Reinforcement Learning project report

Michele Vantini (662998) Fabio Colella (661591)

November 24, 2019

1 Introduction

Reinforcement Learning is nowadays increasingly relevant. Its applications span from robotics [1] to chatbots [2], including also agents able to play old Atari video games [3] and the complex game of Go [4].

The goal of the project was to develop a simple yet effective agent for playing Pong with the agent not having any access to direct information about the environment, its only mean of observation were the frames of the game, updated on each iteration. This report shows that using fine tuned policy gradient approach allows to obtain positive results and beat a generic hardcoded AI in most of the games.

2 Review of external sources

The main external source that we used is [5] where Karpathy explain how to adapt the policy gradient algorithm to the problem of learning how to play Pong. Some of the preprocessing techniques over observations that are suggested in this blog post are further explored in this report.

3 Approach and method

3.1 Setting

As previously mentioned, the Atari game that we are considering is the game of Pong. The game contains two paddles and a ball, the paddles can move up, down or remain still, and each player has to defend their line by making the ball to bump into the paddle. At the same time they need to try to score, which is done by throwing the ball in a way that the opponent cannot reach to bump back. In our setting, one of the players, called the *opponent*, is a simple, "hardcoded" AI, while the other one (*player*) is a Reinforcement Learning Agent. Our goal is to teach *player* how to beat *opponent*. During each iteration of the game the players take some action based on the current state, and the game environment is consequently updated. The state of each

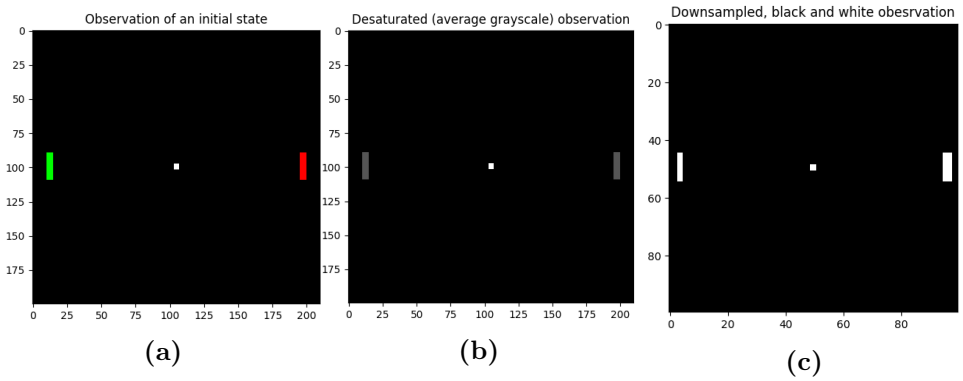

Figure 1: Preprocessing steps from (a) to (c)

timestep of the game consists of the position, direction and speed of the ball and the paddles. While *opponent* knows the current state, *player* has to infer the state from an observation of the state. An observation of such a state consists of a frame of the game state, including the two paddles and the ball. The frame is a three-dimensional tensor composed of three matrices of pixels, one for each colour channel. An example of such observation frames is exemplified in Figure 1a. In this setting there is no reward for actions taken during the game, instead the agent is given a reward at the end of each game. The reward is positive (+10) if the agent wins the game or negative (-10) if the agent loses.

3.2 Observation preprocessing

Different preprocessing techniques have been applied to the observation before feeding it to the agent. Follows a description of each of the steps:

- Resizing: we reshape the observation to be a squared frame;
- Downsampling: this step consists of discarding one pixel every two in both dimension, this results in a frame that continues to provide enough information, but with less data, rendering it computationally lighter;
- Averaged Desaturation: the value of each pixel is replaced by the averaged value of the same pixel over the three colour channels; the resulting frame has a single luminosity colour channel. The colour is not relevant in this setting because all the items can be recognised by their shape and position, so having a grayscale observation retains information while reducing the data amount by a factor three;
- Black and white recolouring: as a harder desaturation, each pixel belonging to the ball or the paddles becomes full white, while everything

else becomes full black. This step is used as an alternative over the averaged desaturation. While it carries the same amount of data, the harder and clearer colour boundaries can make the learning process easier, without sacrificing accuracy.

In addition to the preprocessing, we combined each observation with the observation of the previous state, in order for the agent to have a mean by which to learn direction and speed of the ball, which could not be inferred by a single observation alone.

- Image subtraction: the frame for the previous observation is subtracted from the one for the current observations. In this way the pixels where an object moved to have a positive value, the ones an object moved from are negative and the ones whose state didn't change are zeros. The advantage of this technique is that adds information to a frame without increasing the amount of data. However, it may cause confusion to the agent as both the background and items have zero values when the pixel keeps the same value, so the agent can be misled to confuse one for the other;

- Image stacking: the frame for the previous observation is stacked to the one for the current observation. In this way the agent has a view of the present and the past in the same form. With respect to *Image subtraction* this approach has the disadvantage of doubling the amount of data required, but it also prevents the misleading factors of that approach.

The result of these steps renders the final observation that we feed to our agent. Figure 1b shows an example of a desaturated (grayscale) observation, while Figure 1c shows a downsampled

3.3 Policy gradient

We decided to use the Policy gradient algorithm as suggested by Karpathy [5], but we tried two different policies: one uses a categorical distribution while the other uses a Gaussian one. The policy gradient is updated every 10 games. This allows to accumulate more experience before the policy update "decides" how to update the model. The agent performance were monitored by logging the cumulative reward every 20 games.

We tested different variants of the algorithm:

- Possible actions; for this variable we have two alternatives:
  - all the available actions can be taken: UP, DOWN and STAY;
  - only UP and DOWN: restricts the available actions to the ones for movement. This allows to reduce the size of the neural network

and not consider being still, which can be produced by repeatedly alternating UP and DOWN;

- Policy probability distribution:
  - normal: the neural network policy produces as output a mean and a variance which are used to sample from a Gaussian distribution;
  - categorical: the neural network policy outputs probabilities for different actions. These probabilities are used to sample from a categorical distribution;
- Type of preprocessing:
  - averaged desaturation
  - black and white recolouring and downsampling
- Dealing with the previous observation:
  - Image subtraction
  - Image stacking

In order to understand how much our agent was improving its skills during the game, we logged the cumulative reward every 20 games.

4 Results and performance analysis

In this section we report and analyse the results of the different variants we tried while training our agent, together with considerations about their performance.

4.1 Variants producing poor results

We begin by listing the variants that did not solved the reinforcement learning problem. The first variant that we trained featured: all the actions (UP, DOWN and STAY), normal policy and image subtraction. We discarded this variant because after training for 500,000 episodes the cumulative reward over 20 games was still too low negative. Indeed, you can see in A plot of the learning rate during the full training can be found in Figure 2, and no process of learning is evident from the plot. This result was common to all the models that we tried using the normal distribution, independently from any other factor.

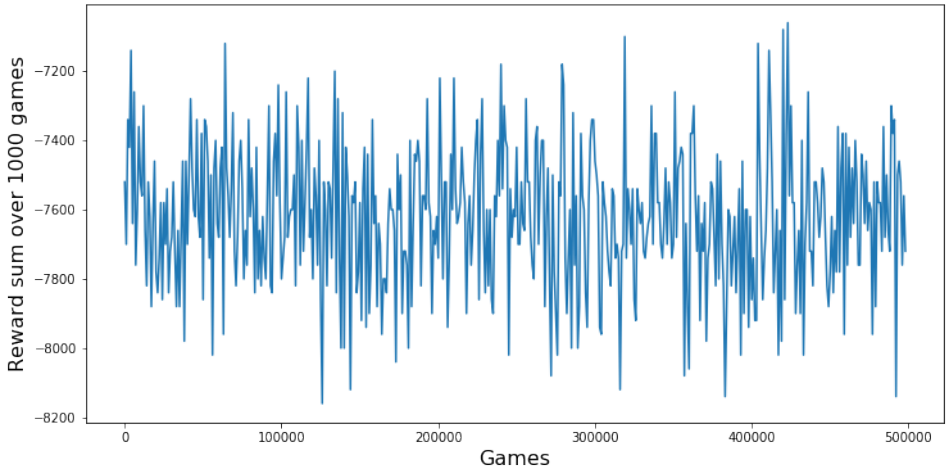

Figure 2: Reward sum grouped by 1000 games - Variant that uses all the possible actions and Normal policy

4.2 Variants able to learn

The first set of trials allowed us to exclude the use of the normal distribution, and as a result, we considered policies implementing only the categorical distribution. Hence, we started comparing variants that were using both image subtraction and image stacking, and also both the full and reduced set of actions. During these new tests we found that combining the observations by image subtractions, while letting the agent learn, did not result in good cumulative rewards over the 20 games interval. As a result we chose image stacking for the following tests. The latest tests hence focused on a comparison between the reduced and the full set of actions. Both the variants were able to consistently achieve positive cumulative rewards, with the reduced outperforming the full set of actions, as it learned earlier and achieved better results at the end of the training.
For what concerns the type of preprocessing, we got very similar learning results for both averaged saturation and black and white recolouring with downsampling, thus we decided to keep using the latter as we found it to be more computationally efficient and hence resulting in a faster training.

4.3 Analysis

Comparing the results obtained with the different variants, we gained some important insights:

- Training with all the three actions is significantly computationally heavier than training with only two;

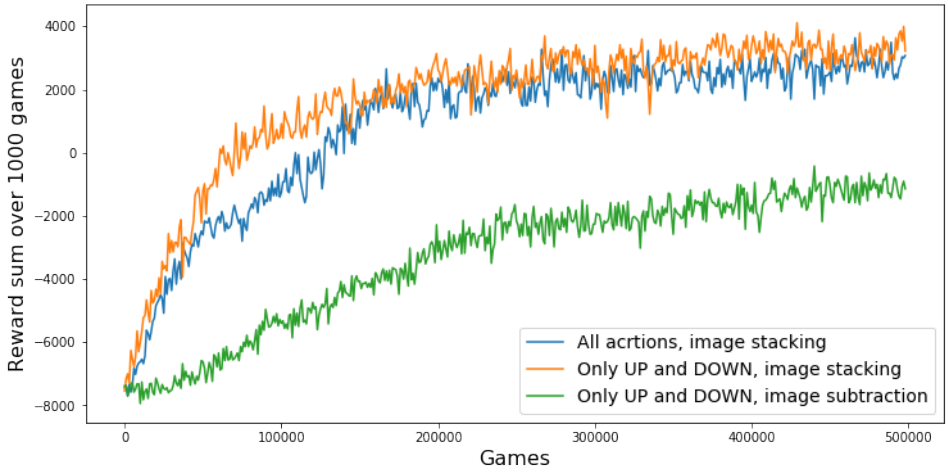

Figure 3

- The agent needs more iterations to learn and achieves lower rewards, when training with all the three actions compared to using only two;
- The agents featuring a normal distribution do not seem to be able to learn to play the game in any meaningful way;
- Using averaged desaturation or black and white did not significantly affect the learning performance, but the downsampling allowed to make the latter computationally lighter.

Some plots comparing some of the most performing variants can be found in Figure 3. It is very clear that the version that is using image stacking and only actions UP and DOWN is outperforming the other variants. As a result, this is our final choice as reinforcement learning model for solving the given task.

4.4 Additional observations on the agent behaviour

After analysing the agent performance we observed how it was able to play the game against the *opponent* by watching some rendered games. It is our opinion that our agent learnt to mimic its opponent in order to beat it. While this strategy seemed surprising at first we considered that it is actually an efficient and effective mean to beat that specific opponent.

5 Conclusions

In this project we explored how preprocessing and fine tuning the same reinforcement algorithm can affect the performance towards achieving a specific

goal. Modifying the settings of the learning algorithm allowed us to obtain a reinforcement learning agent that is able to play pong and beat its opponent around 70% of the times. We tried several different variants and gradually narrowed down to the best combination. At the end of this process, among all the tested variants, we found that using the categorical distribution in the policy, black and white recolouring of observations with downsampling, and image stacking of the current observation with the previous; is the best combination when using Policy Gradient. Moreover, using only the actions that produced movement resulted in a better performing model that could also be trained faster. As an additional discovery, we found that our agent learnt to play by mimicking its opponent. Further variants that we would like to try in the future include hiding the opponent's paddle in the observation and making the agent learn by training only against itself.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[2] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[4] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[5] Deep reinforcement learning: Pong from pixels. http://karpathy.github.io/2016/05/31/rl/. Accessed: 2010-09-30.