Ananas

Deep Learning - Project

Clément, Grégoire, Nathan

January 31, 2025

Our Goal

What is our goal

Our goal is to use Reinforcement Learning to drive a car.

Our implementation will be based on the paper *Playing Atari with Deep Reinforcement Learning* by Google Deepmind.

It will have access to:

- A small vision cone of what is in front of the car.
- The relative position of the goal compared to the car and its direction.
- The speed of the car.

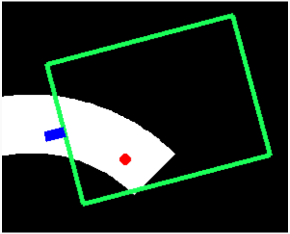


Figure 1: Vision cone of the car

The Environnement

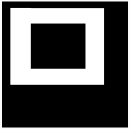
For reasons of time efficiency, we have reimplemented our own environment rather than using a ready-made one. It is optimized with pre-computed maps.

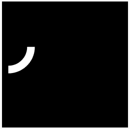
The car has 4 possible actions:

- Accelerate
- Decelerate
- Turn left
- Turn right

The situations we will put our car in

- Learn to drive on 1 long track.
- Choose between multiple roads in function of where the goal is.
- Learn to drive on multiple medium size tracks at once.
- Learn some patterns to be able to drive on tracks never seen before.





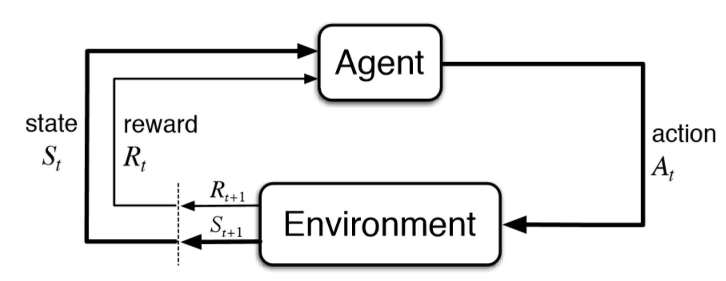
Reinforcement Learning in Therory

What is the purpose?

Agent evolving in an environment

For each action, rewards or penalties.

Objective: Learn to maximize rewards



Mathematical Formulation

The agent interacts with a stochastic environment $\mathcal{E}$ in which it plays games consisting of a set of states, scores, and actions.

At each step, it selects an action a_t from a set $A = \{1, ..., 4\}$ of legal actions, and recieves a reward.

At each step, the agent has access to a set of information $x_t \in \mathbb{R}^d$ and has to make the optimal choice, that will maximize the reward in the long run.

Agent's Objective

We define the expected future return as

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

where T is the time at which the game ends,
and where $0 < \gamma < 1$, generally $\gamma \approx 0.99$.
The agent's goal: maximize this return.

Optimal Action Value Function

We define the optimal action value function $Q^*(s', a')$, which is used to have the best choice to make based on the current situation, meaning:

$$Q^*(s, a) = \mathbb{E}_\pi[R_t | s_t = s, a_t = a, \pi]$$

If we have access to this function, then it is sufficient to always make the best choice.

How to Compute This Function

This function follows Bellman's equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \cdot \max_{a'} Q^*(s', a')|s, a]$$

Thus, if we define

$$Q_{i+1}(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q_i(s', a')|s, a]$$

then Q_i converges to Q^* as i approaches $+\infty$.

Problem: $O(|\mathcal{A}|^i)$ possibilities.

Full training loop

Initialize replay memory M to capacity N; Initialize networks;
for *each episode* **do**
    Initialize state s; **for** *each step in episode* **do**
        Select a using ϵ-greedy policy from $Q(s, a; \theta)$;
        Execute a, observe r, s';
        Store (s, a, r, s') in M;
        Sample mini-batch (s_j, a_j, r_j, s'_j) from M;
        Learn from the mini-batch and update the model.
        If the episode terminates, **Then** break;
    end
    Decrease ϵ
end

How to learn from each minibatch

Input: Mini-batch (s_j, a_j, r_j, s'_j) sampled from memory

Compute target:

- **If** s'_j is terminal, **Then** $y_j = r_j$
- **Else** $y_j = r_j + \gamma \max_{a'} Q'(s'_j, a'; \theta^-)$;

Compute loss: $L(\theta) = \frac{1}{m} \sum_j (y_j - Q(s_j, a_j; \theta))^2$;

Update θ using gradient descent;

Periodically update target network: $\theta^- \leftarrow \theta$;

We use two models, this way we stabilize training.

Results and ablation study

Reward specification

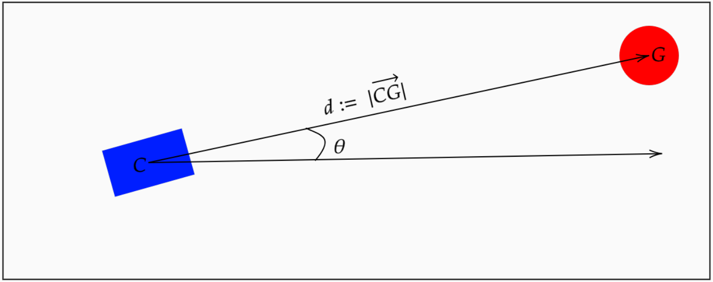


Figure 2: Illustration of needed variables

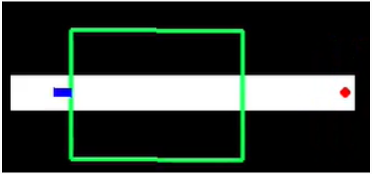
$$\frac{100}{(1 + d/10)^2} + \cos(\theta) + \text{efficiency} + \text{speed penalty}$$

Fixing constants

- Mem Size: 30,000 (100,000 in the lab)
- Epsilon Decay: $10^{-3}/10^{-4}$ (10^{-5} in the lab)
- Cone Dimension: 100 $\times$250

This is already 6 to 15 Go of ram.

Learning to drive a straight road



Even a model without any vision can learn this task.

Learning to drive a curve

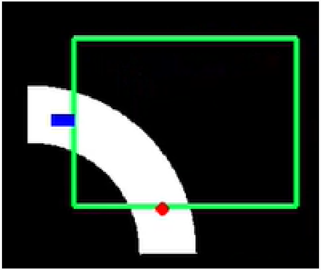


Figure 3: Curved road

Both goal-only and MLP-based models don't work on this example.A simple convolutionnal model (1 convolution and 1 pooling) worked well.

Learning primitives

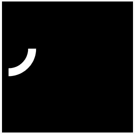


Figure 4: Curved Left

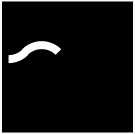


Figure 5: Double Virage

Figure 6: Décaler

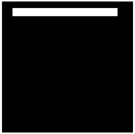


Figure 7: Straight Road

A simple convolutionnal model can learn this training set. But, can it generalize ?

Final Results

Live demonstration.