DOOM&RL

Marco Foschini

Abstract

The scope of the project is to develop a DQN agent, capable of playing different scenarios of the game DOOM. The computed models are able to complete easy and intermediate tasks. By also using additional variants, like Double DQN and Dueling DQN it was also possible to deal with the overestimation of the q-value. However DQN was not able to pass the hardest scenario, since the action-space is large.

1 Introduction

During the course different example were present where RL is applied for some games. Inspired by this I have decided to apply the same approach for one of the most famous first person shooter: DOOM.

The environment used for this tasks is called VizDoom[1] and it provides different scenarios where the agent must perform a particular task. For the project three of them were considered:

- Basic: the player is spawned at the bottom of a rectangular room and its objective is to shoot the demon which is placed in a random position along the opposite wall. The available commands are three: move left, move right, shoot.
- Defend the center: the player is spawned at the center of a circular room while five monster will spawn along the wall, the objective of the player is to survive as long as possible. Since enemies will re-spawn after some time and ammunition is limited, the agent will fail at a certain point. Here the available commands are three: move left, move right, shoot.
- Deadly corridor: the player is spawned at the entrance of a corridor and six demons are placed in different positions, the goal is to reach the checkpoint without dying. Here the available commands are seven: move left, move right, turn left, turn right, move forward, move backward, shoot.

For all the scenario, the RL algorithm considered as baseline is DQN.

2 Pre-processing

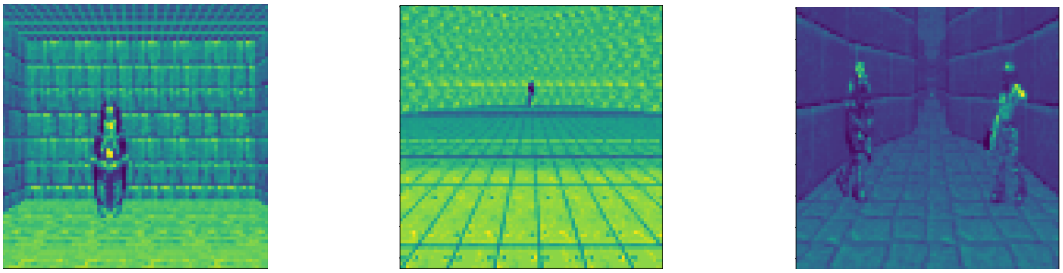
First of all the pre-processing part was handled. In fact, given a gray-scale image, it is cropped by deleting all the unuseful information (like the HUD), then it is normalized and reshaped into an 84*84 image. Last but not lest, a stack of 4 images is created, since it will be the input of the neural network. The operations are similar for each scenario, the only thing that differs is the portion of the cropped image. Also, to speed up the learning process the enviroment has been set to skip 4 frames after selecting an action, this will lead to a stack with less similar images and with a more diverse replay memory [4].

[1] `https://github.com/mwydmuch/ViZDoom`

Preprint. Under review.

(a) Basic (b) Defend the center (c) Deadly corridor

Figure 1: Scenarios considered.



(a) Basic (b) Defend the center (c) Deadly corridor

Figure 2: Pre-processed frames.

3 Network

The network used for all the scenarios is the same and it consists of:

- 4 convolutional layer.
- 2 fully connect layer.

4 DQN

The RL algorithm used is for this project is DQN [2]. Huber Loss has been selected for updating the model, since it is less sensitive to outliers and with the cart-pole environment the training was less time-consuming. Last but not least, in order to have a more stable learning process the gradient has been clipped between -1 and 1, to prevent exploding gradients. The optimizer that was used is RMSprop.

5 Basic

As already said before, the goal of this task is to shoot the demon positioned along the opposite wall. ViZDoom not only offers the enviroment, but it is also possible to get a suggested reward, which in this case has been used. By taking a look at the documentation, it is defined as:

- +101 for shooting the demon.
- -5 for missing the demon.

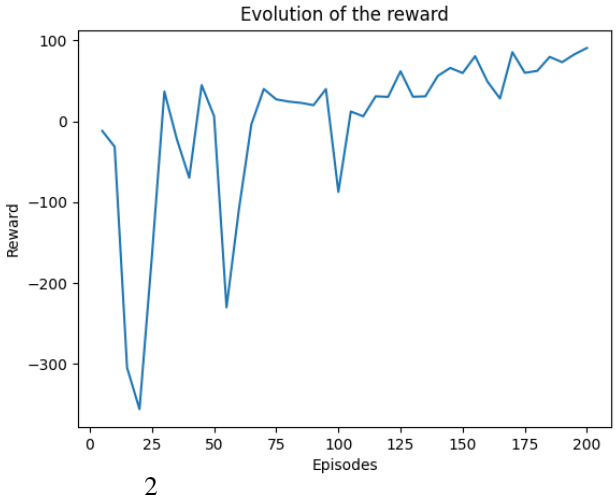


Figure 3: Evolution of the reward.

- -1 for each step.

After only 200 episodes the results are very satisfactory, in fact the model reaches an average of 82 as final reward. By taking a look at the agent while doing the task, we can see that it immediately reaches the enemy, but sometimes it shoot before being in the correct position, but still with this high reward we can say that the task has been completed with success.

In order to obtain this results a small grid search on the hyper-parameter was necessary. The weight of the policy network are updated after each action, this lead to a training time of 2 hours. It can be reduced by increasing the number of steps between the updates, this will be done in the experiment that follows.

6 Defend the center

In this scenario, the objective is to survive as long as possible. The agent is placed in the middle of a circular room and 5 demons will spawn in random locations. Those will try to reach the player and if they are very close, they will attack the agent and decrease its health. Even if they are killed, they will re-spawn after some time and, since ammunition is limited, the agent will fail at a certain point. The reward suggested by the environment is simply defined as:

- +1 for killing the monster.
- -1 if the agent get killed.

For this scenario more episode are needed for the training (around 5000), this is due to the fact that there are different type of enemy, but also the environment is not fully visible by the agent. Here updating the network at each action would lead to a very high training time, so the updates of the policy network are done only at the end of each episode. In order to have a properly trained agent 12 hours are needed. With this configuration the agent is able to survive for around 180 steps as you can see from the table 1; most of the time it dies since it has finished ammunition.

What I immediately noticed is that the agent is inclined to shoot even if the monster is very far and, as a consequence, it might miss the target. So, in order to improve its behaviour, I slightly changed the reward, which became:

- +2 for killing the monster.
- -1 for shooting.
- -1 if the agent get killed.

This should penalize the action of shooting and it is actually the case, however by taking a look at the table 1 we can say that the results are slightly worse, in fact, it will lead to have a lot of demons nearby, as a consequences most of the time it is not able to kill all of them before dying.

Another thing that I noticed is that in particular states the agent stops shooting and follows the movements of a certain demon, this is probably due to the fact the q-value of that particular state is very high, probably waiting would lead to a certain kill, since the monster will be closer. However by waiting, the agent is not able to observe the other part of the environment and as a consequence it will not notice the other closer enemies. In order to prevent this behaviour improved version of DQN were considered.

7 Double DQN

One of the concerns of DQN is that it can overestimate the q-value, since we are always taking the maximum value of the target network as depicted in the equation 1.

$$Y_t^Q = R_{t+1} + \gamma \max_a Q_{\theta_1}(S_{t+1}, a) \tag{1}$$

Double DQN [1] addresses this problem by introducing an additional network Q_{θ_2}, which will be responsible of picking the action, while Q_{θ_1} will be responsible of evaluating the action. As a consequence the target Y_t^Q can be rewritten as the equation 2.

$$Y_t^Q = R_{t+1} + \gamma Q_{\theta_1}(S_{t+1}, \max_a Q_{\theta_2}(S_{t+1}, a)) \tag{2}$$

In the end the new estimate will be more robust to noise and so, it is less likely to overestimate the q-value, as for the implementation Q_{θ_2} is the network used for selecting the action while playing.

8 Dueling DQN

Another thing that I can do in order to try to obtain better results is to separate the representation of the state value with respect to the action advantage, this is the idea at the base of Dueling DQN [5]. The architecture will be quite similar, the only difference is the final part. The network is split into two branches, one will compute the state value (V), while the other one the action advantages (A) as depicted in the image 4.

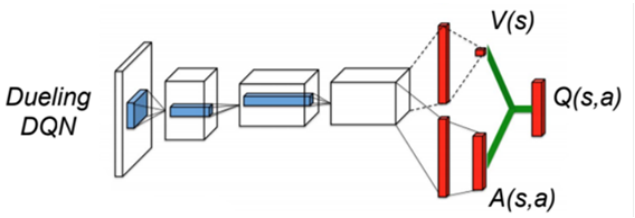


Figure 4: Dueling DQN.

These values are then aggregated according to the formula 3

$$Q(s,a) = V(s,\theta,\beta) + A(s,a,\theta,\alpha) - \frac{1}{|A|}\sum_a A(s,a,\theta,\alpha) \tag{3}$$

After that, the pipeline will be the same.

These changes have been made because it is not always necessary to know the value of each action in a given state, since they would lead to the same results. In fact if the values of each action are very similar, then it is not really important which action is picked.

9 Defend the center - Final Results

Both Double DQN and Dueling DQN can be used at the same time, this technique is called Double Dueling DQN and by applying it, I was able to get an agent capable of outperforming the previous models, as shown in the table 1.

Table 1: Results - Defend the Center

Name	Avg. survive steps
DQN	179.95
DQN + Custom Reward	168.92
DDDQN + Custom Reward	266.95

Also as you can see from the figure 5 the agent is actually learning over time, since the survival time is increasing. Now, we have a behaviour similar to the one obtained by reshaping the reward, but the problems that we had before are now solved: the agent doesn't wait the demons to be nearby it, it usually scout the environment in order to find those that could be a clear shot, as a consequence it is usually not overwhelmed by all the demons.

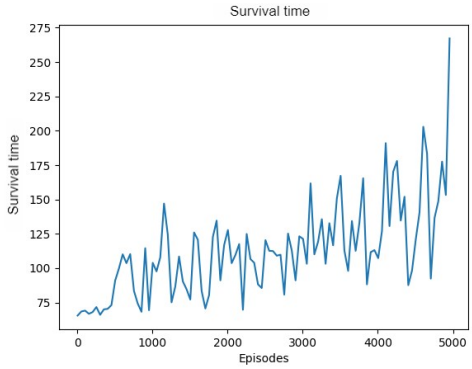


Figure 5: Evolution of the survival time.

10 Deadly corridor

In this scenario the agent is placed at the beginning of a corridor while 6 demons will spawn in a given position. The objective is to reach the checkpoint placed at the end of the path. The reward that VizDoom provides is simply defined as:

- +dx for getting closer to the checkpoint
- -dx for getting further from the checkpoint.
- -100 if you get killed.

First thing that I have done is reshaping the reward by adding three additional variables:

- positive reward for killing one of the demons.
- negative reward for using ammunition.
- negative reward for getting shot.

I have done this in order to have a easier learning process, in fact by adding these three variables the agent can immediately understand that wasting ammunition is a bad action, killing demons as fast as it can is instead good. I suppose I would get the same results with the first reward, besides more episodes would be needed for the training, and as you will see one of the concerns of this task is the training time.

I have tried with different set of weights and also with both the architecture presented before, but the results obtained after the training were quite unsatisfactory. After 5000 episodes the agent is able to shoot one of the two nearby demons, but it is rare, also since around 12 hours are needed for the training, it was not possible to do an accurate grid search over the hyper-parameter space.

However the algorithm is working properly, in fact it possible to decrease the difficulty of the level, by doing that the amount of damage that the enemy will do will decrease. For this scenario it is set by default to the maximum value (5), by decreasing to 3 and by setting the reward as:

- +default reward
- +200 for killing a demon.
- -5 for shooting.
- -10 for getting shot.

The agent will learn that it is possible to reach the checkpoint, without considering the enemy placed in the environment. Probably by decreasing the magnitude of the default reward I would get the expected behaviour. But still since training for this particular task is very time-consuming, I was not able to do it.

11 Conclusion

In the end for the first two tasks, the agent is able to obtain satisfactory results that show the strength of the DQN algorithm. However for the second one, I was able to see that the agent was overestimating certain states, since it was making sub-optimal decision, like waiting the enemy to be closer. In order to deal with this problem two different improved version of DQN were considered: Double DQN and Dueling DQN. By applying both of them at the same time, the results obtained far exceeded the previous ones, so we can consider the problem handled.

However at the same time we can also see the limits of this RL algorithm by taking a look at the last experiment. Here the action space is quite large and also, the scenario is quite difficult to be solved since there is no margin of error. It is necessary to select the appropriate hyper-parameter and an appropriate reward; however this was not possible since the training is very time consuming. Another problem related to this particular task is about the replay memory. When we are storing experiences, there is the risk of storing and sampling elements that will not give a relevant update; for example having an agent which is facing a wall it doesn't give us a lot of information. So future works would be focused on implementing and using a prioritized experience replay [3] in order to solve this issue.

References

[1] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". In: *CoRR* abs/1509.06461 (2015). arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461.

[2] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[3] Tom Schaul et al. *Prioritized Experience Replay*. 2015. DOI: 10.48550/ARXIV.1511.05952. URL: https://arxiv.org/abs/1511.05952.

[4] Christopher Schulze and Marcus Schulze. "ViZDoom: DRQN with Prioritized Experience Replay, Double-Q Learning, & Snapshot Ensembling". In: *CoRR* abs/1801.01000 (2018). arXiv: 1801.01000. URL: http://arxiv.org/abs/1801.01000.

[5] Ziyu Wang, Nando de Freitas, and Marc Lanctot. "Dueling Network Architectures for Deep Reinforcement Learning". In: *CoRR* abs/1511.06581 (2015). arXiv: 1511.06581. URL: http://arxiv.org/abs/1511.06581.