HOMEWORK 3 NNDL

Zambelli Francesco (2029014)

Academic Year 2021-2022

1 CART-POLE

In this section a reinforcement learning algorithm able to solve the Cart-Pole game from the Gym environment was developed, and some characteristics of the training behaviour in different cases are explored. The **goal** of the game consists in keeping a pole straight by moving the base upon which it is hold. The game is lost when the pole angle exceed a given range around 0, or when the base touches the edged of the display. The game is solved when the player is able to not lose for 500 straight time-steps.

The main points of the work are:

1. Definition of the replay memory class and the policy network
2. Definition of functions for action choice based on the state of the net, both with **Softmax** and **ϵ-greedy** strategies
3. Definition of the function for the generation of an **exploration profile** for the temperature τ and ϵ parameters
4. Definition of the function to **update the network** with samples stored in the replay memory
5. Training function which takes in input all the parameters of interest for the system, in which some **adjustment in the reward** computation at each step were added in order to improve the performances
6. Exploring the differences in the score trends with respect to the **reward's computation choices** adopted
7. Different training trials with **different exploration profile**, with the goal to explore the differences between an high exploration and a high exploitation strategy, both using the Softmax and the ϵ-greedy strategies.
8. Study on the **variability of the scores** in each case reported above.
9. Random **hyperparameter search** in two cases. The first one has the goal to minimize the number of iteration to reach the maximum score in the game, the second aims to maximize the mean score in the last iterations of the training.
10. **Test** of the algorithm at the end of the training and display of the **trends of the Q-values** w.r.t. each time step of a single game trial.
11. Trends of the Q-values in each of the test trials.

1.1 Methods

The function and classes that compose the Q-learning algorithm for this first task are almost the same as the ones used in class for lab7. We briefly sum up the function of each one.

The **replay memory** class allows to store a number of samples up to a fixed maximum and randomly retrieve a fixed length batch of elements. In this structure we store some samples from the training experience of the game in order to train the policy net to better predict the future values of the Q-values.

The **policy net** is composed by neural network made of 3 Linear layers with Tanh activations and output dimension equal to the possible actions to take, which in the Cart-Pole case is equal to 2.

Then the 2 functions to choose the action to take given the current state of the environment are defined. The first one uses the **ϵ-greedy** strategy, i.e. the action with highest Q-value given as output by the policy net in that iteration is actually taken with probability $1 - \epsilon$, and with probability ϵ one of the other possible actions is chosen with identical probability (ϵ is a parameters given in input). The second one uses the **Softmax** procedure, i.e. all the possible actions are taken with a probability given by the output of a *softmax* function of the Q-values. The *softmax* function depends also by a temperature parameter τ which quantifies the degree of randomness with which one of the possible actions is chosen. For $\tau = 0$ we have that the action corresponding to the highest Q-value within the prediction of the policy net is certainly chosen, while for $\tau >> 1$ all actions are equiprobable.

Then the function that return the **exploration profile**, i.e. the decreasing schedule for the τ or the ϵ parameters of the previous function w.r.t. the training trials. Some small modification were introduced at this point: the **steepness** parameter is added, which changes the *decreasing speed* of the exponential function. This allows to easily pass from an exploration policy (small steepness values), in which the function decreases slowly and end with a derivative close to 1, to an exploitation one (high values of steepness), in which the function gets really close to 0 after a small number of iterations, so form that point on, it's almost sure to choose the action with the highest Q-value.

The function for the **network updating** trains the policy network using the output of the target net, an identical model with weights kept fixed to the ones obtained from the last training iteration. After the updating, the weights of the policy network are copied into the target net.

In the **manual training function** it's possible to initially define the exploration profile w.r.t. some parameters as the initial value to take (must be in [0,1] in the case of the ϵ-greedy policy), the steepness and the number of iterations. Then the Gym environment is defined and all the commands for the algorithm training are provided. In particular at each iteration some extra terms in the reward were added in addition to the constant contribute gained at each time-step. Alongside to the position contribution, which penalizes states of the game in which the base is far away from the center of the display, an **angular contribution** is added, in order to give a penalty with increases with the incline of the pole. It's also possible to insert a bed penalty in the case in which the game is failed. In addition to the exploration profile parameters, other parameters can be passes to the function, as the policy type (Softmax or ϵ-greedy), the values of position and angular weights, which respectively multiply the corresponding reward contribution to reinforce the importance of one or the other, the bad state penalty value, and some more value related to the structure of the replay memory and the optimizer.

1.2 Results

As regards the different possibility for the reward computation during the training, the cases of 1) **no extra terms** (aside from a constant term always received when the game doesn't fail in that iteration), 2) **only position contribution**, 3) **only angular contribution** and 4) **only bad state penalty** were explored. In all this cases all the other parameters of the training function are kept fixed, in order to better compare the different outcomes. In tab 1 the results are reported, and as can be clearly seen, the case in which only the angular contribution is kept is far better then the others. In particular also the case of the position contribution is able to reach good performances and at the end is able to solve the game producing good scores with a good stability, while in the case of no extra term or bad state penalty, the algorithm struggles to learn useful patterns and the score 500 is reached only a few times, maybe due to stochastic fluctuations.

After this, **different exploration profiles** are tested. In particular for both the polices, ϵ-greedy and Softmax, an *exploration* and an *exploitation* profile are defined. For the first case a high value of **steepness** is chosen, so the exploration profile quickly decreases reaching after a few iterations an almost null probability of choosing not optimal actions, while for the second case the steepness value is low and the exploration profile deceases slowly. To better explore the differences between these two methods, it was chosen not to set the reward's parameters with the most performing combination (high value angular coefficient and all the others small or even null, as was shown before and will be formalized in the hyperparameters search), in order to avoid the quick flattening of the score trend on the constant line of height 500. In general it's possible to see that high exploitation profiles work better in the Cart-Pole game, reaching high scores before than the high exploration profile and having higher mean scores. This is probably due to the small number of possible action to take, which makes the exploration of different possible actions patterns a not so useful element. A possible advantage of the exploration profiles is that it seems that in some cases the trend of the score increases is more stable compared to the one of the exploitation ones, often subject to fast fluctuations. In order to verify this the standard deviation of the last x iterations is computed for all the 4 different cases, making vary x vary from 10 to 200. For x around 130, the high exploration profiles are actually less noisy than the high exploitation ones.

1.3 Hyperparameters search

From the considerations of the previous section, 2 approaches were taken in order to perform the hyperparameter search. In both cases a **Random sampler** inside the Optuna *study* algorithm is used, while the difference lays in the final value of each iteration that the algorithm tries to optimize. In the first case the goal is to minimize the **number of iteration required to obtain the first score of 500**, in the second one to maximize the **mean score obtained in the last 150 iterations**. The results are reported in tab. 2 and 3. In both cases it's possible to see that as regards the reward computation policy, a high value of the angular coefficient is preferred, while the position one and the bad state penalty are not so influential.

As regards the steepness, optimal results are achieved both for high and low values, so we can conclude that this factor is not so determinant if the rewards coefficients are optimally tuned. Similarly for the chosen type of policy (Softmax and ϵgreedy).

1.4 Testing

After all this procedure, the policy network trained with only the "only angular contribution" method is chosen, and some games are played in order to show some results with the Softmax parameter τ set to 0. As expected, in all the cases the perfect score is reached, and in order to better understand this some useful plots are produced.

In the one displayed in fig. 3, the Q-values trends for both the possible actions in time are displayed, and at each time-step the point corresponding to the action chosen by the environment is highlighted with a dot of the corresponding color. As expected, the actions with the highest Q-value are always chosen, and the switch between the one and the other is almost periodic (2 times it moves to the left and 2 times moves to the right), a good strategy to keep in balance an object only subject only to a sort of "gravitational force".

In the fig. 4, instead, the trends for the Q-values relative to a single action are displayed for the 10 test trials. Thanks to this we can have an insight about the behaviour of such quantities in the case of success of the algorithm. In particular it's possible to see that the Q-value for each time-step remains almost constant, fact that tells that the network is capable to correctly foresee what will happen in the next instants based on the current state. Sometimes some decreases can bee seen, related to the fact that the system fell in an unstable state, but, for a well trained algorithm, in most of the cases the action are chosen in order for the Q-values to come back near to the constant value typical of the optimal state.

2 CART-POLE using frames

In the following section it is described the implementation of a reinforcement learning algorithm which trains a net to play the Cart-Pole game by using the **frames** of the display as inputs of the policy network. The main points are:

1. Define a **class** to easily handle the features of the **Cart-Pole Gym environment**. It also allows to perform a *preprocessing* of the image, with the goal to reduce the dimension, and so the computational weight for the network, while trying to retrain the highest quantity of information.
2. Definition of a policy network able to process images.
3. Redefinition of the update step and manual training function in order for them to match the new structure of the problem.
4. Training of the algorithm, both using the standard approach and another one based on the **transfer learning**.
5. Results presentation and analysis of the score trend.
6. **Q-values trends** analysis.

2.1 Methods

Firstly a class is created in order to better manipulate the Cart-Pole Gym environment. This allows to perform basic function, as restarting, closing, rendering and taking actions, but the main reason this class was implemented is to **perform a preprocessing** that allows the images that the "game display" returns at every iteration, to be reduced in size and appropriately modified in order to minimize the number of pixels which provides actual information about the state of the system. In order to do so, a crop around the current position provided by the gym environment is performed, keeping inside the image only the base and the pole, then the dimension in reduced thanks to the Resize function from the *torch.transforms* library and finally the current image is subtracted with the previous one in order to give as output only the pixels which represent the change of the system during that time-step. An example can be visualized in fig. 5.

Then the policy network is redefined in order to deal with images: a **bipartite structure** was chosen. The first block consists in a **convolutional** architecture, with ReLU activations, while the second one is composed by 3 Linear layers with Tanh activations with final output equal to the number of possible actions.

After this the update-step function is also redefined in order to be adapted to the new structure of the problem.

Given the increased complexity of the task, two different methods were explored.

The first one, which is also the most simple one, consists in performing a training very similar to the one of the first task. The chosen number of iteration is equal to 5000, because training the network to correctly predict the best action is much harder in this case in which the algorithm can only access to the images rather than the case in which the input of the net is a vector of compressed information. The exploration profile was chosen with a quite high level of steepness, for the consideration explained before. In order to help the convergence, the reward is computed by considering also the angular contribution as in the previous case. This choice derives from the fact that, given that the images received in input was cropped around the base, it's not possible anymore to access to information related to the position of the pole in the display. All the effort are focused on trying to minimize the inclination of the pole.

In the second case a **transfer learning approach** was used. Initially the first block of the policy network with a couple of extra final Linear layers stacked together was trained to predict the value of angular position and angular speed of the pole, information provided by the Gym environment and ideally accessible from the chosen image's format. This is done by using the policy which best performed in the previous section (the one with only angular weight), adding some noise in the action to choose in order to generate some stochasticity in each game and explore more states. A set composed by pairs image-state was produced, and then the network is trained using the usual methods with MSE loss. The goal was to train an architecture able to infer the main features of the images providing useful information about the state. After this, the weights of the first block composed by convolutional layers were blocked and attached to the second part of the policy network. Then the main Q-learning training was performed in a similarly to what was done in the first method, with the advantage that now it is necessary only to update the 3 last layers of the architecture, while the first part remained unchanged. The number of iteration is equal to 5000 and the exploration profile is the same of the one of the previous method. The policy network used in this method was slightly modified with respect to to the one used before because it was chosen to not reshape the images and only to crop them around the position of the base. In this way we can access to an higher quantity of details and this won't affect so much the speed of the training phase because the weights of the convolutional part are fixed.

2.2 Results

The results of the first method are reported in fig 6. The algorithm is not able to achieve the goal of the game, the 500 score is reached only once, but it's still possible to observe an increasing trend of the score, in particular between iteration 1500 and 2000, meaning that it was able to learn a strategy, at least partially.

The second method performs better, and the algorithm is able to reach the 500 score a not negligible number of times (15), altough not in a stable way. In fig. 8 is reported the trend with respect to the iteration number and the exploration profile, while fig. 7 represents the loss trend relative to the fine-tuning procedure.

2.3 Test

By testing both the algorithm using the Softmax policy with temperature $\tau = 0$, it's possible to see that as regards the first method the reached scores are around 100, while for the second one they have on average grater values, around 200. In order to visualize better what the network does in this last case, in fig 9 are represented the Q-values trends of both the possible actions with respect to the angular position of the pole. For the first part of the trial, the Q-values fluctuate around a value close to 20, as in the case represented in fig. 3, and the angular position fluctuates rapidly around 0, but a certain point some actions made the Q-values decrease suddenly and the instability produced in that moment lead to the failure of the game.

3 MOUNTAIN-CAR

In the following section it is described the implementation of a reinforcement learning algorithm which trains a net to play the Mountain-Car game. It consists in driving a car, subject to the a sort of "gravitational force", up to a hill until it reaches the top of it. The difficulty stays in the fact that for reaching the top it's necessary for the car to go backward until it reaches the top of a smaller hill situated behind it, and from there accelerate in order to gain enough speed to reach the final point. The algorithm was trained in an analogous way to the Cart-Pole one, with some small changes for what concerns the rewards. Some features and statistics are shown at the end, alongside to a graph with shows how the Q-values for a solved trials changes with the position.

3.1 Methods

The function used for this section are the same of the first task. The only differences are the number of possible actions, 3 in this case, accelerate to the left, to the right or not accelerate, and the way to compute the rewards. In particular in order to obtain good results, the reward is added with the value of the **velocity**, which is positive when the car goes to the right and negative in the opposite case, and the **square of the absolute value of the position subtracted with the value corresponding to the bottom of the concavity**. The first term favors high velocity in the positive direction, and this could conceptually lead to a difficulty, because it's necessary for the cart to climb to the top of the hill in the back, and to do this it's necessary to move with a negative speed for at least a short interval of time. This effect is partially mitigated by the position term, which resembles a gravitational potential (the height increases as the square of the position centred in the bottom of the concavity), and favours the states in which the cart is far away form the center, and this happens both when it goes to the positive (right) and negative (left) direction.

3.2 Results

As can be seen in fig. 10, the algorithm is able to reach the goal of the game. Some extra graph are provided to get an insight about what happens during the training. In fig. 12 the distribution of the minimum and the maximum of position and velocity for every time-step are shown; in both cases it's possible to notice that the mean value of the maxima continuously rise with each time-step until iteration nr. 600 more or less, element that indicates that the cart is driven to reach higher and higher positions by the way in which the rewards are computed, then there is an rapidly decreasing and finally it raises again until reaching a mean value even higher than before, meaning that the algorithm found an even better strategy.

3.3 Test

Finally the algorithm is tested, and some games are played using the policy network trained before with Softmax policy with $\tau = 0$. In all the cases the cart is able to reach the final position before the time-limit, and in fig. 13 it's possible to see the trends of the Q-values for one of these trials alongside to the corresponding position of the cart. The chosen action at each time-step is highlighted with a dot of the corresponding color. One thing that can be immediately noticed is that the network almost never chooses the action corresponding to not accelerate, and prefers to always move, probably being able in this way to reach the top faster.

	1)No extra term	2)Pos contr.	3)Ang. contr.	4)Bad state pen.
Mean score	74.3	98.6	402.9	67.6
Game solved at it. #	491	517	126	–
Number of successful games	1	59	435	0

Table 1: Some parameters relative to the score trends for training with different reward's computation policies in the Cart-Pole case.

Init. value	**Steepness**	**Learning rate**	**Targ. net. upd. steps**
13	9	$4.5 \cdot 10^{-4}$	5
Bad state pen.	**Pos. weight**	**Ang. weight**	**Policy**
10	0	13	Softmax

Table 2: Result of the hyperparameters optimization of the Cart Pole training algorithm which tends to minimize the time used to reach the first perfect score.

Init. value	0.18	13	0.45	0.28
Steepness	1	9	7	9
Learning rate	$1.6 \cdot 10^{-4}$	$4.4 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$5.0 \cdot 10^{-5}$
Targ. net. upd. steps	11	5	16	5
Bad state pen.	1	10	1	0
Pos. weight	1	0	1	1
Ang. weight	15	12	9	5
Policy	Greedy	Softmax	Greedy	Greedy
Score	500	500	487.6	497.4

Table 3: Some parameters combination relative to the trials with high scores in the hyperparameters optimization section of the Cart Pole training algorithm which tends to maximize the mean score of the last 150 iterations.

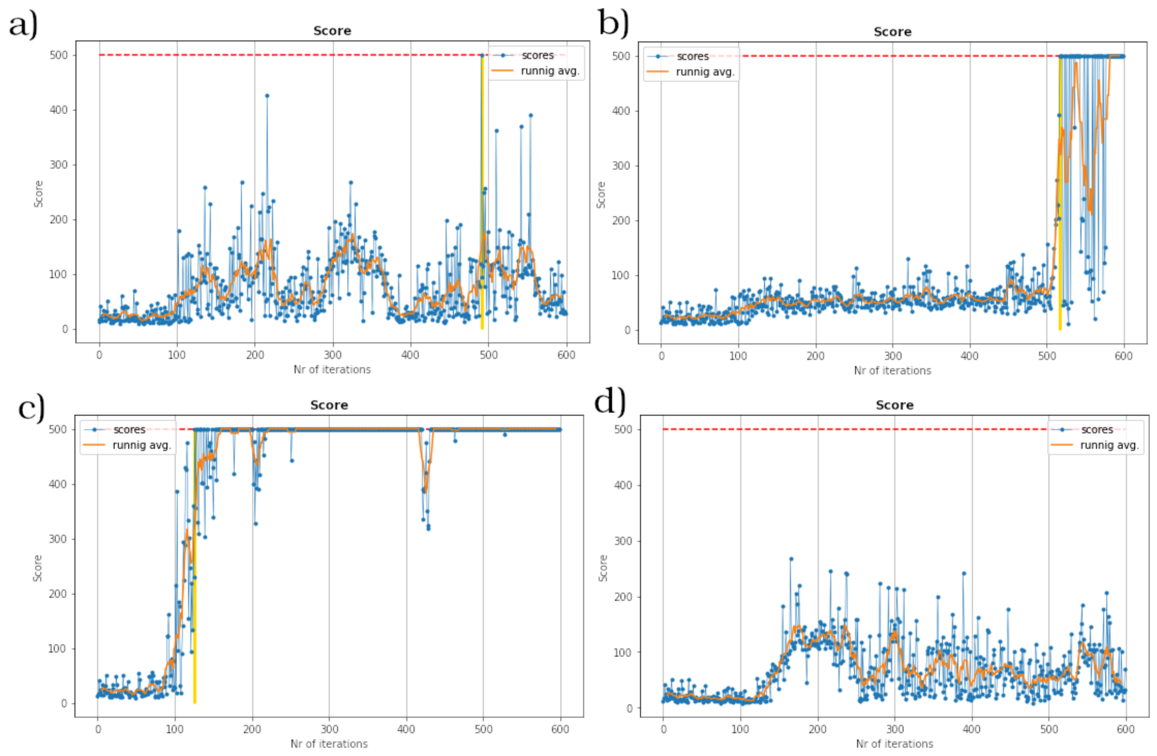


Figure 1: Score trends for training sessions of the Cart-Pole algorithm with different reward computation methods: a) no extra terms b) only position contribution c) angular contribution d) bad state penalty

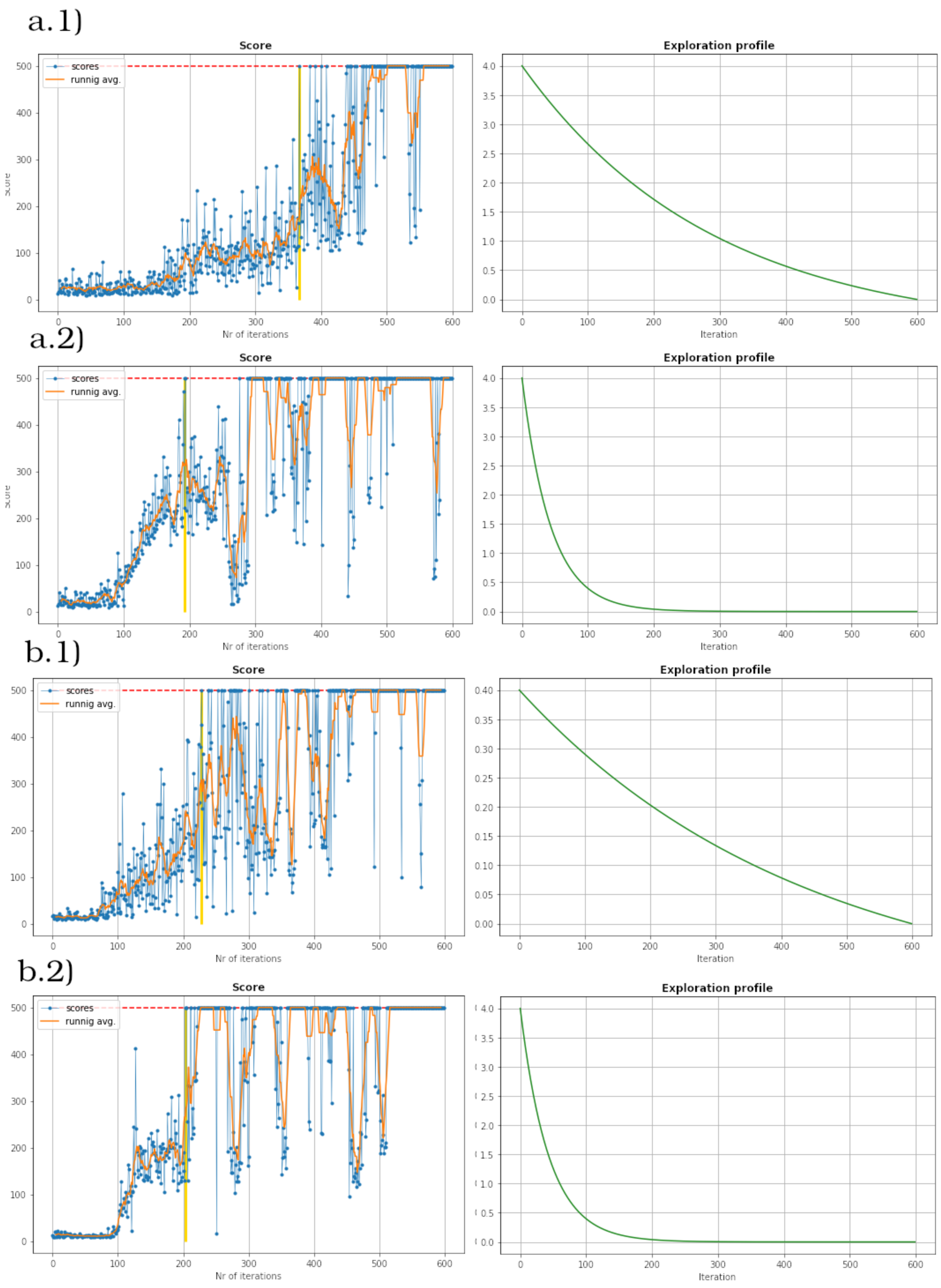
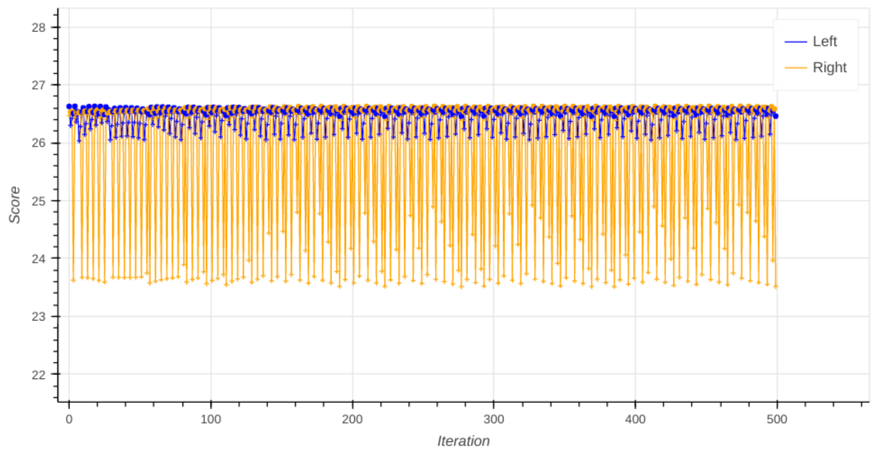


Figure 2: Score trends and exploration profile for different training procedures of the Cart-Pole algorithm: a.1) Softmax policy, high exploration a.2) Softmax, high exploitation b.1) ϵ-greedy, high exploration b.2) ϵ-greedy, high exploitation

a)



b)

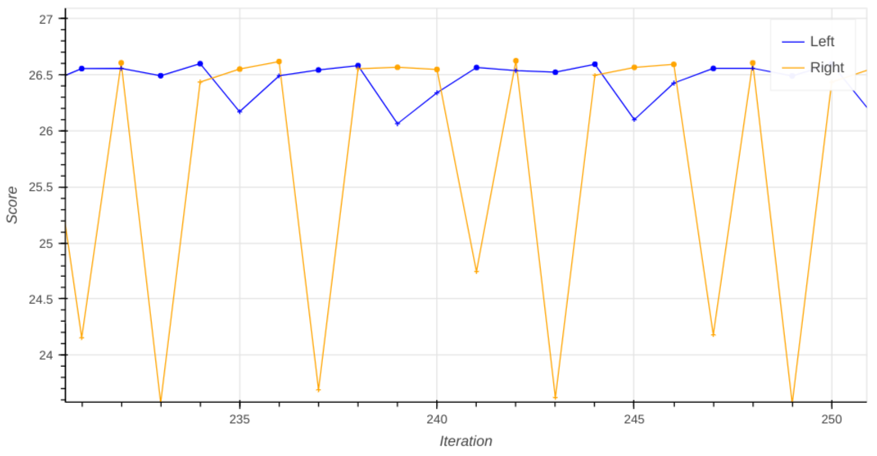


Figure 3: a) Q-values trends of both the possible actions for a test trial of the Cart-Pole algorithm. At each time-step the dot corresponds to the chosen action. b) Zoom of the previous image with the goal to highlight the periodicity that characterize the actions choice.

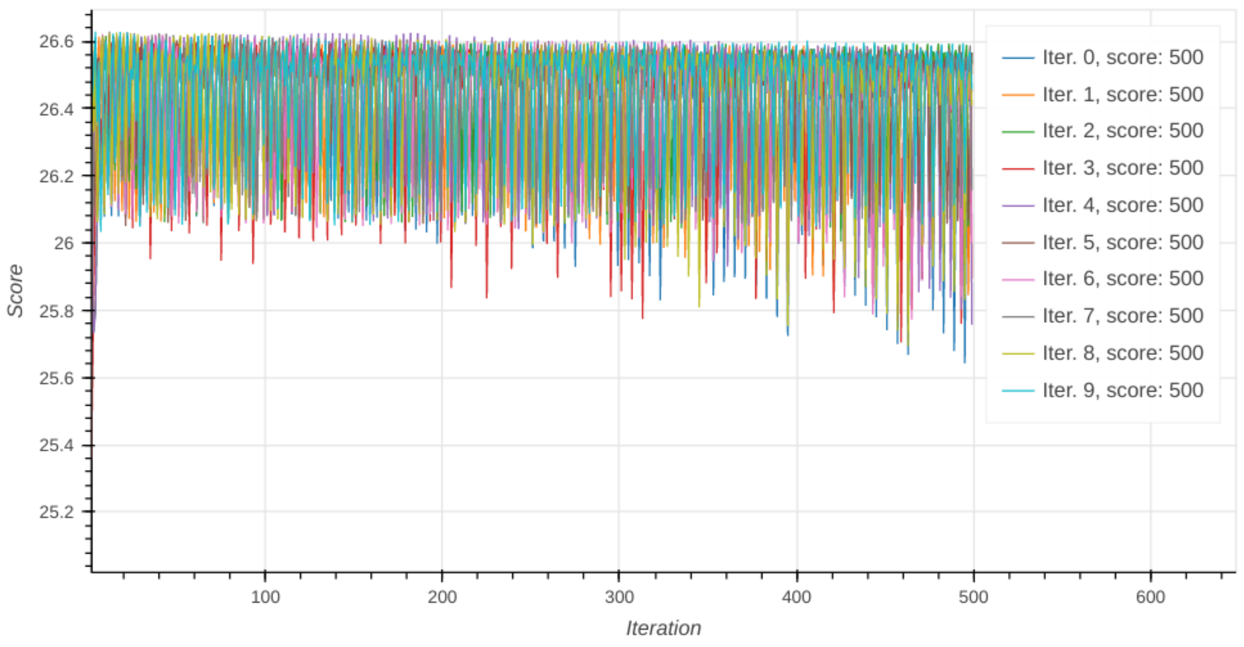


Figure 4: Q-values trends relative to only one action for all the 10 test trials of the Cart-Pole algorithm. It's possible to see that they all tend to keep a constant value

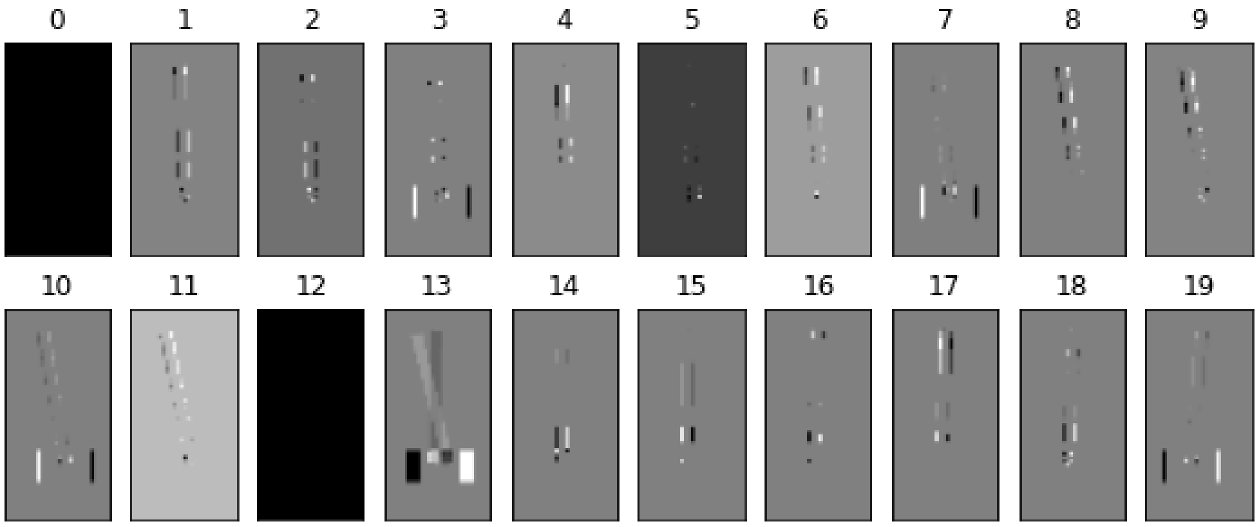


Figure 5: Some examples of processed images from the Cart-Pole environment

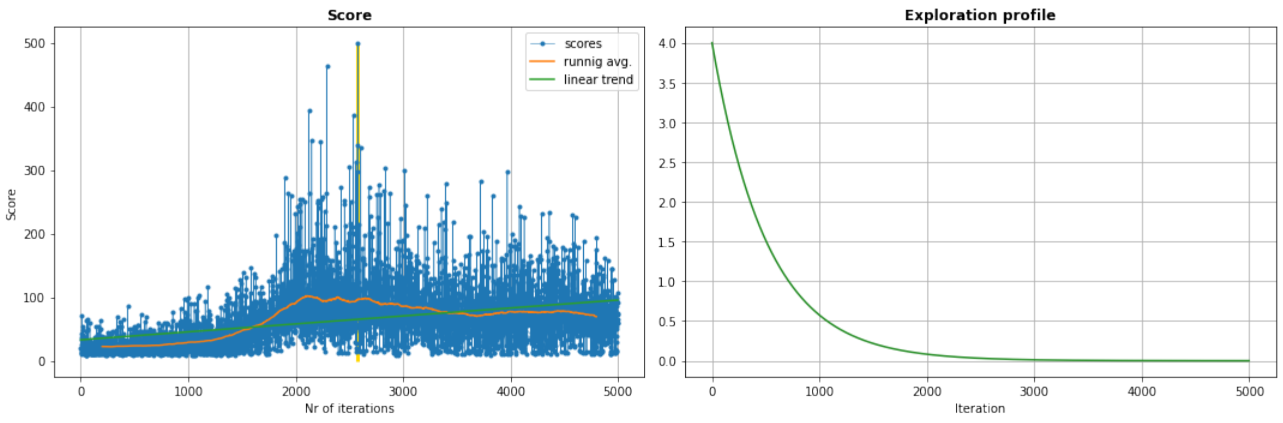


Figure 6: Results of the training of the Cart-Pole algorithm with images as input, first method. On the left: the blue dots represent the score for the Cart-Pole algorithm training at each time-step, the orange line is the running average, the green line the linear fit of the score values and the yellow one highlights the position of the first trail in which the score 500 is reached. On the right: adopted exploration profile.

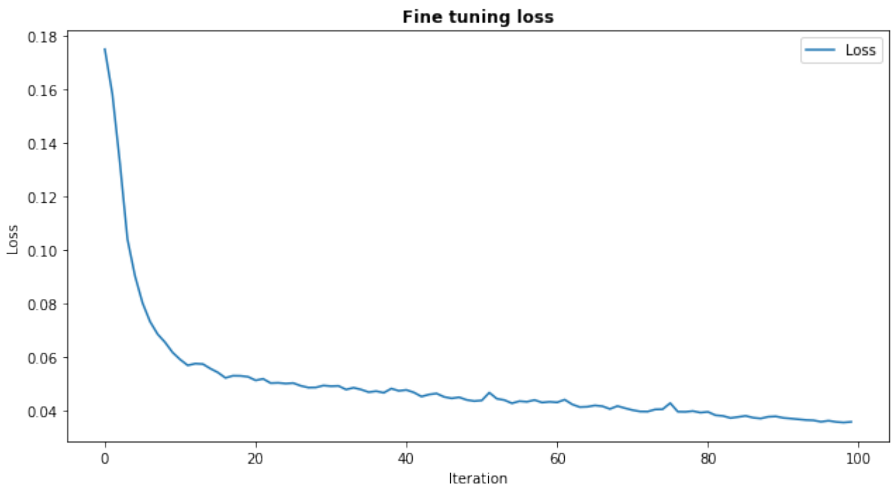


Figure 7: Loss trend for the fine-tuning procedure for the Cart-Pole algorithm trained with images

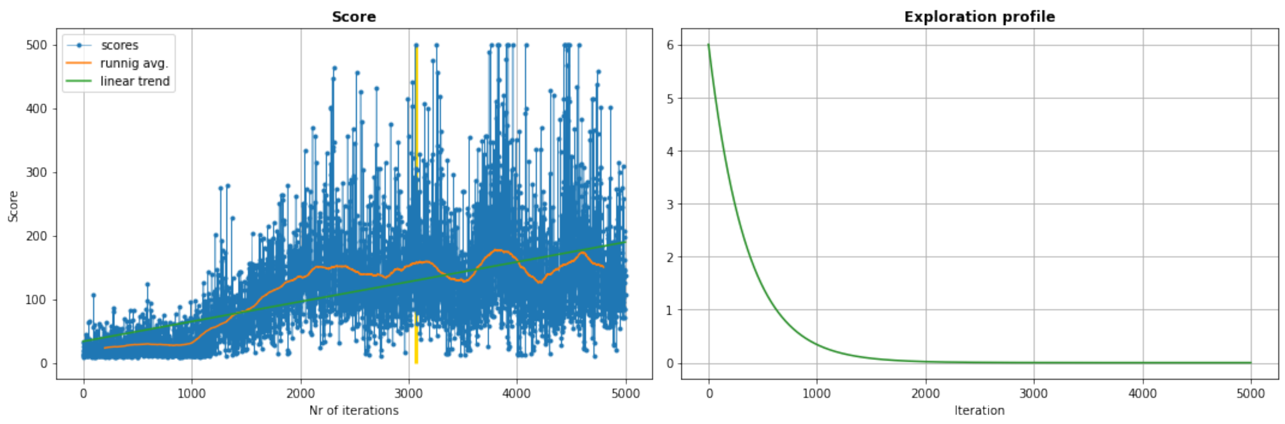


Figure 8: Results of the training of the Cart-Pole algorithm with images as input, second method. On the left: the blue dots represent the score for the Cart-Pole algorithm training at each time-step, the orange line is the running average, the green line the linear fit of the score values and the yellow one highlights the position of the first trail in which the score 500 is reached. On the right: adopted exploration profile.

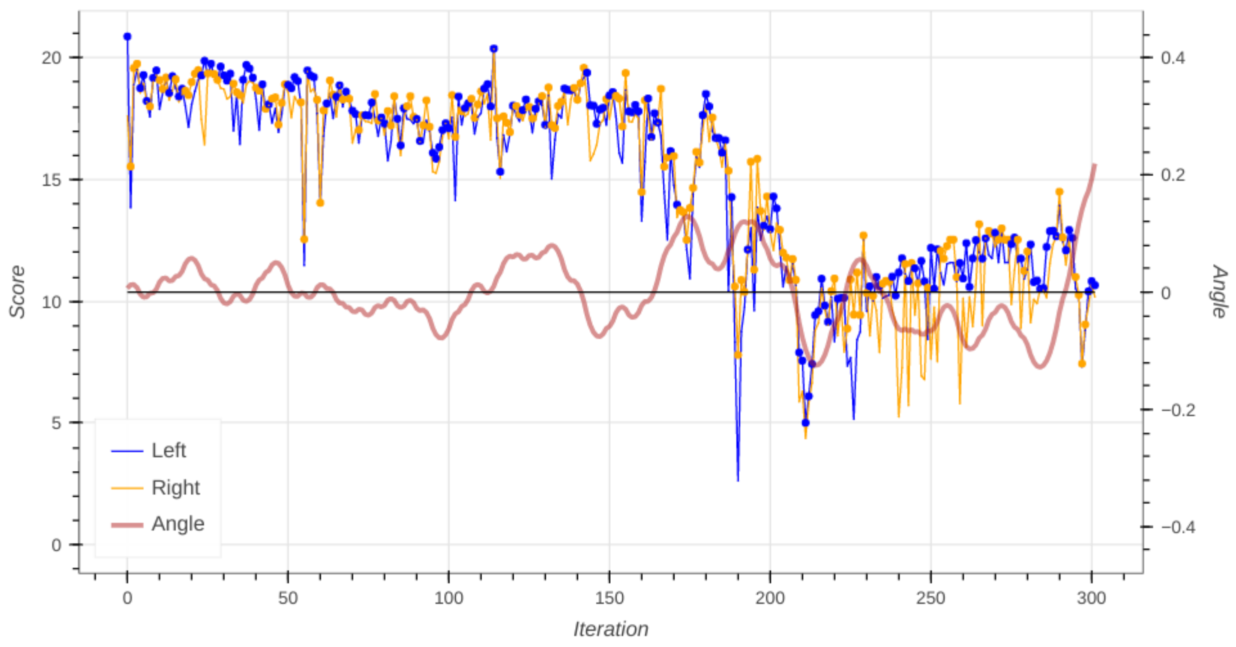


Figure 9: Comparison between the angular position of the pole (red line) and the Q-values trends (orange and blue line). The colored dots represent the corresponding action chosen at that time-step.

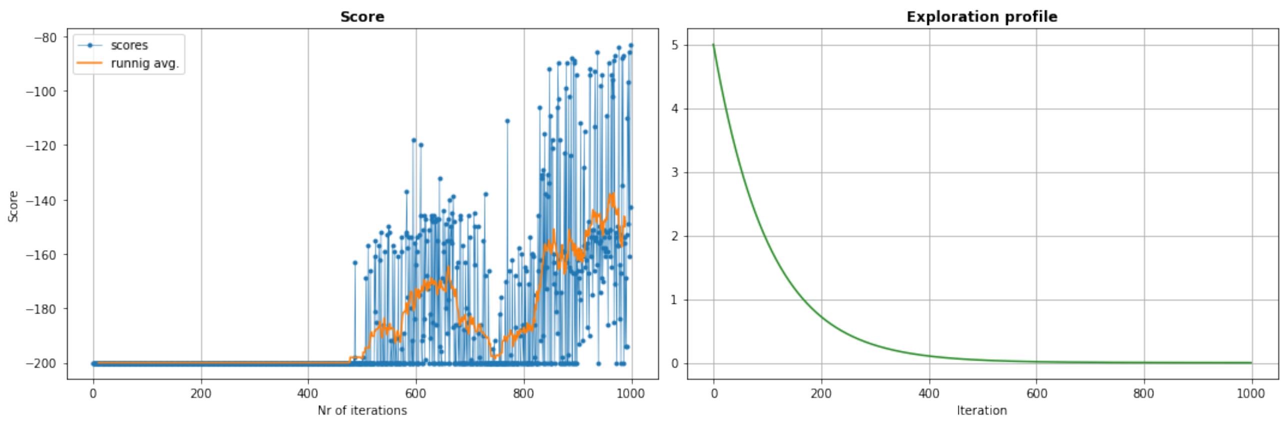


Figure 10: On the left: the blue dots represent the score for the Mountain-Car algorithm training at each time-step, while the orange line is the running average. On the right: exploration profile adopted for the Mountain-Car algorithm training.

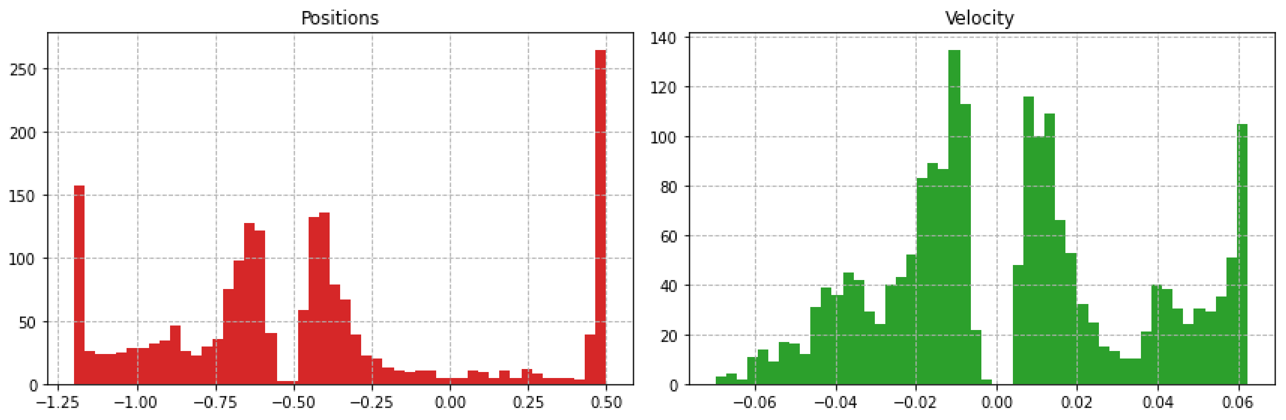


Figure 11: Histograms of position and velocity of the Car at the end of each training trial.

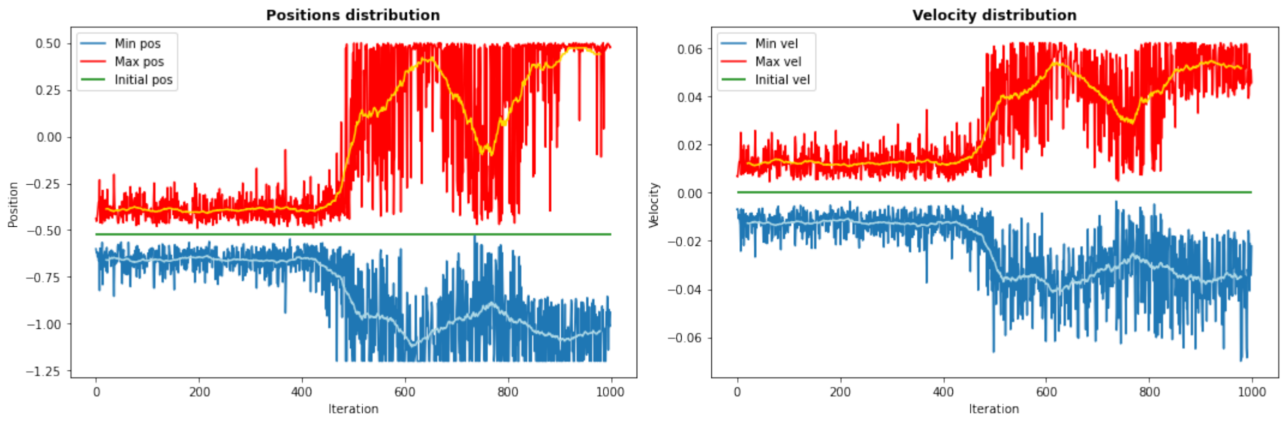


Figure 12: Trends of final position (left) and velocity (right) for each training trial with corresponding running averages.

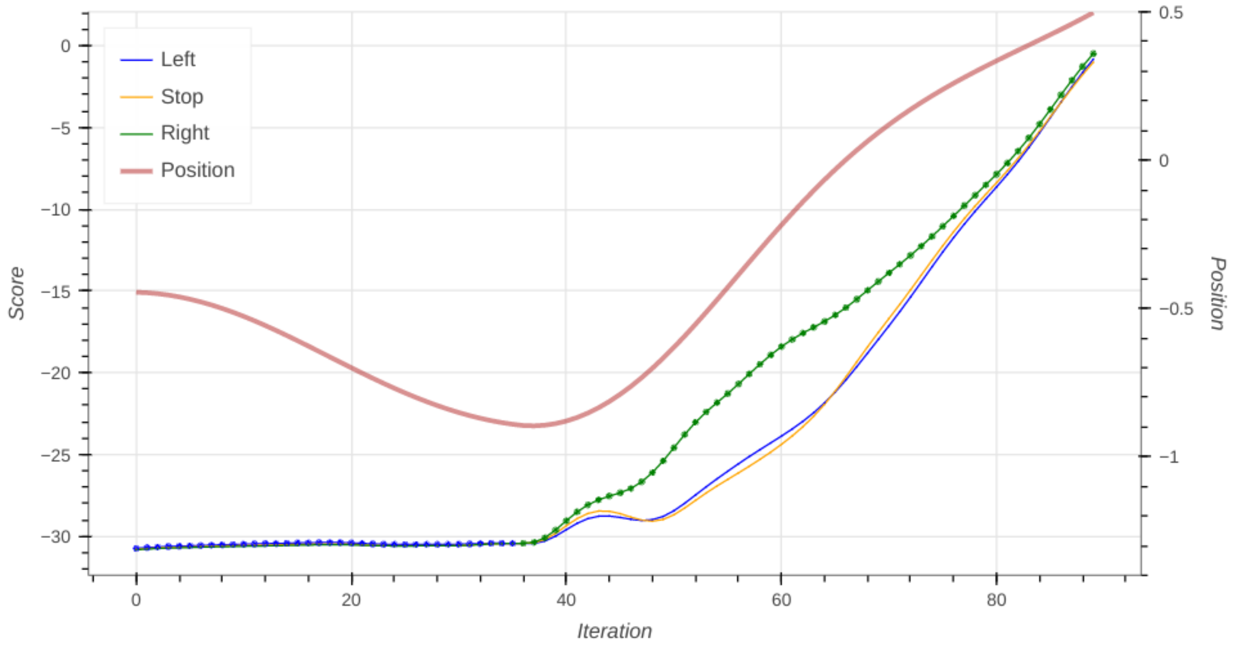


Figure 13: Comparison between the position of the car (red line) and the Q-values trends (gold, green and blue line). The colored dots represent the corresponding action chosen at that time-step.