# Autonomous '*pong*' player-agents

Machine Learning Term Project

**Authors**: Rodrigo Arias <rodarima@gmail.com>
Cedric Bhihe <cedric.bhihe@gmail.com>

Delivery: before 2018.06.22– 23:55
UPC – FIB / MIRI Program

## Table of Contents

# A. Introduction

This report describes the study of a closed system consisting of two autonomous and independent, temporally situated, learning agents, playing a game of *Pong*[i]. Each-agent-player must overcome its opponent by learning to play the game better and faster in order to score points. An agent is computationally autonomous in that it *learns* to interact with its environment by being rewarded, whenever its scores, and penalized whenever its opponent scores. The goal-directed machine learning (ML) methods of choice in our case are reinforced learning (RL) methods, which we set out to implement and benchmark. Pong is a simple game and its rules are outlined in the framed inset at the end of this introductory section. We choose to focus not on the implementation of the game[1], although it is far far from being devoid of interest, but rather on that of the ML methods we propose to study. By endowing the two player-agents with different characteristics and learning method's parameters, we set an explicit objective for them: to maximize their own score. For that we make them aware of their environment in a manner detailed later. Our goal is to compare the relative performances of different ML methods.
Apart from the simplicity of the game, there are two ML-related main reasons to choose Pong to study the relative performance of different RL methods.

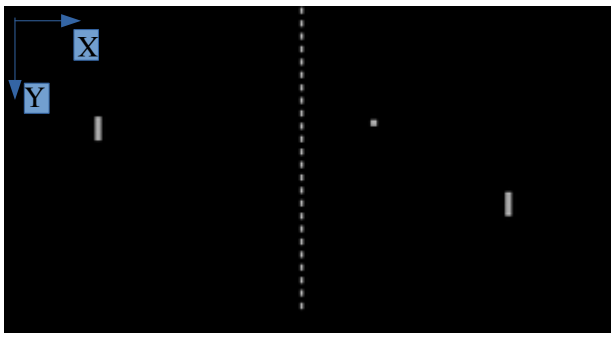
1) Pong has two players. It affords us the possibility to either pit one learning agent against another or to appraise an agent-player's learning curve when opposed to a human player or to a training wall. This permits the design of a parametric study of learning performance, as a function of learning methods' parameters. We can also allow two (differently configured) learning agents to compete in gradually more complex learning environments, i.e. environments with increasing numbers of actions and states.

2) Given the nature of the problem, we can study several RL methods[2], [3], in particular:

- basic, off policy, model-free Q-Learning (QL), parametrized by:
  - reward,
  - discount factor;
  - learning rate or "step size"
- basic on-policy State-Action-Reward-State-Action (SARSA), parametrized by:
  - reward
  - ….
  - ….
- Deep Q Neural-Networks (DQN), parametrized by:
  - reward
  - ….
  - ….

**The simple game of 'pong'**

The game consists of a rectangular arena (in the XY plane), a ball, and two paddles to hit the ball back and forth across the arena. A player-agent (represented by a paddle) scores when the ball flies past the opposite player's paddle and hits the back-wall opposite the scoring player's side. When this occurs a new episode, made of a sequence of exchanges, starts.
Each player can only move vertically (i.e. along direction Y).
The ball can bounce off the paddles as well as the side walls running parallel to axis X.




At writing time, we have no guarantee, that we can include every above-mentioned RL method in our final results.

Our report is organized as follows:

In section B-1 we briefly review the fundamentals of the 3 above-mentionned RL methods.

i The game of 'pong' is one of the earliest video games, released in 1972 by Atari. It is built with simple 2D graphics.

Section B-2 is devoted in some detail to the implementation of Q-Learning from a simple 2-action 3-state problem to an |A| -action, |S| -state one, where:

- |A| denotes cardinality of A, the set of all possible actions *a*, greater than or equal to 2 and
- |S| denotes cardinality of S, the set of all possible states *s*, is greater than or equal to 3.

In particular we give an account of how we experimented moving away from a greedy action-picking policy mechanism to an $\varepsilon$-greedy policy mechanism, based on the current reward matrix. We report intermediate results.

In section C, we outline results and experimental observations.

In section D, we analyze our results and draw conclusions based on them. We further suggest sensible practical extensions to our work, in order to explore new methods in potentially interesting directions.

At the end a "References" section is followed by 4 Appendices. The first one, Appendix A, presents a pseudo code for our general implementation. Appendices B through D list our 3 code sections in their entirety along with numbered code lines for easier referencing throughout this report.

# B. Road-map to implementing Reinforced Learning methods

We wish to explore a computational approach to learning from interaction with one's environment. The circumstances of such goal-directed learning experiments are schematically represented by Fig. 1. We consider only simple model-free systems, where learning agents cannot envision how their action will change their current state.

QL, SARSA and DQN adapt particularly well and in different ways to incomplete knowledge situations and never-seen-before states, where - as is our case - player-agents are memory-less and rely on a limited set of actions to maximize their utility. Each player's action is rewarded by a numerical score, whereby a player-agent learns what the optimal action is in any given state. Q-Learning is about how to perform best in an *a priori* largely unknown environment.

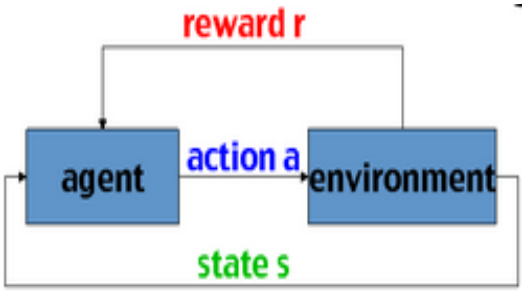



***Fig. 1****: In RL, an autonomous agent, with an explicit objective function optimization goal, perceives its environment (state "s"). Interacting with "s", it explores the rewards "r" its actions "a" may glean.*

## B.1 – Background highlights

Algorithmic learning methods where an agent combines environment sensing, explicit goal-directed action and interaction with its environment through action qualify as Reinforcement Learning (RL). We briefly highlight three such model-free methods hereafter.

### *B-1.1 Q-Learning*

***Off-policy***:
The goal of the Q-Learning agent is to learn how to guide its own actions in an initially unknown environment. In Q-learning, Q stand for "quality" in that a system learns to maximize the quality of its actions by learning an optimal behavior (or action-selection policy) as a function of its state. It is an ***off-policy*** method in that it does not rely on a known policy, but rather creates and shapes its own at it trains and learns.

***Reward – quality of state-action (Bellman equation)***:
The quality of a state-action, also called the current reward, is a relation: $Q: S^{(t)} \times A \rightarrow \mathbb{R}$. The Bellman[i] equation (Eq. 1) defines a instantaneous reward $R_t$, plus an incentive in the form of a discounted future reward calculated as the maximum of

i *Richard Bellman, "A Problem in the Sequential Design of Experiments," Sankhya, 16 (1956), 221-229.*

all potential future rewards attainable from state $s_{t+1}$. It translates the fact that as the agent's environment transitions from $s_t$ to $s_{t+1}$ both in $S$, via an action $a_t$ *in* $A$, the learning agent evaluates its current best action, $a_t$, as a function of a current reward, $R_t$, plus a discounted future reward:

$$Q_t = Q(s_t, a_t) = R_t + \gamma(\max_a(Q(s_{t+1}, a))) \quad \text{(Eq.1)}$$

***Exploration versus exploitation***:

Moreover a player-agent learns during an episodic game, by relying on both:

- exploration of uncharted regions of its environment variable space (never seen before states), and
- exploitation of already seen states for which some measure of learning experience has been accumulated,

such that:

$$Q_t^{new} \leftarrow (1-\alpha)Q_t^{old} + \alpha\left(R_t + \gamma \max_a Q(s_{t+1}, a)\right) \quad \text{(Eq. 2)}$$

where, as before, $Q_t = Q(s_t, a_t)$ is the quality of the state-action, $R_t$ is the instantaneous reward at time $t$ (for the current state), $0 \leq \alpha \leq 1$ is the learning rate, $0 \leq \gamma \leq 1$ is the discount factor. Our learning agent has only one goal in mind to learn the best it can. Our algorithmic interest is to make that happen as fast as possible. It translates in our quality matrix converging, so that:

$$\lim_{t \to +\infty} Q_t = Q^{stationary} \quad \text{(Eq. 3)}$$

The formulation of Eq. 2 essentially translates the Bellman equation into an iterated value update for the state action quality value. It is at the core of the QL algorithm. At every time step, $t$, the update of the state-action reward mapping, or quality matrix $Q_t$, results from the weighted average between the old policy value times $\alpha$, and the learned policy value times $1$-$\alpha$.

- $\alpha = 0$, the agent will not update the $Q$ function (commonly referred to as the quality matrix) mapping state-action to reward, and therefore will not learn a new policy. It remains stuck at: $Q_t^{new} = Q_t^{old}$
- $\alpha = 1$ the agent pays no attention to its previously learned quality matrix $Q_t^{old}$ and only favors its instantaneous reward $R_t$, along with another term, mediated by the discount factor, $\gamma$. The latter term is the maximum discounted potential future reward achievable at future state $s_{t+1}$, reached from $s_t$ by action $a_t$. The discount factor, $\gamma$, effectively parametrizes the importance of future rewards.
- $\gamma = 0$ makes the agent "short-sighted", in that it becomes solely interested in current rewards immediately following its actions.
- $\gamma = 1$ or slightly smaller than 1 on the other hand was shown to produce instability[4], [5] as the agent place the highest possible weight on discounted future rewards.

***Greedy versus pseudo-greedy***:

Calling $Q_t = Q(s_t, a_t)$ the reward to the agent in state "s" for adopting action "a" at time "t": $A_t := \underset{a}{argmax}(Q(s_t, a))$ will select the greedy action which maximizes its immediate reward based from some current (initially unknown) action-to-reward mapping function $Q$. In case of a tie in state, $s_t$, between two actions a and b, such that $Q(s_t,a)=Q(s_t,b)$, we may break the tie in some pre-ordained way, for instance randomly. To favor convergence our algorithmic system may (and will) evolve from a greedy action model to an $\varepsilon$-greedy policy mechanism by introducing uniformly random action selection in $\varepsilon$ % of cases.

***Simplified computational strategies***:

**Caveat 1**: Assume for a moment that the distribution of actions' reward values becomes constant *in time* (after a sound policy has been learned) and, so, that our RL system is in fact stationary, meaning that the true reward values or long-term reward distribution (in the limit of infinite time-steps) do not change. This in turn might lead us to expect that a greedy approach ($\varepsilon = 0$) always yields the best possible choice of action. However non-stationary regimes are one of the principal problems encountered in RL, often (but not exclusively) in the form of periodic, pseudo-periodic or apparently chaotic policy changes as learning proceeds and the agent's policy is updated step-wise. Thus even for apparently deterministic systems (problems with reward probability distribution which do not change over time), it is often preferable to introduce a degree of randomness when choosing an action, to balance exploration and exploitation.

**Caveat 2**: Choosing a constant value of ε can also be far from optimal. Intuitively comparing larger values of ε to smaller ones, we see that the former will lead to faster exploration, but sub-optimal exploitation of decisions with highest rewards. Thus at any time t, it is may be advantageous to use greater values of $\varepsilon$ for larger rewards' distribution variance (examining the set of rewards associated to all possible actions at a given time step). Indexing the instantaneous value of ε on the possible reward distribution variance in addition to a general decreasing trend in $\varepsilon$ values as a function of time may therefore bring the best results.

### *B-1.2 State-Action-Reward-State-Action (SARSA)*

### *B-1.3 Deep Q-Neural Network (DQN)*

From a rapid literature survey, RL, of all ML methods, is the one which most closely approximates the human form of learning, complete with its on- or off-policy, exploration vs. exploitation, and model-based or non-model based learning representations.

## B-2. – Implementation

We placed no particular emphasis on striking the best possible balance between exploration and exploitation for any method in particular. We were content with α values which allowed learning agents to improve over successive time steps, so the iterated computation of the matrix Q would exhibit convergence.

In practice, in order to speed up convergence in the sense of Eq. 3, $\alpha$ was not kept constant. Convergence of Eq. 2, sometimes called *exponential recency-weighted average* with parameter $\alpha$, is governed by the well known recursive update rule's double condition for convergence:

$$\lim_{\tau\to+\infty}\sum_{t=0}^{\tau}\alpha[a_t]=+\infty \quad \text{and} \quad \lim_{\tau\to+\infty}\sum_{t=0}^{\tau}\alpha[a_t]^2<\frac{1}{\tau} \qquad \text{(Eq. 4)}$$

where $\alpha$ is made to depend on the action, $a_t$, taken at time step $t$. Opting for a constant step parameter $\alpha[a_t] = \alpha_o$ does not satisfy the second condition in Eq. 4. Quoting Sutton et al [2] (page 26), *"the first condition is required to guarantee that time-steps are large enough to eventually overcome any initial conditions or random fluctuations. The second condition guarantees that eventually the steps become small enough to assure convergence."*

We also experimented moving away from a greedy action-picking policy mechanism to an $\varepsilon$-greedy policy mechanism, based on the current reward matrix, $Q_t$, where $\varepsilon$ is the probability of taking a random (exploratory) action $a_t$ from a given state at time t, $s_t$. Good results are obtained for a non zero values of $\varepsilon$, while $\varepsilon$ is made to decrease over time. Optimal $\varepsilon$ values are an increasing function of the variance of rewards, calculated from their distribution for all possible actions at any time t. We suggest *ex nihilo* a simple formulation for $\varepsilon$-greedy, of the form:

$$\epsilon_t = c_1\frac{1}{t+1}\left(\frac{\sigma_t^{(reward)}}{\left|\mu_t^{(reward)}\right|+1}\right)^{\frac{1}{c_2}} \qquad \text{(Eq. 5)}$$

where our 2 adjustment parameters are $c_1 \geq 0$, $c_2 \geq 1$ and the algorithmic time step verifies $t \geq 0$.

We may introduce incremental complexity in the decision-making conditions of successive games so as to make learning by the player-agents gradually more difficult. To illustrate this, we only cite:

- paddles with mass to constrain its dynamics.
- paddles' exclusion zones in the areas defended by the players. This constitutes a deliberate vulnerability in the line of defense of each player, which a trained opponent must learn to recognize before taking advantage of it.
- non uniform ball bouncing off the surface of paddles to introduce variety in the trajectories of the ball.
- normally distributed noise in the positioning of the paddle

Performance comparison between learning autonomous agents can take the form of tournaments between players, where the winner will proceed to the next level to face a newly configured player-agent.

**Discretization**

- Highlight the realistic setting where time is discretized, each time step corresponding to a frame or snapshot of the game's state, where Q is recalculated completely or partially, depending o the implementation's particulars.

- Highlight the advantage of symmetrizing states to accelerate learning.

# C – Results and discussion

# D – Concluding remarks

## D-1. – Conclusions

## D-2. – Limitation and extension for this work

Generally speaking we only considered RL settings where systems are stationary, i.e. where agents are not confronted with best actions changing over time for given states, beyond the starting point. The rationale for the study of such non-stationary settings would be to program agents, so they are capable of recognizing new best actions in the face of already seen system's state. Action selection in that case cannot rely on a simplistic greedy mechanism. The ε-greedy mechanism is also not optimal (too slow in particular) in that its random action selection is by definition indiscriminate. Instead the agent might be interested in favoring actions seldom taken in particular states before, in order to explore rapidly the changing state-best-action space. However interesting, this is also markedly more complicated than our earlier proposal of Eq. 5 for ε-greedy and will remain out of the scope of this work.

# References

[1] T. Appleton, "Trevor Appleton: Writing Pong using Python and Pygame," blog post by *Trevor Appleton*, https://trevorappleton.blogspot.com/2014/04/writing-pong-using-python-and-pygame.html, Apr-2014.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[3] C. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer-Verlag, 2006.

[4] L. Baird, "Residual Algorithms: Reinforcement Learning with Function Approximation," in *Proc. of 12th Int'l Conf. on Machine Learning*, 1995, pp. 30–37.

[5] V. François-Lavet, R. Fonteneau, and D. Ernst, "How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies," *arXiv:1512.02011 [cs]*, Dec. 2015.