Adapting Rainbow DQN to Atari games

Willi Menapace
Ms-Pacman
willi.menapace@studenti.unitn.it

Luca Zanella
Atlantis
luca.zanella-3@studenti.unitn.it

Daniele Giuliani
Demon Attack
daniele.giuliani@studenti.unitn.it

1. Introduction

Reinforcement Learning is one of the most active research area in the deep learning community. Its main challenge is finding a way to optimize a target function, usually the cumulative sum of rewards obtained through the policy, through experience and without explicit supervision. This technology finds a variety of possible applications that include continuous control systems such as autonomous cars or robotic arms, stock trading agents or videogame agents. However, despite its relevancy, it is only recently that major improvements have come from the research community allowing the achievement of significant results. In particular, the attention of our group was caught by the victory of novel RL systems in exceptionally challenging games such as Starcraft 2 (DeepMind, December 2018) and Dota 2 (OpenAI, April 2019) against professional players. In order to take a glimpse into the the world of reinforcement learning and try to give a little contribution to the field, we revisit the DQN architecture introduced by DeepMind in [4], and improved upon in [3]. The objective is to train and tune agents able to play at human skill level or better a set of chosen Atari games using only raw pixel input, namely Ms-Pacman, Atlantis and Demon Attack. The task is expected to be particularly challenging for the Ms-Pacman game as shown in [5] where the plain DQN approach is able to achieve only a reported 13% performance compared to a human player and proficient systems in this game usually take as input the internal game state rather than pixel output. Another challenging element is computing time. RL training phases are computationally expensive and reference results taken from [3] are given after 200 million training iterations. From our benchmarks and codebase, a system equipped with a RTX 2070 GPU can process 70 frames per second, giving a projected time of 33 GPU days for training in order to emulate the results. This is a prohibitive time amount, so we need to take steps to increase convergence speed on each particular game.

2. Proposed Solution

We start with a tensorflow based hand implementation of a DQN as described in [4] and test it on Ms-Pacman. We take care of implementing many of the engineering tricks explained in the paper such as reward clipping, frameskipping, frame stacking, different gray conversions and image rescalings. We also experiment with using a Huber loss function instead of MSE to avoid too unbalanced weights updates for the different experience samples. Special care has also been taken to optimize the memory footprint of the replay buffer by storing each preprocessed frame only once and in `int8` format. This allow us to use a large 1M frames replay buffer that can fit in less than 10GB of memory, whereas a naive implementation would use 16 times the space.

As shown in [5], Ms-Pacman is a challenging environment for reinforcement learning algorithms due to its complexity. It requires the agent to be able to navigate in a maze to chase remaining dots to eat while avoiding being eaten by ghosts. An added complexity is given by the fact that after eating a special ‘pill’ the ghosts become eatable for a limited amount of time and the agent needs to identify these timeframes and proactively chase ghosts in order to take them out.

We run the trained agent for 50 episodes, measuring the average game score and, unsurprisingly, this simple architecture is not satisfying, so we decide to implement the optimizations proposed in [3] and generally referred to as Rainbow DQN. Due to the complexity of the first implementation and the desire to migrate our codebase to the PyTorch environment, we decide to base our implementation on top of the ptan agent library [2], taking [1] as the initial codebase.

We start by implementing and integrating together the N-Step, Double DQN and Noisy Layers optimizations, noticing improvements in average reward of the trained agent. In particular the ϵ-greedy exploration strategy proposed in

[4] and replaced by the noisy layers in this implementation is particularly unsuited for Ms-Pacman. This exploration strategy causes a random action to happen with probability $1 - \epsilon$ and is necessary to avoid the agent from blocking in loops of actions while playing, but also causes death every time the agent is chased by a ghost, with the agent escaping successfully until the moment when a random action causes the agent to run into the ghost and be eaten. Noisy layers, on the other hand, work by perturbing the estimation of qvalues. When the agent is stuck in a loop of actions determined by similar qvalues, the agent is still able to escape it thanks to the perturbations introduced by noisy layers, but when it is chased by a ghost and qvalues are far away from each other, the qvalue perturbations introduced by noise do not affect the choice of the agent to escape from the ghost.

Encouraged by the results, we decide to continue integrating the remaining rainbow improvements into our DQN architecture, namely the Prioritized Replay Buffer, Distributional DQN and Dueling DQN, but the likelihood to introduce subtle bugs in our implementation due to the complexity of the task leads us to use the rainbow agent implementation provided by our library as a base for the rest of the experiments.

At this point each group member proceeds to tune the model to its environment.

2.1. Ms-Pacman

We start by performing a hyperparameter search on the model in order to tune the original hyperparameters proposed in [3]. The search is complicated by the very long training times needed to see the effects of each modification. In particular, we notice instability when training the model with the proposed learning rate of $6.25e - 5$, so we lower it to $2e - 5$ obtaining more consistent gains in reward over time. We also set the prioritized buffer β annealing curve to end at 3M frames in order to enable the model to have a large initial time where the prioritized buffer can be more effective. Moreover, the target network is updated every 10k frames like in the initial plain DQN implementation instead of the proposed 32k to stimulate faster convergence. The buffer size is set to 1M frames when training on local machines, while it is slightly reduced when training on Google Colab due to memory limitations.

The agent trained on the rainbow model still shows unsatisfying behaviors after 38 hours of training. Training dynamics are shown in Fig. 1. The agent learns to successfully eat dots across the maze and to eat the special pills that allow it to eat ghosts, but when the special pills terminate and few dots remain in the maze the agent seems unable to detect the presence of attacking ghosts and runs into them. An analysis of qvalues output by the network shows the reason for the behavior. The reward system used for training clips rewards in the range (-1, 1), but never assigns a negative reward to the agent upon death. In an environment such as OpenAI Cartpole, where the agent continuously receives rewards, this is not a problem because death prevents the agent to accumulate future rewards, so qvalues diminish for actions that will lead to death. In our situation however, when there are no dots and special pills in close proximity, the correct qvalue to estimate is 0 for every action. When a ghost approaches, the correct qvalue to estimate is still 0 for every action due to the absence of death penalties, so the agent may well decide to run into the ghost. Moreover, in the original game, there is a difference in color from ghosts that are eatable and ghosts that are not, but due to grayscale conversion we notice that there is only a subtle difference between eatable and not eatable ghosts which may further damage training.

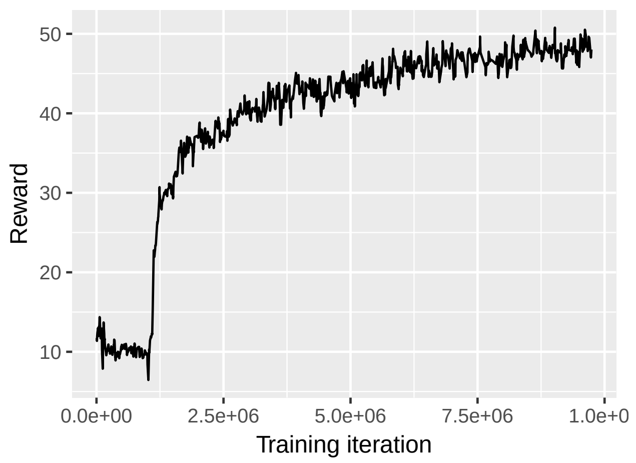

Figure 1. Training reward dynamics of the pacman agent after the first reshaping. The reward refers to the average clipped reward obtained by the agent in the last 100 episodes. The reward increases slowly, but the agent is unaware of ghosts when the number of dots reduces, leading it to unfrequently complete the first level.

In order to address this problems we preprocess each frame in order to make eatable ghosts of a distinguishable gray tonality and we use a common technique in RL called reward reshaping, which consists in changing the rewards in order to drive the policy towards the wanted direction. Care has been taken in order not for expected qvalues to saturate the boundaries imposed to qvalues by the Distributional DQN modification. For the subsequent attempt the reward policy was reshaped in this way:

- Eating dot $1 \rightarrow 0.5$
- Eating pill $1 \rightarrow 1$
- Eating ghost $1 \rightarrow 2$
- Eating fruit $1 \rightarrow 0.5$
- Being eaten $0 \rightarrow -7$

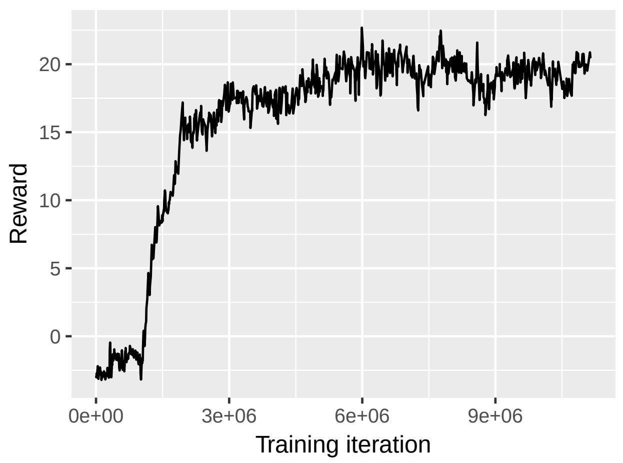

Figure 2. Training reward dynamics of the pacman agent after the first reshaping. The reward refers to the average reshaped reward obtained by the agent in the last 100 episodes. The reward does not increase further after 6M frames because the agent is hindered in chasing the last dots to eat due to high death penalization and does not get to the second level.

- Finishing level $1 \to 7$

The results obtained with these settings, however, were not satisfying. The training dynamics are shown in Fig. 2. An analysis of the behavior of the agent in fact highlighted that, due to the high death penalization, the agent was so hindered to get close to a ghost that it always tried to hide away from them, ignoring nearby dots to eat and not making progress. Moreover, due to the relatively high special pill and ghost eating rewards, the agent tended to immediately eat all the available special pills, and be very aggressive with regards to eatable ghosts. While this strategy provided high game scores, our main objective was to make progress through different levels, so rewards were reshaped again to encourage a more gentle behavior:

- Eating dot $0.5 \to 0.5$
- Eating pill $1 \to 0$
- Eating ghost $2 \to 1.5$
- Eating fruit $0.5 \to 0.5$
- Being eaten $-7 \to -2$
- Finishing level $7 \to 4$

Notably, we didn't reward the agent for eating special pills in order not to encourage it to eat them in close succession, reduced the reward for eating ghosts and considerably reduced the penalty for being eaten.

This settings proved successful. After 90 hours of training the agent proactively searches dots to eat in the maze, eats special pills at reasonable timeframes, hunts eatable ghosts and runs away from dangerous ones. In order to aid convergence, we also reduce the learning rate from $2e-5$ to $5e-6$ after 44 training hours and 11.3M training iterations and from $5e-6$ to $1.25e-6$ after 16.5M training iterations and 74 hours of training. Training reward dynamics are shown in Fig. 3.

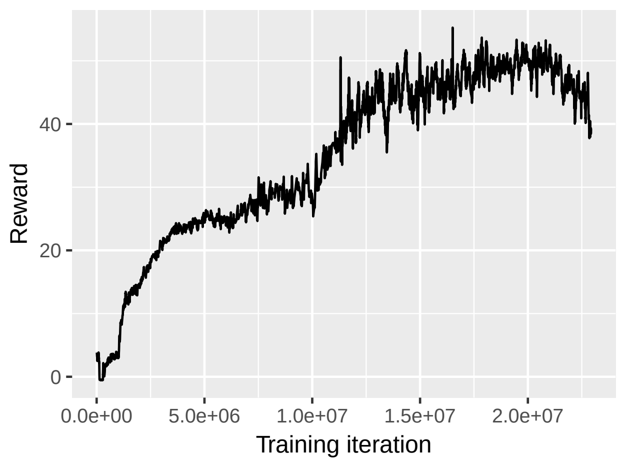

Figure 3. Training reward dynamics of last pacman agent. The reward refers to the average reshaped reward obtained by the agent in the last 100 episodes. The reward increases fastly in the first million of iterations and then stabilizes. When at 10M iterations the agent learns how to clear all the dots and pass to the second level the reward begins to grow quickly again. After 20M iterations reward starts dropping, probably due to the agent trying to adapt to the third level and losing performance on the first two.

2.2. Atlantis

The choice of Atlantis is based on the 449.9% performance obtained by the plain DQN approach compared to a human player as shown in [5].

The game consists in defending the city of Atlantis, made of three defense posts that can be used by the player to attack enemies and six Atlantean installations, from the Gorgon vessels. The game ends when the Acropolis Command Post and all six of Atlantis' installations are destroyed.

In the original Rainbow DQN, observations are greyscaled and cropped to 84 x 84 pixels [3], however, this setting on Atlantis eliminates a relevant portion of the sky and maintains irrelevant information at the bottom such as the score. Based on that, we decided to crop observations to 92 x 84 pixels maintaining only signicant information in the frame.

Gorgon vessels circle down from the highest level to the one closest to Atlantis and can only fire their deathray when close to the city. Since the first hours of training, the agent learns to successfully destroy Gorgon vessels, reaching scores above the ones obtained by a professional human games tester as shown in [5]. One of the most notable aspects encountered during the training was the discovery of a bug in the system that prevents one of the six Atlantean installations to be demolished. Although this strategy leads to the highest possible game score, the main objective was to make the agent strong enough to not rely on this shortcut and we decided to proceed with the training. This choice proved to be successful in improving the policy of the agent: the game score obtained is still the highest possible, but now the agent does not rely on the bug and concludes the game episode with all seven installations being untoched for most of the time. Another significant aspect is that the agent mainly uses the Acropolis Command Post and the right sentry post to defend all the Atlantis' installations.

2.3. Demon Attack

The third game to which we try to apply the Rainbow DQN architecture is Demon Attack. The game mechanics are pretty simple. The player controls a space ship which can be moved horizontally across the playing field. The objective is to shoot down all the enemy spaceships, while avoiding their attacks. As the game progresses the enemies become stronger and stronger, they gain movement speed and use different attack patterns.

Initially, the training of the network for this game was done using Google Colab due to the fact that some of our laptops were not powerful enough to sustain the training at an acceptable speed. Unfortunately, Google Colab has strict limits on how long a program can run. This meant that we had to restart the training every few hours slowing down the process. Furthermore, we noticed that checkpoint restarts cause problems with the implementation of the prioritized replay buffer used by our library, probably due to a sub-optimal initialization of buffer weights. Although it would be possible to save the state of the buffer as a part of the checkpoint, we decided to resort to the plain replay buffer to avoid saving a component of the size of tens of gigabytes.

As a second approach, we retrained the whole network during the weekend using more a pore powerful local machine. This time, we trained the network for approximately 72 hours consecutively on a machine with a GTX 960 GPU which, although not as performing as the RTX 2070 and far from the performance of the Tesla T4 provided by Colab, was still able to process 20 frames per second. With this approach, we could also make use of the prioritized buffer which, as described in the result section, improved noticeably the performance of the network.

	Plain DQN	Rainbow DQN	Our
Mean score	1880	2267	5145
Std dev.	448	504	866
Samples	50	50	50
Train frames	11.4M	9.7M	22.9M
Train time	45h	38h	90h

Table 1. Ms-Pacman agent results. Results refer to the point in time when the agent stopped showing improvements in gathered reward.

	Our Rainbow
Mean score	2,609,433.3
Std dev.	1,250,245.2
Samples	42
Train frames	6.6M
Train time	25h

Table 2. Atlantis agent results.

3. Results

3.1. Ms-Pacman

Table 1 shows average scores obtained by each agent we trained on Ms-Pacman.

Notably, the reward reshaping and frame preprocessing approach we used to train our final agent version allowed the network to obtain an average score of 5145 after 22.9M training iterations. For comparison, [3] reports an average 5380.4 score on Ms-Pacman after 200M training iterations, so we consider our result a success given our limited computational resources. The score corresponds to reaching on average the end of level 2 and sometimes reaching level 3. Interestingly, Ms-Pacman level 3 has a different color pattern and maze structure which may be the reason why the agent struggles to gather further reward. Modifying the Ms-Pacman emulator to start from level 3 instead of level 1 with a certain probability may allow the agent to gather more samples from this level and continue its improvements. Moreover, [6] and [5] reports an average human score for Ms-Pacman of respectively 6951.6 and 15693, meaning we are are still not able to reach human level behavior, but the result is satisfying.

3.2. Atlantis

Table 2 shows average scores obtained by the agent we trained on Atlantis.

DeepMind reports an average score for Rainbow on Atlantis over 200 testing episodes of $826,659.5$ [3]. The frame preprocessing approach we used to train our agent allowed the network to obtain an average score of $2,609,433.3$ after 6.6M training iterations without exploiting the bug mentioned in 2.2 despite the left sentry post is not as precise

	Rainbow	Rainbow + Prio
Mean score	6828	10291
Std dev.	1812	4165
Samples	100	100
Train frames	6.9M	4.1M
Train time	29h (on Tesla T4)	72h (on GTX 960)

Table 3. Demon Attack agents results.

as the others. Proceeding with the training of the network should solve this problem.

3.3. Demon Attack

All of the results are summarize in the Table 3.

With the first approach we were able to obtain a decent score after approximately 29 hours of training on Google Colab. At this point, the network performance was not improving consistently even with more training time. Our intuition was that the network had difficulty defeating the stronger enemies encountered later in the game, due to the fact that their attack pattern was different from all the one seen up to that point and the network didn't encounter this enemy frequently enough to be able to devise a good strategy for defeating them.

With the second approach instead, after 72 hours of training (and only 4.1M frames processed given the outdated hardware) we were able to improve the mean score by a considerable amount over the previous attempt. We consider this a good result if we take into account the fact that fewer frames were processed by the network.

The scores obtained are lower than the one obtained by [3]. In fact, we noticed that the network had a problem learning one particular attack pattern used in the late game which is visually similar to the previous attack but works differently: while all other attacks move only vertically, this attack can track the player and follow him horizontally, so for the agent it's more difficult to avoid.

Because of the huge time requirement and hardware availability, we were not able to train the network any further. Nevertheless, the results obtained are encouraging and training even more the network would probably allow the agent to learn the advanced attack patterns and obtain much higher scores.

4. Conclusions

In this project we were able to take a comprehensive look into the main value based reinforcement learning methods and show the effectiveness of some methods to increase their capabilities and improve convergence speed, using a selection of Atari games as benchmarks.

We achieve human level performance or better in the games of Atlantis and Demon Attack, while we obtain sub human performance in the more complex game of Ms-Pacman, obtaining however scores comparable to the ones reported in [3], but with much shorter training.

In this work we did not address possible modifications to the lower layers of the original DQN network. It would be probably beneficial to create a deeper model, but with smaller convolutional filters and to introduce techniques such as batch normalization to improve performance.

It would also be interesting to see how the model behaves when taking as input the internal game state rather than raw pixel outputs. In more complex games such as Ms-Pacman this should be beneficial and would allow to introduce some carefully handcrafted features such as ghost movement direction, eatable states, number of seconds left for the special pill effect or shortest path distances to the closest ghost for each movement direction. This modification should allow even simpler model to converge faster and obtain higher scores.

References

[1] Deep-reinforcement-learning-hands-on github repository. `https://github.com/PacktPublishing/Deep-Reinforcement-Learning-Hands-On/tree/master/Chapter07`. Accessed: 2019-05-10.

[2] Ptan agent library. `https://github.com/Shmuma/ptan`. Accessed: 2019-05-10.

[3] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and et al. Human-level control through deep reinforcement learning, Feb 2015.

[6] Z. Wang, N. de Freitas, and M. Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.