# Deep reinforcement learning for Snake

Louis Martin & Pierre Stock

Ecole Normale Supérieure de Cachan, MVA

January 17, 2017

# Summary

Reinforcement learning on games:
TD-Gammon in 1992

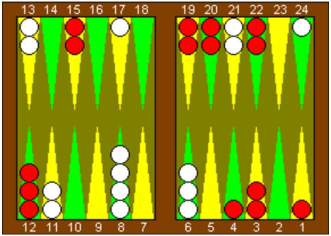


Figure 1: TD-Gammon algorithm from [1]

# Introduction

## Working with images

- Images as input: Hard problem, high dimensional states.
- Historically: Hand crafted low dimensional features
- Recent revival of convolutional networks

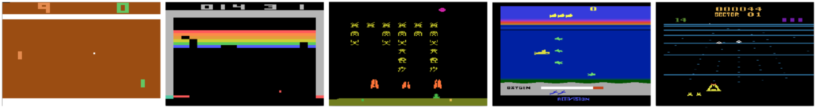

Figure 2: Screen shots from five Atari 2600 Games: (Left-to-right) Pong, Breakout, Space Invaders, Seaquest, Beam Rider, taken from [2]

Policy gradients gained interest over Deep Q-learning.


Figure 3: AlphaGo against the world champion of the game of Go [3]

# Notations and Background

- Current state $s_t = N \times N$ grid, action state $\mathcal{A} = \{$up, down, left, right$\}$.
- Each action $a_t$ modifies the state and triggers a reward $r_t$ : $\{$eat: $-10$, hit: $-10$, fruit: $+10$, nothing: $-1\}$
- Goal of the agent: find optimal policy $\pi^\star$ that maximizes

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

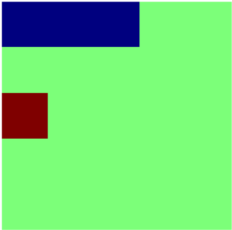

Figure 4: Initial state of the game for $N = 5$

# Policy Networks, introduction

**Challenges**: detect motion, exploration/exploitation dilemma

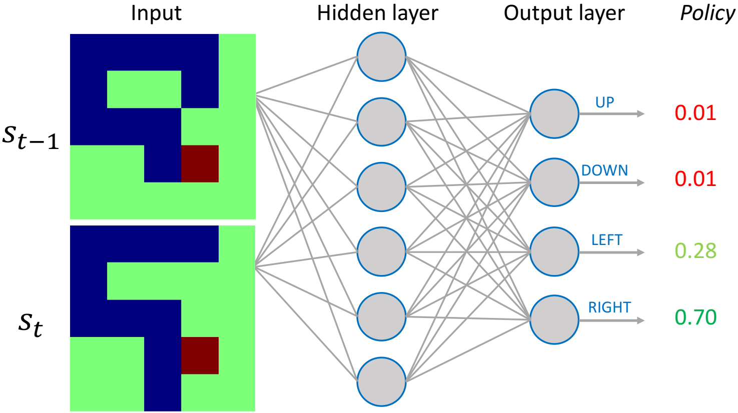


Figure 5: Network architecture

# Policy Networks explained

**Idea**: learn the policy by playing batches of games and updating the network parameters by backpropagation afterwards when we know if a particular move was conclusive

**Algorithm 1** Learning algorithm

Randomly initialize network weights $\theta^0$
Initialize network gradients $d\theta = 0$
**for** $j = 1$ .. #iterations **do**
    **for** $i = 1$ .. $n_b$ **do**
        play game $i$
        store corresponding expected returns $(R_{i,1}, \ldots R_{i,T_i})$
        store corresponding probabilities $\log p(y_{i,t}|x_{i,t})$
    **end**
    compute loss $\ell_j$
    update network parameters $\theta^{j+1} = \theta^j - \eta \frac{\partial \ell}{\partial \theta}$
**end**
**return** network parameters $\theta$

Figure 6: General Policy Gradient Algorithm

# Policy Networks explained

- Loss function depending on the batch

$$\ell_k = -\sum_{i=1}^{n_b} \sum_{t=1}^{T_i} R_{i,t} \log p(y_{i,t}|x_{i,t})$$

  with $p(y_i|x_i)$ is the probability of action $y_i$ given by the network when it sees input $x_i$, $R_{i,t}$ the expected return at time $t$ of the action $y_{i,t}$.
- $R_{t,i} =$ *advantage* of the game $i$ at time $t$, depends on the discount factor $\gamma$.
- Normalize the expected rewards $R_{i,t}$ per batch
  - Diminish the variance of the gradient
  - Roughly half of the games of the batch *bad* the other *good*

# Network architecture & Setup

- Python implementation available online[1].
- **Parameters**: learning rate, set of rewards, batch size, hidden layer size, grid size, $\gamma$, number of iterations
- Each set of parameters was run for 10.000 games
- We experimentally observed low inter-training variability
- **Performance measure**: number of time the Snake played without loosing, along with the number of fruits eaten
- To avoid infinite loops, we set a constraint on the time spent without eating fruits ($3 \times$ *grid_size* actions) after which the game is lost

[1] GitHub repo: `http://github.com/RLSnake/Snake`

# Demo

Show time !

# Results

## Parameter exploration

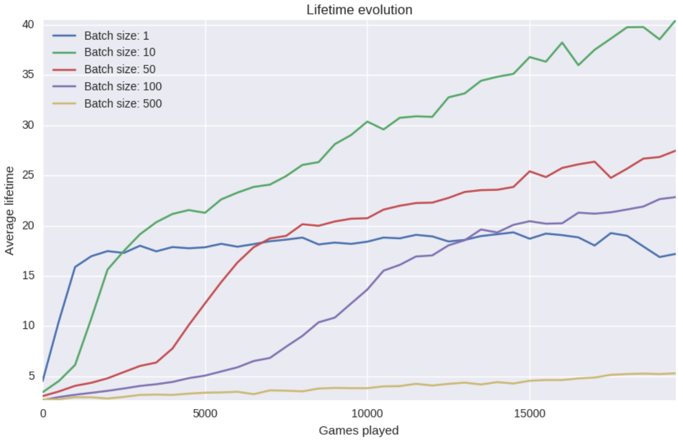


Figure 7: Influence of the batch size

# Results

## Parameter exploration

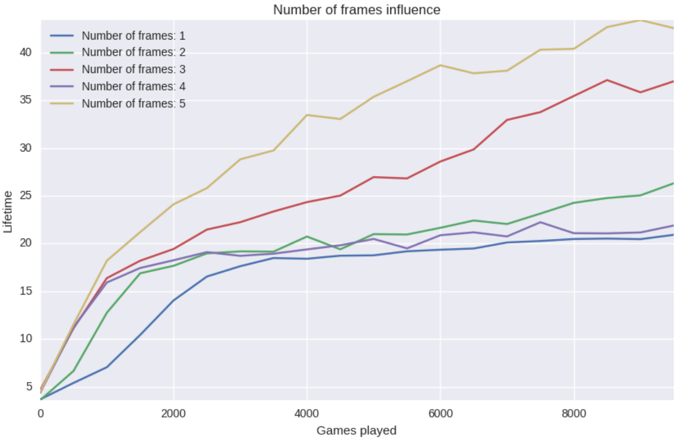


Figure 8: Influence of the number of frames fed to the network

# Results

## Parameter exploration

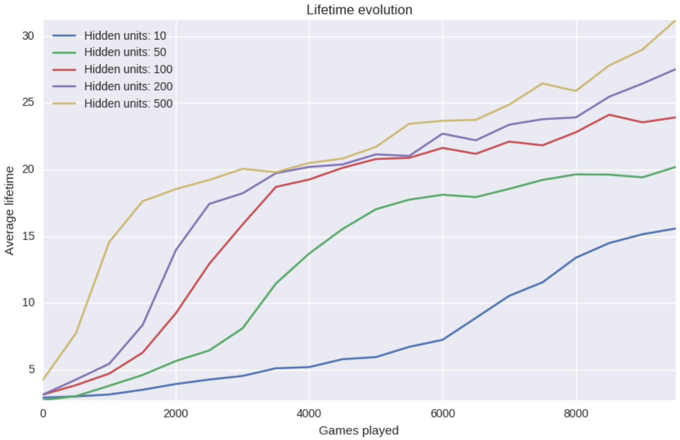


Figure 9: Influence of the number of hidden units

# Conclusion & Perspectives

- Deep Reinforcement techniques yield promising results to play games which require good reflexes and a bit of a long-time strategy
- After training over thousands of games, our snake eats fruits consistently until it grown too much and is eventually stuck by its own body
- Real-world applications (e.g. robotics)

Thank you for listening !
Any questions ?

# References

Gerald Tesauro.
Temporal difference learning and td-gammon.
*Communications of the ACM*, 38(3):58–68, 1995.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller.
Playing atari with deep reinforcement learning.
*arXiv preprint arXiv:1312.5602*, 2013.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis.
Mastering the game of Go with deep neural networks and tree search.