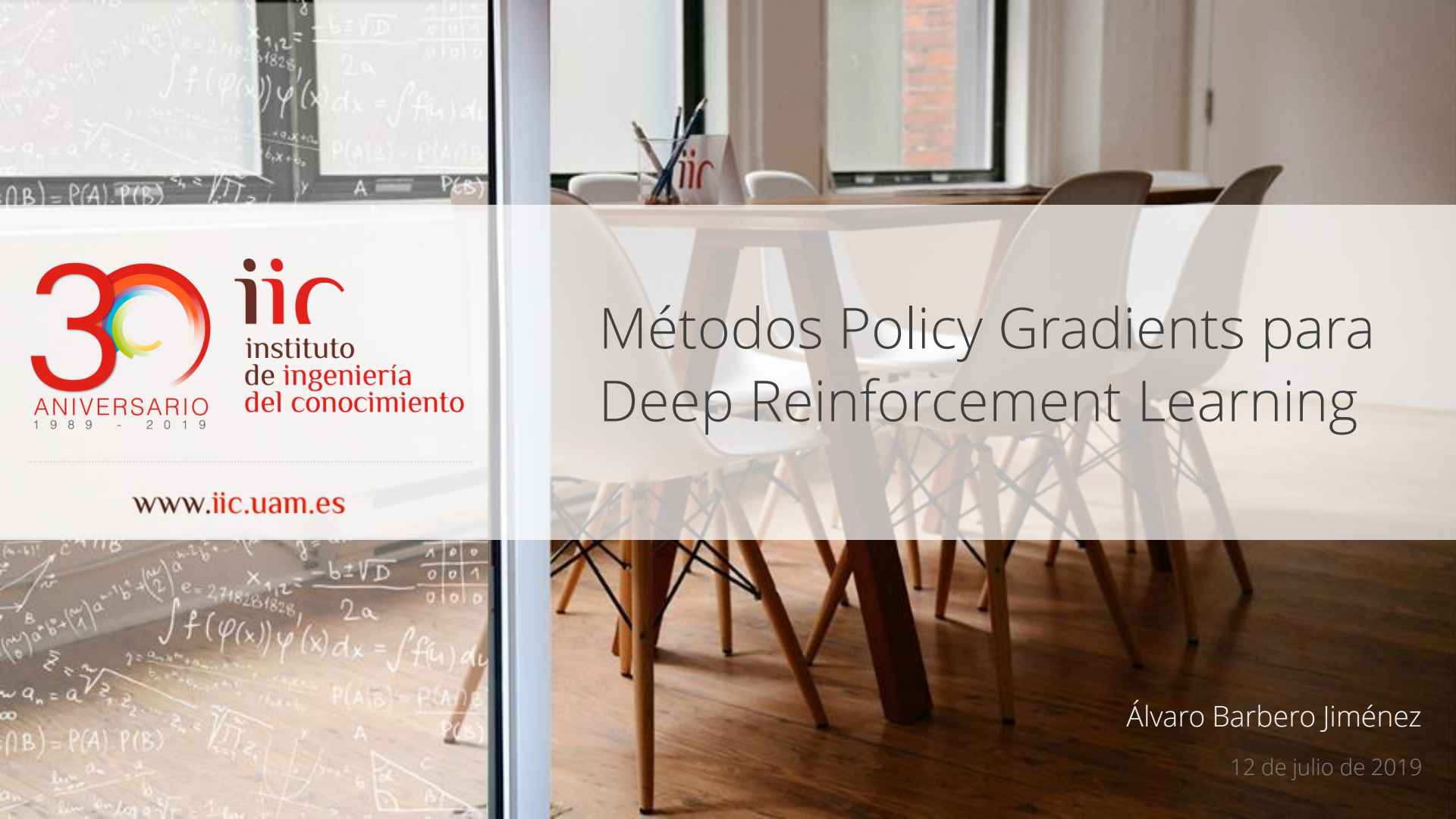


30 ANIVERSARIO 1989 - 2019

iic instituto de ingeniería del conocimiento

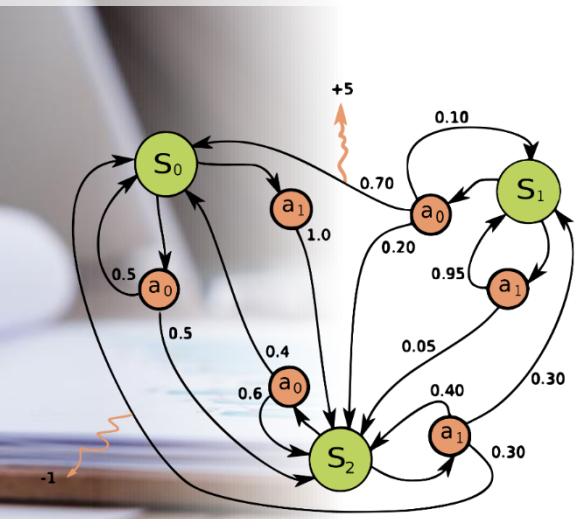
www.iic.uam.es

Métodos Policy Gradients para Deep Reinforcement Learning

Álvaro Barbero Jiménez

12 de julio de 2019

Reinforcement learning formalism



A reinforcement learning problem can be stated in terms of a Markov decision process, defined through:

$\boldsymbol{s_t} \in \mathbb{R}^D$: state time t, which might be a terminal state (process stops)

$\boldsymbol{r_t} \in \mathbb{R}$: reward obtained at time t

$\boldsymbol{p_s}(\boldsymbol{s'}|\boldsymbol{s},\boldsymbol{a}) \in \mathbb{R}$: state transition model, probability of jumping to state s', given current state s and some action a

$\boldsymbol{p_r}(\boldsymbol{r}|\boldsymbol{s},\boldsymbol{a}) \in \mathbb{R}$: reward model, probability of obtaing reward r given current state s and some action a

$\boldsymbol{a_t} \in \mathbb{R}^A$: action performed by the agent at time t

Objective: find an agent or policy function $\pi(a|s)$ (probability of performing action a in state s) that maximizes the expected rewards:

$$\max_{\pi} \mathbb{E}_{p_s,p_r}[\pi(a_0|s_0)p_s(s_1|s_0,a_0)p_r(r_1|s_0,a_0)r_1 + \pi(a_1|s_1)p_s(s_2|s_1,a_1)p_r(r_2|s_1,a_1)r_2 + \pi(a_2|s_2)p_s(s_3|s_2,a_2)p_r(r_3|s_2,a_2)r_3 + \ldots]$$

Wikipedia - Markov Decision Process

Example: Frozen Lake

State: current tile in the lake.

Terminal states: house and holes

Actions: move up, down, left or right.

Probabilistic transition model: random perturbations in chosen action (slippery ice) with unknown distribution.

Deterministic reward model:

 +10  +1  -10

Classic methods

Assuming the state transition model $\boldsymbol{p_s}(\boldsymbol{s'}|\boldsymbol{s},\boldsymbol{a})$ and the reward model $\boldsymbol{p_r}(\boldsymbol{s'}|\boldsymbol{s},\boldsymbol{a})$ are known, the optimal policy $\pi^*(a|s)$ can be found by using dynamic programming methods: **Value Iteration** and **Policy Iteration**.

Essentially, find best 1-step policy for all states, then iterate finding the best n+1-steps policy from the n-steps policy.

However, in real world problems usually $\boldsymbol{p_s}$ and $\boldsymbol{p_r}$ are unknown. There are mainly two options to overcome this:

- **Model-based methods**: learn a model of $(\boldsymbol{p_s}, \boldsymbol{p_r})$, e.g. by Montecarlo sampling, then apply the methods above.
- **Model-free methods**: learn the policy π directly.

! In some problems where the state space is very large, or continuous, or too complex, it is infeasible to apply model-based methods.

Today we will present Policy Gradients Methods, a class of model-free methods.

Policy gradients methods

To learn the policy π directly we will need a reformulation.

Let us define:

Trajectory $\tau = [(s_0, a_0, r_1), (s_1, a_1, r_2), \dots]$

Total reward of a trajectory $r(\tau) = \sum_{r_t \in \tau} r_t$

Differentiable policy $\pi_\theta(a|s)$ with parameters θ

Then we can measure the quality of a policy by the expected rewards of the trajectories produced by that policy

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)] = \int_\tau r(\tau) p(\tau;\theta) d\tau$$

Now, to find the best policy we can follow a simple optimization procedure:

- Start with a random policy
- Update policy parameters through gradient ascent

$$\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta)\, d\tau$$

!!! The integral in this gradient is intractable!

Policy gradients methods – Computing the gradient

$$\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta)\, d\tau$$

To compute this gradient we use a logarithm reformulation

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

and injecting this back

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

This expectation can be approximated through a Monte Carlo method, i.e. by taking sample trajectories from the policy

$$\nabla_\theta J(\theta) \simeq \frac{1}{k}\sum_k r(\tau_k)\nabla_\theta \log p(\tau_k;\theta)$$

The only remaining problem is how to compute $\nabla_\theta \log p(\tau_k;\theta)$.

Policy gradients methods – Computing the gradient

$$\nabla_\theta J(\theta) \simeq \frac{1}{k}\sum_k r(\tau_k)\nabla_\theta \log p(\tau_k;\theta)$$

Since we are in a Markov Decision Process, the probability of a trajectory is

$$p(\tau;\theta) = \prod_t p_s(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)$$

$$\log p(\tau;\theta) = \sum_t (\log p_s(s_{t+1}|s_t,a_t) + \log \pi_\theta(a_t|s_t))$$

Key observation: even if the transition model p_s is unknown, it does not depend on θ!

$$\nabla_\theta \log p(\tau;\theta) = \sum_t (\nabla_\theta \log p_s(s_{t+1}|s_t,a_t) + \nabla_\theta \log \pi_\theta(a_t|s_t))$$

Therefore the gradient of the reinforcement learning objective is simply

$$\nabla_\theta J(\theta) \simeq \frac{1}{k}\sum_k r(\tau_k) \sum_t \nabla_\theta \log \pi_\theta(a_{kt}|s_{kt})$$

Policy gradients methods – REINFORCE algorithm

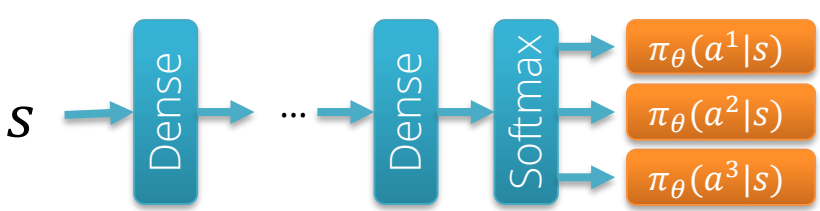
$$\nabla_\theta J(\theta) \simeq \frac{1}{k}\sum_k r(\tau_k)\sum_t \nabla_\theta \log \pi_\theta(a_{kt}|s_{kt})$$

Implementing this gradient calculation is easy: use a neural network to implement $\pi_\theta(a|s)$, with one output neuron per action and softmax activation. Then the training loop is:

- Sample k trajectories from the environment under the current policy $\pi_\theta(a|s)$. Store trajectories as lists of steps $[s_{kt}, a_{kt}, r_{kt}, \pi_\theta(a_{kt}|s_{kt})]$.

- Compute $\nabla_\theta \log \pi_\theta(a_{kt}|s_{kt})$ for each timestep. This is trivial using an automated differentiation tool such as TensorFlow or PyTorch.

- Update policy parameters θ using the gradient above.



TensorFlow

PYTORCH

Policy Gradients intuition

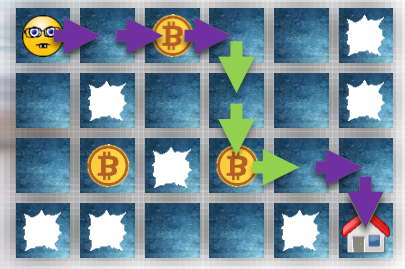
$$\theta \mathrel{+}= \frac{1}{k}\sum_{k} r(\tau_k) \sum_{t} \nabla_\theta \log \pi_\theta(a_{kt}|s_{kt})$$

Sample many trajectories, then equally promote all the actions $\pi_\theta(a|s)$ performed in high reward trajectories, and equally demotivate all the actions $\pi_\theta(a|s)$ performed in low reward trajectories. Eventually we converge to a local maximum where the actions that are consistently useful among many trajectories should be performed with high probability.

+11

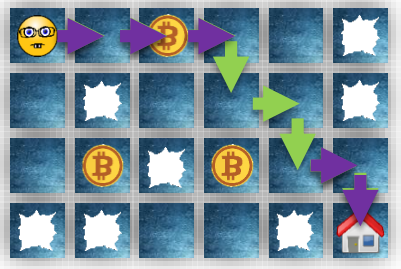
-10

-9



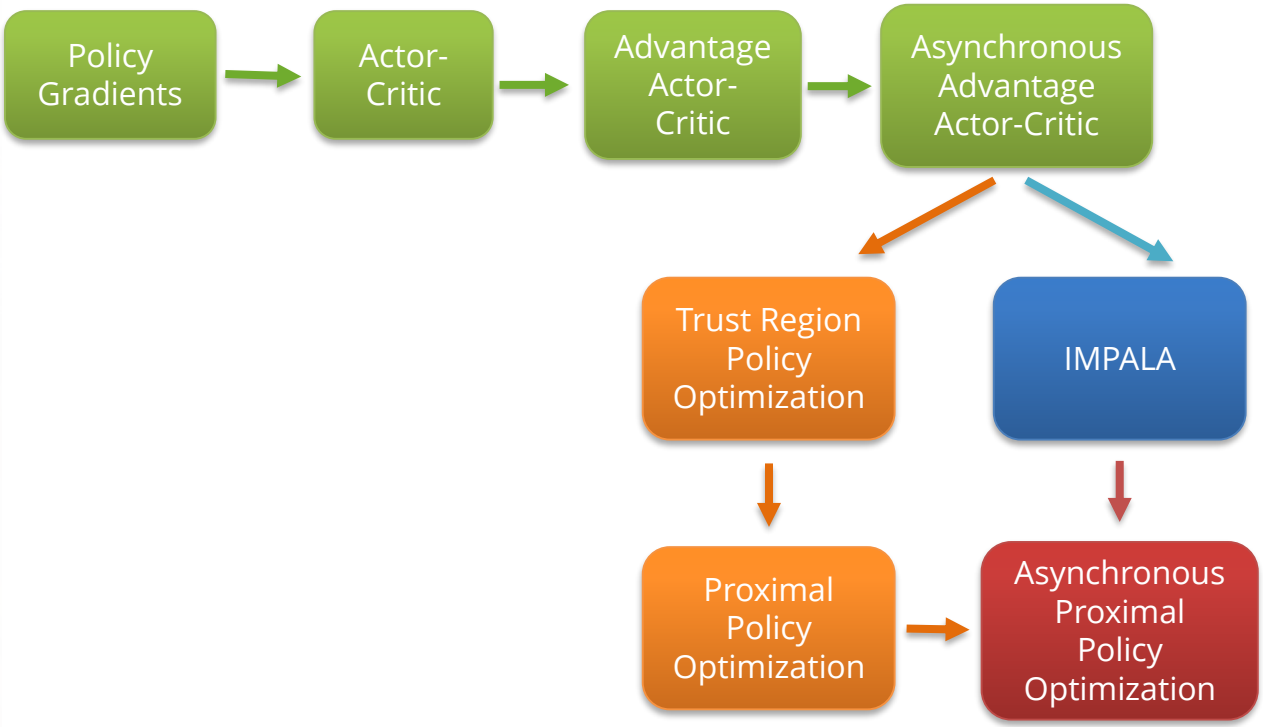
+12

-9



+11

Evolution of Policy Gradients algorithms



Better credit assignment

The update

$$\theta \mathrel{+}= \frac{1}{k}\sum_{k} r(\tau_k) \sum_{t} \nabla_\theta \log \pi_\theta(a_{kt}|s_{kt})$$

reinforces all actions in the trajectory by the total trajectory reward $r(\tau_k)$.

However, most actions may have no real impact on the total reward, while a few actions might be key. It might take many samples for the policy to learn which are those.

It would be useful to implement some intuition into what actions are significant. An intelligent player is reasonably good at estimating the probable outcome of an in-progress trajectory.

Actor-Critic methods

The Actor model defines the policy $\pi_\theta(a|s)$, and learns to improve the expected rewards.

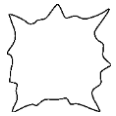
Same as the usual Policy Gradients policy.

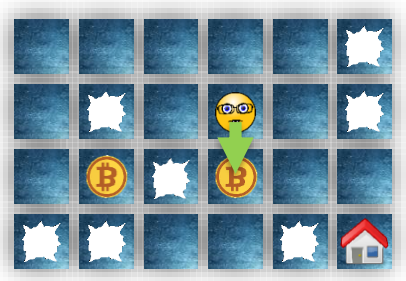
The Critic model defines some value function $Q_\phi(s, a, \pi)$ estimating the expected reward to be obtained when applying the policy π after using action a in state s. Improves its estimates through training.

Critic example


+10


+1


-10



Actor-Critic methods – Value function

The ideal value function $Q(s, a, \pi)$ is the total expected reward to be obtained from state s, after applying action a, until the trajectory finishes.

$$Q(s, a, \pi) = \mathbb{E}_{\tau \sim p(\tau;\pi)} \left[\sum_{t \geq 0, r \in \tau} \gamma^t r_t \, | s_0 = s, a_0 = a \right]$$

with $\gamma \in (0,1)$ a discount factor to give less weight to rewards in the far future.

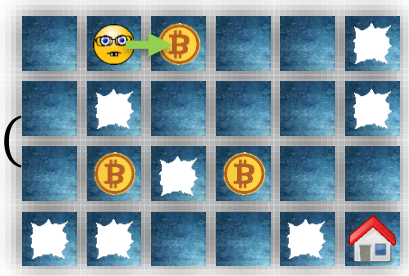
Learning this is posible through a Temporal Difference (TD) error strategy: in the optimum the following must hold:

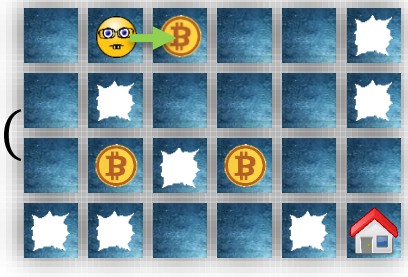
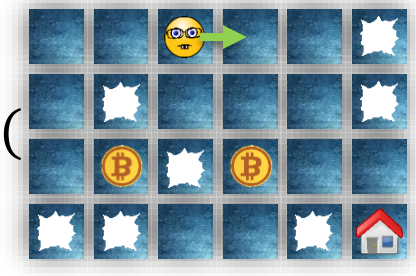
$$Q_\phi(s_t, a_t, \pi) = r_t + \gamma Q_\phi(s_{t+1}, a_{t+1}, \pi)$$

So we train the network to reduce the misalignments observed in Q in the sampled trajectories

$$\tau = [(s_0, a_0, r_1), (s_1, a_1, r_2), \dots]$$

Temporal Different learning example

$$Q_\phi\left(\text{  }\right) = 0 + \gamma Q_\phi\left(\text{  }\right)$$

$$Q_\phi\left(\text{  }\right) = +1 + \gamma Q_\phi\left(\text{  }\right)$$

$$Q_\phi\left(\text{  }\right) = -10 + \gamma Q_\phi\left(\text{  }\right)$$

Actor-Critic methods – Learning from the Critic

Now, we can improve the function the policy is optimizing by making use of the Critic. Instead of the vanilla Policy Gradients equation

$$\nabla_\theta J(\theta) \simeq \frac{1}{k}\sum_k r(\tau_k) \sum_t \nabla_\theta \log \pi_\theta(a_{kt}|s_{kt})$$

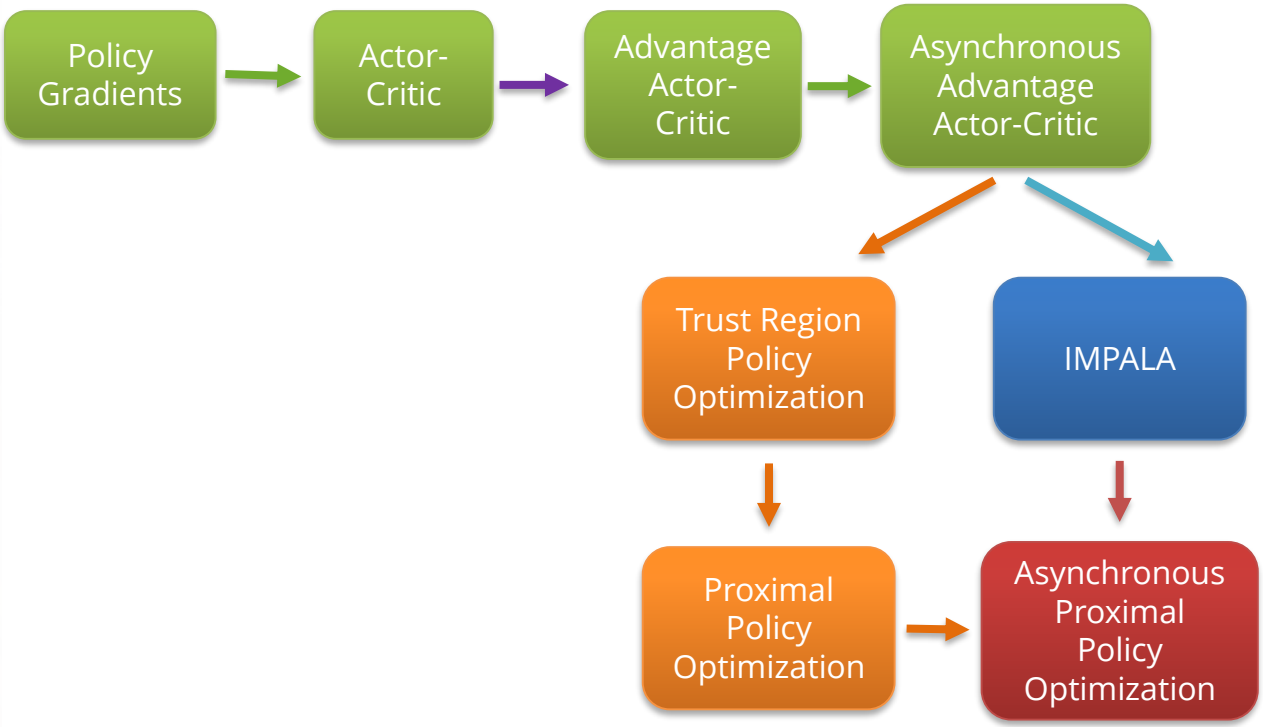
we will weigh each action by the Critic value function

$$\nabla_\theta J(\theta) \simeq \frac{1}{k}\sum_k \sum_t Q(s_{kt}, a_{kt}, \pi) \nabla_\theta \log \pi_\theta(a_{kt}|s_{kt})$$

In this way we motivate policy decisions that will lead to high rewards in the future, while demotivating decisions that will lead to negative rewards.

Also, we can learn without requiring full trajectories! → easier parallelization

Evolution of Policy Gradients algorithms



Baseline rewards

Note that even when using a Critic there is a fundamental problem when working in environments without negative rewards ($Q(s,a,\pi) \geq 0$)

$$\nabla_\theta J(\theta) \simeq \frac{1}{k}\sum_k \sum_t Q(s_{kt}, a_{kt}, \pi) \nabla_\theta \log \pi_\theta(a_{kt}|s_{kt})$$

The algorithm will always try to increase the probability of every decision taken (each with different weights), but won't explicitly demotivate any action.

This is sometimes fixed by substracting a baseline b

$$\nabla_\theta J(\theta) \simeq \frac{1}{k}\sum_k \sum_t (Q(s_{kt}, a_{kt}, \pi) - b) \nabla_\theta \log \pi_\theta(a_{kt}|s_{kt})$$

as b we can use the average reward, thus promoting actions with over-average rewards, while demotivaing actions below-par.

Advantage function – Expectations VS Reality

Better than using the average as baseline, we would like to promote actions that are advantageous, in the sense that they produce better rewards than we expected. We define the Advantage $A_\phi(s,a)$ as kind of Temporal Difference error:

$$A_\phi(s_t, a_t, \pi) = r_t + \gamma V_\phi(s_{t+1}, \pi) - V_\phi(s_t, \pi)$$

with $V_\phi(s,\pi)$ a modified Critic that accounts for the expected rewards when applying policy π from state s on.

$$V(s,\pi) = \mathbb{E}_{\tau\sim p(\tau;\pi)}\left[\sum_{t\geq 0, r\in\tau} \gamma^t r_t \,|s_0 = s\right]$$

When $A_\phi(s_t, a_t, \pi) \neq 0$ the Critic still has not yet learned the implications of doing a_t at s_t, and if $A_\phi(s_t, a_t, \pi) > 0$ then this is an unexplored and rewarding action. We want to promote that in the policy!

Advantage function - intuition

$s_t =$  $\rightarrow a_t \rightarrow s_{t+1} =$ 

$s_t =$  $\rightarrow a_t \rightarrow s_{t+1} =$ 

Learning with Advantages

The Actor (policy) can learn from a Critic based on advantages, as

$$\nabla_\theta J(\theta) \simeq \frac{1}{k}\sum_k \sum_t A(s_{kt}, a_{kt}, \pi)\nabla_\theta \log \pi_\theta(a_{kt}|s_{kt})$$

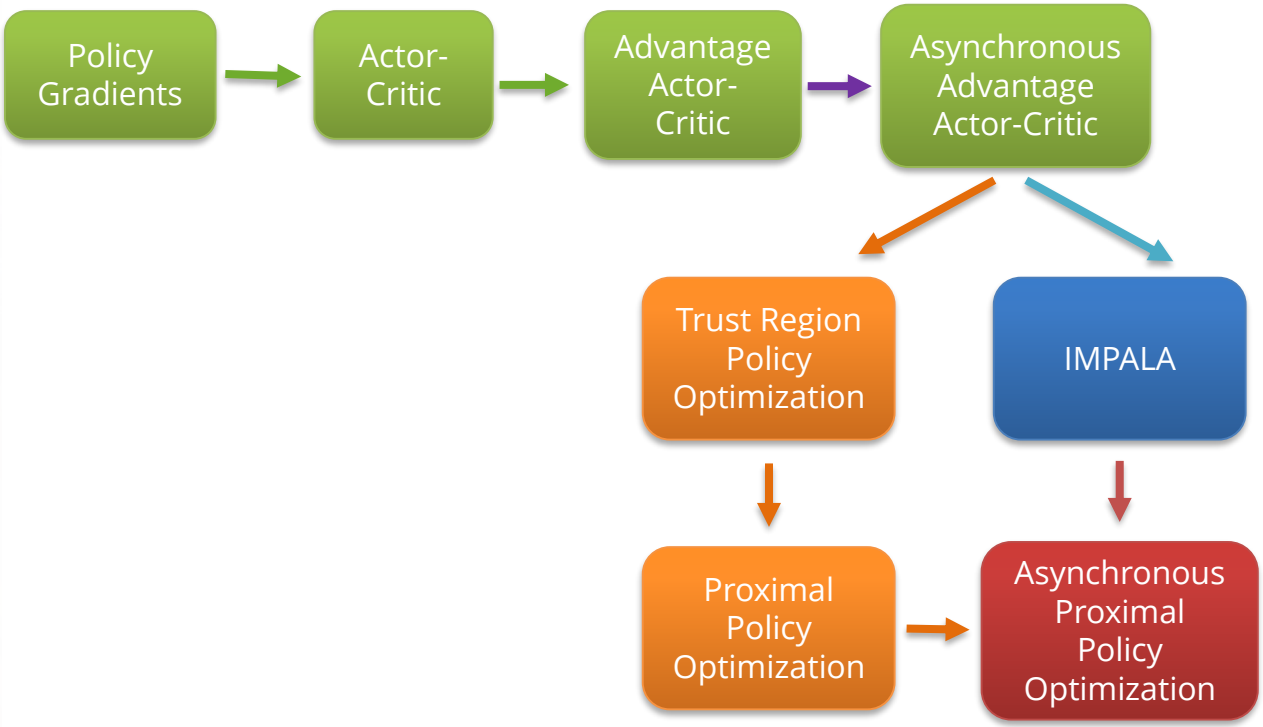
and the Critic can learn again by minimizing the Temporal Difference error, that is, the Advantages

$$\nabla_\phi \frac{1}{2}\left|\left|A(s_{kt}, a_{kt}, \pi)\right|\right|_2^2$$

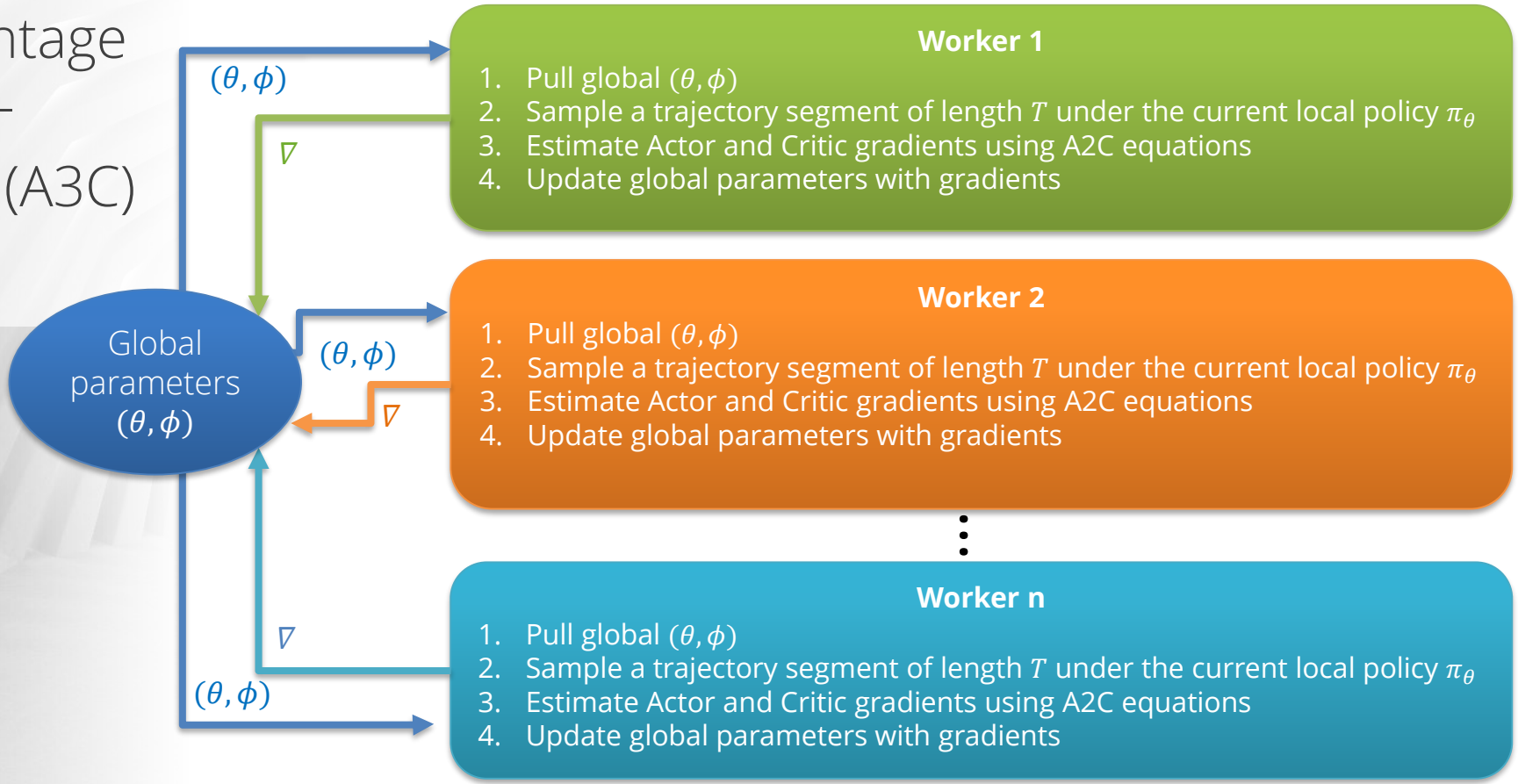
$$= \nabla_\phi \frac{1}{2}\left|\left|r_{kt} + \gamma\, V_{\hat{\phi}}(s_{kt+1}, \pi) - V_\phi(s_{kt}, \pi)\right|\right|_2^2$$

$$= -\left(r_{kt} + \gamma\, V_{\hat{\phi}}(s_{kt+1}, \pi) - V_\phi(s_{kt}, \pi)\right)\nabla_\phi V_\phi(s_{kt}, \pi)$$

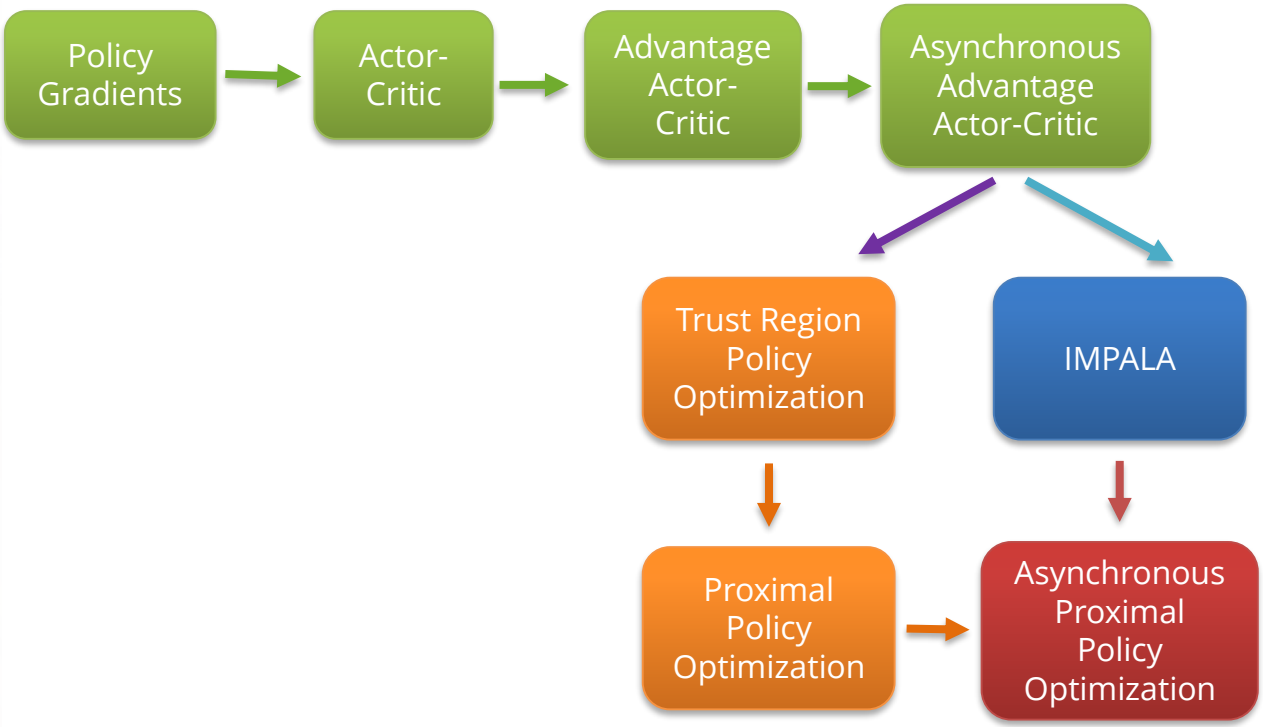
Evolution of Policy Gradients algorithms



Asynchronous Advantage Actor-Critic (A3C) loop



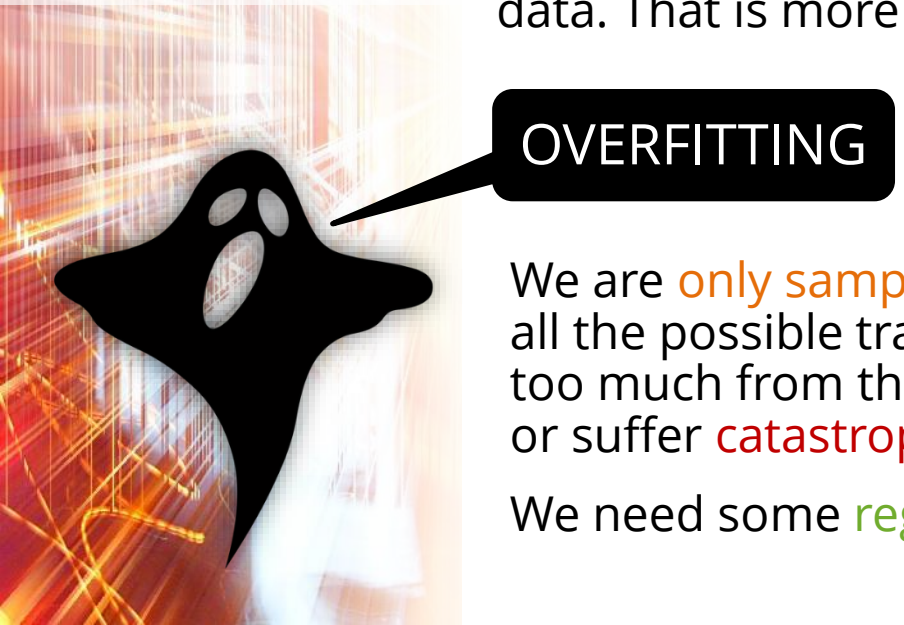
Evolution of Policy Gradients algorithms



Sample efficiency

A significant problem with all algorithms seen so far is sample efficiency. Each sample taken from the environment is only used once to compute a gradient, then it is discarded.

But in supervised learning it is quite common to perform several passes (epochs) over the training data. That is more efficient, why don't do it here?



We are only sampling a few trajectories (or segments) from all the possible trajectories in the environment. If we learn too much from them we will overfit to a suboptimal strategy, or suffer catastrophic forgetting.

We need some regularization mechanism!

Trust Region Policy Optimization (TRPO)

We can allow several training passes over the sampled data if we make sure the policy π_θ doesn't change too much. First, note that the gradient that we use in A2C (or A3C) can be rephramed as the optimization problem

$$\nabla_\theta J(\theta) = \sum_t A(s_t, a_t, \pi) \nabla_\theta \log \pi_\theta(a_t|s_t) \rightarrow \max_\theta \mathbb{E}_t[A(s_t, a_t, \pi) \log \pi_\theta(a_t|s_t)]$$

To avoid learning too much, we introduce a penalty in the form of Kullback-Leibler divergence

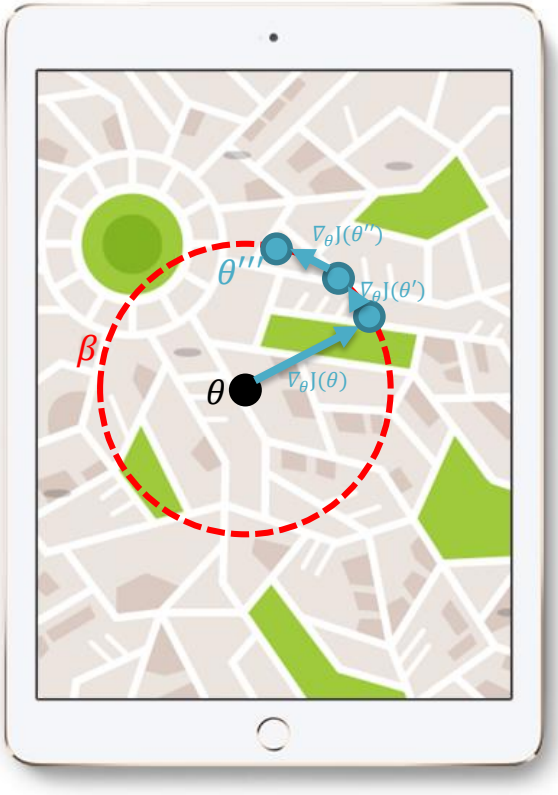
$$\max_\theta \mathbb{E}_t \left[A(s_t, a_t, \pi) \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} - \beta KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right]$$

with $\pi_{\theta_{old}}$ the previous version of the policy, and β a regularization parameter that controls the strength of the penalty.

TRPO intuition

A2C / A3C

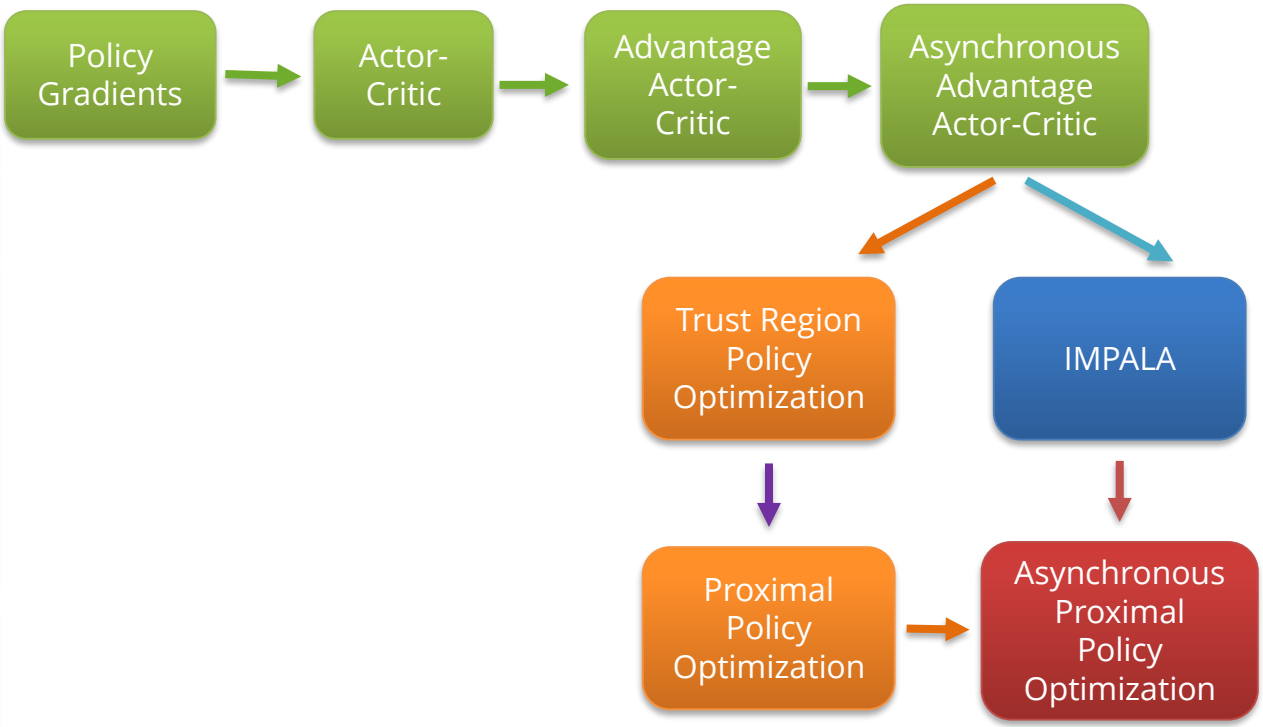
TRPO



Many steps per sample, higher sample efficiency!

But choosing an appropriate β value is not easy...

Evolution of Policy Gradients algorithms



Proximal Policy Optimization (PPO)

PPO is the algorithm of choice at OpenAI. It is similar to TRPO but easier to configure.

Let us define the ratio of probabilities $R_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$.

The objective is changed to look as

$$\max_\theta \mathbb{E}_t[\min(A(s_t, a_t, \pi)\, R_t(\theta), A(s_t, a_t, \pi) clip(R_t(\theta), 1-\epsilon, 1+\epsilon))]$$

this objective imposes a pessimistic bound: we can optimize θ freely, but we ignore any advantage obtained after modifying the probability of an action by a factor smaller than $1-\epsilon$ or larger than $1+\epsilon$.

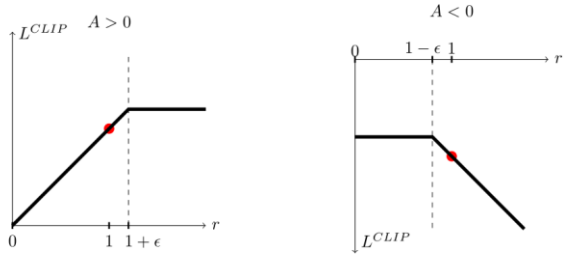
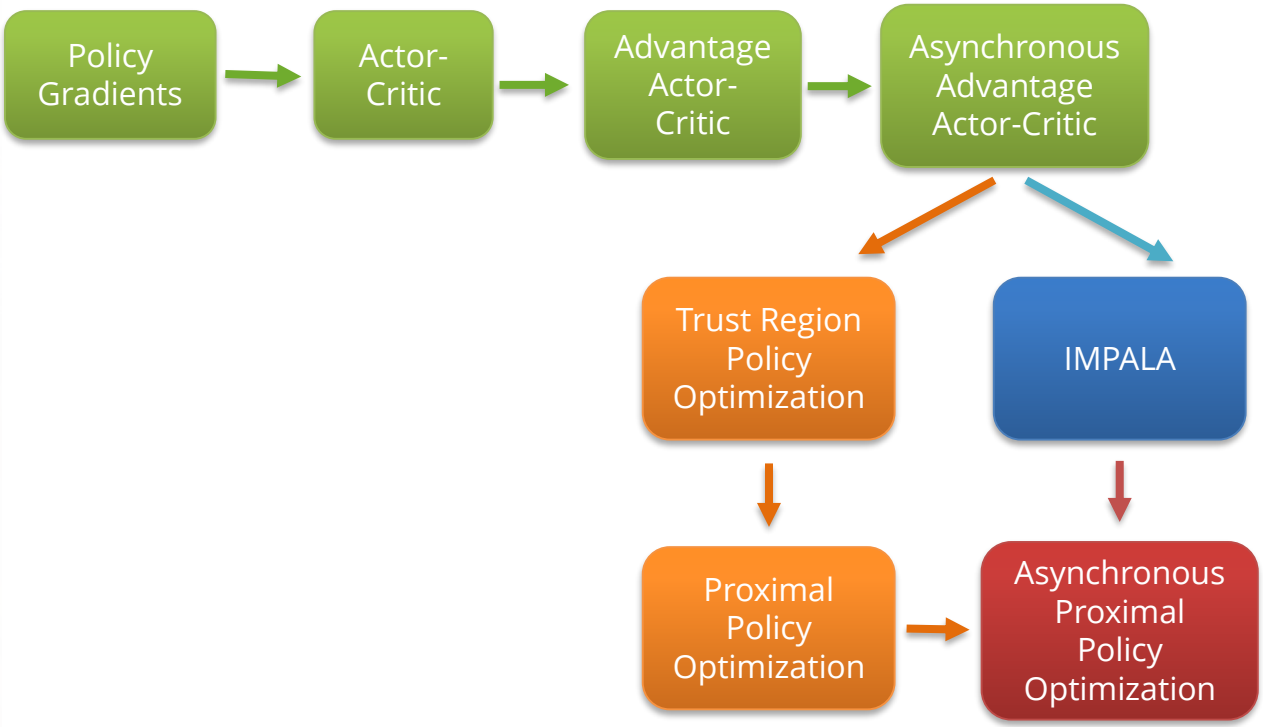


Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that L^{CLIP} sums many of these terms.

Schulman et al – Proximal Policy Optimization Algorithms

Evolution of Policy Gradients algorithms



OpenAI Five

Competitive: 7.215–42 (99.4% winrate, 15.019 total players)

Note: During the live stream, the game count incorrectly omitted games abandoned by the human side.

Cooperative: 35.466 games (18.689 total players)

OpenAI – How to train your OpenAI Five

Some home-made results

Joint work by

Álvaro Barbero

Ainhoa Goñi

Juan Montesino

Alba Segurado

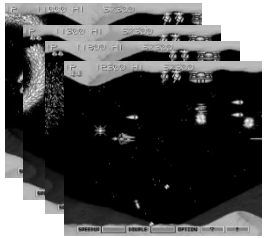
Jorge López

Rubén García

Data processing

Rescale to 84x84
Grayscale

Keep 1 out of 4 frames

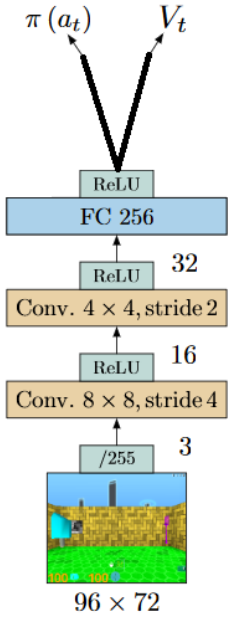


State: stack of last 4 frames

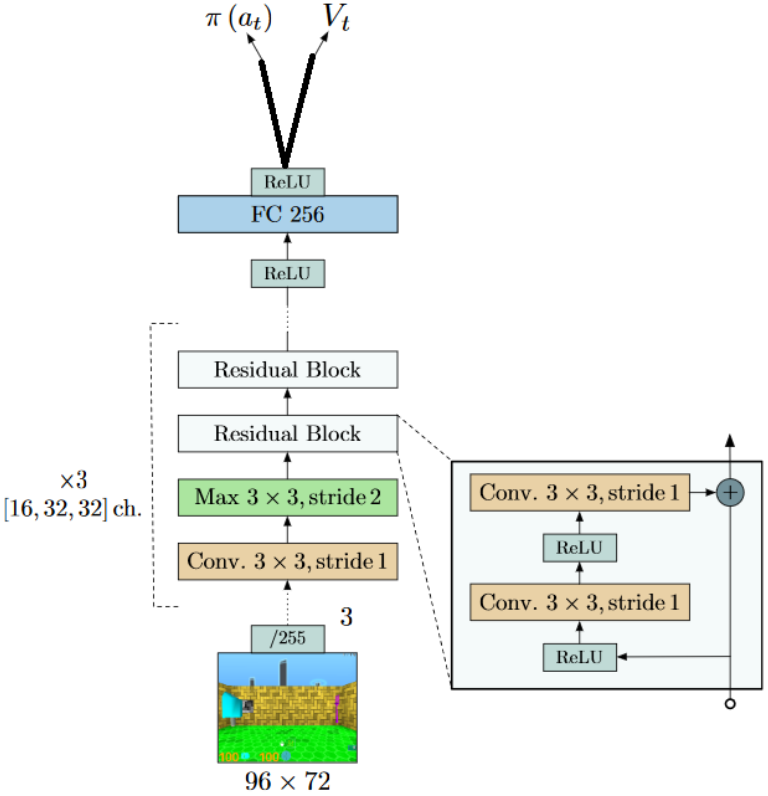
Reference code: https://github.com/albarji/deeprl-snes/

Neural network

Small

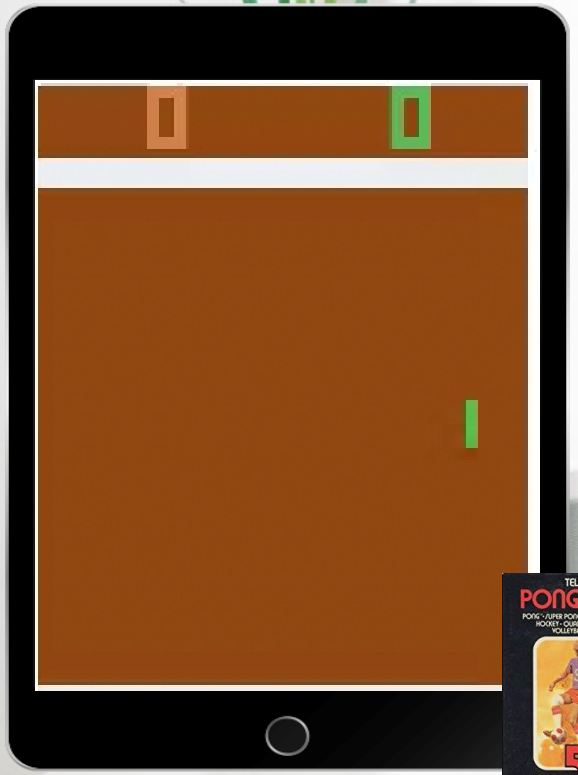


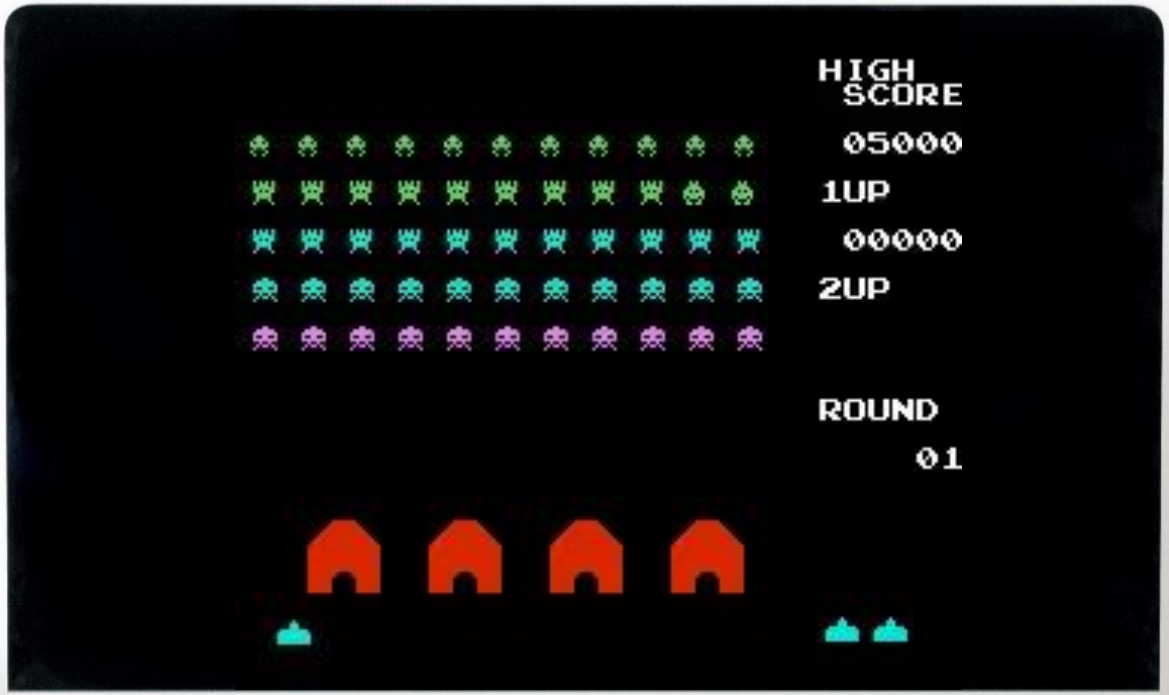
Large

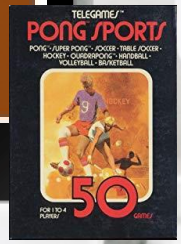


Espeholt et al - IMPALA: Scalable Distributed Deep-RL with Importance WeightedActor-Learner Architectures

RL on simpler videogames








ATARI 2600 VIDEO GAME CARTRIDGE
Q*bert
by JVW Electronics

Reinforcement Learning in Gradius 3

Reward design:

$$reward = increment\ in\ score + pick\ powerup + reduction\ in\ boss\ life\ - death$$

Learning architecture:

OpenAI Gym ›

- Environment definition
- Reward design
- Gym Retro: extensión to emulate retro video games

RAY ›

- Cluster computing library
- Specialized in reinforcement learning
- Includes PPO, IMPALA and many others

- 1 GPU + 32-200 CPUs cluster

Experiencia
BiciMAD

Foto obtenida en: europapress.es

Hemos supuesto que:

- Hay 6 camiones en funcionamiento 24 horas al día
- En cada camión se pueden cargar hasta 22 bicicletas
- Estos camiones pueden visitar, como mucho, 2 estaciones en cada hora
- Cada camión tiene un radio de acción de 3,5 km

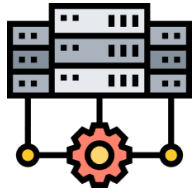
Resolución del problema

Foto obtenida en: europapress.es

Esquema de la solución:

Open Data
EMT Madrid

Frecuencia
horaria



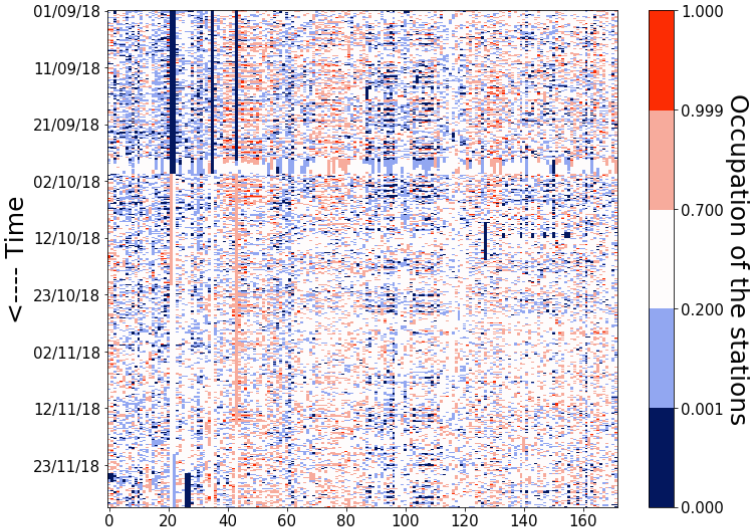
Machine Learning
Predictor

Optimización
Planificador

Los resultados (I)

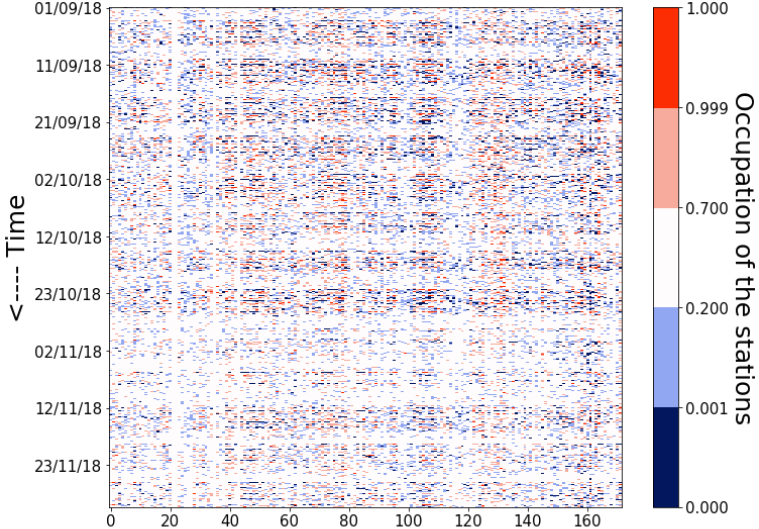
Ocupaciones

Reales del
1 de septiembre al
30 de noviembre de 2018



BiciMAD
white: 60.087
red and dark blue: 6.152
dark blue: 5.046
red: 1.063

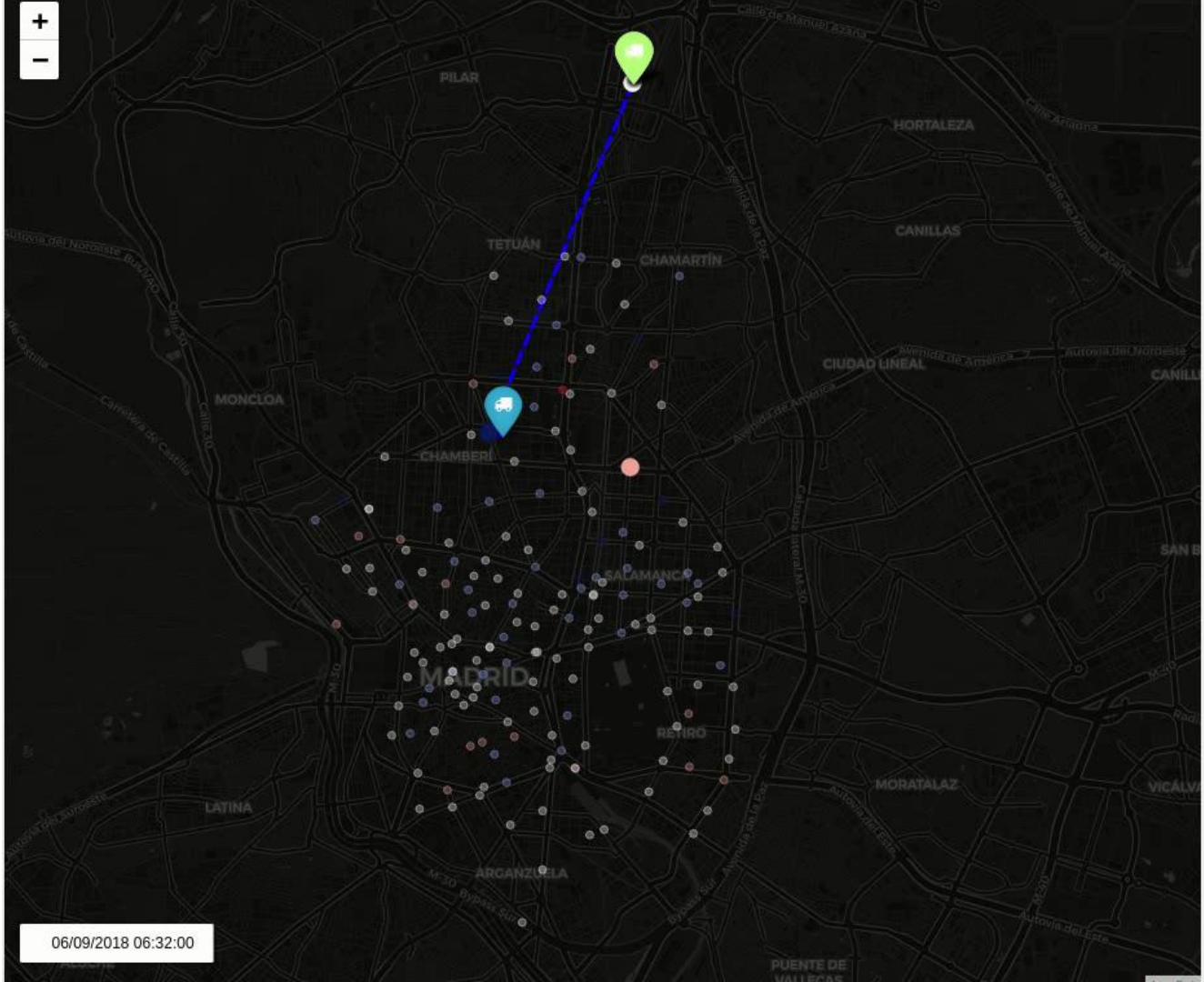
Con nuestra primera solución y 6 camiones del 1 de septiembre al
30 de noviembre de 2018



Resultado 1 con 6 camiones
white: 74.642
red and dark blue: 5.156
dark blue: 3.784
red: 1.371

Demostración

Foto obtenida en: perfil de Twitter Bicimad



Alvaro Barbero

@albarjip

https://github.com/albarji

albarji.deviantart.com

Álvaro Barbero Jiménez

Chief Data Scientist en el Instituto de Ingeniería del Conocimiento

You can check more articles of innovation on our Blog:
www.iic.uam.es/blog/

C/ Francisco Tomás y Valiente, nº 11,
EPS, Edificio B, 5ª planta
UAM Cantoblanco. 28049 Madrid
Tel.: (+34) 91 497 2323

Graphic elements of support obtained in :

Elementos gráficos de apoyo obtenidos en:
designed by freepik.com
pixabay