Deep Reinforcement Learning

Pr. Fabien MOUTARDE
Center for Robotics
Mines Paris
PSL Université Paris

Fabien.Moutarde@minesparis.psl.eu
http://people.minesparis.psl.eu/fabien.moutarde

Outline

- <u>Introduction</u>
- Basics of Reinforcement Learning (RL)
- RL as a Markov Decision Process (MDP)
  ➔ RL algos for tabular policies
- Deep RL algos
  - Policy Gradient: REINFORCE, TRPO, PPO
  - Deep Q-learning: DQN
  - Actor-Critic: A3C, SAC, DDPG
- Example of DRL application:
  learning to drive from vision in urban area

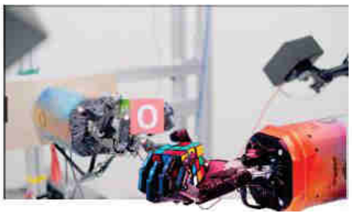
Recent striking successes of Reinforcement Learning

DM - Atari DQN
(2013, 2015)

DM - AlphaGo
(2016, 2017)

DM - AlphaZero
(2018)

OpenAI - Dexterity
(2018, 2019)



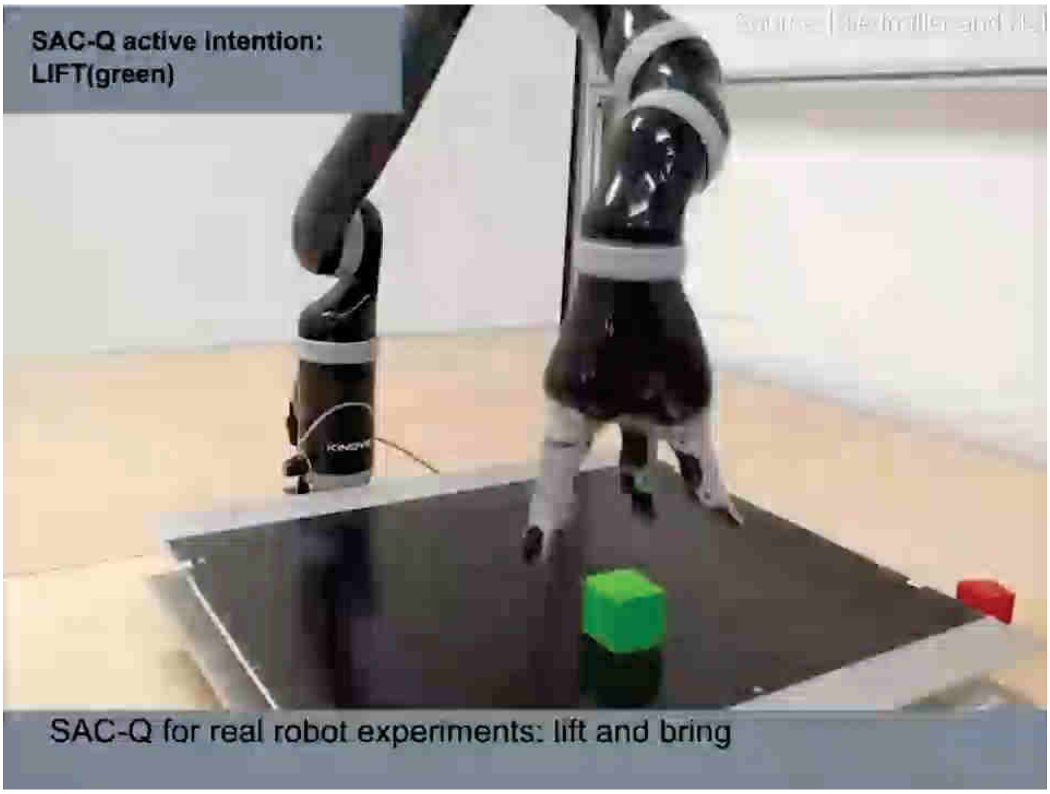
OpenAI - Five
(Dota 2 - 2019)

DM - AlphaStar
(StarCraft II -2019)

RL for Training Robots

Combination of Learning from Demonstration (LfD) and Reinforcement Learning
[*Robot Motor Skill Coordination with EM-based Reinforcement Learning,* Kormushev et al. (IROS'2010)]

Learning complex robotic behavior with Deep Reinforcement Learning



Work by Google DeepMind
[Learning by Playing Solving Sparse Reward Tasks from Scratch, Riedmiller et al. (ICML'2018)]

RL for Locomotion Learning

Work by Google DeepMind
[Emergence of Locomotion Behaviours in Rich Environments, Heess et al. 2017]

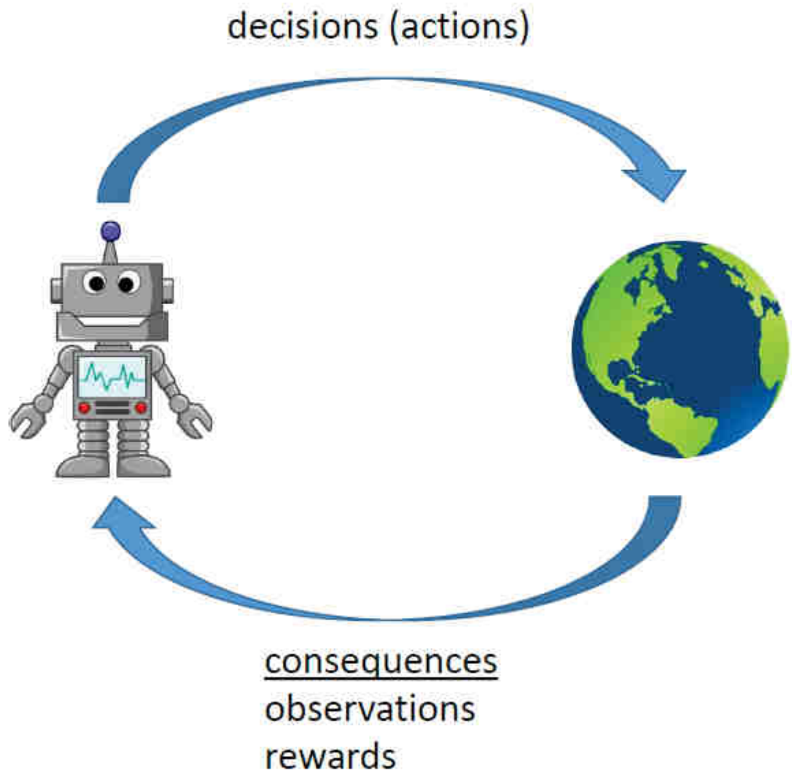
RL for Automated Driving: towards intelligent visual servoing

Preliminary experiment conducted by PhD student Marin Toromanoff
(CIFRE Valeo/MINES_Paris)

Outline

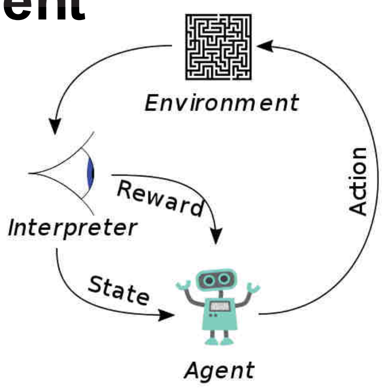
- Introduction
- <u>Basics of Reinforcement Learning (RL)</u>
- RL as a Markov Decision Process (MDP)
  ➔ RL algos for tabular policies
- Deep RL algos
  – Policy Gradient: REINFORCE, TRPO, PPO
  – Deep Q-learning: DQN
  – Actor-Critic: A3C, SAC, DDPG
- Example of DRL application:
  learning to drive from vision in urban area

What is Reinforcement Learning?



Reinforcement Learning (RL)

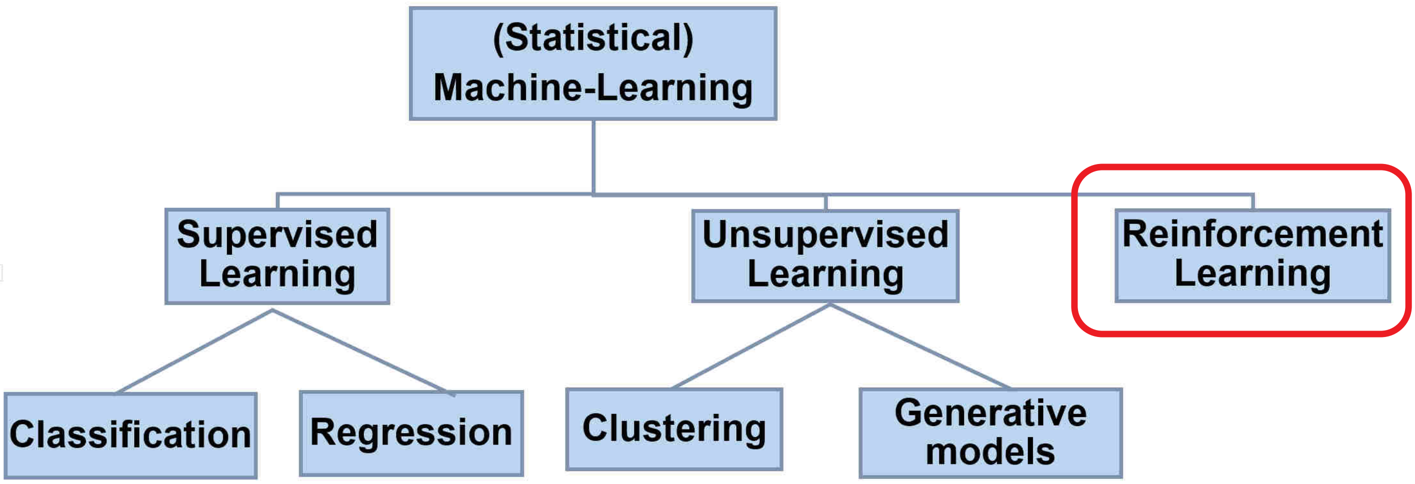
- **GOAL:**
  learn a BEHAVIOUR, i.e. being able to *make sequential decisions* that realizes a goal task
- **HOW?**
  By interaction with the environment



- ***Reward hypothesis:***
  Any goal can be formalized as the outcome of maximizing a cumulative reward

MINES PARIS | PSL

Reinforcement Learning in Machine-Learning typology

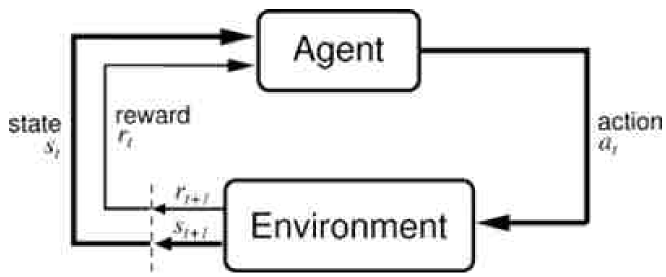


<u>Specificities of Reinforcement Learning:</u>

- No supervision, only a **reward** signal
- Feedback can be delayed, not instantaneous
- Time matters
- Earlier decisions affect later interactions

MINES PARIS | PSL

Principle of Reinforcement Learning (RL)



Goal: find a "policy" $a_t=\pi(s_t)$ that maximizes the *cumulated* reward (="return") $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \gamma \in [0,1[$

<u>Deep</u> Reinforcement Learning (<u>DRL</u>) if Deep NeuralNet used as model (for policy and/or its "value"): DQN, Actor-Critic A3C, etc

Key elements of RL

- **STATE (of environment)**
  - Fully *vs* Partially observable
  - Discrete *vs* Continuous
- **POLICY**
  - Deterministic: $a=\pi(s)$
  - *vs* Stochastic: conditional probability $\pi(a|s)$
- **ENVIRONMENT**
  - With/without known MODEL giving s_{t+1} = model(s_t,a_t)
  - Stochastic vs Deterministic
- **REWARD**
  - Scalar
  - Must be hand-crafted so that
    Max(cumulated_reward) $\Leftrightarrow$ goal-task perfectly performed

Value-function and Q-function

- **State-value of a policy** = expected cumulated reward if applying policy π starting from a given state s

$$V_\pi(s) = \mathbb{E}_\pi[R_t|s_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{T} \gamma^t r_{t+k}|s_t = s]$$

- **Action-value (Q-function) of a policy** = expected cumulated reward if applying policy π after taking action a when in state s

$$Q_\pi(s,a) = \mathbb{E}_\pi[R_t|s_t = s, a_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{T} \gamma^t r_{t+k}|s_t = s, a_t = a]$$

Note that $V_\pi(s) = Q_\pi(s\,,\,\pi(s)\,)$ and
$Q_\pi(s\,,a) = \Sigma_{s'}\, p(s'\,|\,s\,,a)\,[\,r(s\,,a) + \gamma\, V_\pi\,(s'\,)]$

MINES PARIS | PSL

Optimal policy and values

A policy π_2 is better than another policy π_1 iff for all states s, $V\pi_2(s) \geq V\pi_1(s)$

A <u>policy π_* is *optimal*</u> iff better than all others

$$\forall \pi, \forall s \in S, V_{\pi^*}(s) \geqslant V_\pi(s)$$

➔ <u>Optimal state-value and action-value functions</u>

$$V_*(s) = \max_\pi(V_\pi(s))$$
$$Q_*(s, a) = \max_\pi(Q_\pi(s, a))$$

MINES PARIS | PSL

Bellman equations (deterministic case)

$$V_\pi(s_t) = E_\pi(r_{t+1} + \gamma V_\pi(s_{t+1}) \mid s_t = s)$$
$$= r(s_t, \pi(s_t)) + \gamma V_\pi(s_{t+1})$$

The state-value function V, and action-value Q-function, can be recursively estimated from their <u>*future*</u> values

$$Q_\pi(s_t, a) = E_\pi(r_{t+1} + \gamma V_\pi(s_{t+1}) \mid s_t = s, a_t = a)$$
$$= r(s_t, a) + \gamma V_\pi(s_{t+1}) = r(s_t, a) + \gamma Q_\pi(s_{t+1}, \pi(s_{t+1}))$$

<u>Bellman *optimality* equations:</u>

$$\left.\begin{aligned} V^*(s) &= \max_a(r(s,a) + \gamma V^*(s')) \\ Q^*(s,a) &= r(s,a) + \gamma \max_{a'} Q^*(s',a') \end{aligned}\right\} \text{where } s' = \text{state after action } a \text{ taken in state } s$$

Bellman equations (stochastic case)

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a) + \gamma V_\pi(s')]$$

The state-value function V, and action-value Q-function, can be recursively estimated from their *future* values

$$Q^\pi(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[R(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s',a') \right]$$

Bellman *optimality* equations:

$$V^*(s) = \max_a (\Sigma_{s'} p(s'|s,a)[r(s,a) + \gamma V^*(s')])$$

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right]$$

Approaches for RL

- **Policy-based RL**

  Search *directly* for the optimal policy π*
  (= the policy achieving maximum cumulated reward)

- **Value-based RL**

  Estimate first the maximal state-action value function $Q^*(s,a)$ and then apply $\pi^*(s) = \mathrm{argMax}_a(Q^*(s,a))$

- **Model-based RL**

  Build (or use) a model of the environment $s_{t+1} = m(s_t, a_t)$ then choose actions by planning (e.g. look-ahead)

Outline

- Introduction
- Basics of Reinforcement Learning (RL)
- RL as a Markov Decision Process (MDP)
  ➔ RL algos for tabular policies
- Deep RL algos
  - Policy Gradient: REINFORCE, TRPO, PPO
  - Deep Q-learning: DQN
  - Actor-Critic: A3C, SAC, DDPG
- Example of DRL application:
  learning to drive from vision in urban area

RL as a Markov Decision Process

Finite RL problems can be mathematically formalized as a Markov Decision Process (MDP), i.e. a < S, A, P, R > tuple where

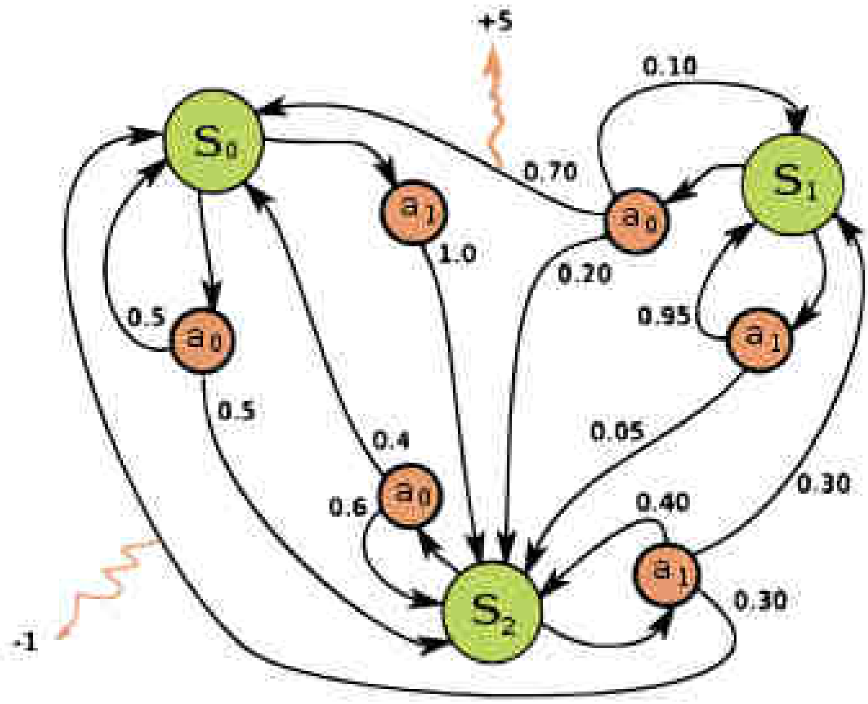
- S = Finite set of states
- A = Finite set of actions
- P = Transition Probabilities (Markov property):

$$\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

- R = Reward function:

$$\mathcal{R}_{s}^{a} = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

Markov Decision Process (MDP)



Finding optimal policy: Dynamic Programming

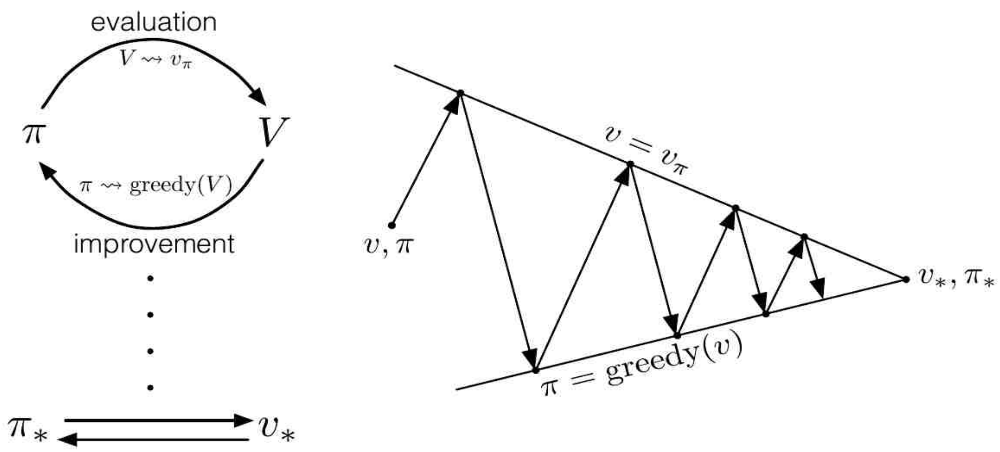
Policy Iteration

1. **Given an initial policy π, <u>policy evaluation</u> by *iterating Bellman equation*:**

   $$V_{k+1}(s) = \Sigma_a \pi(a|s) \Sigma_{s'} p(s' \mid s, a) \left[r(s, a) + \gamma V_k(s') \right]$$

   → converges to fixed point $V_\pi(s)$

2. **<u>Improve policy</u> greedily: $\pi'(s) = \mathrm{argMax}_a(Q_\pi(s, a))$**



Value iteration

- Drawback of policy iteration = computation cost, due to nested iterations for policy evaluation

➔ directly estimate *optimal* state-value function with 1 sweep of states by iterating the *Bellman* *optimality* *equation*:

$$V_{k+1}(s) = \max_a\left(\Sigma_{s'}\, p(s' \mid s, a)\, [\, r(s, a) + \gamma\, V_k(s')]\, \right)$$

→ converges to fixed point $V^*(s)$

- Then, deduce optimal policy from:

$$\pi^*(s) = \mathrm{argMax}_a(\, Q^*(s, a)\,)$$

Temporal Difference (TD) learning

- Faster algo for *estimating* V_π of a policy
- Idea: instead of waiting estimation of return (= final cumulated reward), update $V_\pi(s)$ *at every step during episods*, until *ordinary* Bellman equation becomes true
- Run episodes of policy π
  - For each episod, at every step, use $a_t=\pi(s_t)$ to observe s_{t+1} and r_{t+1}, then update V_π by:

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

TD target

SARSA

- **Acronym for State Action Reward State Action**
- **On-policy TD-learning of Q_π:**

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[\underbrace{\boxed{r_{t+1} + \gamma Q(S_{t+1}, A_{t+1})}}_{\text{TD target}} - Q(S_t, A_t)]$$

Sarsa: An on-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

- **Policy: greedy from current Q**

epsilon-greedy policy

$$\pi(s) = \begin{cases} \texttt{argMax}_a(\ Q(s,a)\ ) \text{ with proba } 1-\varepsilon \\ \text{random with proba } \varepsilon \end{cases}$$

- **The random part allows to maintain exploration**
- **Used during several RL training approaches**

Q-learning

- ***Off-policy*** **TD learning of Q*, by using as the *optimality* Bellman equation as target:**

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[\boxed{r_{t+1} + \gamma \max_a(Q(S_{t+1}, a))} - Q(S_t, A_t)]$$

TD target

Q-learning: An off-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$
  until S is terminal

Final policy π^* = greedy(Q*)

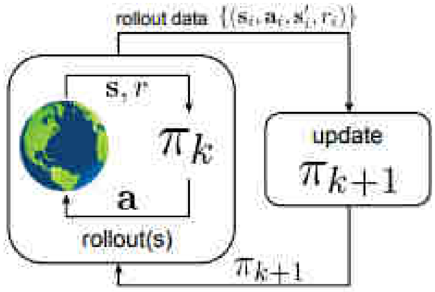
Summary of main *tabular* RL algorithms

Type	Algo name	Based on	Episods
Policy-based	Policy Iteration	Dynamic Programming	ON-policy
Value-based	Value Iteration	Dynamic Programming	OFF-policy
	SARSA	TD-learning	ON-policy
	Q-learning	TD-learning	OFF-policy

On-policy vs Off-policy

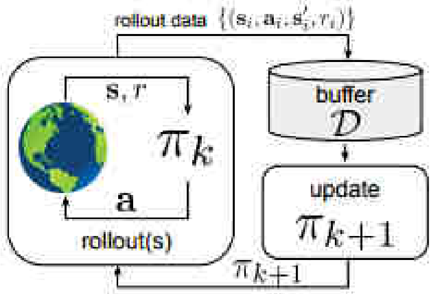
<u>On-policy:</u> training episods are generated with policy being learnt

- **Interaction trajectories (s0,a0,r0, s1,a1,r1, …) used for only ONE update of the target policy ➔ *less efficient***
- **BUT training generally *more stable***



<u>Off-policy:</u> training episods can be generated with other policies

- **Easier to explore better**
- ***More sample-efficient* (episods can be used several times)**
- **BUT training can be unstable**



Curse of dimensionality

$Q(s,a)$	a_1	a_2	$\dots$	$a_{\|\mathcal{A}\|}$
s_1	$Q(s_1,a_1)$	$Q(s_1,a_2)$		
s_2	$Q(s_2,a_1)$	$Q(s_2,a_2)$		
$\dots$				
$s_{\|\mathcal{S}\|}$				

➔ Instead of tabular, use <u>*parameterized function*</u> form for V, Q, and π

Outline

- Introduction
- Basics of Reinforcement Learning (RL)
- RL as a Markov Decision Process (MDP)
  ➔ RL algos for tabular policies
- <u>Deep RL algos</u>
  – <u>Policy Gradient: REINFORCE, TRPO, PPO</u>
  – Deep Q-learning: DQN
  – Actor-Critic: A3C, SAC, DDPG
- Example of DRL application:
  learning to drive from vision in urban area

Deep Reinforcement Learning (DRL)

- <u>DRL = DL + RL</u>
  = use (Deep) Neural Network as
  parameterized function for π and/or V, Q
- Learn using gradient-based optimization
- Possibility of image-based policy,
  i.e. observed state = image(s), by using
  Convolutional Network

3 families of DRL algorithms

- **Policy-based** $\pi_\theta \approx \pi^*$
  optimize a parameterized policy

- **Value-based** $Q(s, a, \theta) \simeq Q^{\pi^*}(s, a)$
  find the optimal (parameterized) Q-value

Model-free
i.e. learn exclusively by trial-and-error

- **Model-based** $m(s_t, a_t, \theta') \approx s_{t+1}, r_{t+1}$
  → choose actions by planning

Policy gradient

- **Principle:** **find** argMax$_\theta$ $\mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$

  by gradient ASCENT on θ of $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$
  $$= \int_\tau r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

  where r(τ) = cumulated reward on trajectory
  $$\tau = (s_0, a_0, r_0, s_1, \ldots)$$

➔ Need to compute $\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$

REINFORCE algorithm

- **The « vanilla » Policy Gradient**

- **Trick to compute $\nabla_\theta p(\tau;\theta)$:**

$$\nabla_\theta p(\tau;\theta) = p(\tau;\theta)\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$$

$$\Rightarrow \nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

REINFORCE algorithm (2)

Computation of $\nabla_\theta \log p(\tau;\theta)]$:

We have: $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)$

Thus: $\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t,a_t) + \log \pi_\theta(a_t|s_t)$

And when differentiating: $\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Therefore when sampling a trajectory τ,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$
$$\approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

REINFORCE algorithm:

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

Return along trajectory estimated by Monte-Carlo random rollouts

Drawbacks of REINFORCE

- **High variance of gradient → slow convergence**
- **Rewards are relative, not absolute**

  ➔ REINFORCE *with «baseline»:*
  Substract a reward bias to reduce variance, and push-up only trajectories with high rewards
  + discount rewards along trajectories

- **Need to avoid large gradient steps**

  Step too far → bad policy
  → Next batch: collected under bad policy
  → Can't recover, collapse in performance!

- **Sample-inefficient**

Policy gradient variants to mitigate drawbacks

➔ TRPO (Trust Region Policy Optimization):
Add KL divergence constraint to *avoid too large policy updates*

➔ PPO (Proximal Policy Optimization):
Add KL div. penalty to *reduce big policy updates*

+ Another way to reduce gradient variance and improve sample-efficience = *estimate cumulated rewards with a parameterized function,* rather than by Monte Carlo (random rollouts) → Actor-Critic

Outline

- Introduction
- Basics of Reinforcement Learning (RL)
- RL as a Markov Decision Process (MDP)
  ➔ RL algos for tabular policies
- <u>Deep RL algos</u>
  - Policy Gradient: REINFORCE, TRPO, PPO
  - <u>Deep Q-learning: DQN</u>
  - Actor-Critic: A3C, SAC, DDPG
- Example of DRL application:
  learning to drive from vision in urban area

Deep Q-learning (DQN, etc)

<u>Value-based DRL:</u>
<u>Q-learning with parameterized Q-function</u>

- In standard Q-learning:

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

converges to Q*

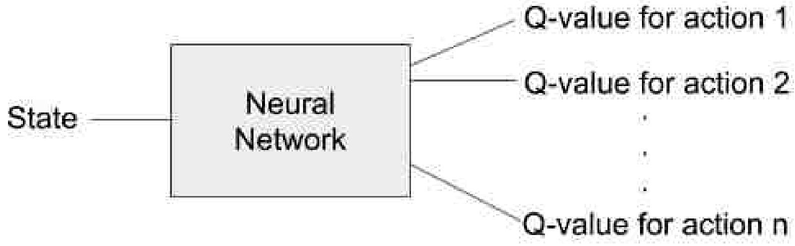
➔ $\boxed{\text{Loss for learning parameterized } Q^*_\theta \text{ with gradient: } L(s_t,a_t,r_{t+1},s_{t+1},\theta) = (\underbrace{r_{t+1}+\gamma \max_a Q(s_{t+1},a,\theta)}_{target} - Q(s_t,a_t,\theta))^2}$

- Once Q^*_θ trained, optimal policy:

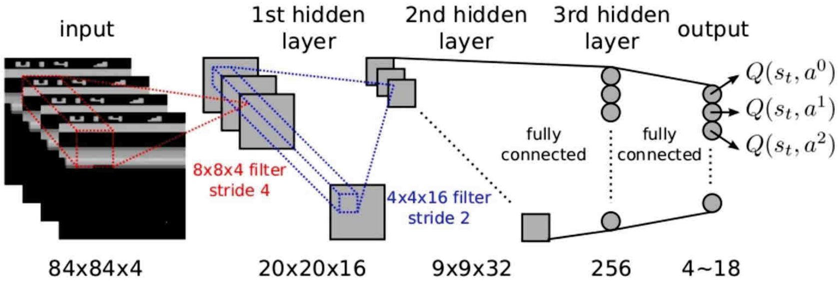
$$\pi^*(s) = arg \max_a Q_{\pi^*}(s,a)$$

Q-function Neural Network

Use one output per possible action
(rather than using action as 2nd input)



If state = image(s), use Convolutional Network



Stability issues of Deep Q-learning

Naive Q-learning oscillates or diverges with neural nets

1. Data is sequential
   - Successive samples are correlated, non-iid
2. Policy changes rapidly with slight changes to Q-values
   - Policy may oscillate
   - Distribution of data can swing from one extreme to another
3. Scale of rewards and Q-values is unknown
   - Naive Q-learning gradients can be large unstable when backpropagated

http://videolectures.net/rldm2015_silver_reinforcement_learning/

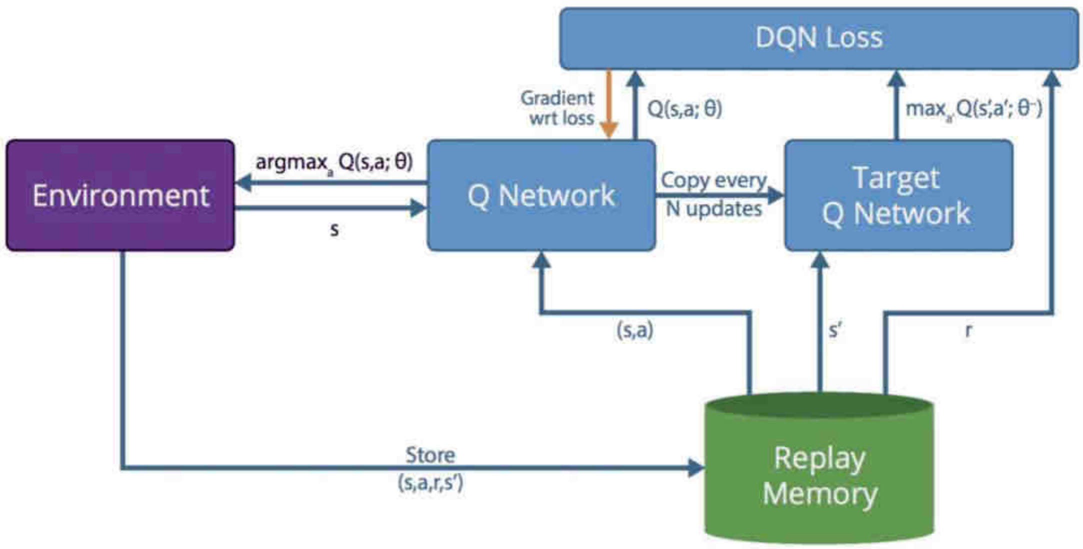
DQN tricks that make Deep Q learning work

DQN provides a stable solution to deep value-based RL

1. Use experience replay
   - Break correlations in data, bring us back to iid setting
   - Learn from all past policies
   - Using off-policy Q-learning
2. Freeze target Q-network
   - Avoid oscillations
   - Break correlations between Q-network and target
3. Clip rewards or normalize network adaptively to sensible range
   - Robust gradients

http://videolectures.net/rldm2015_silver_reinforcement_learning/

Double DQN and (Prioritized) Experience Replay



+ Prioritized Replay: select in replay memory with higher probability the transitions with larger TD $\left(r_t + \gamma \max_a Q_{w_{target}}(x_{t+1}, a) - Q_w(x_t, a_t)\right)$

Pros and Cons of DQN

- **Among most sample-efficient DRL algo**
- **Only one Neural Network to train**
  (≠ Actor-Critic)
- **But *limited to discrete output***

DQN improvement variants

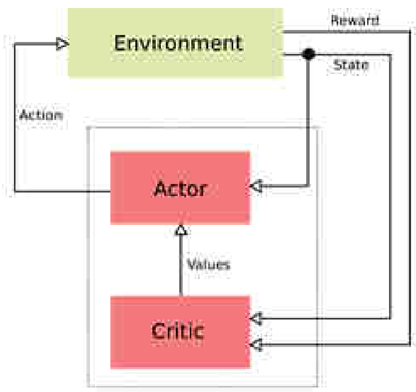
- **Double DQN**
- **Duelling DQN**
- **Rainbow**
- **IQN (Implicit Quantile Network)**
- **…**

Outline

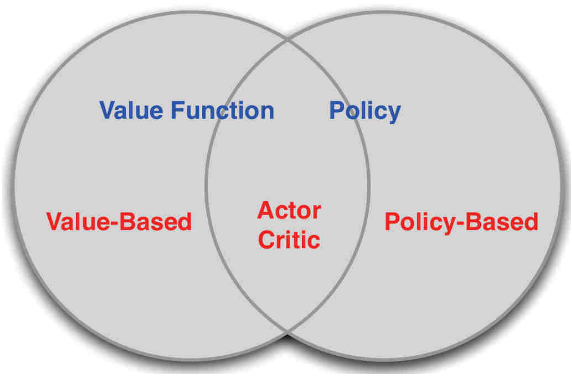
- Introduction
- Basics of Reinforcement Learning (RL)
- RL as a Markov Decision Process (MDP)
  ➔ RL algos for tabular policies
- <u>Deep RL algos</u>
  - Policy Gradient: REINFORCE, TRPO, PPO
  - Deep Q-learning: DQN
  - <u>Actor-Critic: A3C, SAC, DDPG</u>
- Example of DRL application:
  learning to drive from vision in urban area

Actor-Critic

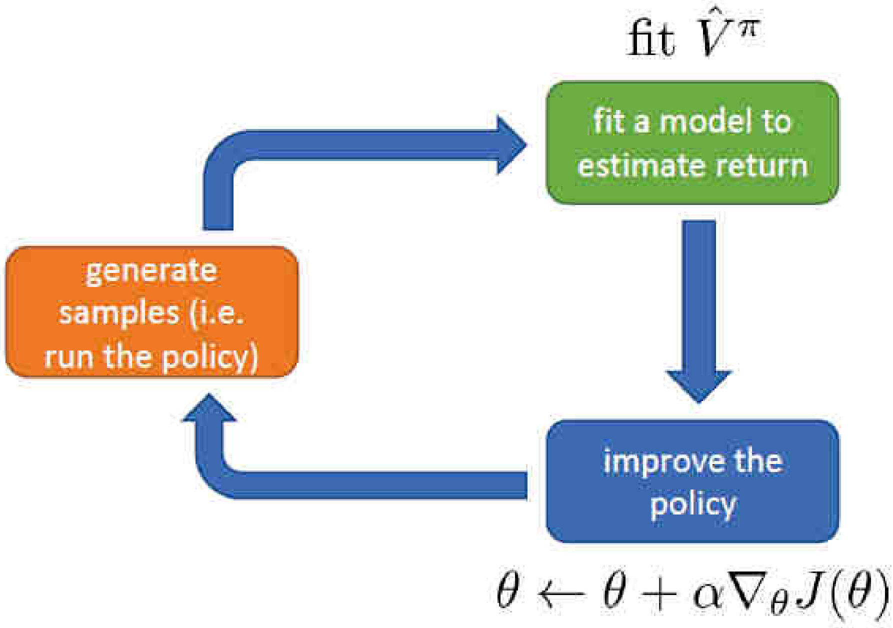
Actor-Critic principle =
use 2 parameterized functions:
- policy $\pi_\theta(s)$
- state-value function $V^\pi_\Phi(s)$



Algo combines policy-gradient & Q-learning:
- Learn $\pi_\theta \approx \pi^*$ with *policy gradient* using V^π_Φ
- V^π_Φ is learnt to fit observed cumulated rewards



Actor-Critic Algo



Estimating cumulated rewards with a parameterized function,
rather than by Monte Carlo (random rollouts)
➔ reduces policy gradient variance + improves sample-efficience

Advantage Actor-Critic (A2C)

Advantage of a policy: $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$
measures how better it is to choose action a instead of π(s)

A2C: replace r(τ) by A^{π}_{Φ} in Policy Gradient estimate

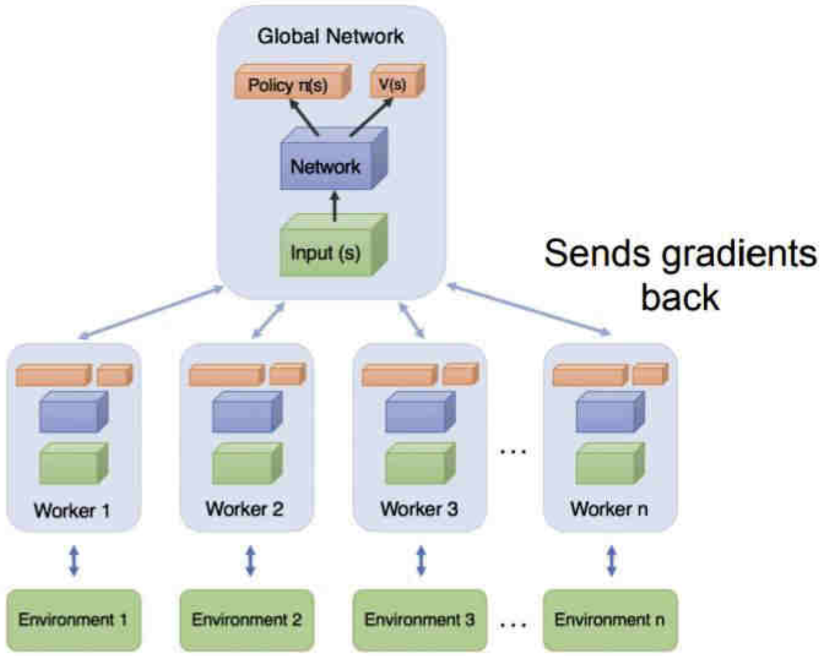
<u>Batch A2C algo:</u>

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$
2. fit $\hat{V}^{\pi}_{\phi}(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^{\pi}(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}'_i) - \hat{V}^{\pi}_{\phi}(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^{\pi}(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

<u>Online A2C algo:</u>

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}^{\pi}_{\phi}$ using target $r + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}')$
3. evaluate $\hat{A}^{\pi}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}') - \hat{V}^{\pi}_{\phi}(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^{\pi}(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

A3C: Asynchronous Advantage Actor Critic



➔ **Parallel learning in several instances of environment**

Soft Actor Critic (SAC)

- **Off-policy Actor-Critic (~soft Q-learning + PG)**
- **Maximizes not only return, but also *entropy* of the policy π (for better exploration):**

$$J(\pi_\theta) = E_{\pi_\theta}[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t) + \alpha H(\pi(.|s_t))]$$

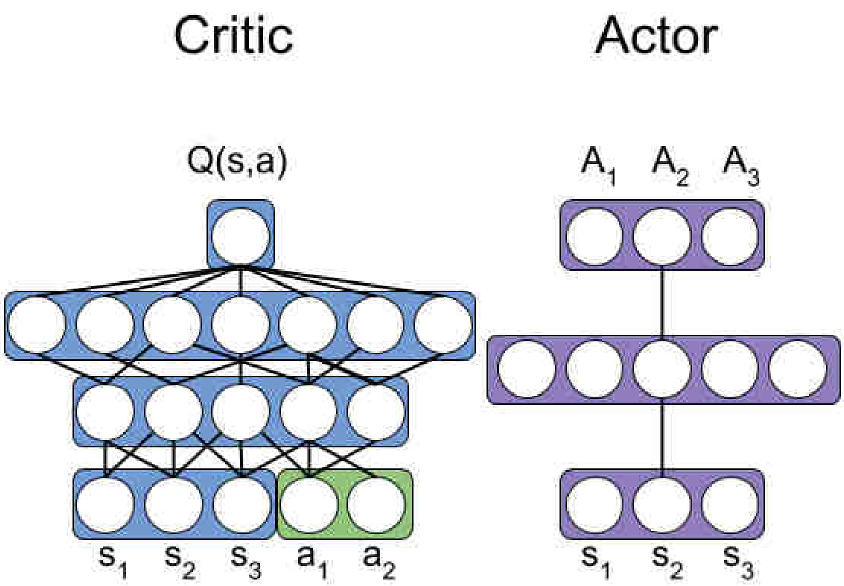
- **Learn 3 Neural Networks: π_θ, Q_Φ, V_ψ**

Algorithm 1 Soft Actor-Critic

Inputs: The learning rates, λ_π, λ_Q, and λ_V for functions π_θ, Q_w, and V_ψ respectively; the weighting factor τ for exponential moving average.

Initialize parameters θ, w, ψ, and $\bar{\psi}$.
for each iteration **do**
  (In practice, a combination of a single environment step and multiple gradient steps is found to work best.)
  for each environment setup **do**
    $a_t \sim \pi_\theta(a_t|s_t)$
    $s_{t+1} \sim \rho_\pi(s_{t+1}|s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1}\}$
  for each gradient update step **do**
    $\psi \leftarrow \psi - \lambda_V \nabla_\psi J_V(\psi)$.
    $w \leftarrow w - \lambda_Q \nabla_w J_Q(w)$.
    $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta J_\pi(\theta)$.
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$.

Deep Deterministic Policy Gradient (DDPG)



Off-policy Actor-Critic

~ A3C+DQN combined

DDPG algorithm

- Incorporate replay buffer and target network ideas from DQN for increased stability
- Use lagged (Polyak-averaging) version of Q_ϕ and π_θ for fitting Q_ϕ (towards $Q^{\pi,\gamma}$) with TD(0)

$$\hat{Q}_t = r_t + \gamma Q_{\phi'}(s_{t+1}, \pi(s_{t+1}; \theta'))$$

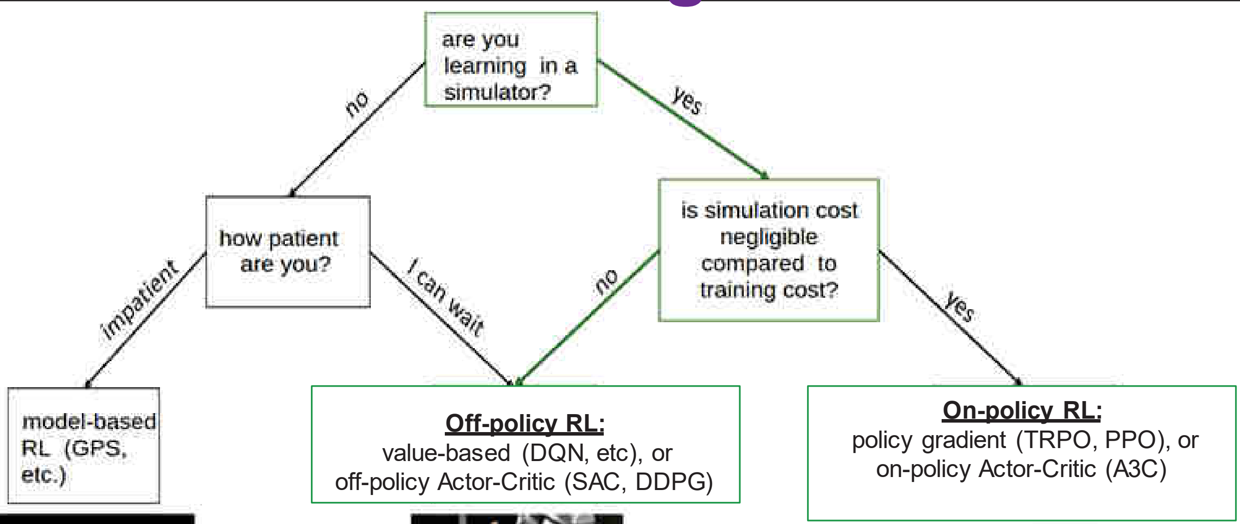
- Pseudocode:

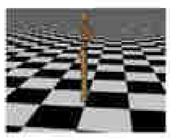
for iteration=$1, 2, \ldots$ **do**
  Act for several timesteps, add data to replay buffer
  Sample minibatch
  Update π_θ using $g \propto \nabla_\theta \sum_{t=1}^{T} Q(s_t, \pi(s_t, z_t; \theta))$
  Update Q_ϕ using $g \propto \nabla_\phi \sum_{t=1}^{T} (Q_\phi(s_t, a_t) - \hat{Q}_t)^2$,
end for

Summary of main DRL algorithm types

Type	Algo name	Based on	Output type
Policy-based	REINFORCE	ON-policy	Continuous
	TRPO	ON-policy	Continuous
	PPO	ON-policy	Continuous
Actor-Critic	A3C	ON-policy	Continuous
	SAC	OFF-policy	Continuous
	DDPG	OFF-policy	Continuous
Value-based	DQN & variants	OFF-policy	Discrete

How to choose DRL algorithm?





+ Need continuous-valued output of policy *vs.* Can use discrete actions

Policy Gradient vs Q-learning

- **Policy gradients**: very general but suffer from high variance so requires a lot of samples. **Challenge**: sample-efficiency
- **Q-learning**: does not always work but when it works, usually more sample-efficient. **Challenge**: exploration

- Guarantees:
  - **Policy Gradients**: Converges to a local minima of J(θ), often good enough!
  - **Q-learning**: Zero guarantees since you are approximating Bellman equation with a complicated function approximator

Main State-of-the-Art DRL algos:

DDPG or SAC (off-policy Actor-Critic continuous ouput)

OR

DQN-rainbow/IQN (off-policy Q-learning, discrete output)

Outline

- **Introduction**
- **Basics of Reinforcement Learning (RL)**
- **RL as a Markov Decision Process (MDP)**
  ➔ RL algos for tabular policies
- **Deep RL algos**
  - **Policy Gradient: REINFORCE, TRPO, PPO**
  - **DQN**
  - **Actor-Critic: A3C, SAC**
- **Example of DRL application:**
  learning to drive from vision in urban area

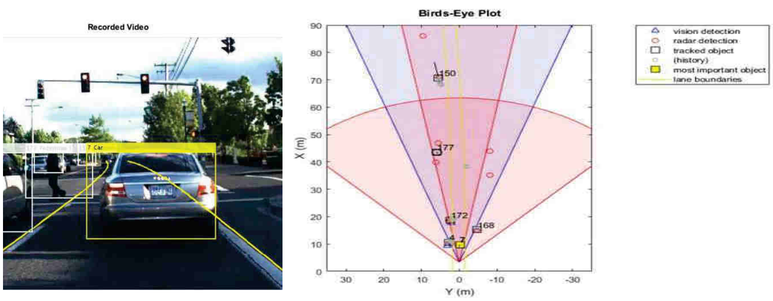
Application domains for RL

- Playing games
- Robots:
  - Locomotion Learning
  - Task Learning
  - Navigation/path-planning
- Automated Driving

MINES PARIS | PSL

Idea of end-to-end driving

Current architecture for automated driving



vs. HUMAN driving =
turn/accelerate-brake by just looking in front
≈ *"intelligent" visual servoing*

Principle of end-to-end driving

Sensors (cameras, radar, LIDAR, etc…) → Perception → Trajectory Planning → Control → Driving wheel, Acceleration or braking

Current architecture for automated driving

vs. **HUMAN driving: turn/brake *by just looking in front!***

≈ "intelligent" visual servoing

Sensors (cameras, radar, LIDAR, etc…) → Imitation Learning or Reinforcement Learning → Driving wheel, Acceleration or braking

Center camera → CNN → Network computed steering command → Drive by wire interface

Deep RL for automated driving

- **Until recently, very few published research, and mostly in racing games:**

  Asynchronous methods for deep reinforcement learning, V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, ICML'2016.

  End-to-End Race Driving with Deep Reinforcement Learning, Maximilian Jaritz, Raoul De Charette, Marin Toromanoff, Etienne Perot, Fawzi Nashashibi, *ICRA 2018 - IEEE International Conference on Robotics and Automation*, Brisbane, Australia, May 2018.

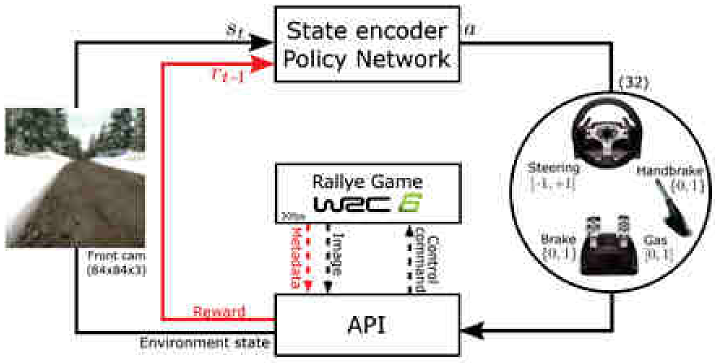
- **Up to now, only real driving with RL:**

  1. **"*Learning to Drive in a Day*" (Kendall et al., 2018)** [Cambridge]
     - Embed DRL in a real car, and learn « *from scratch* »
     - But *VERY SIMPLE CASE: lane keeping along 250m!*
     - Simulation used before to design architecture + tune hyper-parameters
  2. **"*Learning Robust Control Policies for End-to-End Autonomous Driving from Data-Driven Simulation*" (Amini et al., 2020)** [MIT]

Preliminary DRL experiment for end-to-end driving

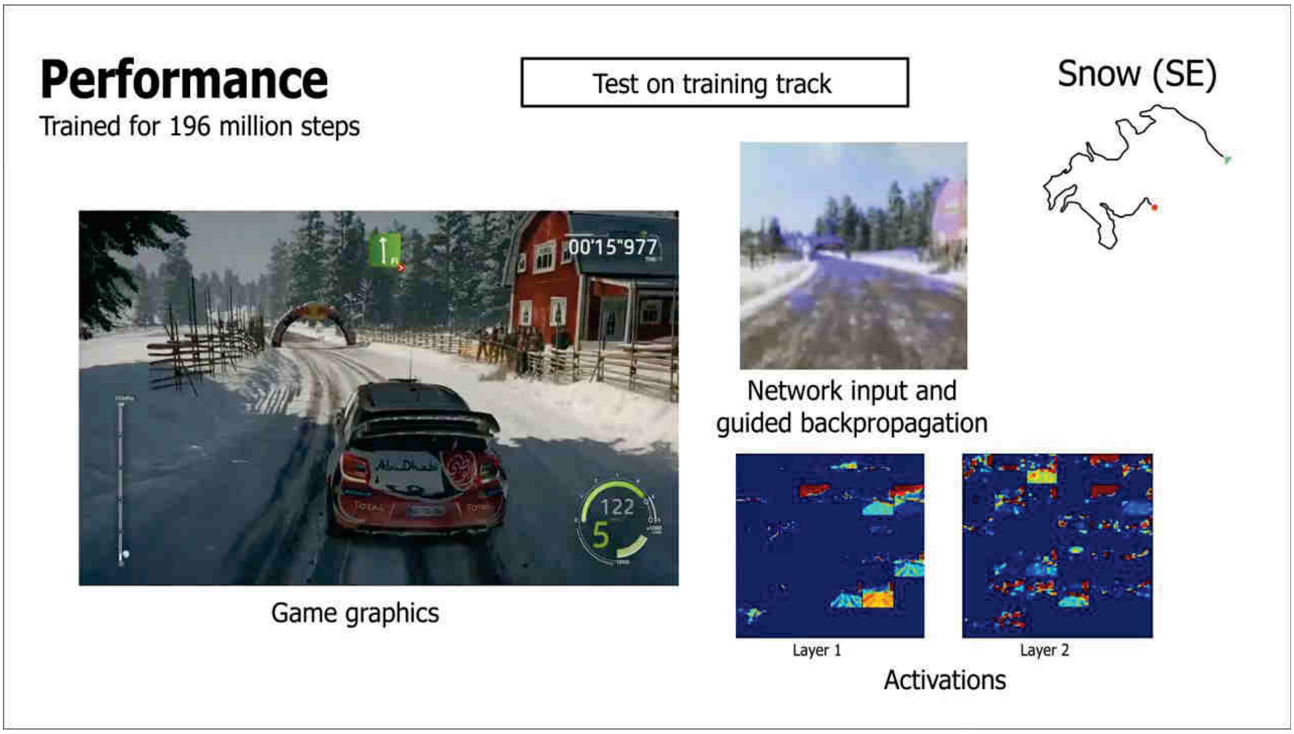
[Work by my Valeo CIFRE PhD student Marin Toromanoff]

End-to-end driving learning by RL in racing-car simulator



Etienne Perot, Maximilian Jaritz, Marin Toromanoff, Raoul De Charette. End-to-End Driving in a Realistic Racing Game with Deep Reinforcement Learning, International conference on Computer Vision and Pattern Recognition - Workshop, Honolulu, United States, Jul. 2017.

End-to-end driving learnt by RL in racing-car simulator



End-to-End Race Driving with Deep Reinforcement Learning, **Maximilian Jaritz, Raoul De Charette, Marin Toromanoff, Etienne Perot, Fawzi Nashashibi, *ICRA 2018 - IEEE International Conference on Robotics and Automation*, Brisbane, Australia, May 2018.**

RL for Automated Driving: why learn in a simulator?

- **RL require HUGE amount of trial & error, and initial policy = very bad driving!**
  ⇒ *Learn in <u>simulation</u>* (for safety + speed)
- **Still few driving simulators adapted for DL and RL, and best ones not totally mature**

Simulateur	GTA	DeepDrive.io	AirSim	CARLA [1]
Flexibilité	−−	++	++	++
Variété	++	−−	−	+
Complexité/Réalisme	++	−−	−	−
Objets mobiles	++	−−	−−	+
Vitesse exécution	−−	+	+	+
Multi-agent	−−	−	−	++

→ Choice of CARLA

[1] A. Dosovitskiy: CARLA: An Open Urban Driving Simulator (2017)

CARLA simulator

- **Open source, flexible** http://carla.org/

CARLA Autonomous Driving challenge

- **Itinerary to be followed in a city** (given by 4 possible orders at intersections: Left, Straight, Right, Follow_Lane) **BUT must <u>stay on the road</u>, in the lane, respecting Traffic Lights, and no collision with pedestrians and other cars!**
- **Evaluation metrics = Task completion & Distance between infractions, in an UNSEEN CITY**

Our Valeo-MINES_Paris approach for Carla AD challenge

- Value-based (DQN family) SotA and optimized Deep Reinforcement Learning algorithm
- Specific architecture for driving ConvNet
- Image-encoding part of convNet pre-trained with supervised learning
- Rewards as Natural as possible (close to human description of driving task)

"*End-to-End Model-Free Reinforcement Learning for Urban Driving using Implicit Affordances*", M.Toromanoff, E.Wirbel & F.Moutarde, CVPR'2020

DRL used: Rainbow + IQN + ApeX

- **Rainbow** [1] = combination of many improvements of DQN [4] ➔ currently SoA on ATARI benchmark
- **IQN** [2] = learning with ***probability distributions*** rather than just expectation of average

	Mean	Median	Human Gap	Seeds
DQN	228%	79%	0.334	1
PRIOR.	434%	124%	0.178	1
C51	701%	178%	0.152	1
RAINBOW	**1189%**	**230%**	0.144	2
QR-DQN	864%	193%	0.165	3
IQN	1019%	218%	**0.141**	5

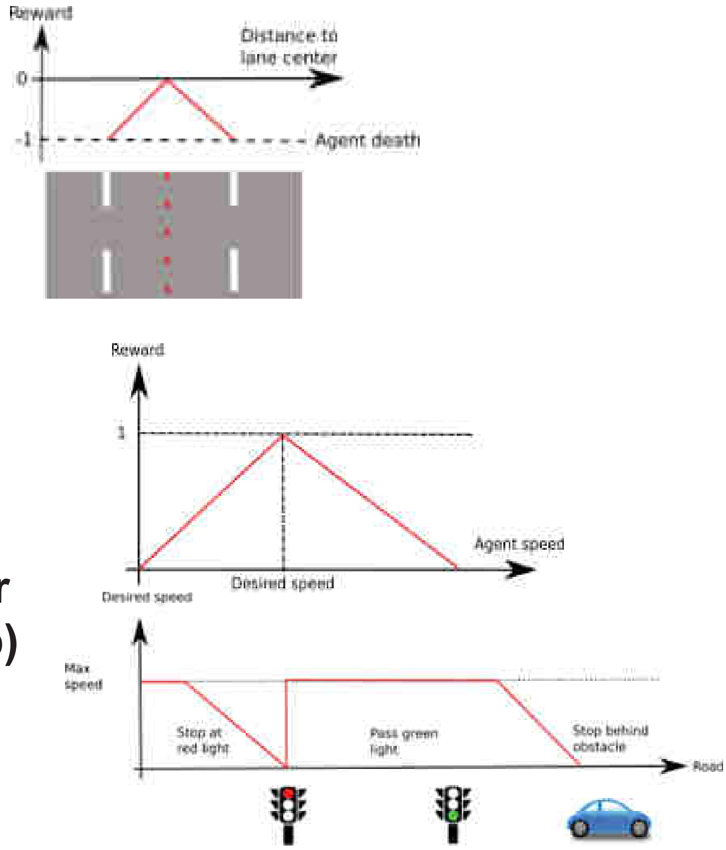
- **Ape-X** [3] ***multi-agent*** version of DQN allowing massively parallel distributed learning
  ⇒ Largely better performance, but typically require 22 billions of frames (vs. 200 millions)

[1] M. Hessel et al : Rainbow: Combining Improvements in Deep Reinforcement Learning Matteo (2017)
[2] D. Silver et al : Implicit Quantile Networks for Distributional Reinforcement Learning (2018)
[3] B. Horgan et al : Distributed Prioritized Experience Replay (2018)
[4] V. Mnih et al : Human-level control through deep reinforcement learning (2015)

Reward shaping

Rewards scaled in [-1, 1]:

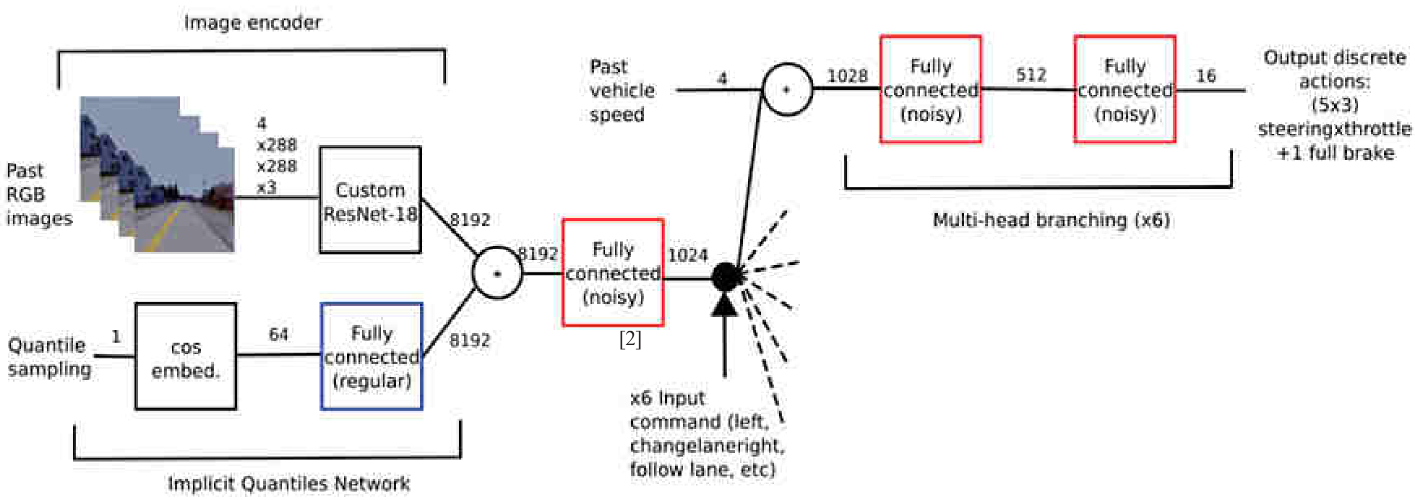
- **Lateral position: negative reward depending on distance to lane center**
- **Speed: positive reward to follow speed, depends on obstacles & traffic light**
- **Episode terminates on collision, running red traffic light, too far from lane center or stuck (if no reason to stop)**



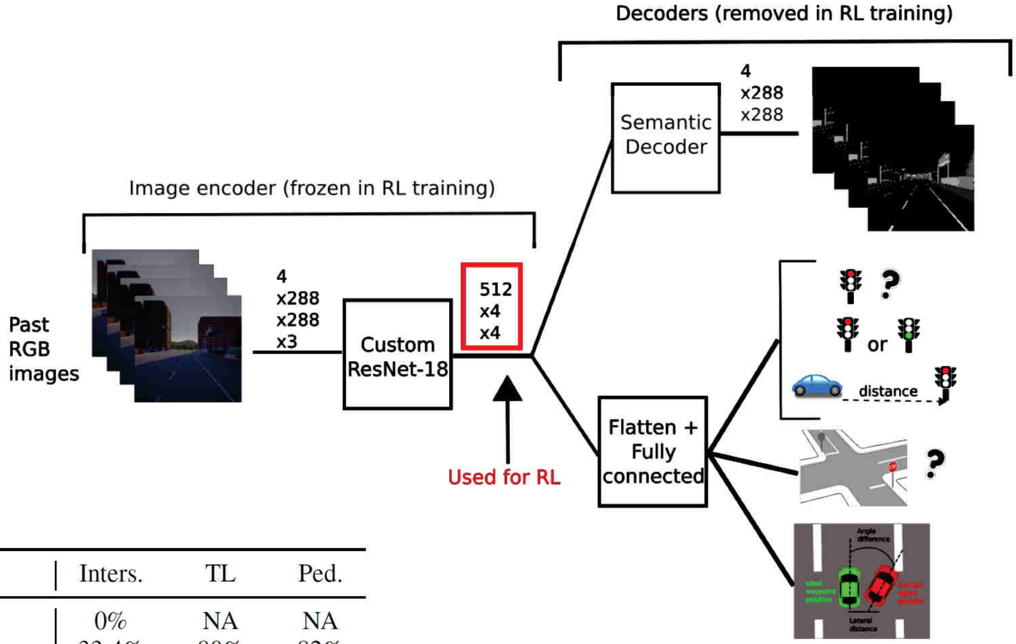
Network architecture

- **U.S. Traffic lights ➔ Need to use *COLOR and high-enough resolution* ⇒ big network, hard and slow to train**
- **Use a resnet-18 (10 times more weight than previously used in *DQN-like* network)**
- **Handle turn-orders (at intersections) with multi-head branching [1]**

[1] Codevilla et al., *End-to-end driving via Conditional Imitation Learning*, 2017
[2] M. Fortunato et al.. *Noisy Networks for Exploration*, 2017



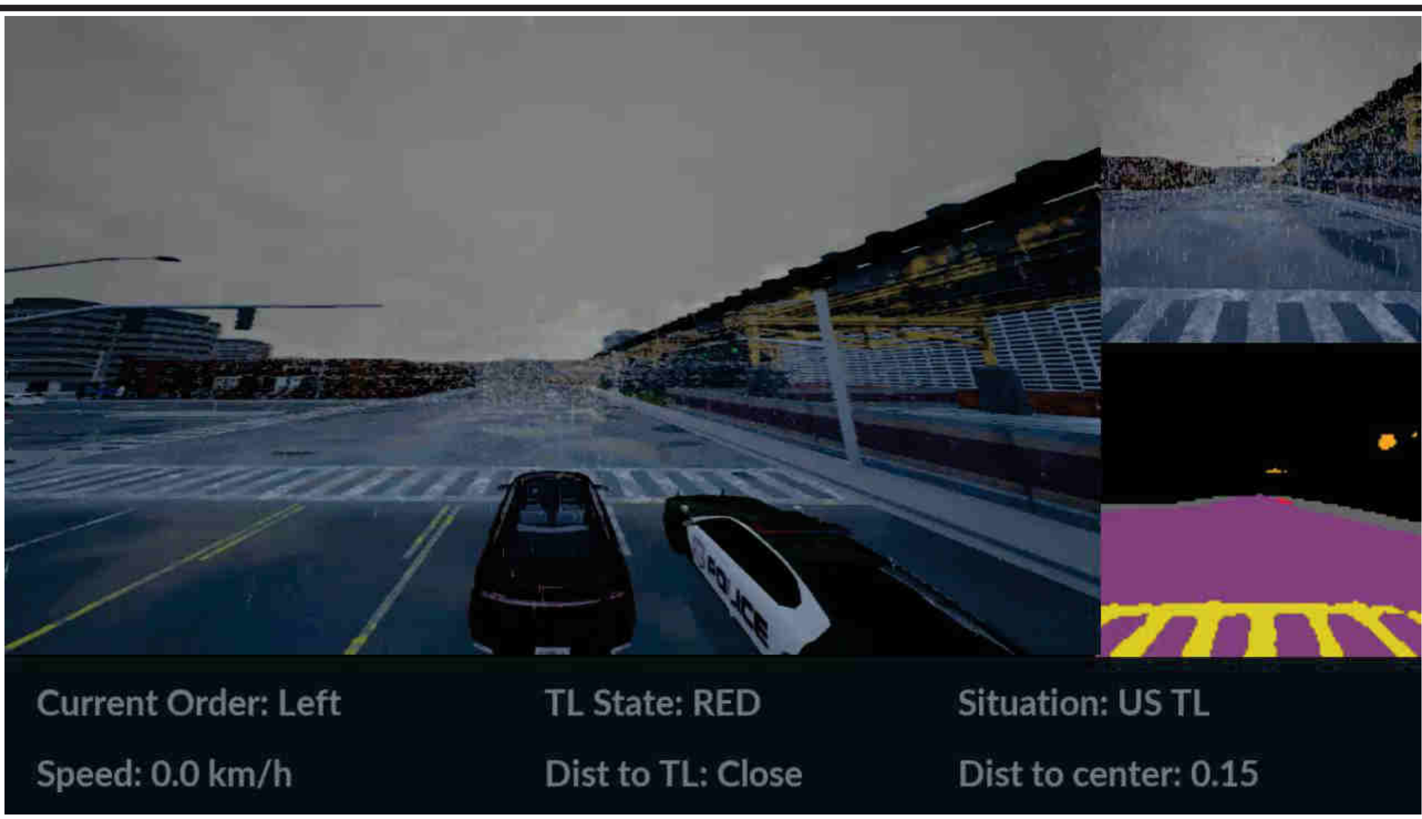
Offline encoder SUPERVISED training



Encoder used	Inters.	TL	Ped.
Random	0%	NA	NA
No TL state	33.4%	80%	82%
No segmentation	41.6%	96.5%	63%
All affordances	61.9%	97.6%	76%

Table 1. Comparison of agent performance with regards to encoder training loss (random weights, trained without traffic light loss, without semantic segmentation loss, or with all affordance losses)

Examples of Autonomous Driving obtained with our DRL



"*End-to-End Model-Free Reinforcement Learning for Urban Driving using Implicit Affordances*", M.Toromanoff, E.Wirbel & F.Moutarde, CVPR'2020

Conclusions & perspectives on DRL for Automated driving

- DRL allows to learn driving behavior *without any example provided by human*
- Only the REWARD needed to define objectives
- Very encouraging first results in simulation: able to learn a kind of "*Intelligent visual servoing*" avoiding collisions & respecting traffic lights + high-level orders *(e.g. turn-left at next intersection)*
- Winner of "vision-only" track at CARLA "Autonomous Driving challenge" 2019 & 2020 !!
- Future work:
  - transferrability to real-world videos
  - Combination of Imitation-Learning and RL?

General conclusion on DRL

- Many variants of algorithms
- Generally necessary to learn in some simulator
- Allows to learn intelligent BEHAVIORS (real AI?)
- Big potential of Deep Reinforcement Learning in particular in Robotics and Automated Vehicles