Deep Learning Lab - Exercise 4

Baohe Zhang (4652986), Guilherme A. A. Miotto (4653503)

January 7, 2019

Abstract: In this exercise we used Deep Q-network (DQN) on three Open AI environments: cart-pole, mountain car and car racing. Moreover, we investigate the influence of different exploring methods (ϵ-greedy, ϵ-annealing and Boltzmann exploration) and also different ways to select the greedy actions (standard-Q and double-Q). The code was based on a stub provided by the tutors and implements the neural networks using Tensorflow.

Contents

1 Introduction

In this section we quickly review some of the concepts used in this report. The rushed reader may prefer to skip to section 2 and go straight to the results.

1.1 Q-Learning

A table containing all possible action-value function is arbitrarily populated, but successively updated using the following rule: every time a action is taken, a new estimate of the action-value function (δ) is calculated using the observed reward and an estimate of the sum of discounted rewards from the next state until the end of the episode. The table is then updated using a certain learning rate (α). More formally:

$$\delta = R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$$

$$Q(s_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\delta$$

1.2 Deep Q-Learning Network

For many problems, like those with a too large or continuous state space, it is unfeasible to use a table to keep track of the action-value function (q-value). In those situations it may be better to use q-value approximation. Deep Q-networks, or DQN, is an algorithm that uses neural networks as the approximator.

In our implementation we used the concept of replay buffer, i.e the network is trained on batches of stored state transitions that the agent has seen. This helps breaking the temporal correlation of subsequent state transitions while training the network. We also used distinct target ($Q_{\Phi'}$) and behaviour (Q_Φ) networks to avoid the problem of moving target yielding a more stable gradient descent.

$$y_i = r_j + \gamma \max_{a'_j} Q_{\Phi'}(s'_j, a'_j)$$

$$\Phi \leftarrow \Phi - \alpha \sum_j \frac{dQ_\Phi}{d\Phi}(s_j, a_j)(Q_\Phi(s_j, a_j) - y_i)$$

1.3 Double Q-Learning

The formulation shown above tends to overestimate the next q-value because it uses the target network for both picking the greedy action and calculating the correspondent q-value. This problem can be avoided if we instead use the behaviour network for picking the greedy action.

$$y_i = r_j + \gamma Q_{\Phi'}(s'_j, \arg\max_{a'_j} Q_{\Phi'}(s'_j, a'_j)) \qquad \text{Standard-Q}$$

$$y_i = r_j + \gamma Q_{\Phi'}(s'_j, \arg\max_{a'_j} Q_{\Phi}(s'_j, a'_j)) \qquad \text{Double-Q}$$

1.4 Exploration methods

The algorithms just presented will always enforce the agent to take the action with maximum action-value, i.e . to greedily maximize the expected return. However, in order to increase the chance of convergence to a global optimal solution, the agent has to be able to explore the action-state space during the training phase. Here we tested three exploration methods:

1. **ϵ-greedy**: Very straightfoward method. Take random actions with probability ϵ and greedy actions with probability $1 - \epsilon$

2. **ϵ-annealing**: Same logic of the the basic ϵ-greedy. The difference is that here ϵ decays as training progresses — the agent starts more exploratory and gradually becomes more and more explotatory.

3. **Boltzmann exploration**: In this method, the probability of selecting each action is proportional to the exponential of its estimated q-value divided by a constant τ. If $\tau \to 0$, then the greedy action selected. If $\tau \to \infty$, all actions are equally likely.

$$P_t(a) = \frac{exp(q_t(a)/\tau)}{\sum_i exp(q_t(i)/\tau)}$$

1.5 Environments

We applied these concepts to three Open AI Gym problems (Figure 1):

1. **Cart-pole**: A pole is passively attached to cart that can be moved left and right. The aim is to keep the pole in the upward position.

2. **Mountain car**: The objective is to drive a car, initially in a valley, up a mountain. Nevertheless the car's engine is no so powerful, which means that it has first to gain moment by swinging between the two mountains.

3. **Car racing**: A car in a racing track. The aim is to drive as fast as possible while avoiding leaving the track.

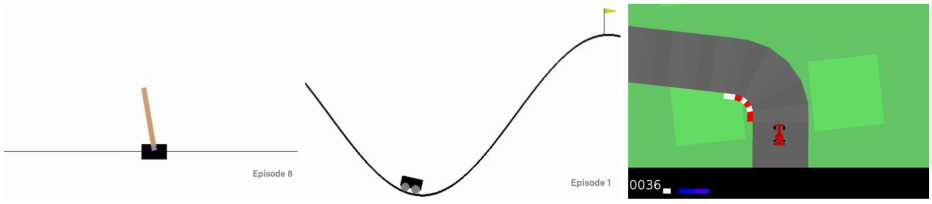


Figure 1: Open AI Gym environments. From left to right: cart-pole, mountain car, car racing

2 Methods and results

In this section we present the results of using DQN on all three environments. Moreover, we compare the effects of using standard-Q vs double-Q and different exploration methods. We first contrast the standard and double formulations using a fixed exploration method. After that, we do the opposited: experiment with different exploation methods using a fixed DQN formulation.

2.1 Comparing standard-Q vs. double-Q

In order to compare the influence of using the standard or the double formulation of DQN, we fixed the exploration method as ϵ-annealing with a decay of 0.99, spanning the interval of [0.30, 0.04]. Other fixed parameters were learning rate (3×10^{-4}) and discount factor (0.98). Adam was the choosen optimizer.

The same neural network was used for cart-pole and mountain car: a two-layer MPL with ReLU activations and 100 units per layer (as provided on the code stub). The network used for car racing is a bit more complex and is described on Table 1.

#	Type	Size	Stride	Activation
1	Conv2D	7x7x16 filter	1x1	ReLU
2	Maxpool	2x2 filter	2x2	—
3	Conv2D	5x5x32 filter	1x1	ReLU
4	Maxpool	2x2 filter	2x2	—
5	Conv2D	3x3x48 filter	1x1	ReLU
6	Maxpool	2x2 filter	2x2	—
7	Dense	512 units	—	ReLU
8	Dense	128 units	—	ReLU
9	Dense	5 units	—	Linear

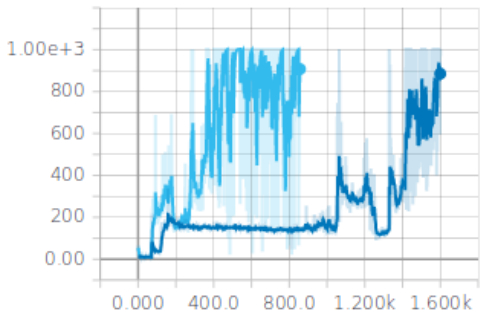
Table 1: Layers of the neural network used on car racing

The results are shown on Figures 2, 3 and 4 for cart-pole, mountain car and car racing respectively. On these figures the left sub-figure displays the rewards obtained during training (ϵ-greedy). The right sub-figure displays rewards obtained on validation episodes during training (greedy maximization of the action-value function). Each point in the validation plot represents an average over three episodes.

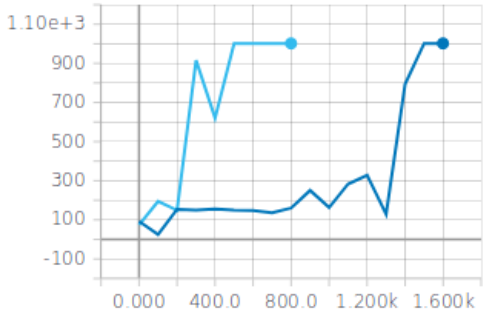
Notice that not all agents were trained for the same number of episodes. This is due to an early stop logic we implemented that would interrupt training once the validation performance consistently scored above a certain threshold. This proved to be very valuable, because in some cases over-training would degrade performance of the agent.

For car racing, whose training is way more expensive than the other environments, we ensured that the first training episodes were just 300 steps long. At later episodes, the number of steps is gradually increased until it reaches 1000. Validation episodes are always 1000 steps long.

On all three environments, both double-Q and standard-Q have yielded agents with roughly the same final reward performance. Neverthess, using double-Q resulted in faster convergence, this was specially noticeable on cart-pole (Figure 2) and car racing (Figure 4). On mountain car (Figure 3), double-Q made the agent be more stable, scoring high rewards more consistently than the standard-Q agent. To summarize, we observed that double-Q, although didn't result on better final agents, made the convergence substantially more stable and faster.

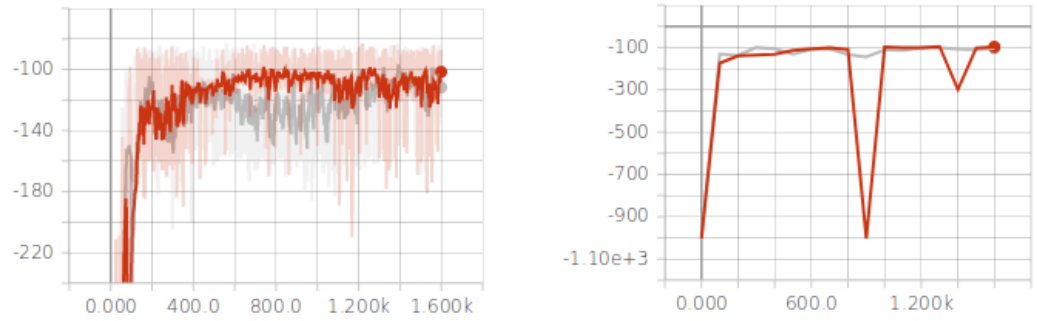


(a) Training (ϵ-greedy)



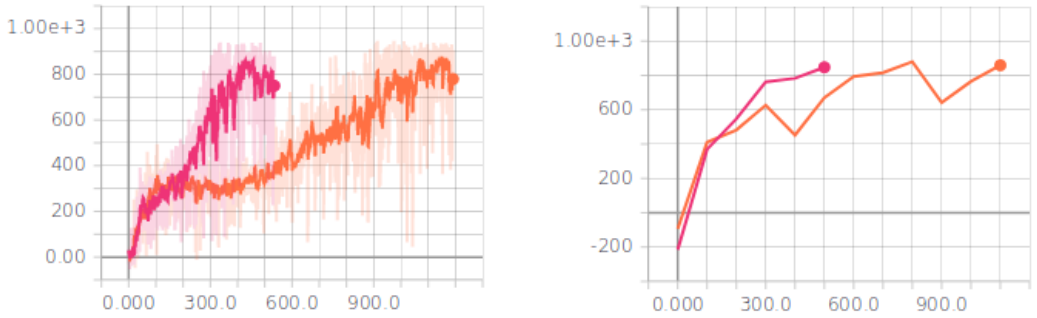
(b) Validation (greedy)

Figure 2: Cart-pole rewards during training with standard-Q (dark blue) double-Q (light blue).



(a) Training (ϵ-greedy) (b) Validation (greedy)

Figure 3: Mountain car rewards during training with standard-Q (brown) double-Q (gray).

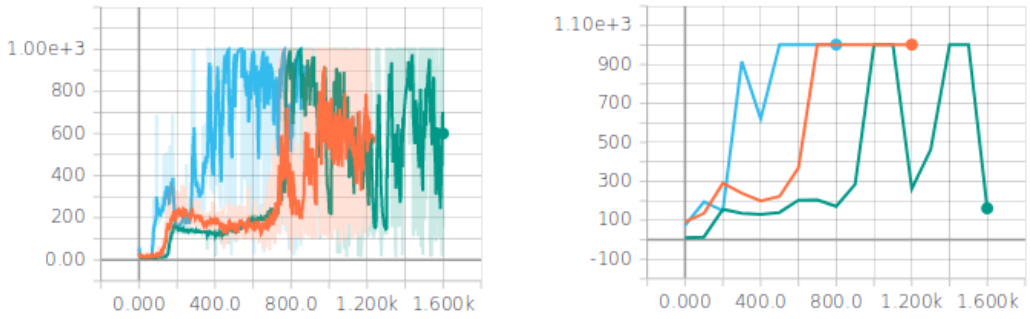


(a) Training (ϵ-greedy) (b) Validation (greedy)

Figure 4: Car racing rewards during training with standard-Q (orange) double-Q (pink).

2.2 Comparing different exploration methods

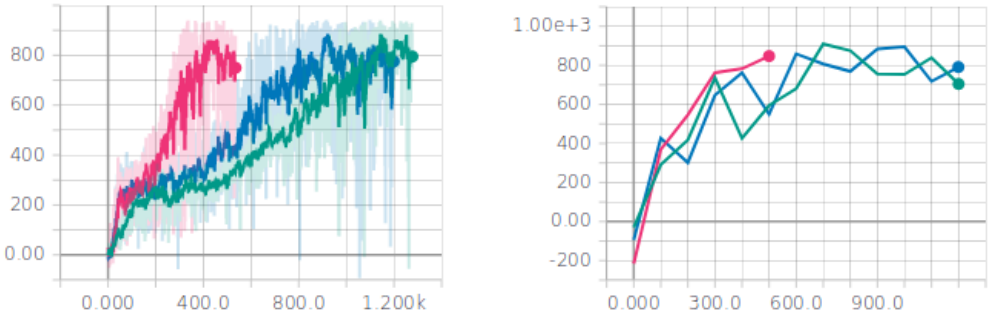
In order to compare the influence of the different exploration methods, we fixed the q-value evaluation to standard-Q and tested the three methods presented on Section 1.4 on cart-pole and car racing. The results are shown on Figure 5 and 6, respectvely. On both environments, ϵ-annealing gave the best results, i.e. the fastest convergence. Specifically on car racing, Boltzmann and ϵ-greedy (no decay) yielded similar results. On cart-pole, the second place was the Boltzmann exploration and coming as last is ϵ-greedy, which actually suffered some performance degradation towards the end of the training, probably a consequence of being too exploratory even at late episodes.



(a) Training (exploratory)

(b) Validation (greedy)

Figure 5: Car racing rewards during training with ϵ-greedy (green), ϵ-annealing (blue) and Boltzmann exploration (orange).



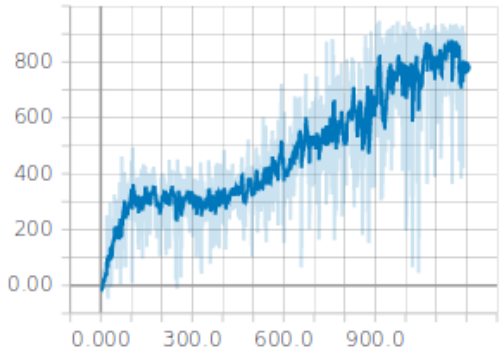
(a) Training (exploratory)

(b) Validation (greedy)

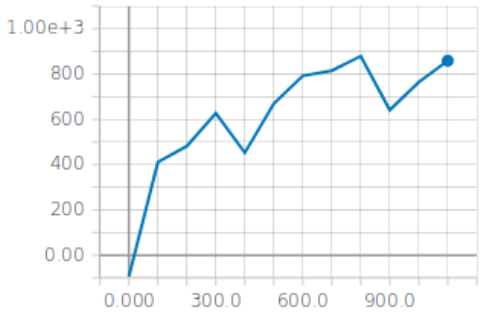
Figure 6: Car racing rewards during training with ϵ-greedy (blue), ϵ-annealing (pink) and Boltzmann exploration (green).

2.3 Our best car racing agent

Based on the results presented on sections 2.1 and 2.2, it seems that for the considered problems a combination of double-Q and ϵ-annealing gives the best results. We tested this combination on car racing and the results are presented on Figure 7. Indeed the outcome was pretty satisfactory: the agent improves consistently during training and reaches the best performance we could obtain. Over 15 episodes the **average final reward was 820.6** with standard deviation of 55.3.



(a) Training (exploratory)



(b) Validation (greedy)

Figure 7: Car racing rewards during training with double-Q and ϵ-annealing

3 Conclusions

In this assignment we tested two DQN formulations with three exploration methods on three environments. We have seen that a good choice of formulation and exploration method is fundamental for having more stable and faster convergence. More specifically, double-Q and ϵ-annealing seemed to yield the best results. Nevertheless, a definite confirmation of this finding would require the repetition of these experiments many times[1], given that DQN is a method that suffers strong influence of its initial values. All things considered, DQN showed very good results on the problems studied.

4 Acknowledgements

We are very thankful to Alex Rose and Ingmar Baetge for our extensive Whatsapp interactions on practical DQN tricks. Also special thanks to the tutors for the constructive feeedbacks.

[1]This demands an amount of computing power that the authors currently don't have at their disposal