Homework 3 - Deep Reinforcement Learning

Simionato Giuseppe (2029013)

Academic Year 2021-2022

1 Introduction

The main goal of this work is to learn how to implement and test neural network models for solving reinforcement learning (RL) problems. It is shown how these models depend on some fundamental building blocks and how the proper implementation of the all framework of the reinforcement learning problem affects the final performances. The most general setup problem is made of an agent and an environment which interact through a loop. At every timestep, the agent receives a new observation of the environment which consist in the current state and a reward. The agent then chooses the next action to take from the set of actions available which lead to the next state in the environment. The goal of a such algorithm is to maximize the accumulated reward.

A powerful way to solve this kind of task is the Q-learning which allows to consider the policy used for predicting future rewards by relating actions to states. Accordingly with some policies, the actions are chosen probabilistically, in this work are tested two different types of policy: the ϵ-greedy and the softmax policy. Moreover, the exploration and exploitation dilemma is studied to find the best exploration profile for the training.

In this work, the RL problems are based on two different environments provided by the Open AI Gym library. Firstly, it is implemented a simple Deep Q Network (DQN) to solve the *"CartPole-v1"* game ([1]), where it is tested the impact of the exploration profile and of different reward types during the training. The second one is a more complex environment, an Atari 2600 game, called *"MsPacman-v0"* ([2]) where the acquired knowledge in the previous problem is applied to see how the difficulty in reaching good performances increase with the complexity of the space of observations and actions.

1.1 CartPole-V1

The cart pole is a game where a pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center. In this case, the **agent** is the cart on top of the which stands the pole and the **environment** is everything that can be observed, which in this problem includes four different variables:

- the cart's horizontal position where the point (0.0) indicates the center
- the cart's velocity where (0.0) means the agent is stopped
- the angle of the pole where (0.0) means it stands perfectly vertical
- the pole's angular velocity where (0.0) indicates the pole is not moving

In this simple environment, as already mentioned, are only possible two different discrete **actions:** move the cart toward left or right. Finally, the agent obtains a reward of +1 every time it takes an action until the pole does not fall and its goal is to avoid this event to happen, or at least, to make it happen as late as possible. The game can be consider completed if the total score reaches 500 points.

1.2 MsPacman-V0

MsPacman was released in 1982 as an arcade video game, and has immediately become one of the most popular video games of all time. In this second problem, the goal is to maximize the score in the Atari 2600 game MsPacman. In this environment, the **observation** is an RGB image of the screen,

which is an array of shape (210, 160, 3). Each action is repeatedly performed for a duration of K frames, where K is uniformly sampled from {2, 3, 4}. The environment is difficult to predict, because the ghost behaviour is stochastic. The **reward** can be well defined by relating it to in-game events, such as collecting a pill (+10) or eating a ghost (it depends on how many consecutive ghosts are eaten, see 7). Furthermore, there is an **action space** which consists of nine decisions which MsPacman can take: stay, move up, right, left, down or along the four diagonals. The episode ends when MsPacman finishes all her lives (three) or when she has eaten all the pills. In this last case the labyrinth can be considered completed and it will automatically pass to the next one, for a total of 4 different scenarios. The goal is to obtain the maximum possible score which is cumulative along the labyrinths.

2 Methods

2.1 DQN architectures

Starting with **CartPole** game, the chosen architecture is made by three linear layers each of them followed by the ReLU activation function and finally the model ends with another linear layer with the number of filters equal to the action space dimension, in this case two. The first layer takes in input a vector composed by four elements as the state space dimension, the number of filters for the linear layers are 512, 256, 64. The model is very simple due to the fact that the problem is not too complex and a more deep architecture with an higher number of trainable parameters may increase the computational time without any improvement in the final performances.

The constructed DQN for the second environment, the **MsPacman** game, is a Convolutional Neural Network (CNN). It takes in input a 3 channels RGB image and gives in output the value of the q-values of the corresponding actions. The possible actions will not be all the action in the space, because there are nine actions where only 5 are useful to win the game. Indeed, the last four actions correspond to the diagonal movements that increase only the complexity of the problem. The architecture is made of three convolutional layers followed by a maxpooling and the ReLU activation function, then there are two linear layers and the output one. The input images are normalized to make the model converge faster. Indeed, when the data is not normalized, the shared weights of the network have different calibrations for different features, which can make the loss function to converge very slowly and ineffectively.

States pre-processing: the images of the states are transformed in order to have a faster training and to simplify the learning of the agent. The states are firstly cropped from (210, 160, 3) to (170, 160, 3) where is removed the black box containing the representation of the lives of MsPacman: indeed this is not a relevant information and increase only the size of the image. Then the states are rescaled to the size of (100, 90, 3) using the cv2 resize function (with interpolation= INTER CUBIC) to obtain an lighter image. The last operation is to give in input to the DQN not the state, but the difference between the current state and the previous one. In this way the agent should see easily the changes after an action and should be able to extract the relevant information to improve the reward.

Replay memory: in the training of the DQN are used two different replay memories: one for the positive rewards and another for the negative ones. The samples for training are randomly chosen from these two memories (0.9 of the batch size from the positive one and 0.1 from the negative), in this way the agent can learn from both positive and negative results with a not arbitrary splitting for the two.

In both problems is implemented as **loss function** the Huber loss. This function uses a squared term if the absolute element-wise error falls below beta and an L1 term otherwise. It is less sensitive to outliers than the MSE loss and in some cases prevents exploding gradients.

2.2 Policy profiles

Starting from the estimated Q-values (one for each action), we need to choose the proper action. This action may be the one expected to provide the highest long term reward (exploitation), or maybe we want to find a better policy by choosing a different action (exploration). The correct implementation of the exploration profile is crucial in the Q-learning approach and there are several techniques to use. In the **first problem** are tested the performances using the ϵ-greedy policy and the softmax policy.

The first one chose a non optimal action with probability ϵ otherwise chose the action with highest Q-value. The second one chooses the action to perform according to their Q-value distribution where it is applied a softmax function with temperature τ. The most common actions will be the one with the highest Q-value. The stochasticity increases with the parameter τ.

Hence, the two policies have both a parameter to set the randomness of the decision. It is known that the agent needs to explore the environment in the first episodes to learn as much as it can, while it has to exploit what it learned in the latter part of the training. For this reason, the two parameters decrease exponentially in order to define the exploration profiles. What is the best exploration profile is one of the goal of this work and they will be tested and compared with the same hyperparameters in the same environment. In Figure 1 and 2 are shown the exploration profiles used to compare the two polices, while in Figure 3 and 4 there are the profiles of the best policy used for understanding the best trade-off between exploration and exploitation and to see the impact during the training.

In the **second problem** is only used one of the two policies (the epsilon greedy) with the exploration profile reported in Figure 5.

2.3 Rewards

2.3.1 CartPole-V1 rewards

In this first environment has been tried some different rewards to see the better ones to have a fast convergence and speed up the training, but also to obtain a robust and stable agent which can complete the game with an high probability. The tested rewards are:

- a bad penalty if loses the game
- an increment in the default reward each timestep
- a bad penalty if it is far from center
- a bad penalty if the cart velocity is far from zero
- a bad penalty if the pole angle is different from zero
- a bad penalty if the pole angular velocity is far from zero

The amount of penalty is decided experimentally.

2.3.2 MsPacman rewards

The rewards in this second problem are fixed. Rewards are determined based on the original MsPacman game. That is, rewards are offered when a game event occurs, such as collecting a pill or colliding with a ghost, for more details of standard game rewards see 7. The additional rewards used in this work are listed in Table 1. In the case of an action which collects multiple rewards, these rewards are added and then treated as if they were a single reward. The reward for reversing on a direction was added to discourage the agent from hanging around an empty corridor in the maze, while the step penalty is necessary to avoid inactivity.

Event	Reward	Description
pill	10	MsPacman ate a pill
powerpill	5	MsPacman ate a powerpill
ghosts	30	MsPacman and a scared ghost collided
step	-5	MsPacman performed a move
reverse	-6	MsPacman reverse on her path
lose	-350	MsPacman and a non scared ghost have collided

Table 1: MsPacman rewards

2.4 Hyperparameters optimization

The optimization of the hyperparameters is done experimentally with many trainings without an explicit implementation of a gridsearch or of other hyperparameters optimization techniques. However, the impact of the different values of the following variables is tested in the **CartPole** problem to see which are the most important ones and which are the less relevant for the final performances. The hyperparameters are the followings:

- the discount rate γ
- the replay memory capacity
- the type of optimizer
- the learning rate
- the target net update steps (after how much steps the target net is updated)
- the batch size
- the minimum sample for the training

The same hyperparameters are optimized in the **MsPacman** environment, however thanks to the study made on the first game, it is put more attention to the most relevant ones.

3 Results

3.1 CartPole-V1 results

In the following, it is important to keep in mind that the official CartPole webpage defines that the problem is "solved" if the score >195 over 100 consecutive trials.

Firstly, the environment is tested with a random agent that it is not able to solve the task (Figure 6). The first training is done using the DQN without updating the rewards and the results are shown in Figure 7.

3.1.1 Reward study

In the following, it is presented the results of the study of the impact of the different rewards to find what are the most important to implement to have a fast and robust training. Firstly, each reward is tested individually to see if the agent is able to learn how to win the game and in how many episodes. The results are shown in Table 2, where the last four rewards are multiplied by the value of the corresponding state variable. The three rewards that lead to complete the game do not train in a robust way the agent, in the sense that the game is completed more and more with the increasing of the episodes but even after 300 episodes it is not able to win enough times to consider the CartPole challenge solved. In order to solve the game, a combination of these rewards is used: in the second

Event	Reward	First game completed in
lose the game	-100	190 episodes
be alive	+100	>300 episodes
far from center	-100	>300 episodes
cart velocity $\neq 0$	-100	>300 episodes
pole angle far from vertical	-100	33 episodes
pole angular velocity $\neq 0$	-100	21 episodes

Table 2: CartPole rewards impact

stage of the reward study are implemented both the loss game reward and the pole angular velocity penalty with the same value of -100. The result is a more robust agent, able to complete the game in few episodes with an increasing rate of success with the played episodes. The problem of using these two rewards lies in the absence of constraints in the cart, indeed it is clear from the video of the games after 300 episodes that the cart moves faster outside the screen. For this reason, they are introduced two penalties in the cart leading to the final rewards implementation in Table 3.

Event	Reward
lose the game	-100
far from center	-100
cart velocity $\neq 0$	-50
pole angular velocity $\neq 0$	-100

Table 3: CartPole rewards final implementation

3.1.2 Hyperparameters impact

After different trials, the best hyperparameters found are reported in Table 4. In the following studies, the hyperparameters will be kept constant to these best values.

Hyperparameter	Best value	Considerations
discount rate	0.99	if < 0.99 the game is not completed in $<$300 episodes with these rewards
replay memory capacity	5000	increasing or decreasing this parameter does not influence a lot the final performance
optimizer	Adam	it does not influence significantly the performance
learning rate	$5 \cdot 10^{-3}$	if smaller does not converge, if grater complete the game in more episodes
the target net update steps	5	a small value reduces drastically the required episodes for completing the game (≈ 100 fewer episodes from 10 to 5 steps)
batch size	32	it does not affect too much the performances but with the available resources is required a small value
min sample for training	100	if too high increase the episodes required for complete the game, if too small the training is slow

Table 4: CartPole best hyperparameters

3.1.3 Policy study

The two different policy were trained with the same hyperparameters with the profiles already presented. The score with respect to the episodes are plotted in Figure 8. It can be seen that the faster and more robust agent is trained with the softmax policy.

After the training, the two agents are tested over 100 episodes to see if they are able to obtain a score $>$195 and solve the game. In Table 5 are reported the scores for the first 10 episodes and if the game is solved with that policy. From now on, it has been used only the softmax policy because it is the best to train the agent fro this problem.

3.1.4 Exploration profile impact

The exploration and exploitation, as already mentioned, is a crucial part in the RL framework. For this reason two different exploration profiles have been tested and the score trends are reported in Figure 9 and 10. From these results it is clear the paramount importance of the exploration (with this profile can solve the game), in fact without an adequate exploration the agent is not robust (the second profile cannot solve the game) as it is highlighted in the low exploration training. However, also the exploitation plays an important role, if the agent does not exploit what he learnt he cannot obtain high performances in the test. Hence, the best profile must include both of them, in these specific problem more exploration is required respect to the exploitation.

Episode nr.	ϵ-greedy policy	softmax policy
1	75	500
2	49	500
3	47	500
4	125	500
5	81	500
6	51	500
7	58	500
8	111	500
9	149	500
10	274	500
Solved?	no	yes

Table 5: CartPole test policy results

3.2 MsPacman-V0 results

The MsPacman agent is trained more times with the shown policy each time for 25 episodes due to the available resources. The best hyperparameters are: gamma = 0.99, replay_memory_capacity = 10000, lr = 2.5e-4, target_net_update_steps = 5, batch_size = 64, min_samples_for_training = 100. Every session is tested after the training: the agent plays 10 games and the average score is computed. The results are reported in Table 6. It is also reported the result for ten episodes for the random agent (Figure 11) to see how are the scores without any training. The training score trend of the last session is reported in Figure 12. It can be seen that the results are not good as in the previous

Session Number	1	2	3	4	5	6	7	8	9	10
Cumulative episodes	25	50	75	100	125	150	175	200	225	250
Average score	290	347	428	272	326	388	643	310	272	320

Table 6: MsPacman training results. Each session is composed by 25 episodes (with a training time on average of 450 seconds without the render) for a total number of 250 episodes

environment and the reason lies in the increasing complexity of the problem due to the higher spaces of actions and observations but also to the higher number of possible sequence of actions that can lead to various consequences. There is also the presence of the ghosts and the fruits which contributes to increase the complexity of the game. Furthermore, the training is slow and unstable.

One of the main problems in learning in this environment is the required computational effort for training the agent which lead to limit the training episodes and thus to limit the final performances. The selected rewards seems to be able to help the DQN to learn properly how to play, but for more precise considerations it should be trained for more episodes.

Moreover, the solutions adopted in this work are really simple compared to the ones that can be found in literature ([3], [4]) where it is shown that the reason that modern deep RL struggle with MsPacMan is not related to learning from pixels; the underlying issue is that the optimal value function for MsPacMan cannot easily be mapped to a low-dimensional representation. Starting from this considerations, more advanced strategies were build, as for example to train the agent in a simpler environment or create for each object a separate input channel, encoding its location extracted from the whole image.

References

[1] https://gym.openai.com/envs/CartPole-v1/

[2] https://gym.openai.com/envs/MsPacman-v0/

[3] https://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/MS_PACMAN_RL.pdf

[4] https://arxiv.org/pdf/1706.04208.pdf

Appendix

Object	**Reward**
pillow	10
power pillow	50
1^{st} blue ghost	200
2^{nd} blue ghost	400
3^{rd} blue ghost	800
4^{th} blue ghost	1600
cherry	100
strawberry	200
orange	500
pretzel	700
apple	1000
pear	2000
banana	5000

Table 7: MsPacman game rewards

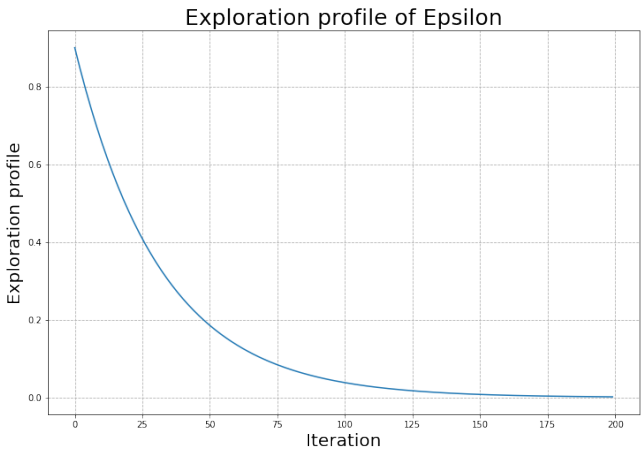

Figure 1: CartPole Epsilon policy profile

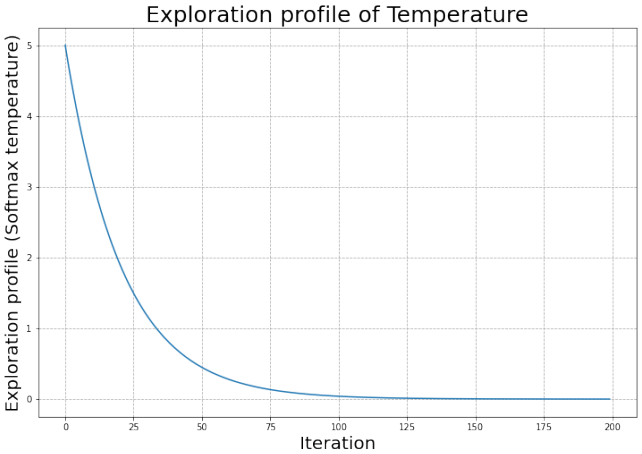

Figure 2: CartPole Softmax policy profile

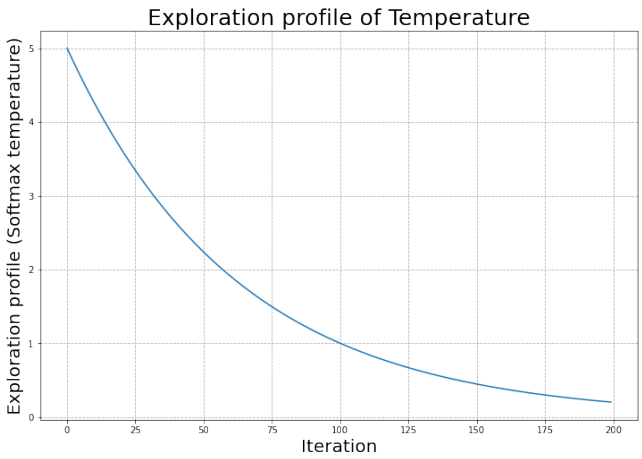

Figure 3: CartPole High exploration, low exploita-tion

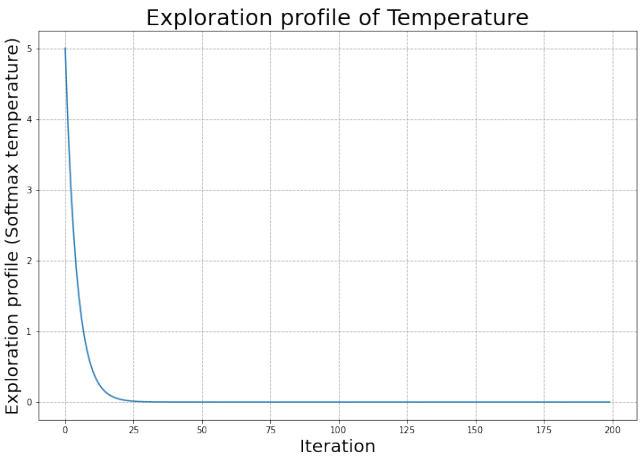

Figure 4: CartPole Low exploration, High exploitation

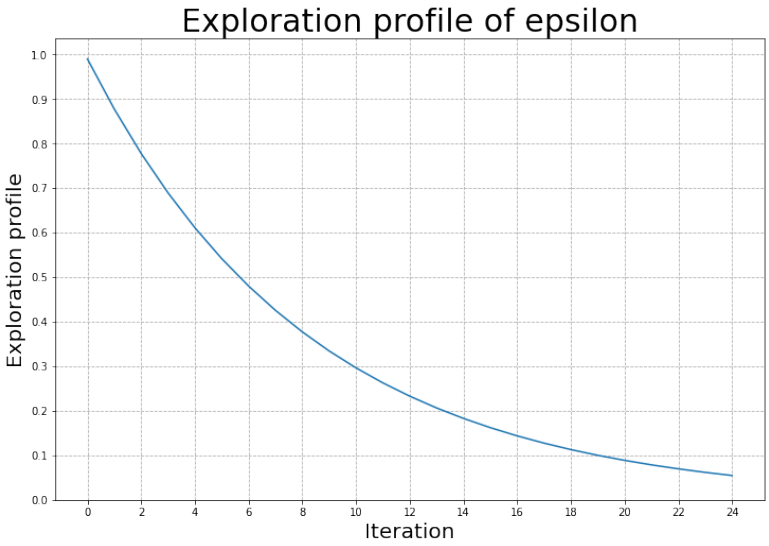

Figure 5: MsPacman exploration profile

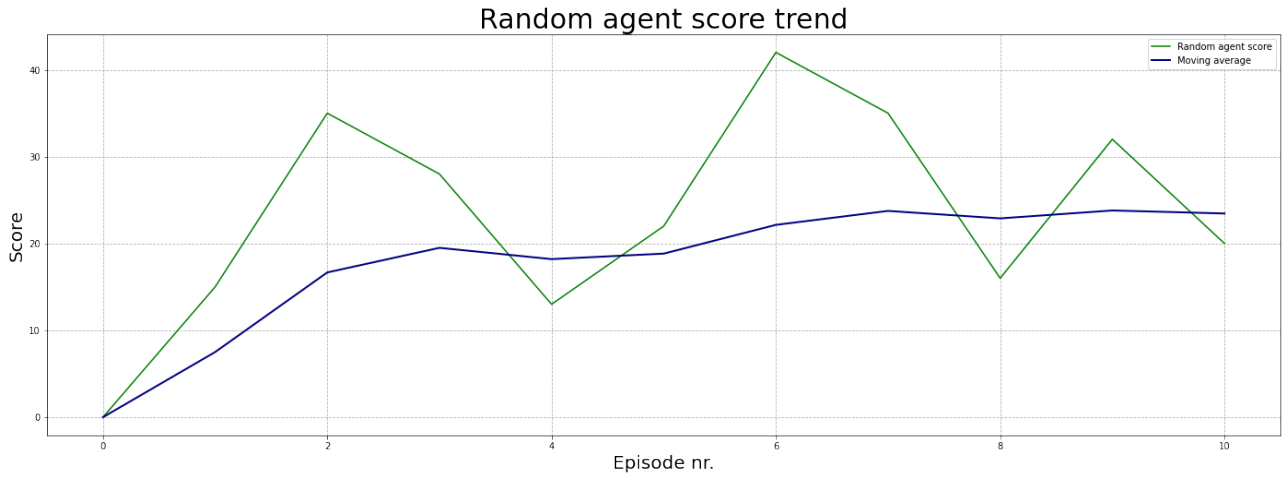

Figure 6: CartPole Random agent score trend

Figure 7: CartPole First training score trend

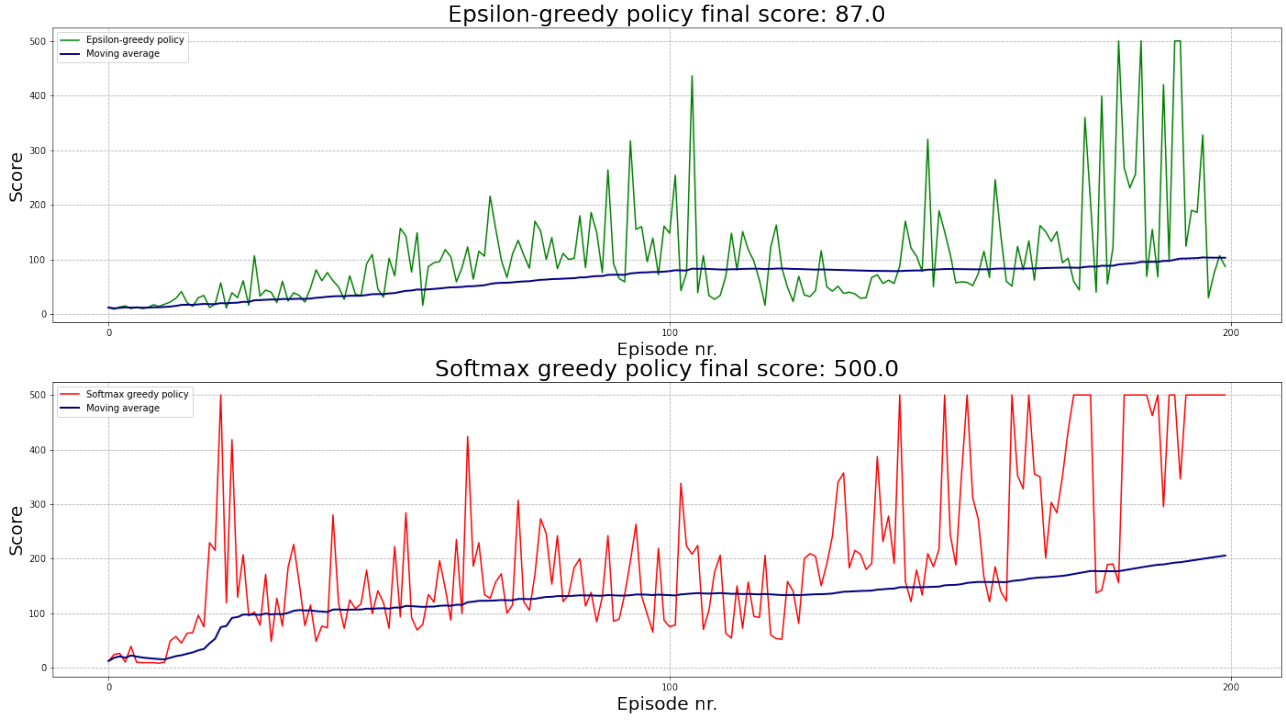

Figure 8: CartPole Different policy score trend

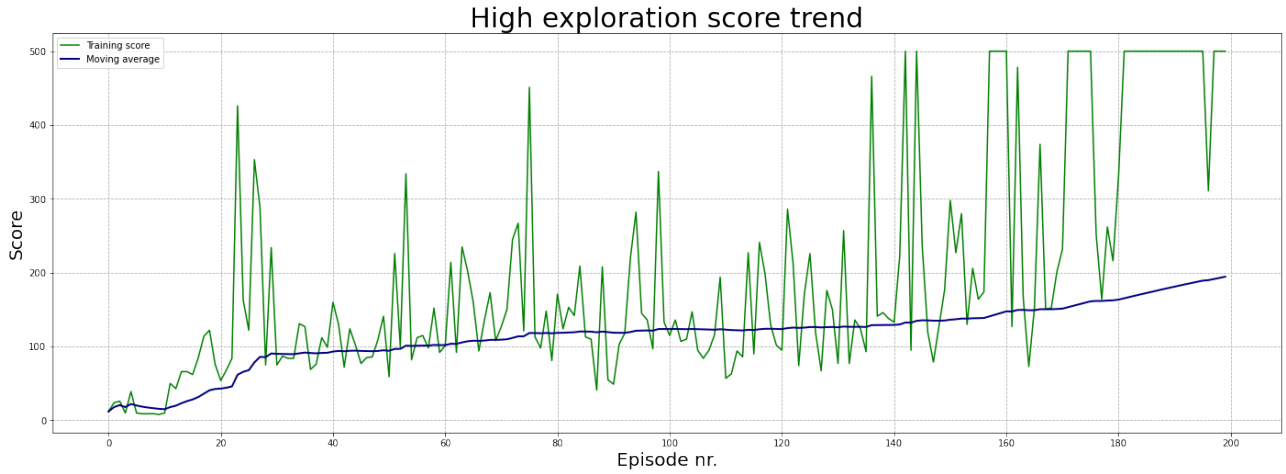

Figure 9: CartPole High exploration score trend

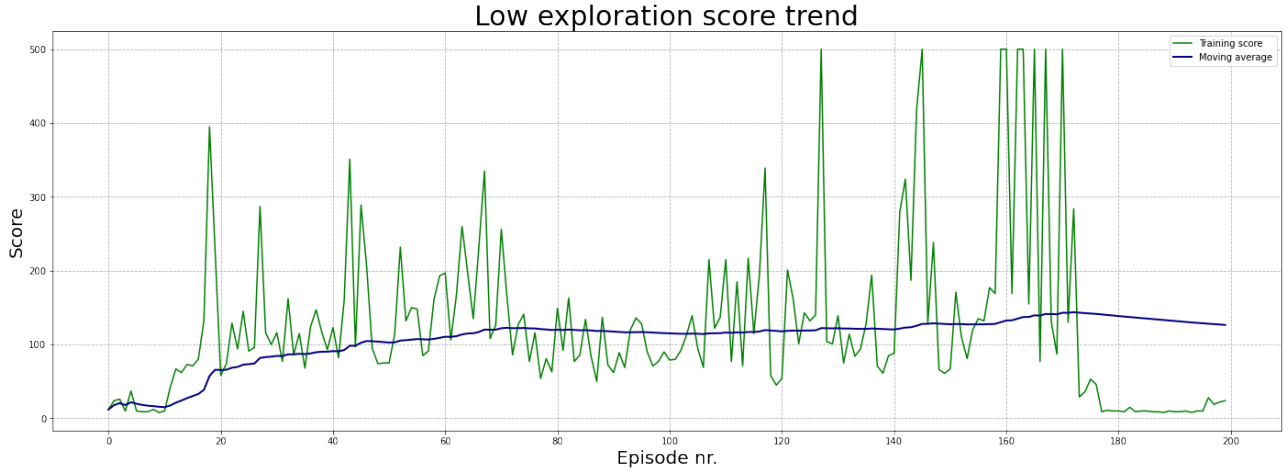

Figure 10: CartPole Low exploration score trend

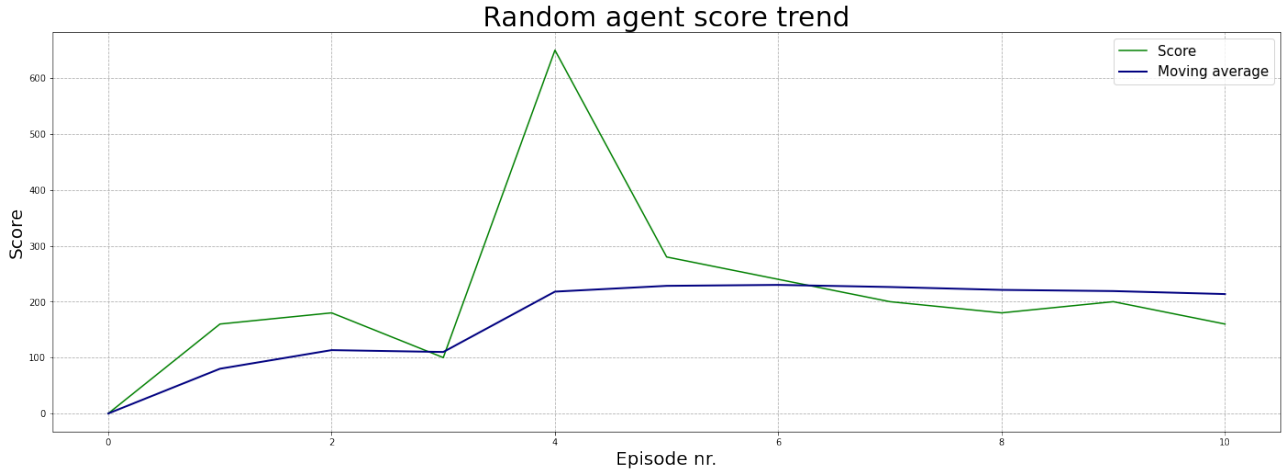

Figure 11: MsPacman random agent ten episodes score

Figure 12: MsPacman DQN score, last training session