Deep
Reinforcement Learning

About me

About me

Gabriel Muñoz

Lead data scientist en Intelygenz/Terminus7 

`https://www.terminus7.com/`

Msc. Artificial Intelligence & Computer Science, Universidad de Sevilla

Lecturer in the Msc. Big Data & Data Science, Universidad de Sevilla

Roadmap

1. Objective of the talk
2. Introduction
3. Basic concepts
4. DQN: Deep Q-Networks
5. Policy gradient networks
6. Let's do it
    a. Atari environment
    b. DQN using Keras-rl
7. Conclusions

Objective of the talk

Objective of the talk

The objective of the talk is to give an introduction about deep reinforcement learning.

We will review two of the most successful approaches that join deep neural networks and reinforcement learning algorithms.

Also, we will see some available frameworks for implementing this type of solutions.

Introduction

Definition

“Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence.”

“Reinforcement learning [...] is simultaneously a problem, a class of solution methods that work well on the class of problems, and the field that studies these problems and their solution methods.”

Reinforcement learning: An Introduction, R. Sutton & A. Barto

“The objective of reinforcement learning is to train an intelligent agent that is capable of interacting with an environment intelligently”

Deep Q Network vd Policy gradients, Felix Yu

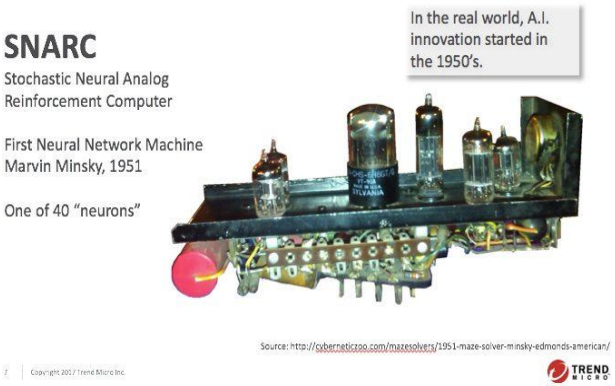
Why reinforcement learning

In my opinion: *The natural way of learning*

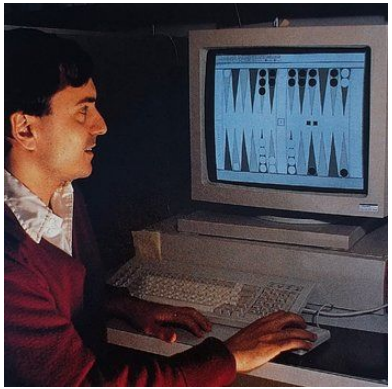
It allows adaptation to an environment

Could help to reduce the human-bias

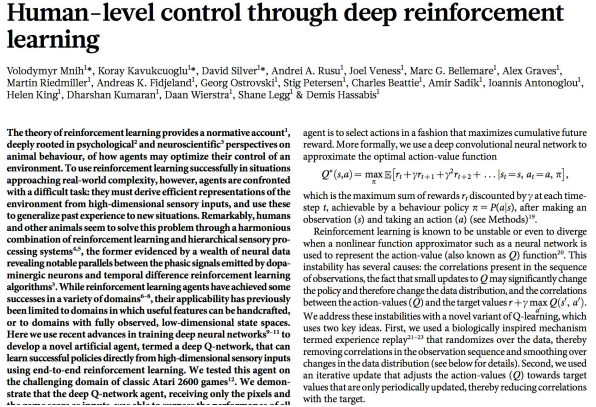
History



Minsky's PHD Thesis (1954)



TD-Gammon (1992) (temporal difference learning)



Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

The theory of reinforcement learning provides a normative account[1], deeply rooted in psychological[2] and neuroscientific[3] perspectives on animal behaviour, of how agents may optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted with a difficult task: they must derive efficient representations of the environment from high-dimensional sensory inputs, and use these to generalize past experience to new situations. Remarkably, humans and other animals seem to solve this problem through a harmonious combination of reinforcement learning and hierarchical sensory processing systems[4,5], the former evidenced by a wealth of neural data revealing notable parallels between the phasic signals emitted by dopaminergic neurons and temporal difference reinforcement learning algorithms[3]. While reinforcement learning agents have achieved some successes in a variety of domains[6–8], their applicability has previously been limited to domains in which useful features can be handcrafted, or to domains with fully observed, low-dimensional state spaces. Here we use recent advances in training deep neural networks[9–11] to develop a novel artificial agent, termed a deep Q-network, that can learn successful policies directly from high-dimensional sensory inputs using end-to-end reinforcement learning. We tested this agent on the challenging domain of classic Atari 2600 games[12]. We demonstrate that the deep Q-network agent, receiving only the pixels and the

agent is to select actions in a fashion that maximizes cumulative future reward. More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi\right],$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a) (see Methods)[19].

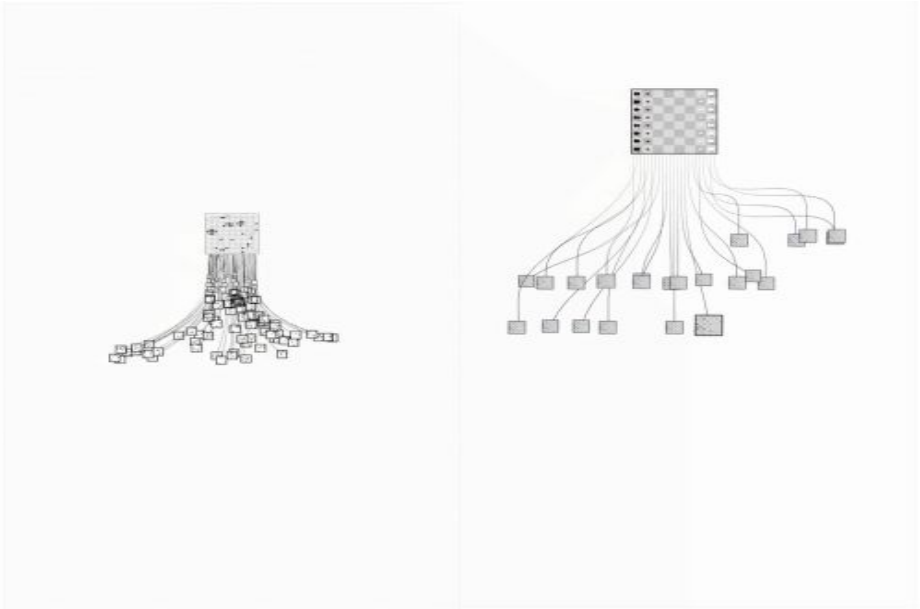
Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) function[20]. This instability has several causes: the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between the action-values (Q) and the target values $r + \gamma \max_{a'} Q(s', a')$. We address these instabilities with a novel variant of Q-learning, which uses two key ideas. First, we used a biologically inspired mechanism termed experience replay[21–23] that randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution (see below for details). Second, we used an iterative update that adjusts the action-values (Q) towards target values that are only periodically updated, thereby reducing correlations with the target.

Deepmind (2015)

Milestones

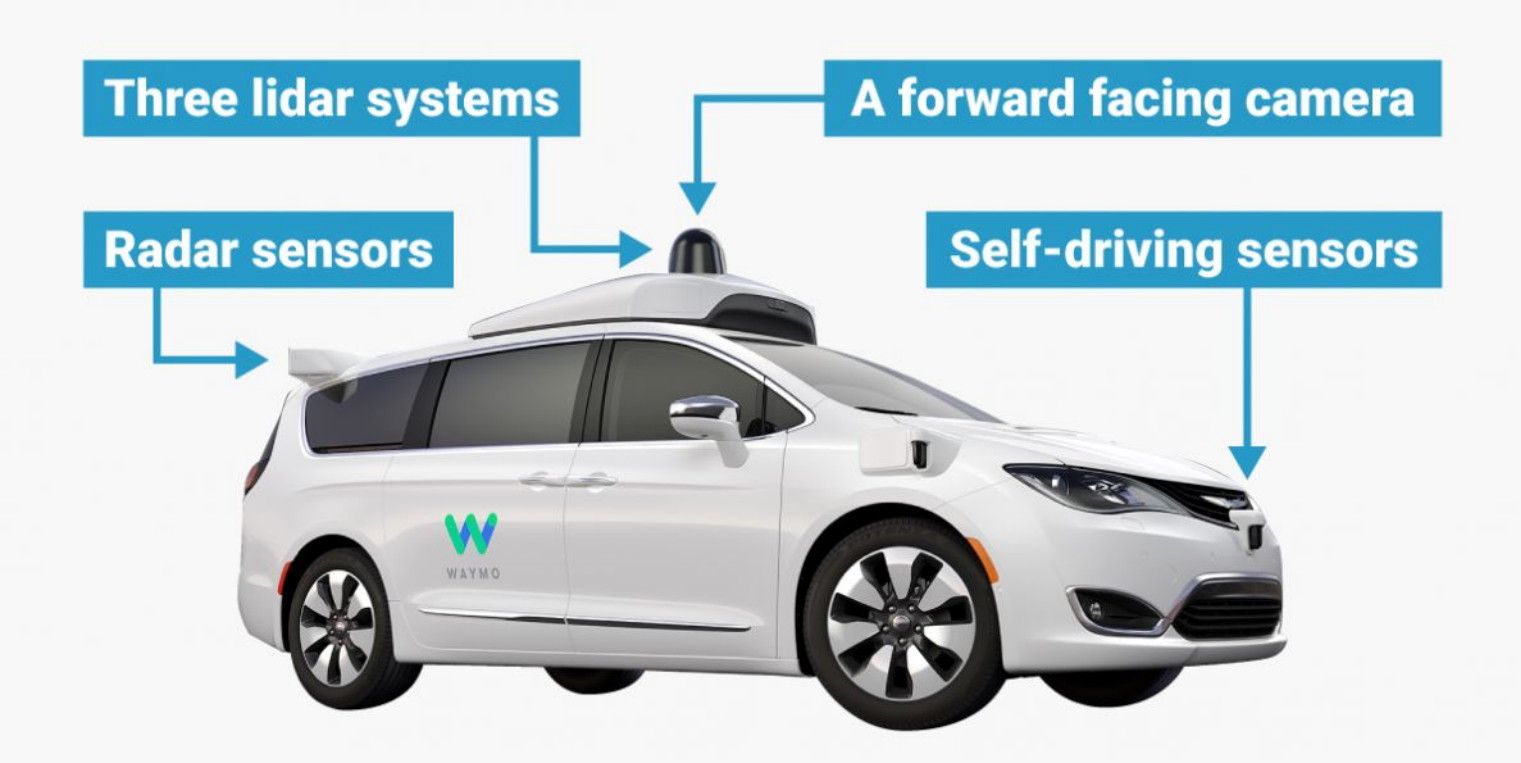
Google DeepMind

Milestones



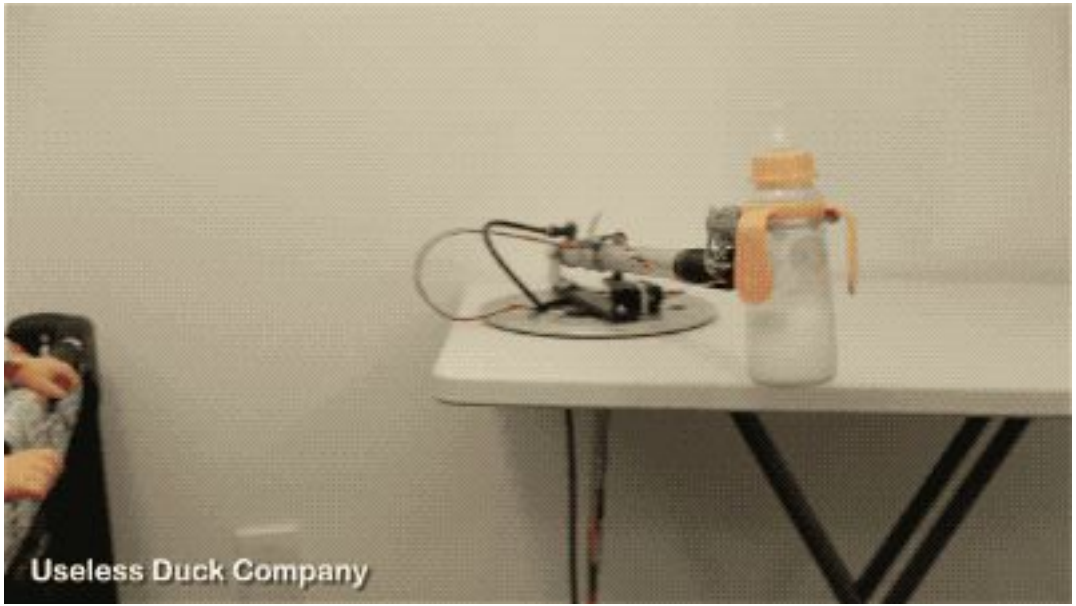
AlphaGo

Milestones


Three lidar systems
A forward facing camera
Radar sensors
Self-driving sensors
WAYMO

Milestones

(Not so good) Milestones



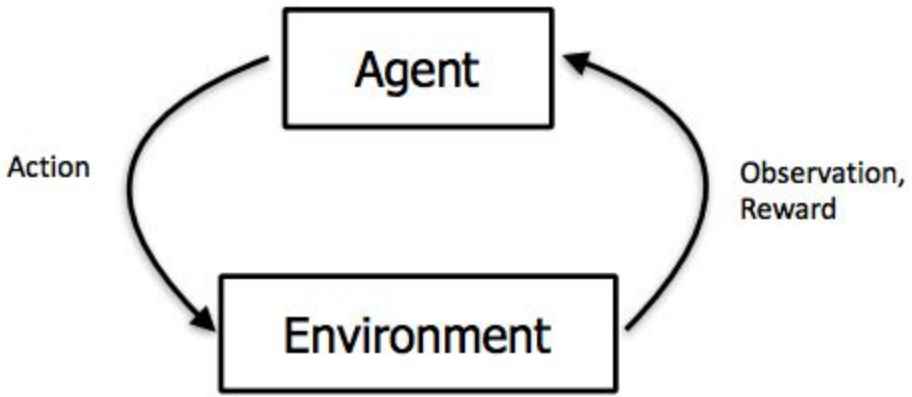
The future

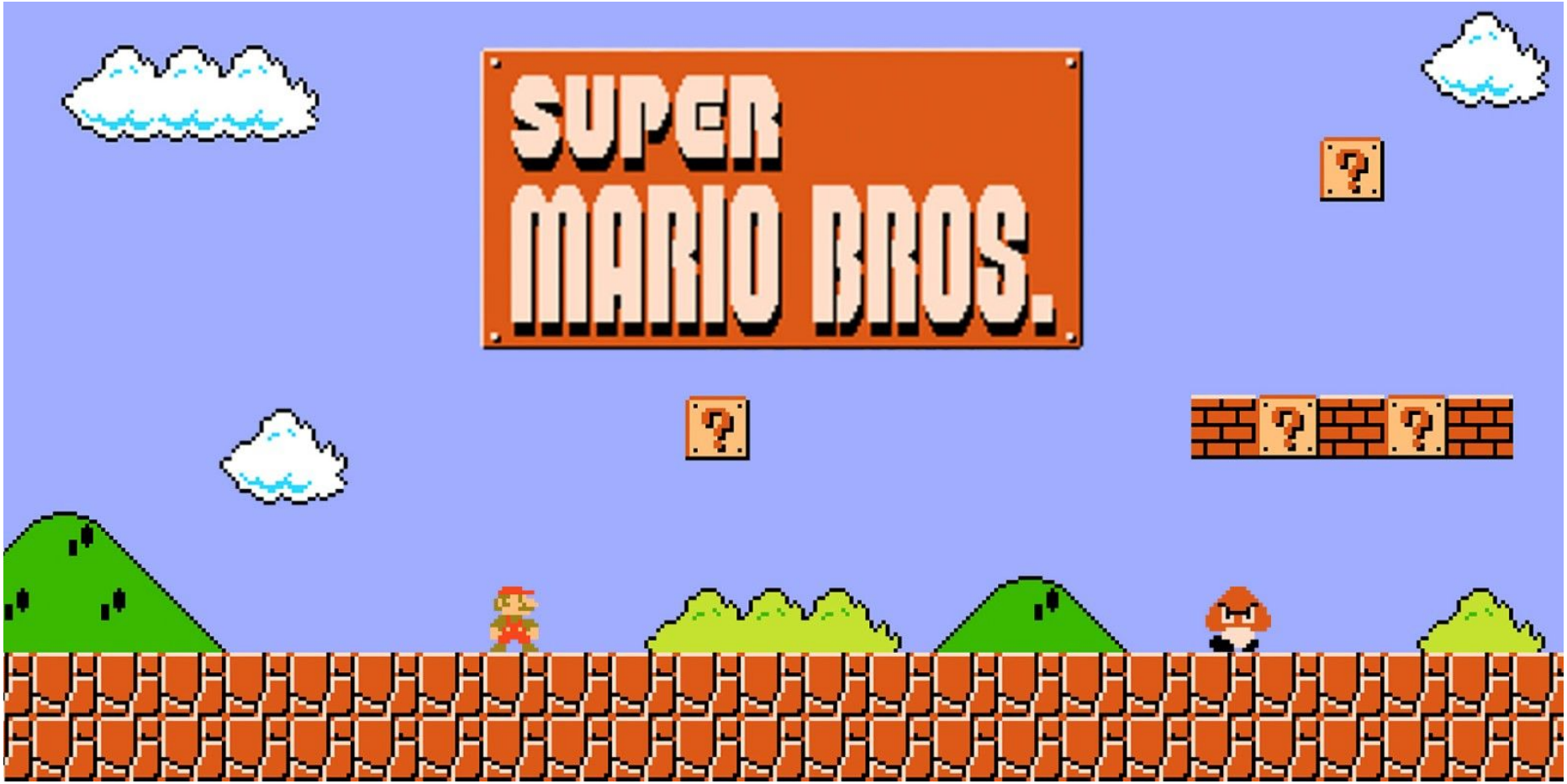
One of the hottest fields for researching.

It could push really interesting data science branches like transfer learning

It is (really) complex. We need to take into account a lot of concepts for developing a correct solution for a problem.

Basic concepts

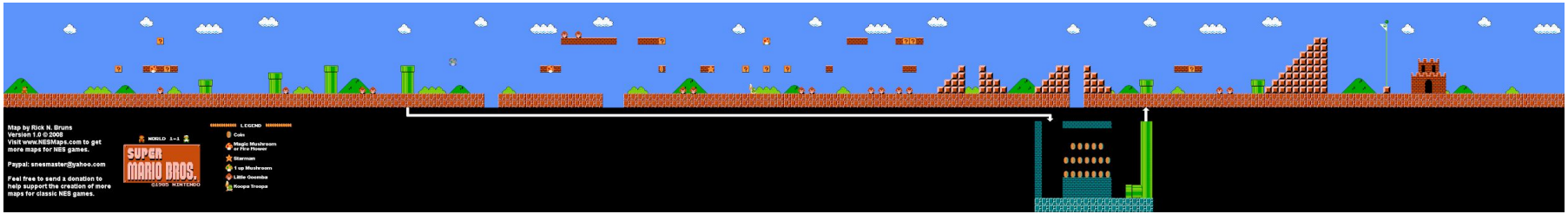

Agent
Action
Observation,
Reward
Environment


SUPER
MARIO BROS.

Environment

The environment is the set of elements where the agent *lives*.

The agent can interact with some of these elements.



Agent

An agent is a computational element that interacts with elements of an environment for achieving a goal.

Generally, the goal is to maximizing a reward (IE: games).

This will be the intelligent element that will use the deep neural network that we will design.

Action

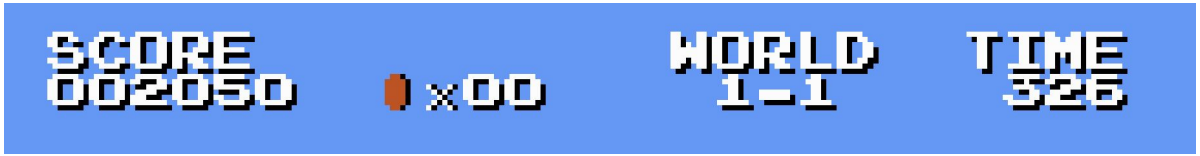
When an agent interacts with an environment, it has a list of available actions that it can perform.

The actions by themselves is a subproblem on reinforcement learning.

Reward

For guide the agent behaviour, we define a reward for the actions.

Using the example of games, the reward could be “get the highest score” or “do not die”.

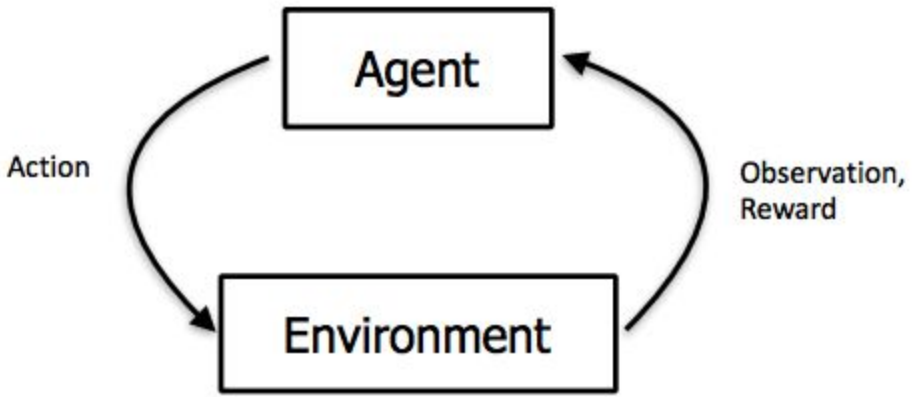


State

Every time that an agent chooses an action, the environment changes.

The snapshot of the environment for every moment is called state.

There's a relationship between the actions chosen by an agent and the states that arise.


Agent
Action
Observation,
Reward
Environment

Needed concepts

S-Space: State space

Categorization or discretization of the situation and the environment in different states: S_1 , S_2 , S_3 ...

A-Space: Action space

Categorization or discretization of the possible actions that the agent can carry out: a_1 , a_2 , a_3 ....

Iteration (or step):

Iterative process of trial-and-error observing the changes of state that occur when executing actions.

$$\text{State } S_t \text{, Action } a_t \rightarrow \text{Next State } S_{t+1} \text{, Reward } r_t$$

Needed concepts

Episode

Complete set of iterations followed in a time interval.

π : Policy

Function that defines how the agent will behave in a state

π(S) ---> action

The agent aspires to achieve the best behavior or what is called the **optimal policy** π^*

Needed concepts

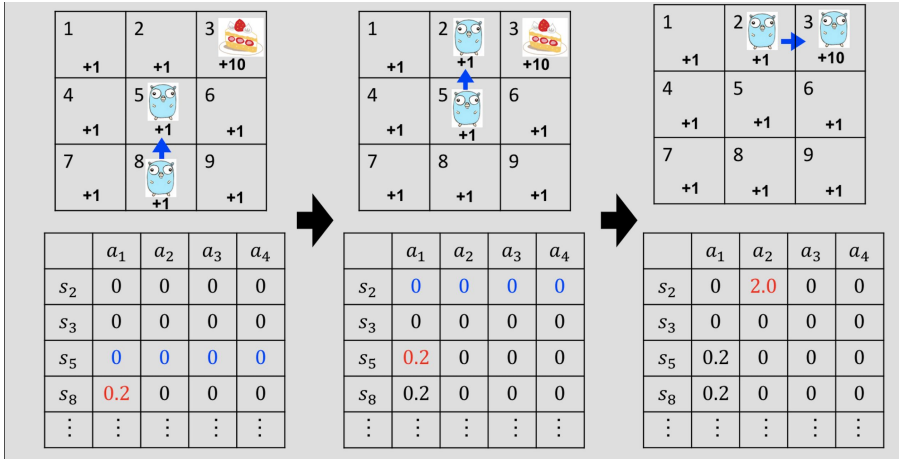
For our algorithms, we will need a tuple called **Experience**.

In each time step:

$$\textbf{State } S_t, \textbf{ Action } a_t, \textbf{ Reward } r_t, \textbf{ Next State } S_{t+1}$$

Deep Q-networks

General view



The first algorithm that we are going to study is Q-learning.

We can see our problem as a Markov chain. So, based on theory, there's a function able to model which action is better in each state.

Using this function, we can find the optimal policy.

General view

$$Q^{\pi}(s,a) = E[R_t]$$

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$\pi^*(s) = arg\max_a Q(s,a)$$

The Q-function relates states with actions. The Q means *quality*.

Basically, we have a function that returns "the expected value of the sum of future rewards" taking an action *A* in a state *S*.

But...what if the problem is really complex? How we obtain that function?

We approximate the function using DNN.

General view

Ok, but...

How we choose the actions at the beginning?

How I know what action will be better in the future?

Exploration

First, **exploration** phase.

We set an Epsilon value to perform random choices. With these random choices, we explore the environment to check out what actions are better for a state.

This exploration returns the rewards for the pairs (state, action).

Exploitation

Once Epsilon is small enough, we start the next phase called **exploitation**.

In this phase, we have explored the environment for a number of iterations and we have knowledge for exploiting the optimal actions for the states.

Bellman equation

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

Bellman provides a recursive definition for the optimal Q-function.

The $Q^*(s, a)$ is equal to the summation of immediate reward after performing action a while in state s and the discounted expected future reward after transition to a next state s'.

Bellman equation

$$L = \frac{1}{2}[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

As we want to approximate the Q-function with our DNN, we need a loss for minimizing the error.

We can use the Bellman equation to **iteratively approximate the Q function through temporal difference learning**. Specially, at each time step, we seek to minimize the mean squared error of the Q(s,a) (prediction term) and Bellman equation (target term)

DQN algorithm challenge

This will be the algorithm that we are going to use in the practical block.

Actually, the environment will be the Atari environment, so we will teach our agent to play by itself to Atari games!

But...

If you try this algorithm like it is defined, you will have issues with convergence and learning. Let's check some needed advanced concepts.

Advanced concepts

- Target network
- Memory buffer

The objective is to stabilize the network learning.

(We can go into details with our practical example)

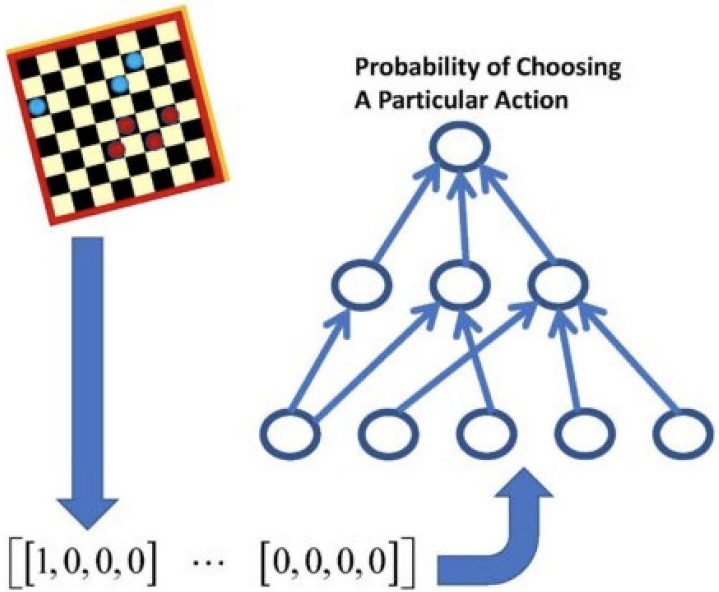
DQN algorithm

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Policy gradient networks

Policy gradients



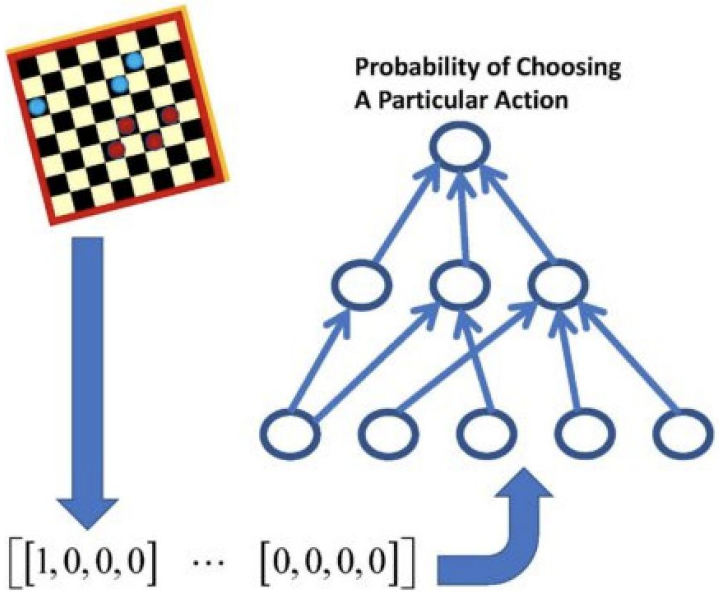
In the previous algorithm, we wanted to approximate a function, the Q-function.

Then, we used this function to get the optimal policy for our problem.

What if we approximate directly the policy?

Welcome policy gradients.

Policy gradients



In this method, we want to model the action-selection.

We will obtain an array of probabilities related to each available action for an agent in a state.

So, we update the weights of the actions in each iteration according to maximizing the expected reward.

Policy gradients

$$\nabla_\theta E[R_t] = E[\nabla_\theta log P(a) R_t]$$

How we achieve this?

"R_t is the scaling factor which dictates how the P(a) should change in order to maximize the expected future rewards. If action a is good (i.e. large Rt), P(a) will get pushed up by a large magnitude, on the other hand, if action a is bad (i.e. small or negative Rt), P(a) will be discouraged. Eventually, good actions will have an increased likelihood to get sampled in future iterations".

Policy gradients algorithm

function REINFORCE
    Initialise θ arbitrarily
    for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
        for $t = 1$ to $T - 1$ **do**
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
        end for
    end for
    return θ
end function

Let’s do it

Atari environment

Our environment will be the Atari environment.

This environment has become itself almost a benchmark for reinforcement learning solutions.

It includes +50 Atari 2600 games.

For installing the package:

```
pip install atari-py
```

```
https://github.com/openai/atari-py
```

Atari environment

This environment is ready to use it. It works like we have studied, with a wrapper of the environment that allows us to interact with the environment easily.

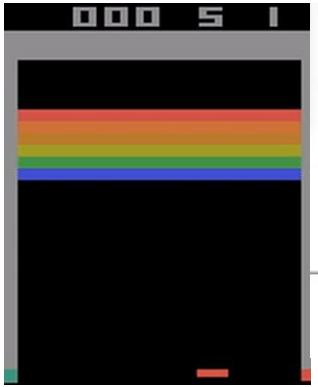
- How to change the state?
- Actions?
- Reward?

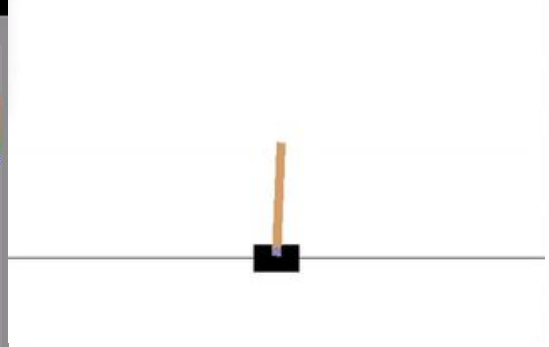
All of these questions are performed by the environment.

Keras-rl

Once we have the environment ready, we need a framework to use the concepts.

The selected framework is **keras-rl**. Keras-rl is a high level framework that acts like a wrapper for RL algorithms.





```
pip install keras-rl
```

```
https://github.com/keras-rl/keras-rl
```

Other frameworks

Tensorforce

https://github.com/reinforceio/tensorforce

Baselines

https://github.com/openai/baselines

Our example

We are going to develop a reinforcement learning agent for playing to Breakout!

We will follow the code line by line for explaining what's happening.

Conclusions

Conclusions

1) Now, Reinforcement learning has a lot of opportunities for researching
2) It is very hard to develop a good Reinforcement learning solution
    a) Reward
    b) Exploration-Exploitation tradeoff
    c) ...
3) From a learning perspective, is a natural way of learning. It allows adaptation.
4) It is fun!

Bibliography

Bibliography

Reinforcement learning: An Introduction, R. Sutton & A. Barto

“Deep Q Network vs Policy Gradients - An Experiment on VizDoom with Keras”, Felix yu
https://goo.gl/Vc76Yn

“Deep Reinforcement Learning: Pong from Pixels“, Andrej Karpathy
https://goo.gl/8ggArD