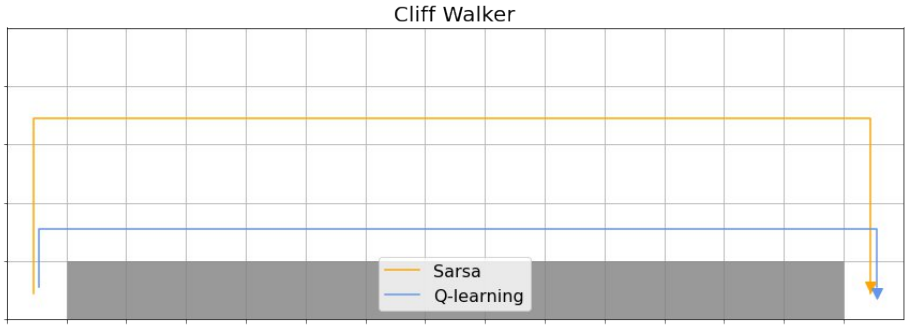
At the end of the training

Good Q*table

Example : Cliff Walking



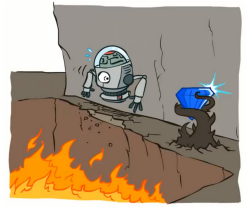
[RL1] Cliff Walking : Sarsa vs Q-learning

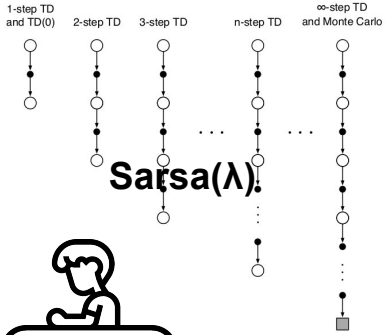


- Q-learning: Off-Policy (1 x ε-greedy)
- SARSA: On-Policy (2 x ε-greedy)

To have in Mind

- Bellman's Value function, Q (value-action) function and discount rate
- Exploration strategy : Ɛ-greedy
- Temporal Difference, TD(n) with tail, and Monte Carlo without tail
- On-policy, Off-policy concepts





Tabular RL resolved environments

Deterministic	Fully Observable	Episodic	Static	Discrete Control	Mono-Agent
OK Bellman's equations	**OK** Trajectory Exploration	**OK** TD Learning	**OK** Lifelong Exploration	**NOK**	**NOK**
Stochastic	**Partially Observable**	**Sequential**	**Dynamic**	**Continuous Control**	**Multi-Agents Adversarial Collaborative**

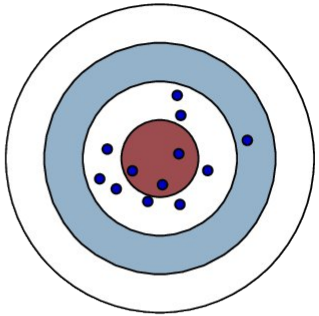
Questions Break

From RL to Deep RL

FIDLE

Variance vs Bias trade-off

High Variance



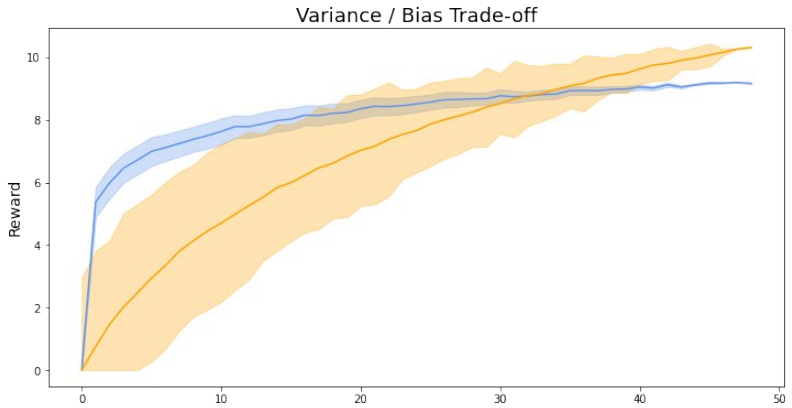
More Explorative
Slower Learning
Hard convergence

On-policy

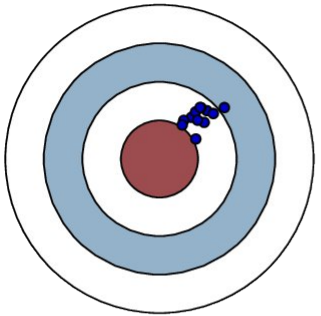
Monte Carlo

Actor

Reinforcement Learning Trade-off



High Bias



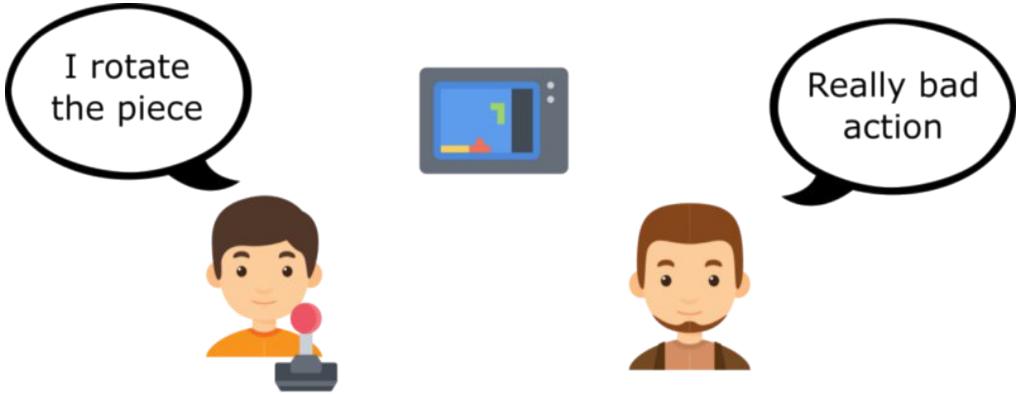
More Exploitative
Faster Learning
Easy convergence

Off-policy

TD Learning

Critic

Actor-Critic - TD Learning



Actor

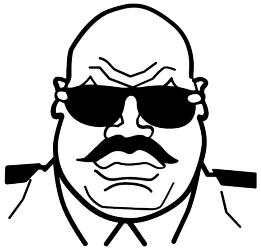
decides which action to take

Critic

tells the actor how good its action was and how it should adjust

Variance-Bias Balance Learning

Advantage function

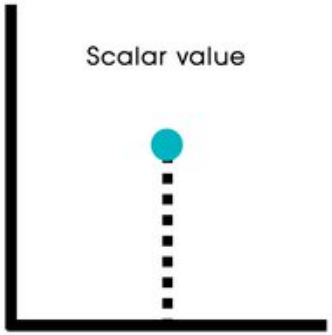


Q(s, a)
Value-Action function
Q guevara

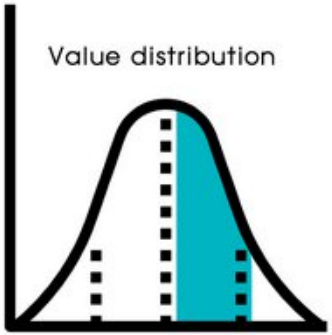
V(s)
Value function
Strategist
Visionary
Critic
How reach the best state at the end ?

A(s,a)
Advantage function
Tactician
Actor
From every state, what is the best action ?

Distributional RL



General RL



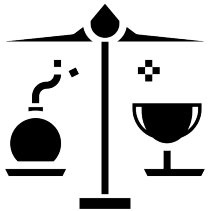
Distributional RL

Bellman Equation: $Q(s,a) = R(s,a) + \gamma Q(s',a')$

Distributional Bellman Equation: $Z(s,a) = R(s,a) + \gamma Z(s',a')$

Distributional Q-learning evaluate:

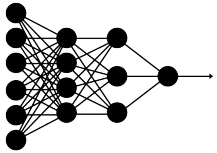
- the expected reward
- **and the risk to reach it**



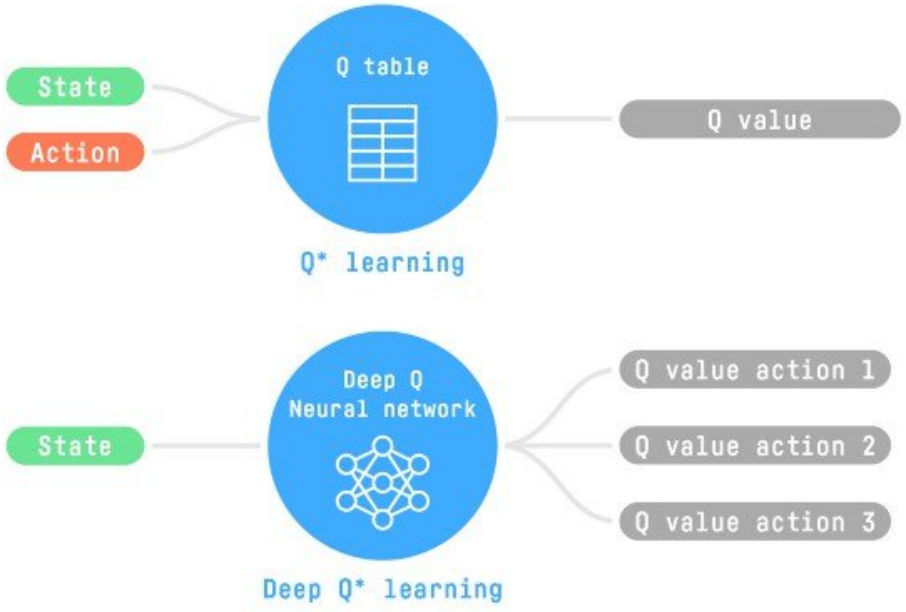
Q Tabular / Approximation Q function

Q table is a nonlinear function…

!?



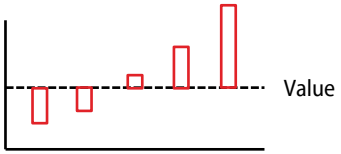
Why don't use a neural network ?



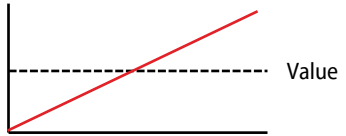
Approximation function benefit

Imagine this state-value function V=[-2.5, -1.1, 0.7, 3.2, 7.6]

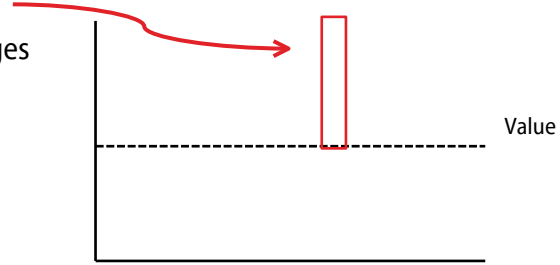
With Q-table each value is independent.



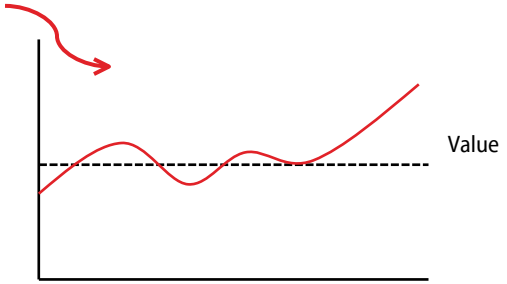
With function approximation, the underlying relationship of the states can be learned and exploited



With Q-table, the update only changes one step.

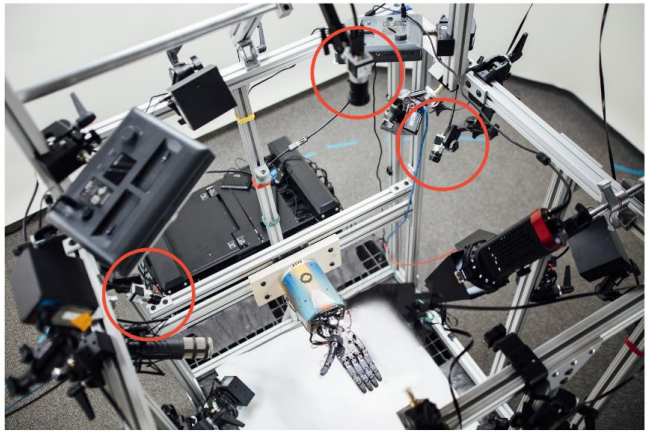


With function approximation, the update changes multiple steps.



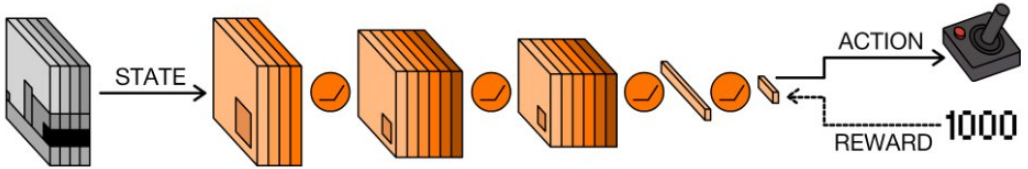
Deep learning perspective

Tabular representation is limited. But Deep learning enable to use **high dimensional data** : Image, text, ...

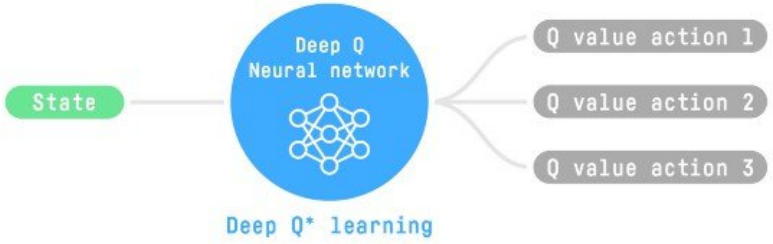


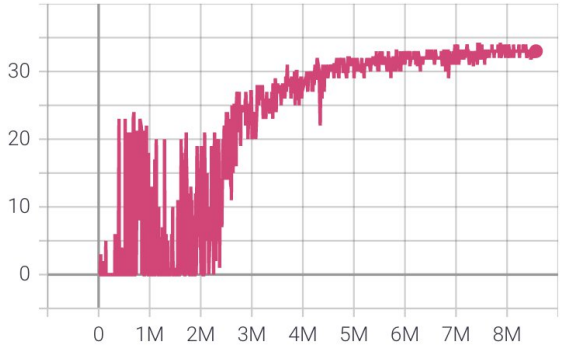
DQN : Deep Q-learning Network

[Playing Atari with Deep Reinforcement Learning (2013)]



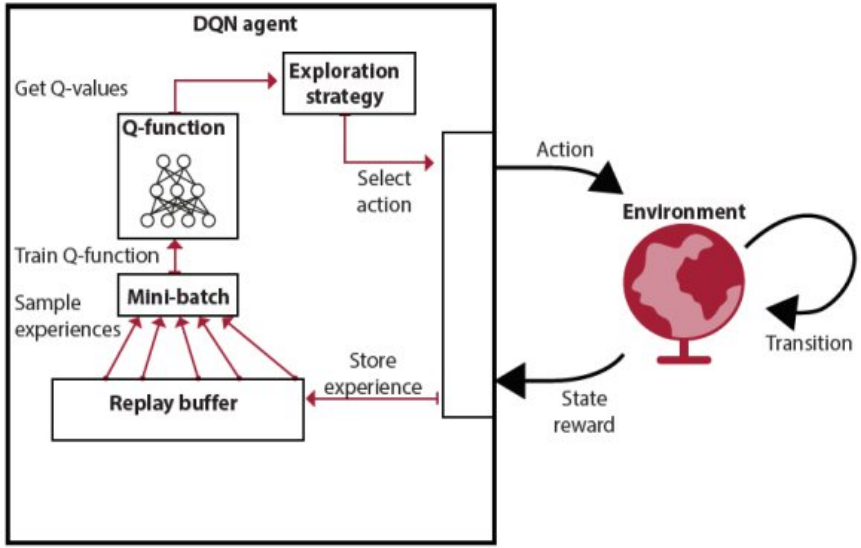
Freeway Game





DQN

Off-policy



Initialize Network

Initialize network Q
Initialize target network $\hat{Q}$
Initialize experience replay memory D
Initialize the *Agent* to interact with the Environment

↓

Choose an action a using policy ϵ-greedy(Q)

↓

Perform action

↓

Measure reward

↓

Update Q

Calculate the loss $\mathcal{L} = 1/N \sum_{i=0}^{N-1} (Q(s_i, a_i) - y_i)^2$
Update Q using the SGD algorithm by minimizing the loss $\mathcal{L}$

Store transition (*s, a, r, s', done*) in the experience replay memory *D*

$$\mathcal{L} = 1/N \sum_{i=0}^{N-1} (Q(s_i, a_i) - y_i)^2$$

Q-label

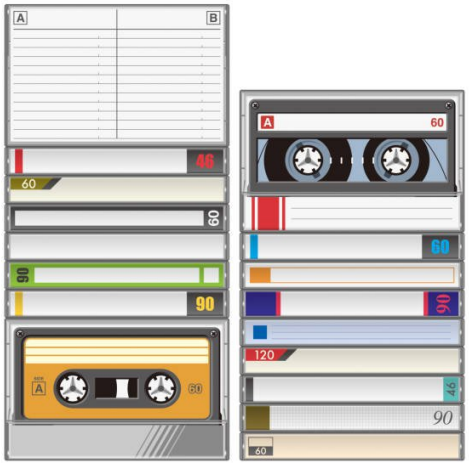
$$y_i = r + \gamma \max_a Q(s', a)$$

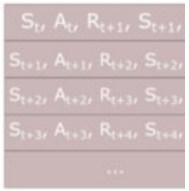
At the end of the training

Good Q Network

DQN - Replay Buffer

Experience Replay Buffer





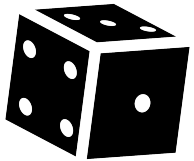
Experience replay Buffer

Sample

Batch of experiences

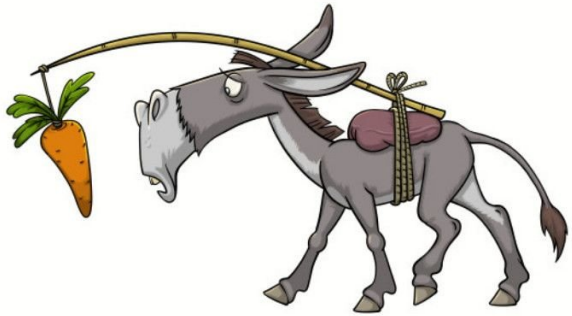
DQN Agent

Random sampling

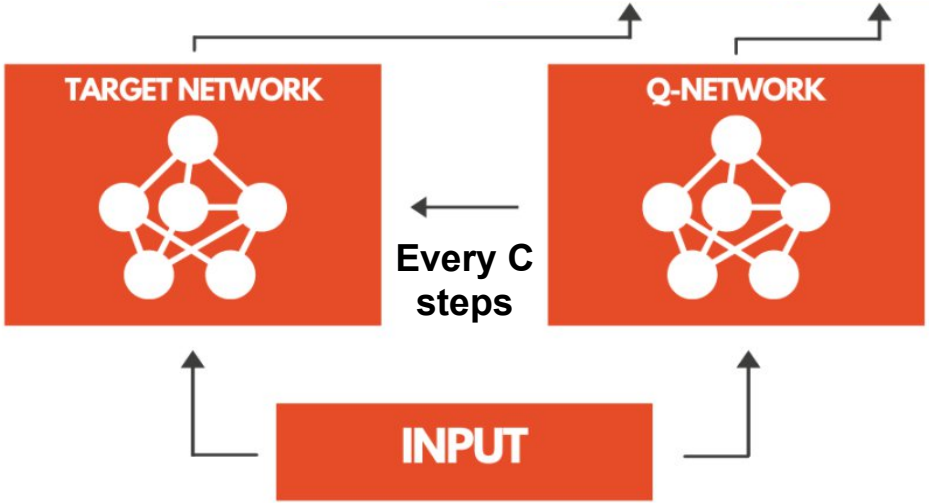


DQN - Target network

Target network / online network



$$\text{Loss} = \left(R_{t+1} + \gamma_{t+1} \max_{a'} q_{\bar{\theta}}(S_{t+1}, a') - q_{\theta}(S_t, A_t)\right)^2$$



Prioritized DQN

Prioritizing Experience Replay Buffer

Prioritize the examples with larger TD error, and so with more information

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

Probability sampling

$$p_i = \frac{1}{\text{rank}(i)}$$

rank according to how big is the TD error

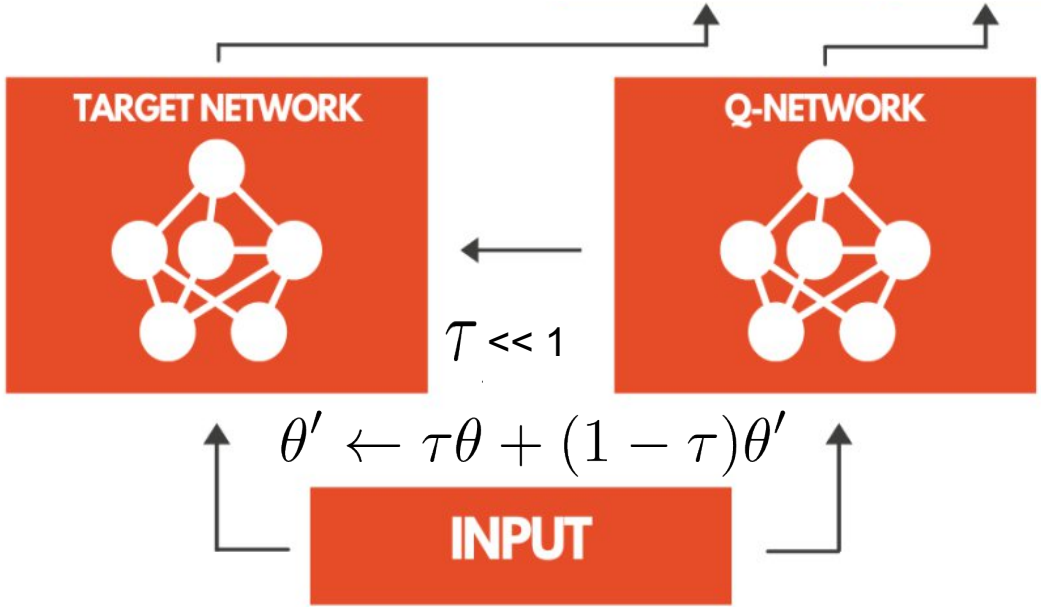
Double DQN

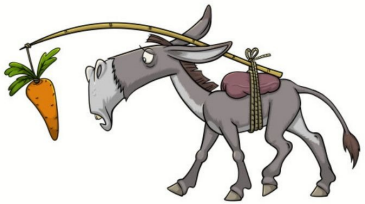
To fix Q value overestimation issue and to smooth the learning

$$\text{Loss} = \left(R_{t+1} + \gamma_{t+1} \max_{a'} q_{\bar{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t)\right)^2$$

$$\leq$$

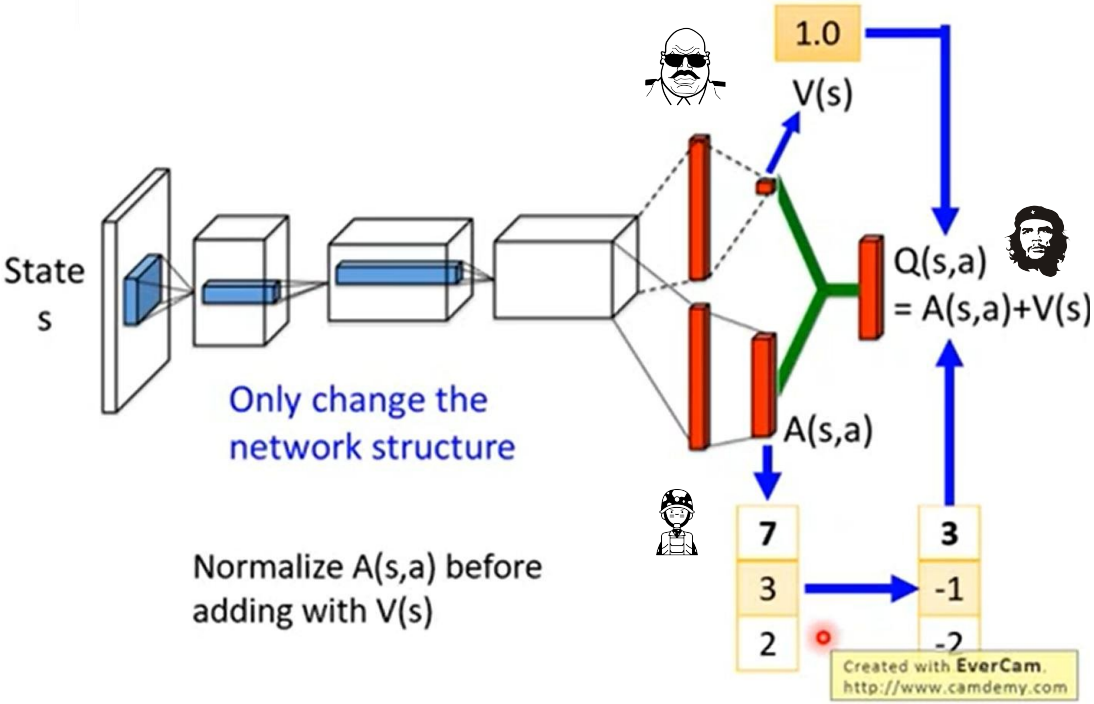
$$\text{Loss} = \left(R_{t+1} + \gamma_{t+1} q_{\bar{\theta}}(S_{t+1}, \operatorname{argmax}_{a'} q_\theta(S_{t+1}, a')) - q_\theta(S_t, A_t)\right)^2$$

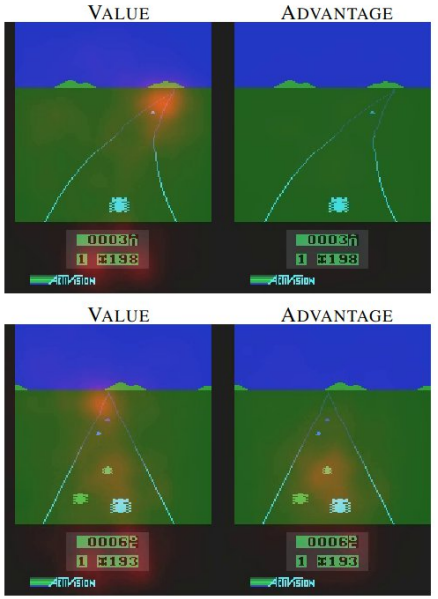




Dueling-DDQN

$$Q(s,a) = V(s) + A(s,a)$$



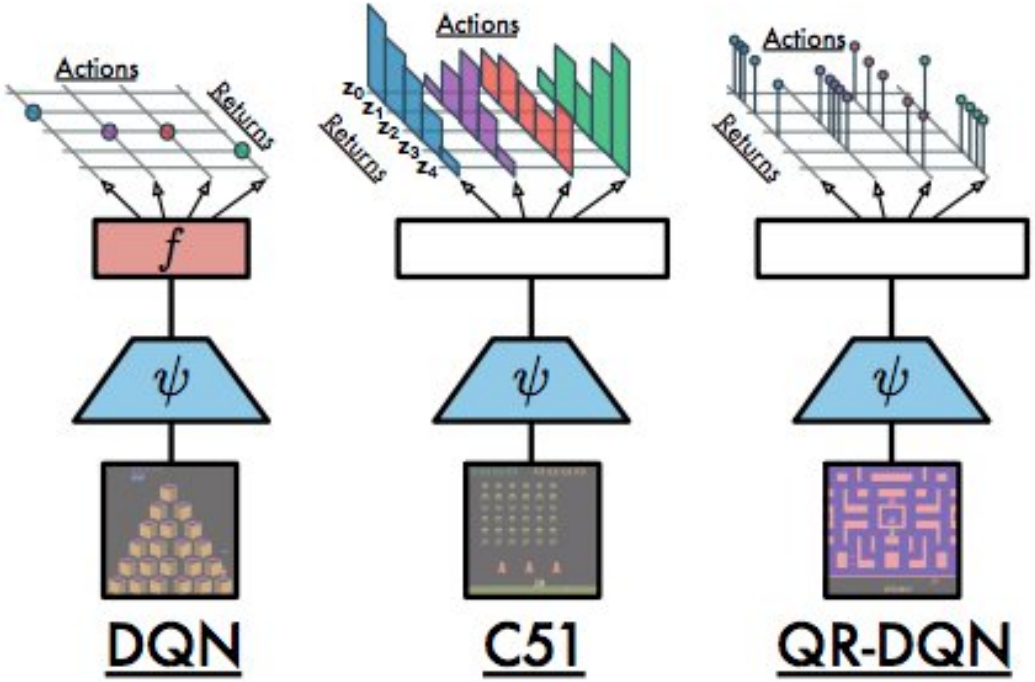


The value stream learns to pay **attention to the road**.
The advantage stream learns to pay attention only when there are cars immediately in front, so as to **avoid collisions**.

Distributional DQNs

Distributional RL aims to model the distribution over returns…

Distributional Bellman Equation: $Z(s,a) = R(s,a) + \gamma Z(s',a')$



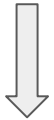
Categorical

Quantile Method

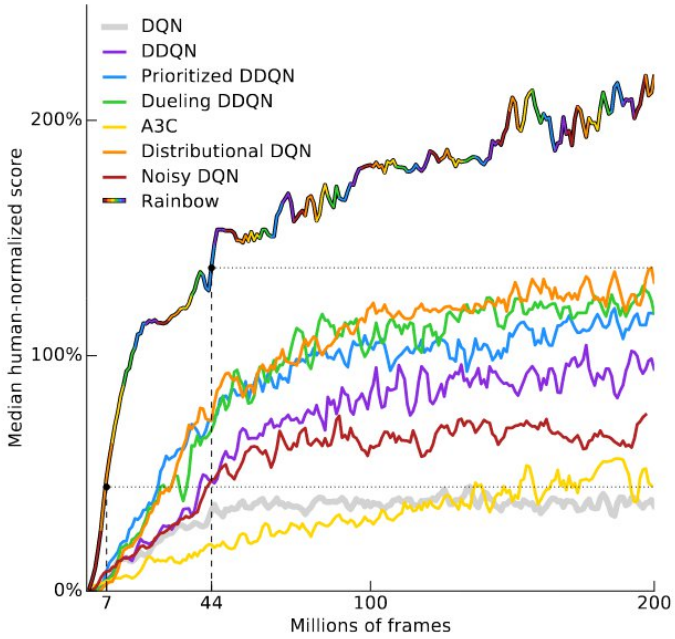
Rainbow DQN

https://arxiv.org/pdf/1710.02298.pdf

- DQN
- DDQN
- Prioritized DDQN
- Dueling DDQN
- Distributional DQN
- Noisy DQN (Add noise for exploration)



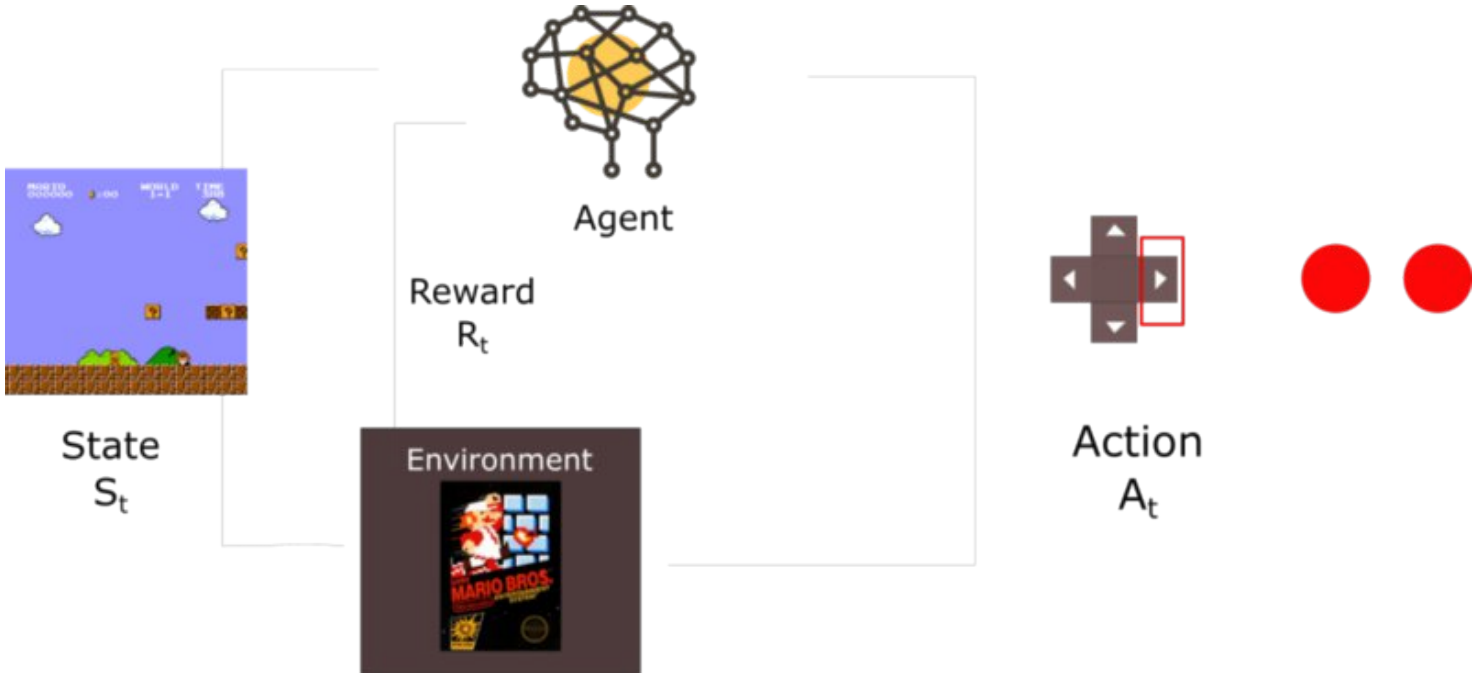
Rainbow DQN



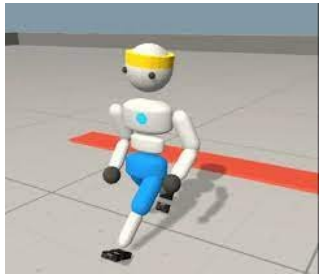
Median human-normalized performance across 57 Atari games

Limit of DQNs

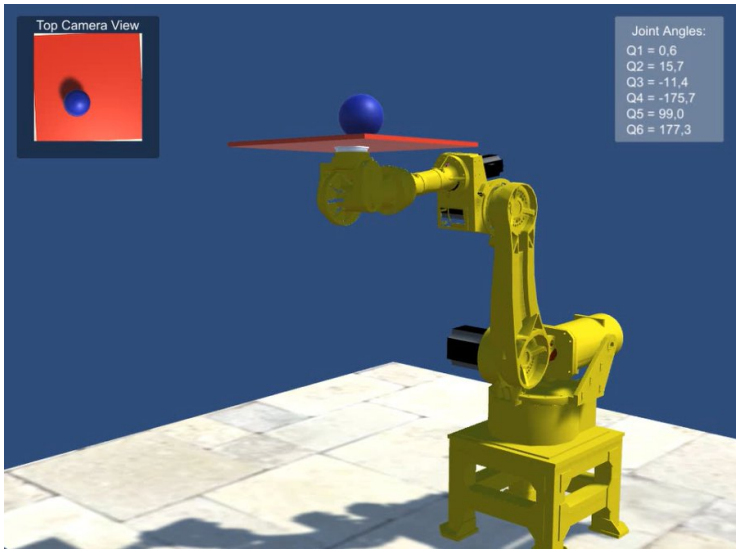
Environment's actions space must be **discrete** !!!

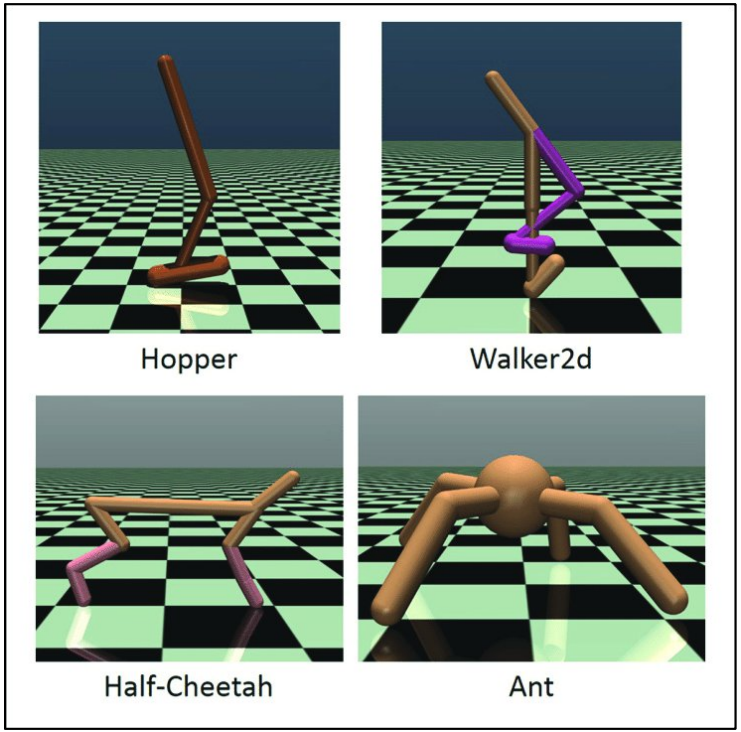


Limit of DQNs



Environment's actions space **cannot be** continuous !!!





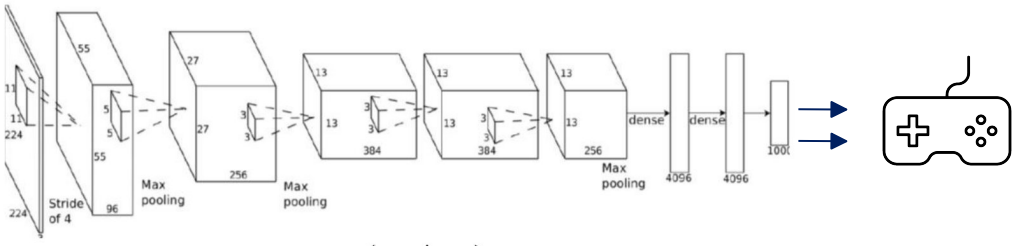
Questions Break

Policy Gradient

FIDLE

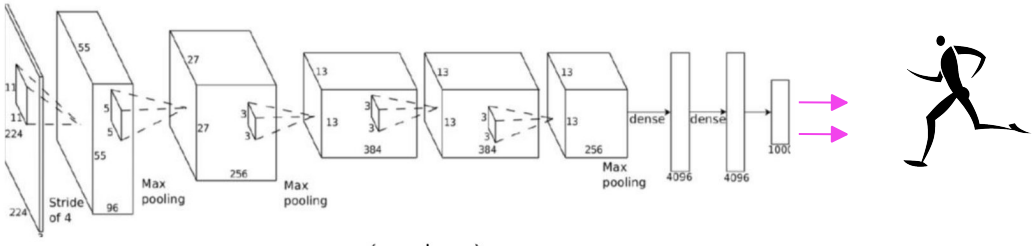
Policy Gradient

=> Action Inference



Discrete control
last activation: Softmax, Sigmoid
Stochastic policy
Probability of taking that action

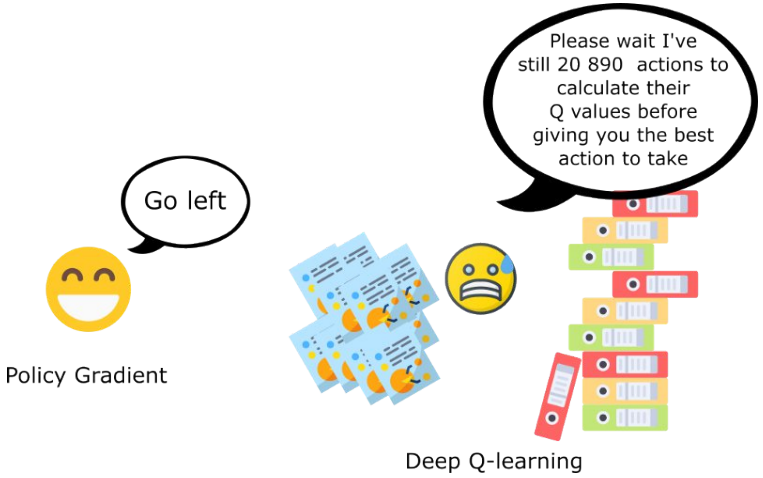
or



Continuous control
last activation : Sigmoid [0;1] or Tanh [-1;1]
Deterministic policy
directly the accurate action value

Policy Gradient

On-policy



Policy gradient **simpler** (straightforward) than Q or V based method
Easy to use for **continuous control & high-dimensional action spaces**

Exploration is few efficient especially for deterministic/continuous PG (no ε-greedy)

~~Loss~~ Objective function :

$$J(\theta) = E_{\tau \sim \pi}[R(\tau)]$$

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots$$

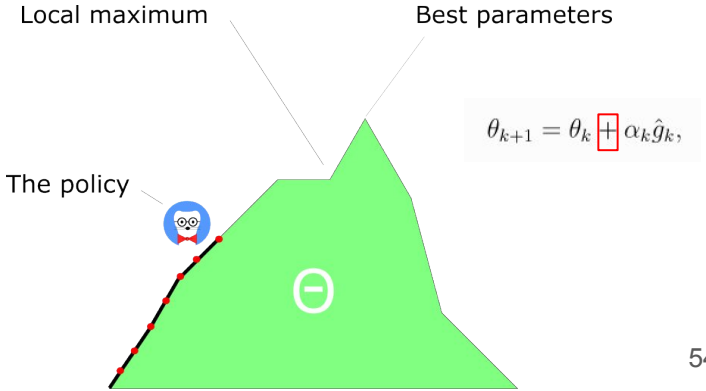
Return: cumulative reward

Gamma: discount rate

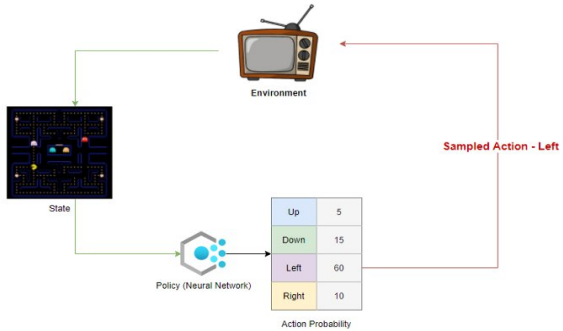
Trajectory (read Tau)
Sequence of states and actions

Gradient Ascent
= negative gradient descent

```
(-obj).backward()
optimizer.step()
```



REINFORCE - Pre-DQN Policy Gradient



Monte-Carlo policy gradient

1. Initialize the policy parameter θ at random.
2. Generate one trajectory on policy π_θ: $S_1, A_1, R_2, S_2, A_2, \ldots, S_T$.
3. For t=1, 2, ... , T:
    1. Estimate the the return G_t;
    2. Update policy parameters: $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)$

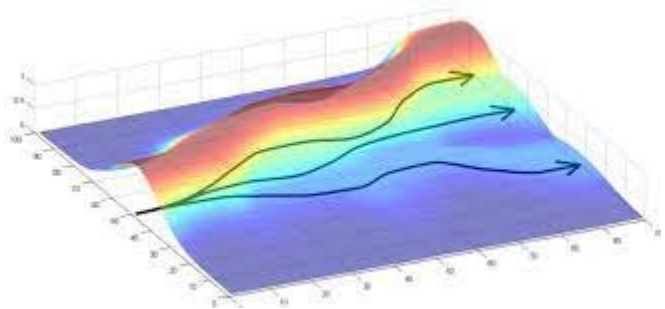
Good stuff is made more likely or more intensively
Bad stuff is made less likely or less intensively

$$Policy\ gradient: E_\pi[\nabla_\theta(log\pi(s,a,\theta))R(\tau)]$$

Policy function — Score function

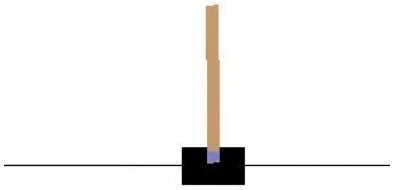
$$Update\ rule: \Delta\theta = \alpha * \nabla_\theta(log\pi(s,a,\theta))R(\tau)$$

Change in parameters — Learning rate



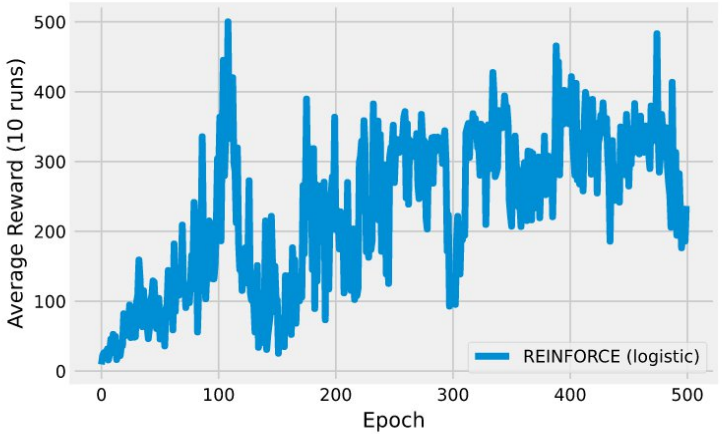
(See Policy Gradient Theorem)

REINFORCE - Pre-DQN Policy Gradient



CartPole
Discrete or Continuous environment

Big issues :
Huge variance & Local maxima !!



REINFORCE with Baseline

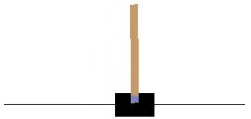
On-policy

= **Vanilla Policy Gradient (VPG)**

reduce the variance of the policy gradient by using an estimate of the **advantage function**, instead of the actual return

$$A_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$$

baseline baseline



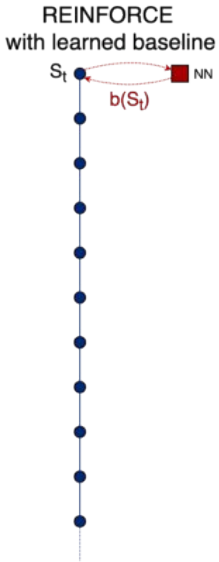
$$L^{PG}(\theta) = E_t[\log \pi_\theta(a_t|s_t) * A_t]$$

Policy Loss

Expected

log probability of taking that action at that state

Advantage if A>0, this action is better than the other action possible at that state

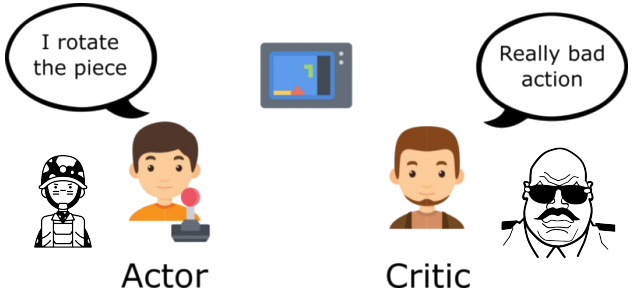


Instead of waiting until the end of the episode as we do in Monte Carlo REINFORCE, **we make an update at each step (TD Learning).**

TD Learning needs a tail !!

VPG = Actor-Critic

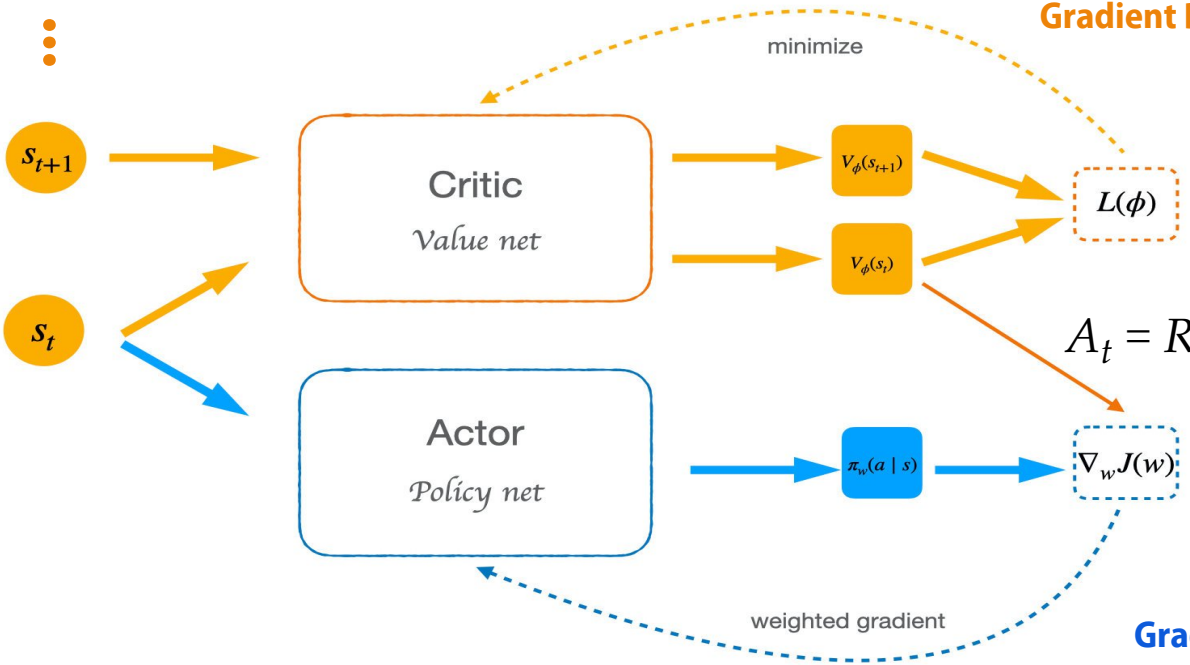
On-policy



TD(n-step)

$$R_{t(n)} = \sum_{l=0}^{n-1} \gamma^l r_{t+l} + \gamma^n V_\phi(s_{t+n})$$

Gradient Descent



$$R_{t(1)} = r_t + \gamma V_\phi(s_{t+1})$$

$$L(\phi) = (R_t - V_\phi(s_t))^2$$

$$A_t = R_t - V_\phi(s_t)$$

$$\nabla_w J(w) \approx \nabla_w \log \pi_w(a|s)\, A_t$$

vector scalar

Gradient Ascent

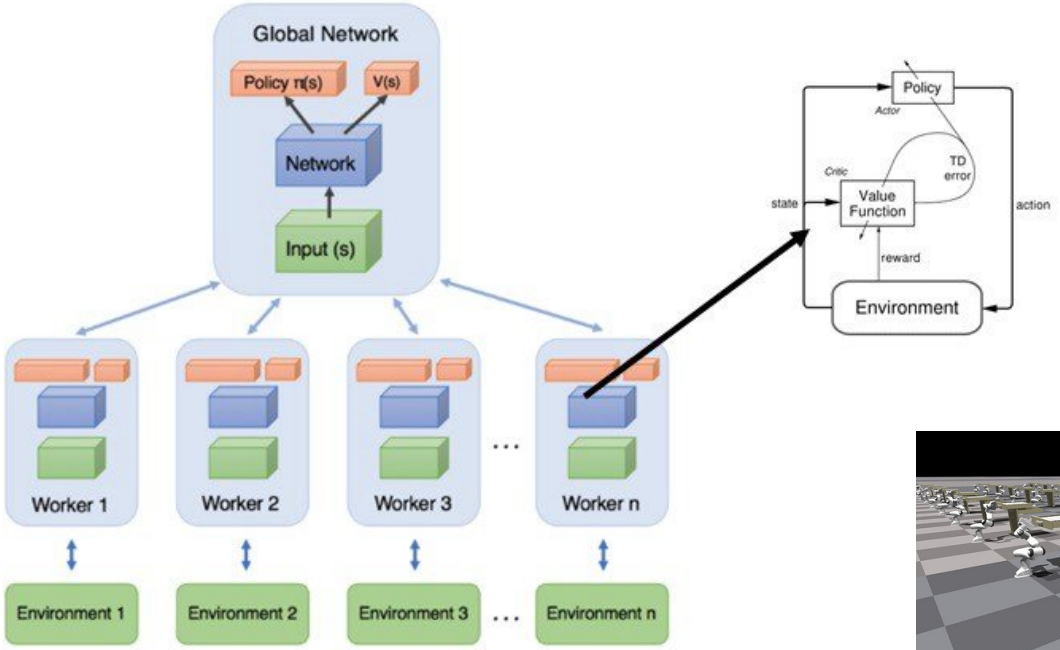
A3C - Asynchronous Advantage Actor-Critic

On-policy

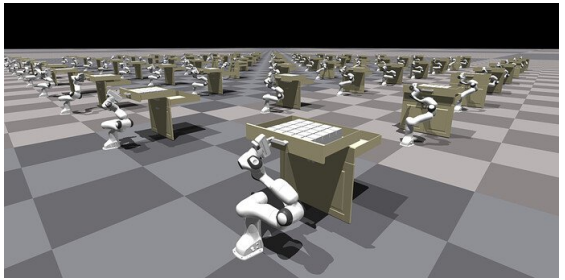
like REINFORCE with Baseline (VPG) :

- **TD(n-step)** Actor Critic

- using **Advantage function**



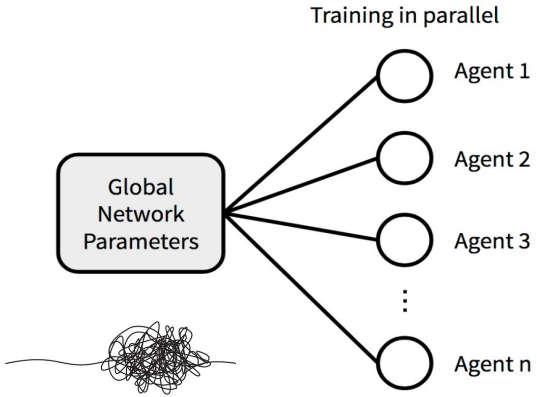
+ Asynchronous **multi-workers for exploration**



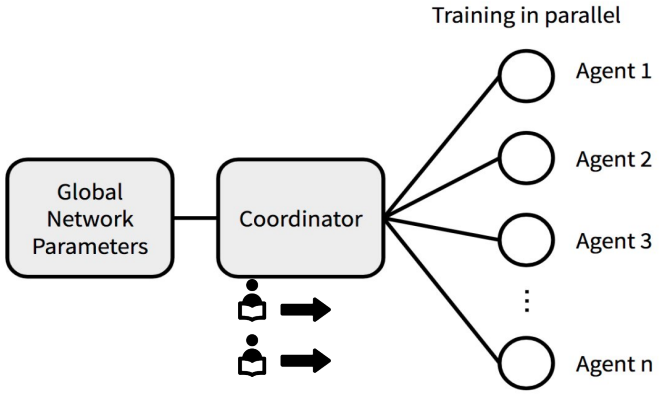
A2C - Synchronous Advantage Actor-Critic

On-policy

- ~~Asynchronous~~ Synchronous **multi-workers for exploration**



A3C (Async)



A2C (Sync)

Sometimes the thread-specific agents would be playing with **policies of different versions.** Update would not be optimal.

The synchronized gradient update keeps **the training more cohesive** and potentially to make **convergence faster**.

Trust Region Policy Optimization - TRPO

On-policy

Line search
(like gradient ascent)

Trust region

Maximize

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\log \pi_\theta(a_t \mid s_t)\hat{A}_t\right].$$

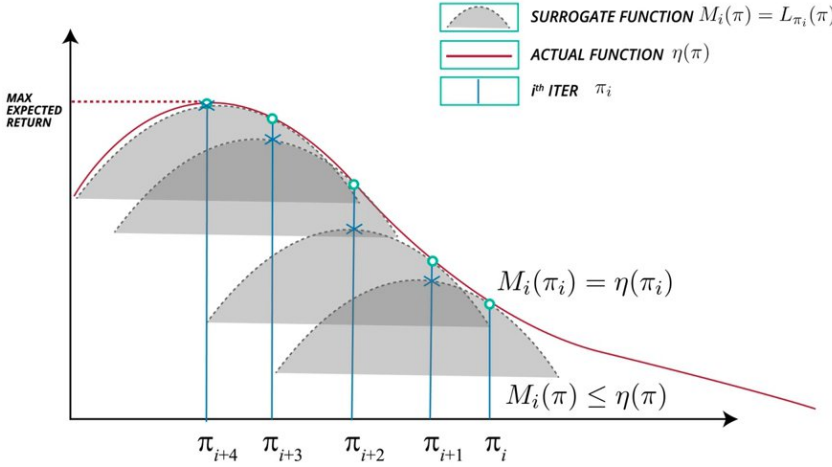
Expected

Policy Loss

log(probabilities) from the output of the policy network

Estimate of the relative value of selected action

"destructively large policy updates."



The Trust Region Constraint

$$\max_{\pi'} \mathcal{L}_\pi(\pi')$$

trust region

$$\text{s.t.} \underset{s\sim d^\pi}{\mathrm{E}}\left[D_{KL}(\pi' \| \pi)[s]\right] \leq \delta$$

kullback-leibler divergence

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

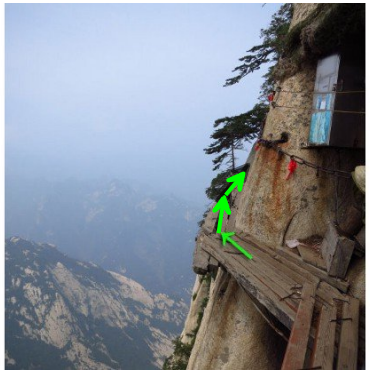
natural policy gradient

Computationally expensive with big model but use conjugate gradient algorithm to approximate the natural policy gradient !!

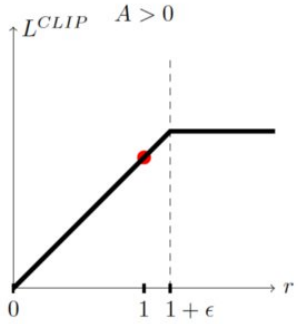
Proximal Policy Optimization - PPO

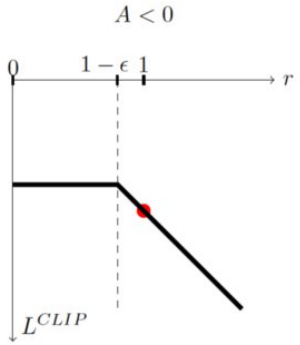
On-policy

Line search



Clipped search





Maximize

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\log \pi_\theta(a_t \mid s_t)\hat{A}_t\right].$$

Expected

Policy Loss

log(probabilities) from the output of the policy network

Estimate of the relative value of selected action

"destructively large policy updates."

The Clipped Surrogate Objective Function

$$L^{PG}(\theta) \approx L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right],$$

clipped PG objective

MSE of value function

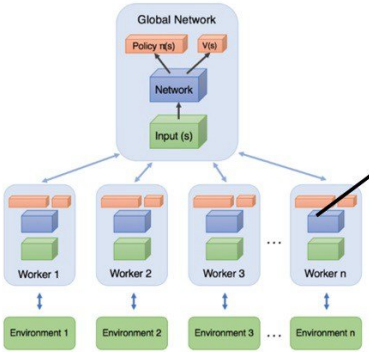
entropy term to ensure enough exploration for many agents

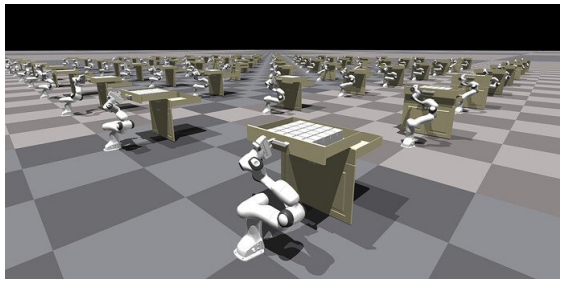
How to Minibatch Policy gradient ?

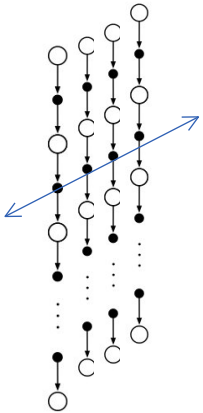
On-policy

Batches on n epochs

Collect set of trajectories by running policy in the different environments.

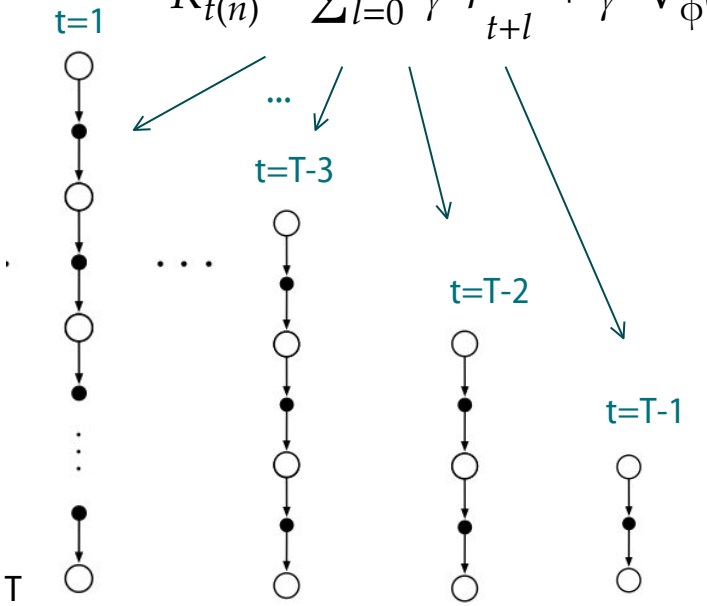






At each timestep in each trajectory, compute the return of each TD(n-step) trajectory candidates :

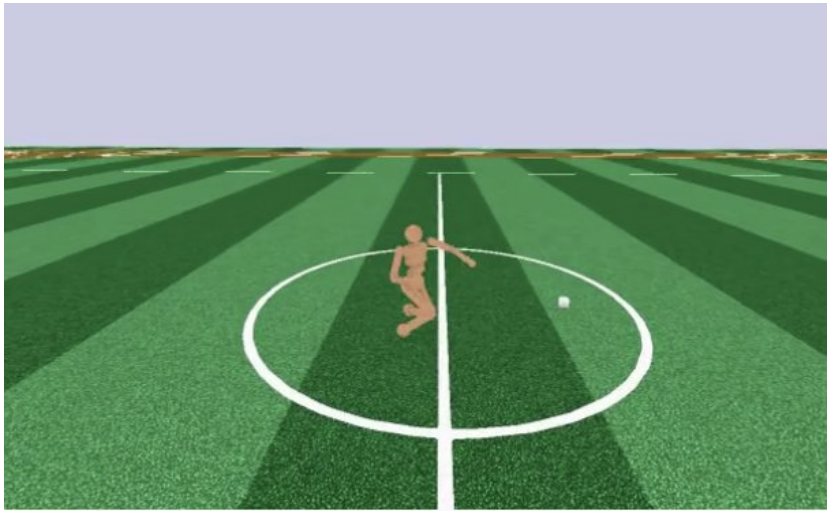
$$R_{t(n)} = \sum_{l=0}^{n-1} \gamma^{l} r_{t+l} + \gamma^{n} V_{\phi}(s_{t+n})$$



PPO Applications

On-policy

"PPO has become the default reinforcement learning algorithm at OpenAI because of its ease of use and good performance. July 20, 2017"



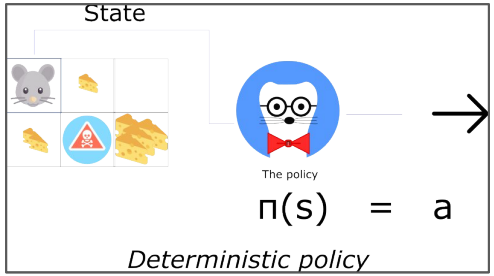
OpenAI Five DOTA2 winners

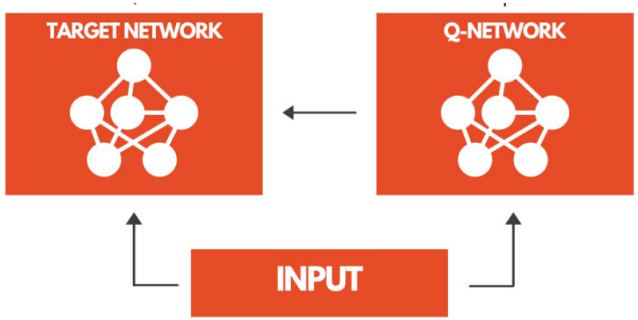
Questions Break

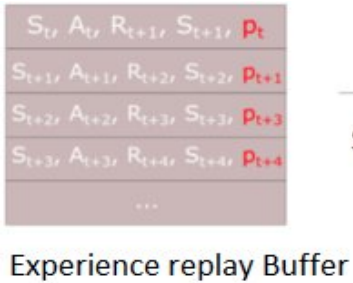
Deep Deterministic Policy Gradient

DDPG is an **off-policy** TD learning algorithm

DDPG can be thought of as being **Deep Q-learning Network for continuous control**

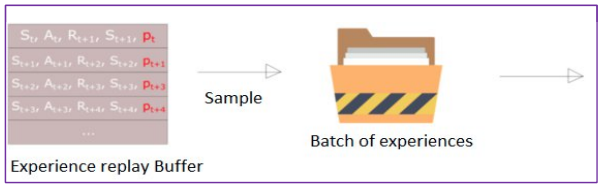


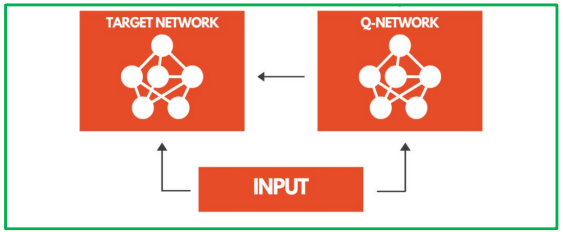


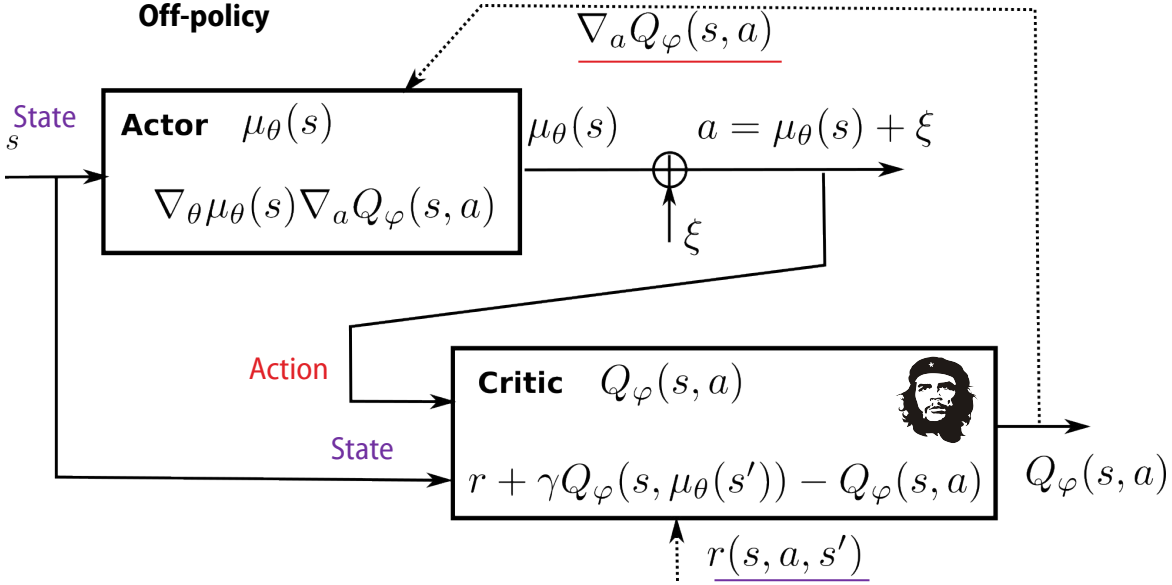


Deep Deterministic Policy Gradient

Off-policy







Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

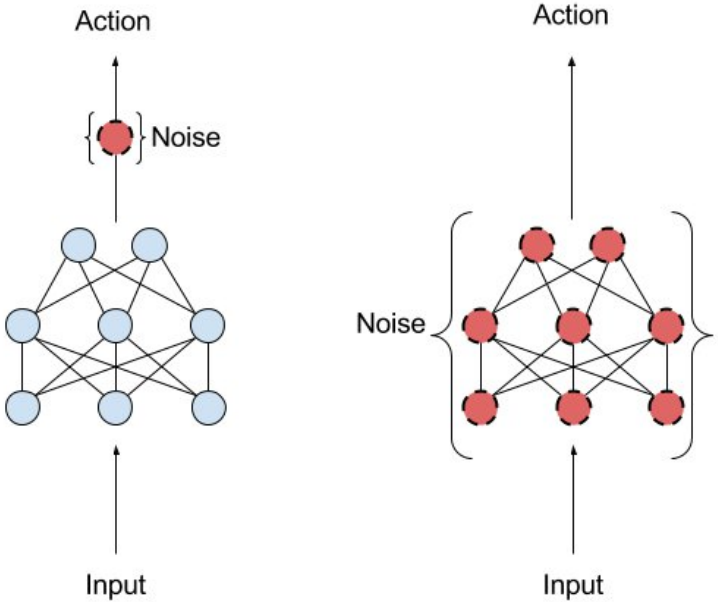
Update the actor policy using the sampled policy gradient:

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

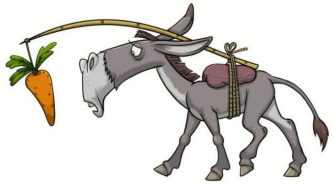
Deep Deterministic Policy Gradient

Noisy exploration

Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise



TD3 : Twin Delayed Deep Deterministic

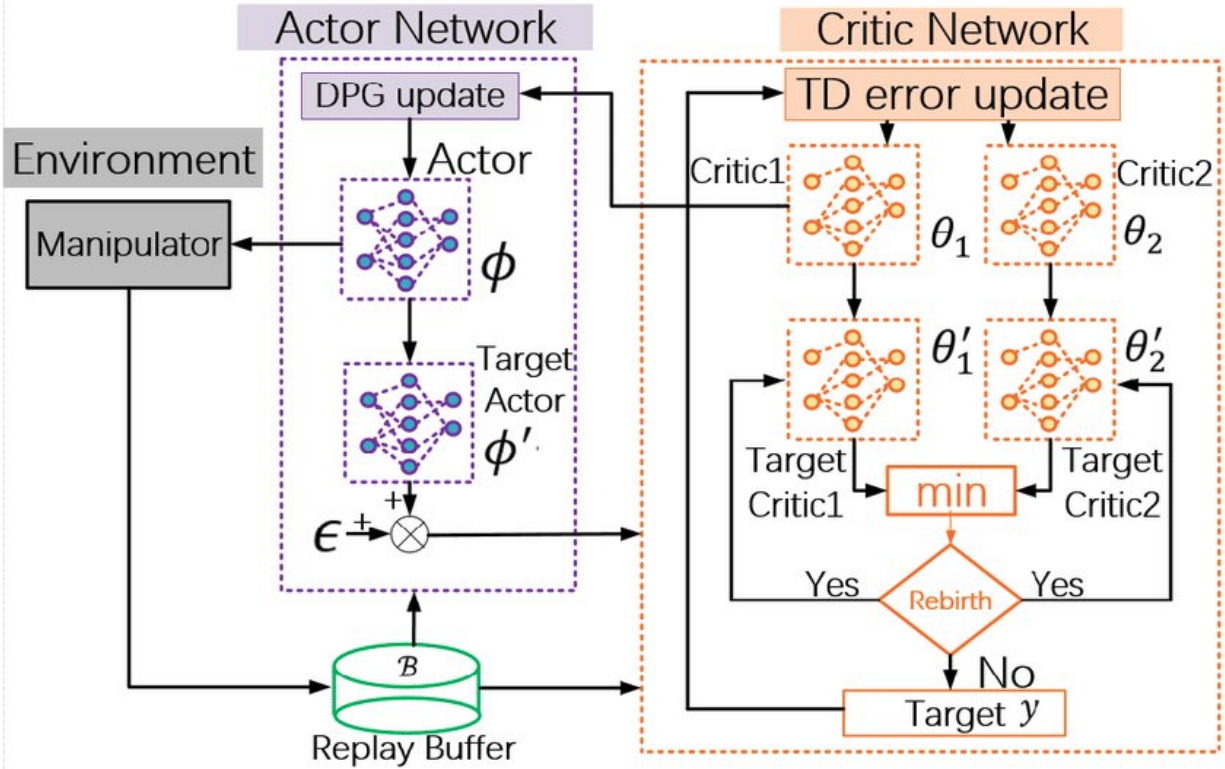


Off-policy

To fix Q value overestimation issue and variance issue : use Twin critic

Double Q-learning

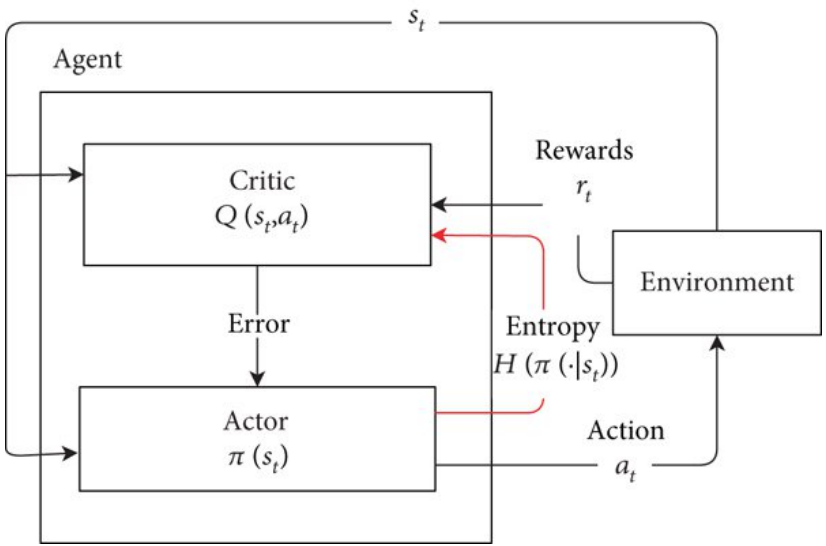
Delayed update of target and policy networks



Soft Actor Critic

incorporates the **entropy measure of the policy** into the reward to encourage exploration

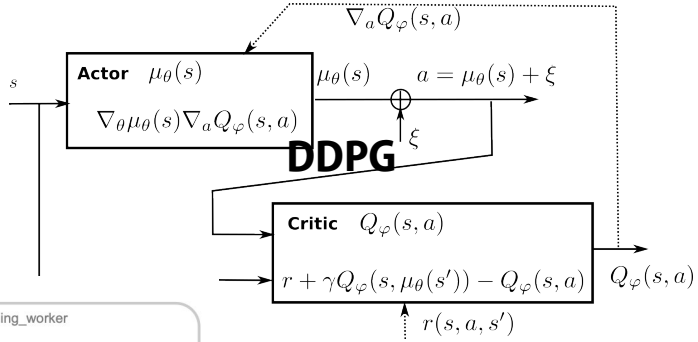
$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\theta}}[r(s_t,a_t) + \alpha\mathcal{H}(\pi_\theta(.\,|s_t))]$$



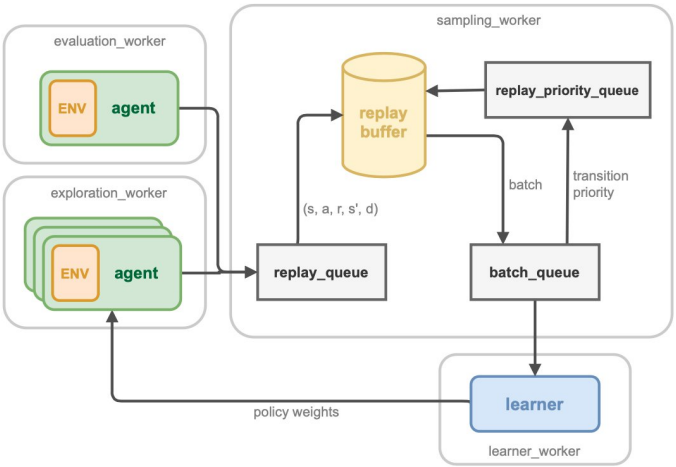
D4PG

Off-policy

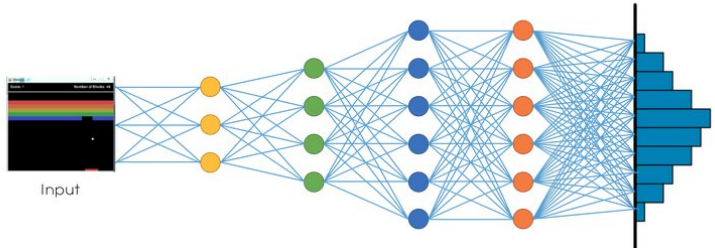
Distributed Distributional Deep Deterministic Policy Gradients



+Multi-Workers

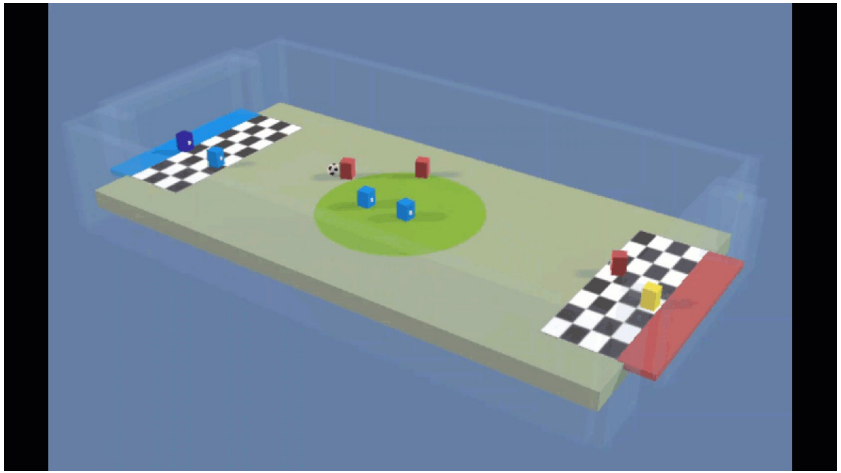


+Distributional RL

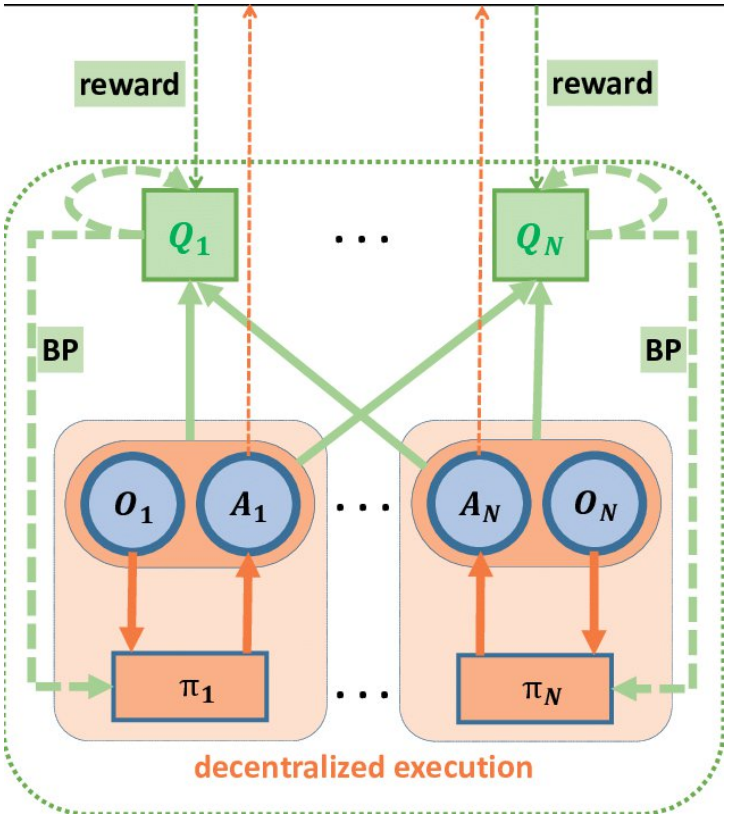


Multi Agent DDPG

Collaborative and/or adversarial

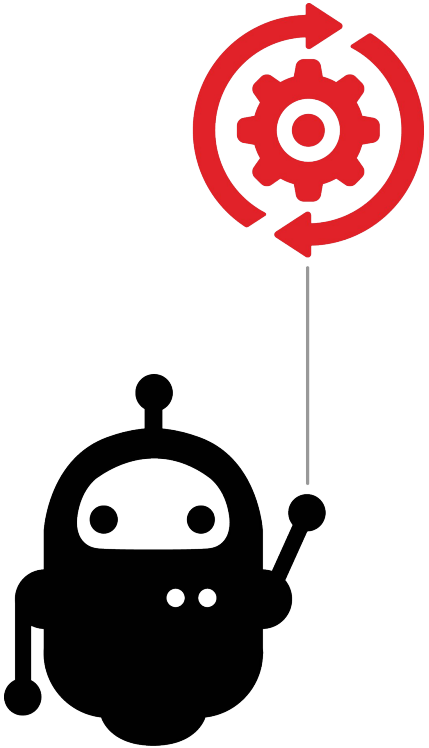


Critics take the actions of every actor

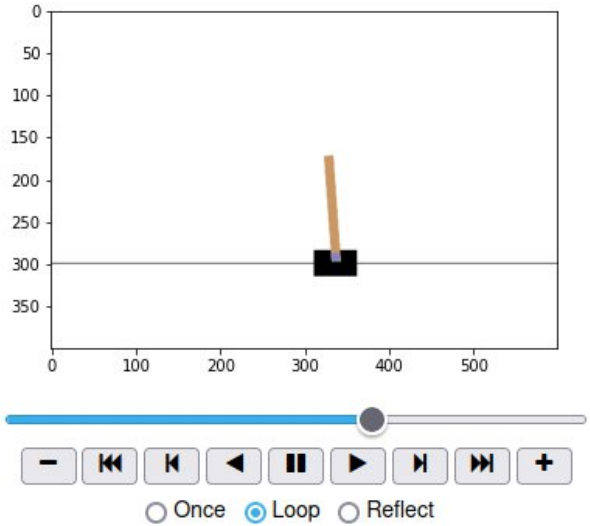


Questions Break

Example : Car Pole



[DRL1] - Solving CartPole with DDQN & Vanilla Policy Gradient



Success & Perspectives

FIDLE

DRL Python libraries - Model Zoo

Dopamine on Tensorflow

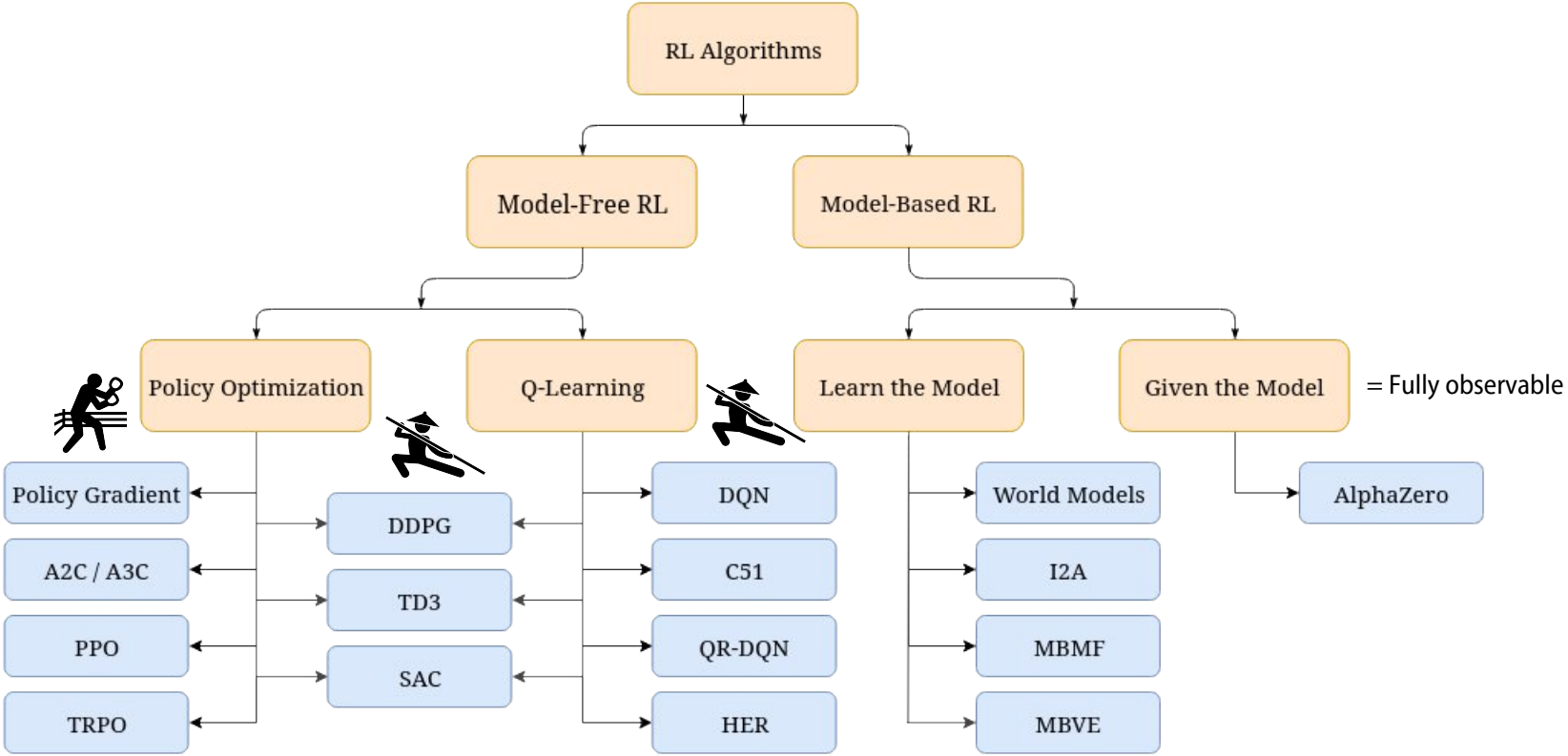
Stable Baseline3 on Pytorch

Which algorithm should I use ?

	Discrete control	Continuous control
Single Process	DQN and Distributional DQN. DQN is usually **slower** to train (regarding wall clock time) but is the **most sample efficient** (because of its replay buffer)	SAC, TD3 and TQC Please use the hyperparameters in the RL zoo for best results
Multiprocessed	PPO and A2C	PPO, TRPO and A2C Please use the hyperparameters in the RL zoo for best results

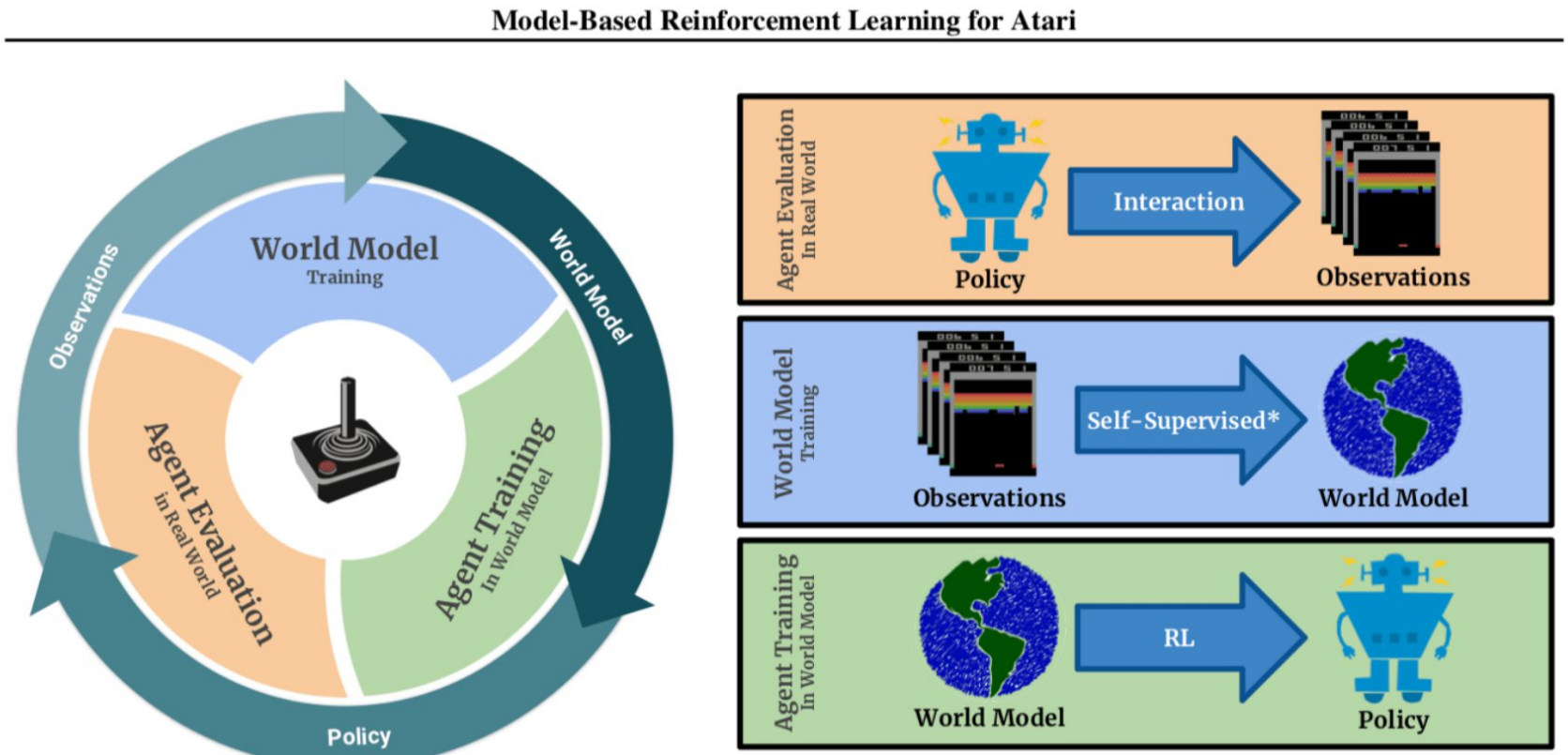
Source: Stable Baseline 3

Kinds of DRL Algorithms



Source : [OpenAI Spinning Up]

Model-Based DRL



Which algorithm should I use ?

Model-based
(100 time steps)

Off-policy
Q-learning
(1 M time steps)

Actor-Critic

On-policy
Policy Gradient
(10 M time steps)

Evolutionary
(100 M time steps)

DDPG & co

PPO & co

Better Sample Efficient
More Computationally Expensive
Experience Replay

DEEPMIND

OpenAI

Less Sample Efficient
Less Computationally Expensive
Multiple Live Experience

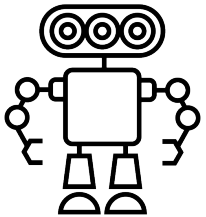
Stochastic
(Distributional RL)

Dynamic
(no Replay)

Strategic

Multi-Agents
Adversarial
Collaborative

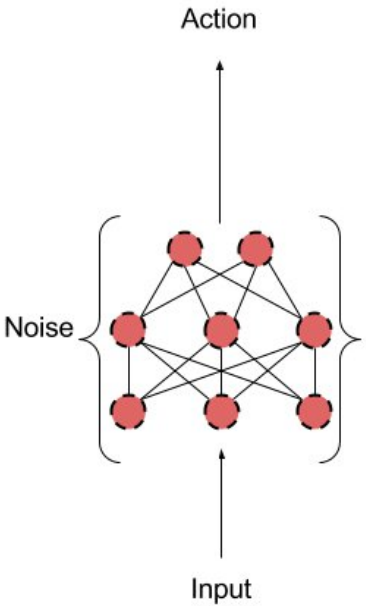
Tactical



Augmented Random Search (2018)

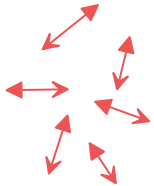
ARS is combination of :

- Evolution Strategies – **derivative free optimization** method, parallelized training
- Natural policy gradients for training linear policies



Exploration in the policy space:

Apply several +/- noises δ



Collect r[+] , r[-]
Update the weights Θ += α(r[+] – r[-]). δ

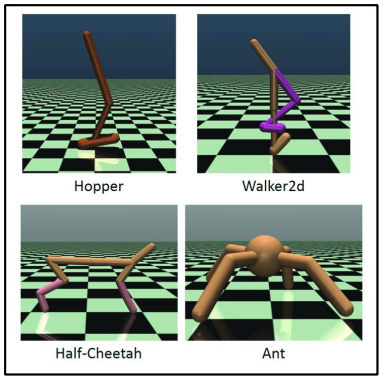
Augmented with:

- Dividing by the Standard Deviation $\boldsymbol{\sigma}_r$,
- Normalizing the States,
- Using top performing directions

Sample Efficient

Computationally Cheap

Outperforms largely PPO, DDPG on MuJoCo locomotion environments !!



Going Forward & Ressources

- **Reinforcement Learning - Book of R. S. Sutton and A. G. Barto**
- **Grokking Deep Reinforcement Learning - Book of M. Morales**
- OpenAI Spinning Up
- Berkeley's Deep Reinforcement Learning course
- More resources

- Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning

Lilian Weng's Blog about DRL

Policy Gradient Algorithms (2018)

…, PPG, ACER, ACTKR, SVPG, IMPALA

Meta Reinforcement Learning (2019)

generalize to new tasks or new environments with memory

Curriculum for Reinforcement Learning (2020)

step by step complexity guided learning

Exploration Strategies in Deep Reinforcement Learning (2020)

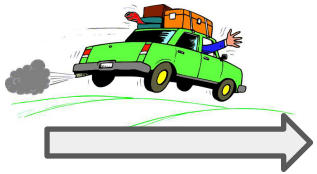
hard environments : rarely provides rewards or have distracting noise

And after 2020 ?

And what ?

After 2020 are we in one **Deep Reinforcement Learning Winter ?**

OpenAI DEEPMIND



????

TRANSFORMERS

ex : **GATO**, AlphaFold, GPT, PALM-E

New trend is **RL fine-tuning** !!

Imitation Learning

For impossible tasks with Reinforcement Learning from scratch !!

Video labeled pre-train then fine-tune with RL !

Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos

Bowen Baker*† bowen@openai.com
Ilge Akkaya*† ilge@openai.com
Peter Zhokhov*† peterz@openai.com
Joost Huizinga*† joost@openai.com
Jie Tang*† jietang@openai.com
Adrien Ecoffet*† adrien@openai.com
Brandon Houghton*† brandon@openai.com
Raul Sampedro*† raulsamg@gmail.com
Jeff Clune*†‡ jclune@gmail.com

jun. 2022

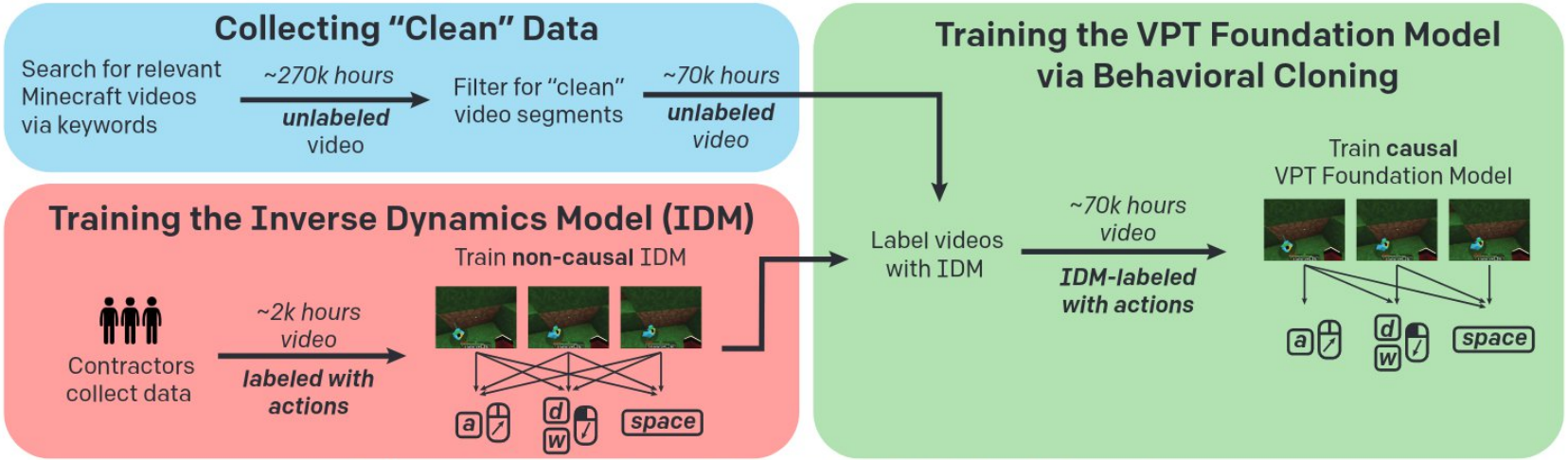


Figure 2: Video Pretraining (VPT) Method Overview.

chat GPT / Instruct GPT - RLHF

Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.

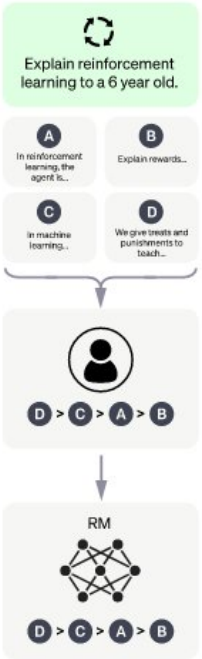
Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

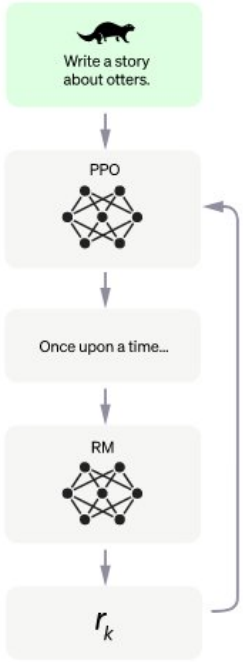
A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

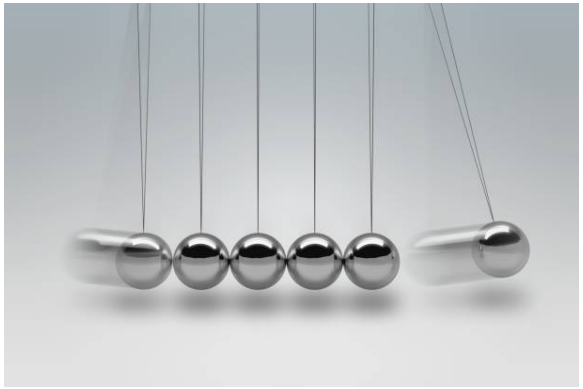
The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

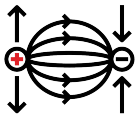


Questions Break

Next, on Fidle :



19 New !



Physics-Informed Neural Networks

PINNS

Jeudi 4 mai,

Episode 19 :

Des neurones pour la physique : les PINNs (New !)

Que sont les Physic-Informed Neural Networks ?
Equilibrage de la Loss pour améliorer l'exactitude
Sampling & autres astuces

Durée : 2h

19 New !

Physics-Informed Neural Networks
PINNS

Jeudi 4 mai,

Episode 19 :

Des neurones pour la physique : les PINNs (New !)

To be continued...

Contact@fidle.cnrs.fr
FIDLE https://fidle.cnrs.fr
YouTube https://fidle.cnrs.fr/youtube

FIDLE