Deep Reinforcement Learning for the game of Snake

Yannick Terme

ENSAE ParisTech

07/05/2018

Introduction: the game of Snake

The game got famous in 1998 when it appeared on a Nokia phone.

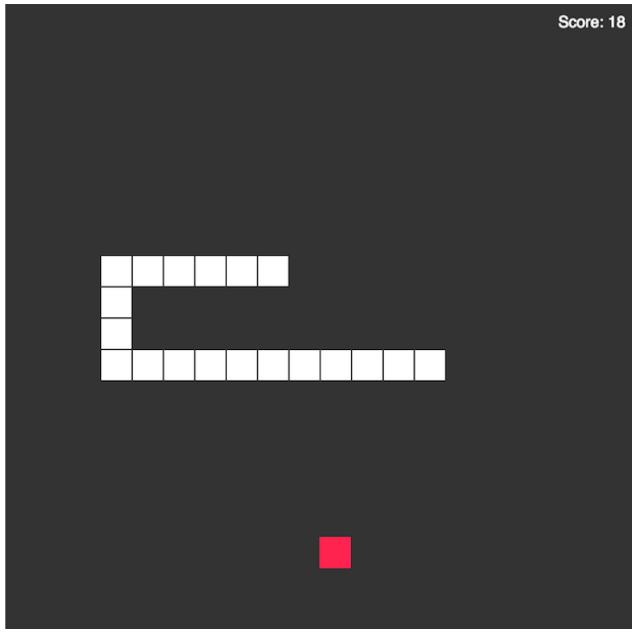


Figure 1: The snake must eat the apple

A simple solution: shortest path

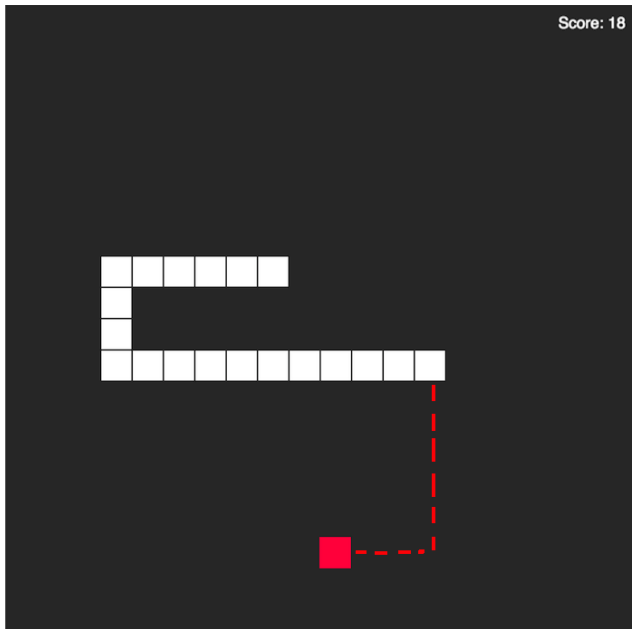


Figure 2: Snake agent working with the shortest path

- simple to implement
- no training
- but ... may not perform well in the advanced steps of the game

$\rightarrow$ We will use it as a benchmark

The Reinforcement Learning approach

Reinforcement Learning (RL) in games recently had much success and produced superhuman results in hard games such as Go, chess.
I will present Deep RL for Snake.

The RL approach

$$\text{Agent (player)} \xrightarrow{\text{action}} \text{Environment}$$
$$\text{,,} \quad \xleftarrow{\text{reward,state}} \quad \text{,,}$$

Rewards policy:

- Apple: $+1$
- Death: -1
- Otherwise: 0

The agent

We don't want to explicitly teach the agent how to play. At every time step, the input of the agent is:

- the state: a NxN matrix of pixels
- the reward

At every timestep, its output is:

- an action in A=$\{$right, left, up, down$\}$.

A game forms is a set of states, actions and rewards: $(s_0, a_0), (s_1, a_1, r_1), (s_2, a_2, r_2), ..., (s_n, a_n, r_n)$ The final reward of the game is:

$$R = r_1 + r_2 + ... + r_n$$

The Q function

We introduce the discounted reward at time t:

$$\begin{aligned} R_t &= r_t + \gamma r_{t+1} + ... + \gamma^{n-t} r_n \\ &= r_t + \gamma R_{t+1} \end{aligned}$$

γ represents the uncertainty in the future. We prefer to have a reward now than later. We pick $\gamma \approx 0.9$.
Let's define the **Q function**:

$$Q(s_t, a_t) = \max_{\pi} R_{t+1}$$

It is the maximum total reward that can be reached if we play with the optimal policy. Q will assign a score to a set (state, action). If we know Q, the agent according to a greedy policy:

$$\pi(s) = \arg\max_{a \in A} Q(s, a) \tag{1}$$

Q learning: estimate the Q function

We know that Q verifies the **Bellman equation**:

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

The idea of Q-learning is to iteratively approximate the Q function using the Bellman equation.

initialize Q[s,a] arbitrarily
while convergence condition is not met **do**
select and carry out an action a
observe reward r and new state s'
$Q[s, a] = Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
s = s'
end while

Deep Q network

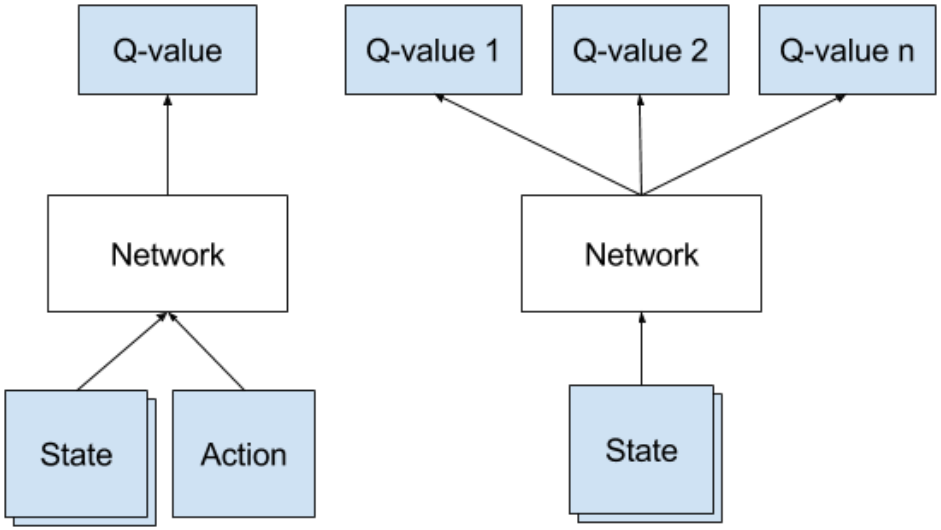


Figure 3: *Left*: Naive formulation of deep Q-network. *Right*: More optimal architecture of deep Q-network, used in DeepMind paper.

We will use the network on the right. Its output is a 4-uplet. The loss is:

$$L = \frac{1}{2}(\overbrace{r + \gamma \max_{a'} Q(s', a')}^{\text{target}} - \overbrace{Q(s, a)}^{\textit{prediction}})^2$$

Compute the target of the Q Network

For a sample (s, a, r, s'), the target is computed:

if s' is terminal **then**
    $y[a] = r$
else
    $y[a] = r + \gamma \max_{a'} Q(s', a')$
end if
for $a' \neq a$ **do**
    $y[a'] = Q(s, a')$
end for

Figure 4: Calculate the target y

Training the agent: some important points

- **The exploration/exploitation dilemma**: the agent will play a random action with a probability ϵ. The higher ϵ the higher the exploration.
- **Experience replay**: the data used for training is randomly sampled from the past experiences. Thus, the network does not "forget" past experience.

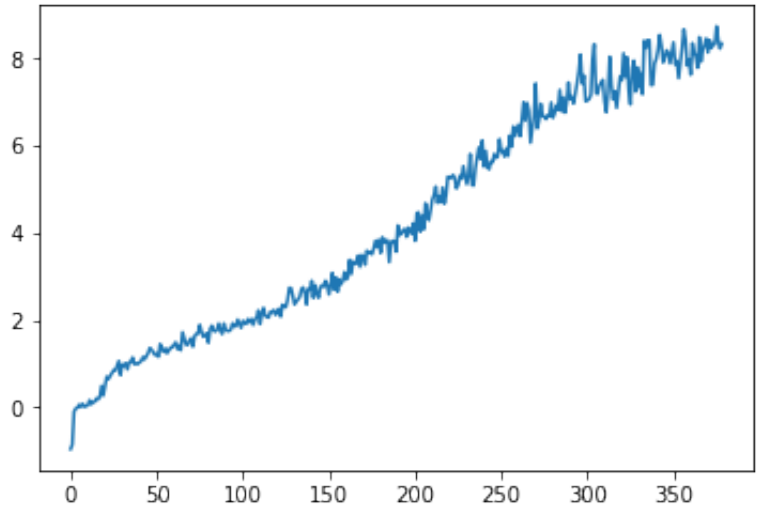
Training the agent

The training of the agent is done with the following procedure:

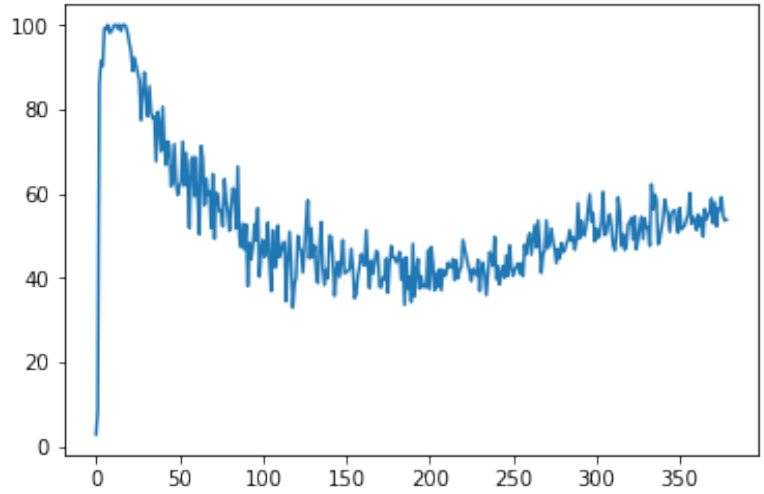
while training **do**
    for n experiences **do**
        Experience episodes
        Add experiences to the dataset
    end for
    Sample random experiences from the dataset
    Compute the target of the Q network
    Train the Deep Q network for 1 epoch
end while

Experiments

Training the network on a 5x5 grid, with a 2-layer fully connected neural network.



(a) Final scores



(b) Lengths of episodes

2000 rounds, 4 hours of training.

1. In the first rounds of training, the agent learns to survive.
2. Then it slowly learns to eat the apples.

After training, mean score of 10 to 11 apples per game. Better than shortest path (≈ 8.5 per game) but far from optimal.

Now the difficulty is about:

- finding the adapted network architecture
- properly tuning the hyper-parameters (learning rate, exploration ϵ, ...)
- dealing with the computational problems

Deepmind uses CNN because of the translation invariant properties.