Advanced Topics in Machine Learning
Assignment 3: Reinforcement Learning

Sofiane Mahiou
ucabsm1

April 11, 2017

I - Problem A

Question 1

As required, you can find **the cumulative reward** as well as **the episode length** for three trajectories generated under a random policy bellow :

Table 1: Three example of trajectories under a random policy

	Reward	episode length
trajectory 1	-0.8775	13
trajectory 2	-0.8345	18
trajectory 3	-0.7778	25

Question 2

The exercice done above have been repeated over 100 episodes, below is the mean and variance of **the cumulative reward** as well as **the episode length** :

Table 2: mean, standard deviation

	Reward	episode length
mean	-0.7957	23.63
std	0,098	13.97
variance	0.0098	195.23

Question 3

Before training each model implemented for the various learning rates required, we first started by running **2000 episodes** under a random policy, recording every transition : [**state, action, reward, nextState**]. Once this was done, two type of models were trained, one with a Single layer **(2 neurones)** and a second with a hidden non-linear layer **100 neurones** followed by a linear layer.

After experimentation, the use of biases was avoided as it did not help in any of the cases attempted.
Bellow you can find the results of each model trained for the following learning rates : [10^{-5}**,**10^{-4}**,**10^{-3}**,**10^{-2}**,**10^{-1}**,**0.5]

Part 1 : Single Layer

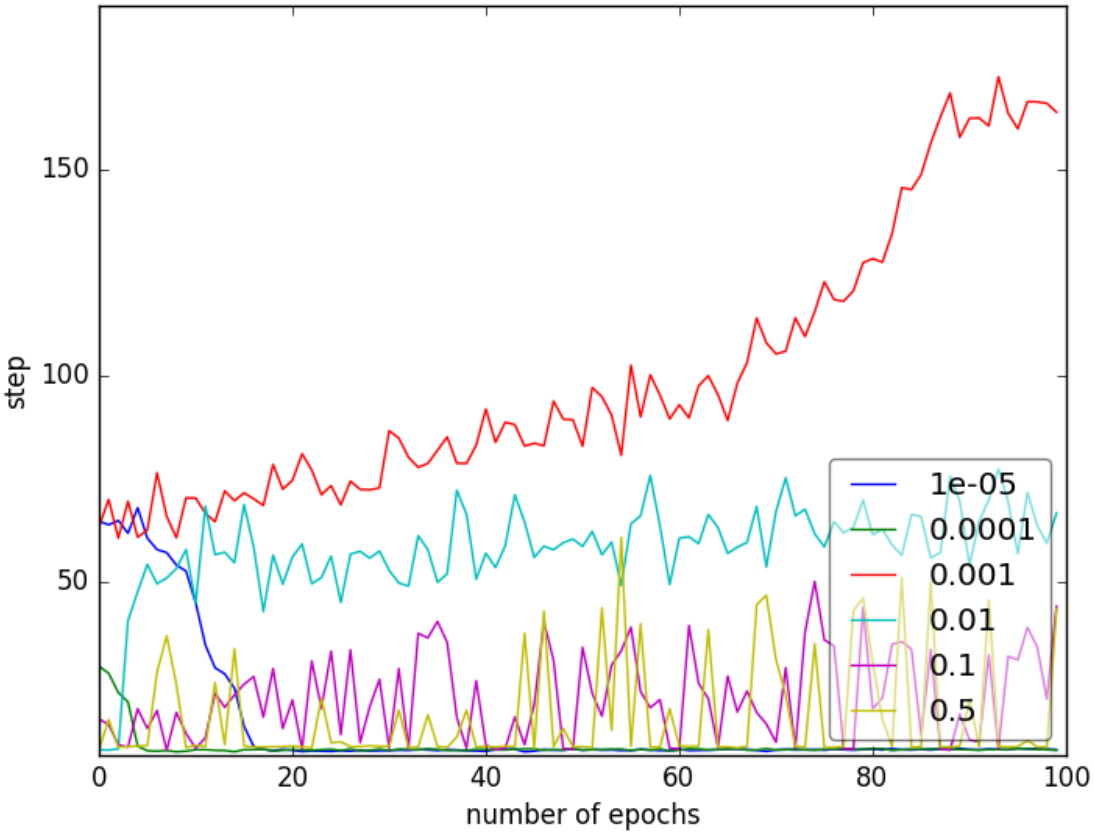

Figure 1: test performance - average number of steps over 100 episodes

As it can be seen on the plot above, the model seems to learn best for learning rates of **0.001** or **0.01**. However, over 100 epochs, the systems doesn't converge yet which suggests it would require a longer training time.

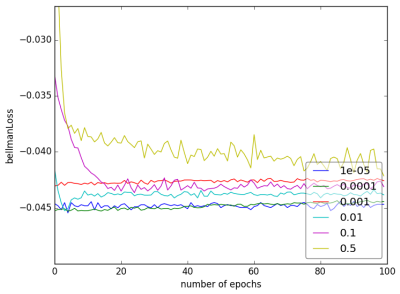

Figure 2: bellman residual

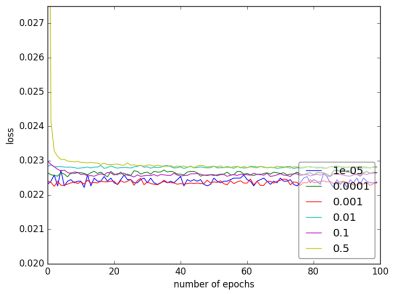

Figure 3: Training Loss

Part 2 : One hidden Layer

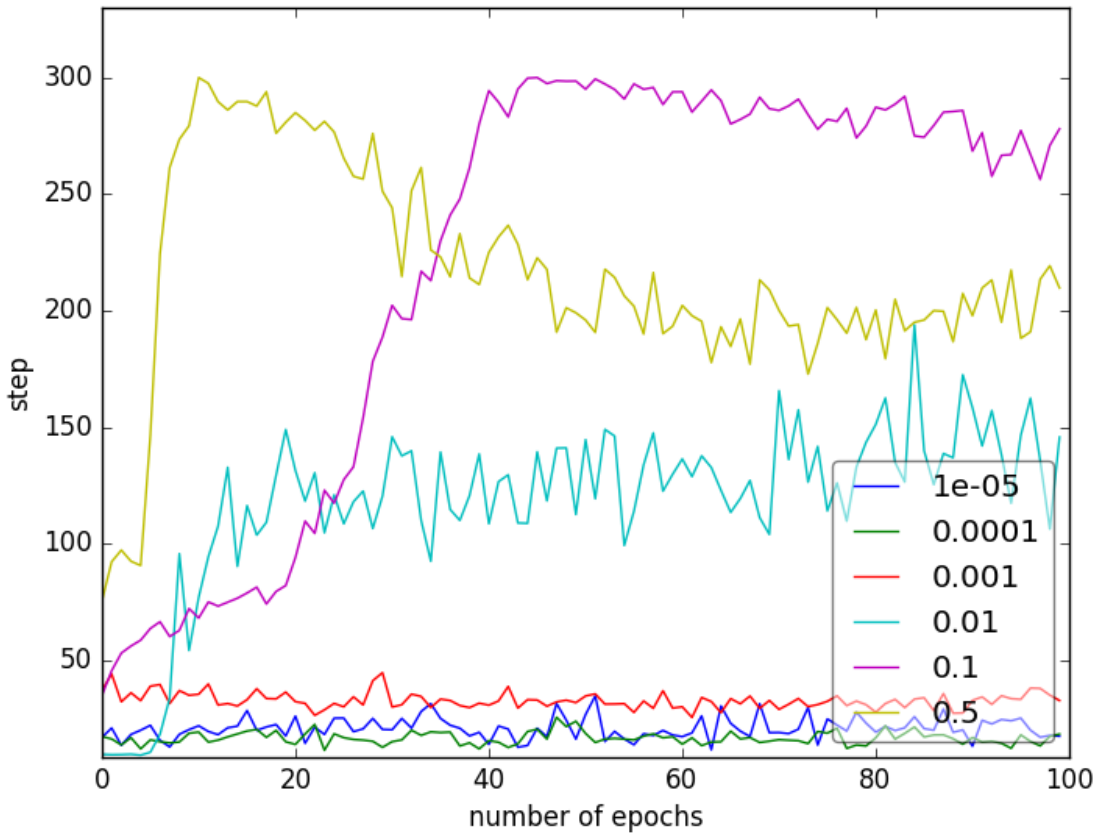

Figure 4: test performance - average number of steps over 100 episodes

The second model seems to behave far better as it reaches the **300 steps** threshold set up for some of the attempted learning rates. In this case, the higher learning rates **(0.5,0.1,0.01)**. However, it seems that it quickly over fits for a learning rate of **0.5** and doesn't learn fast enough **0.01**. A learning rate of **0.1** then seems to be the optimal value.

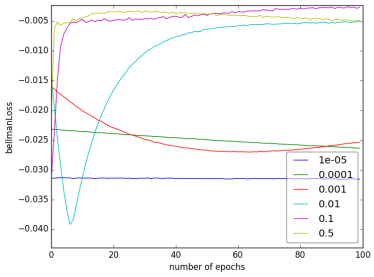

Figure 5: bellman residual

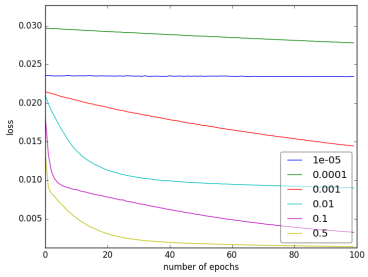

Figure 6: Training Loss

Question 4

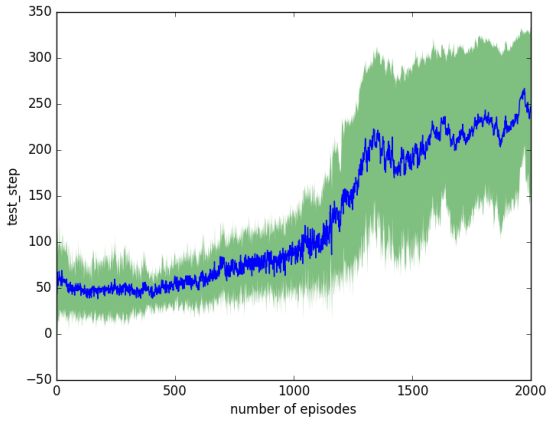

Figure 7: test performance

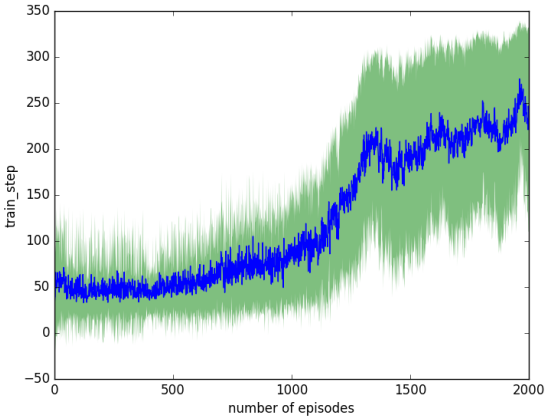

Figure 8: train performance

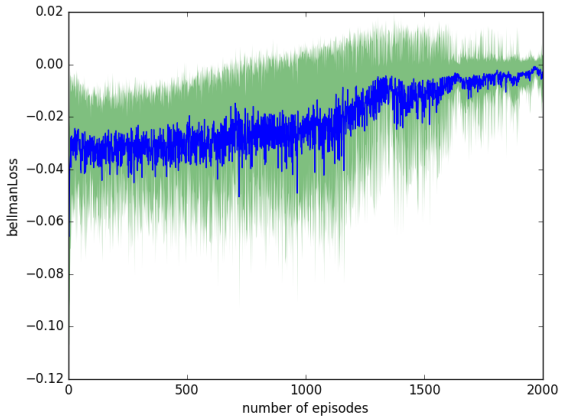

Figure 9: bellman residual

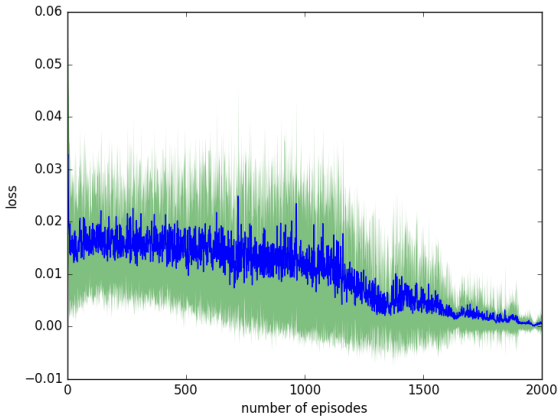

Figure 10: Training Loss

the plots displayed above are the results a Q-learning agent following an ϵ**-greedy** policy during training with $\epsilon = 0.05$ trained at every transition over 2000 episodes The learning Rate was chosen emperically to be **0.01** . This process was iterated 100 times and averaged, these plots display both the mean and the standard deviation of the following metrics : [**bellman residual, training loss, train episode lenght, test episodes length averaged over 10 episodes**]. As it can be noticed, the model does seem to learn properly as the average number of steps increases and the losses decreases, however, this process seems to be quite noise as the **standard deviation** remains high.

Question 5

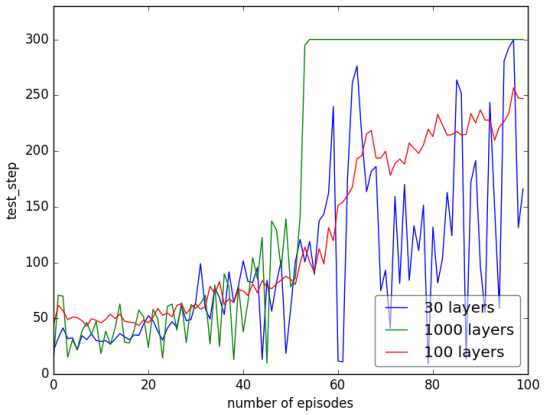

Figure 11: test performance

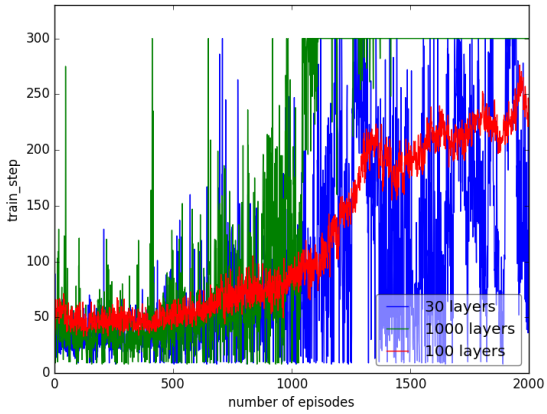

Figure 12: train performance

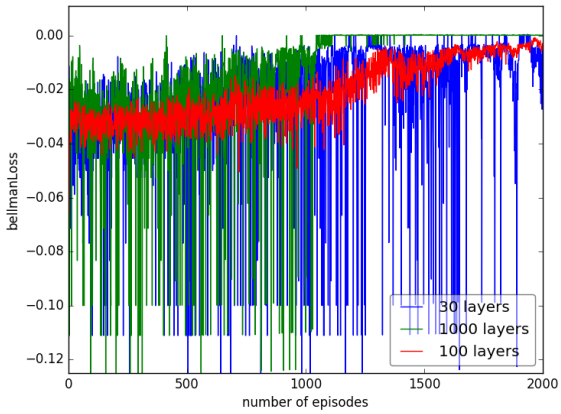

Figure 13: bellman residual

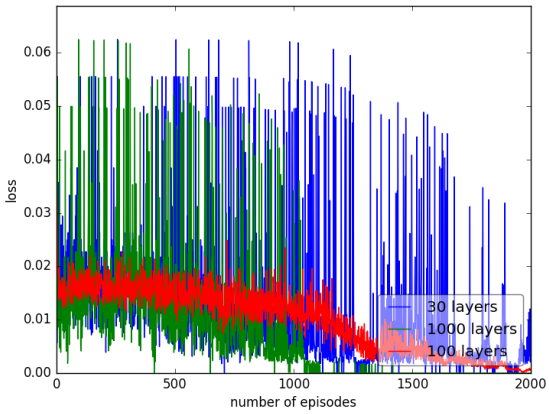

Figure 14: Training Loss

We then tried to assess the importance of the hidden layer size, by comparing the previous results to a the same network with first a **hidden layer size of 30** and then a **hidden layer size of 1000**. In order to be able to fairly compare the models, we keep the learning rate fixed to **0.01**. As expected, increasing the hidden size seems to improve the learning process, as the model with 30 neurones displays worse performance than the initial one and the 1000 neurones model converges faster *within 1200 episodes* and remains reliably at the maximal performance possible .

Question 6

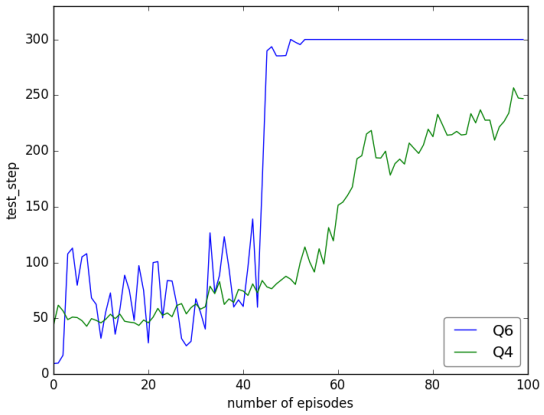

Figure 15: test performance

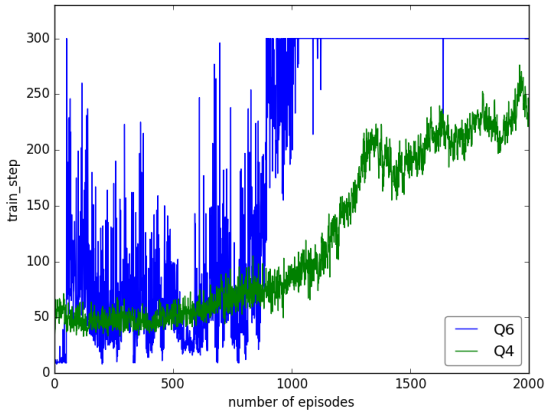

Figure 16: train performance

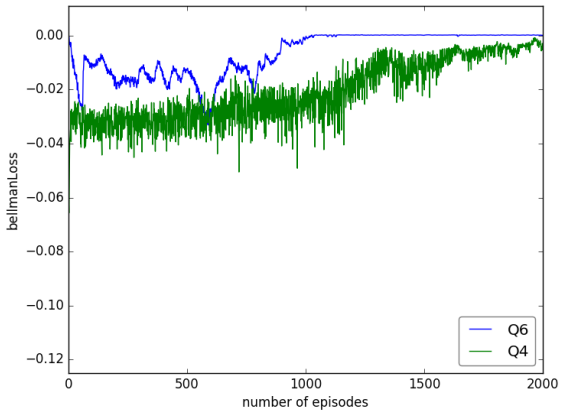

Figure 17: bellman residual

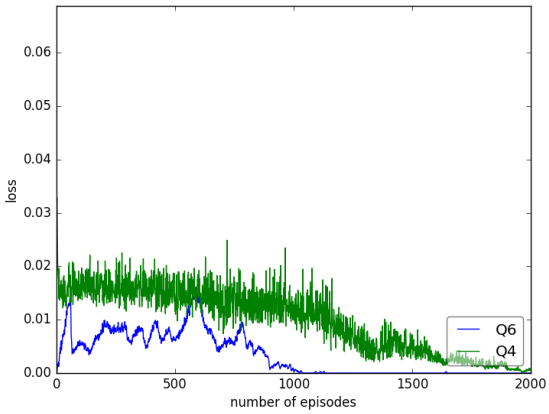

Figure 18: Training Loss

We then implemented a replay buffer, in order to try to improve the results obtained above. After a few attempts, The **buffer size was set to 2000** and the **batch size to 500** and the **learning rate to 0.015**. As it can be seen above, this model was more performant than any of the three models attempted in question 4 and 5. It converged within 1000 episodes and remained consistent. As the loss can prove it, this model also seems to be less noisy than the one implemented before.

Question 7

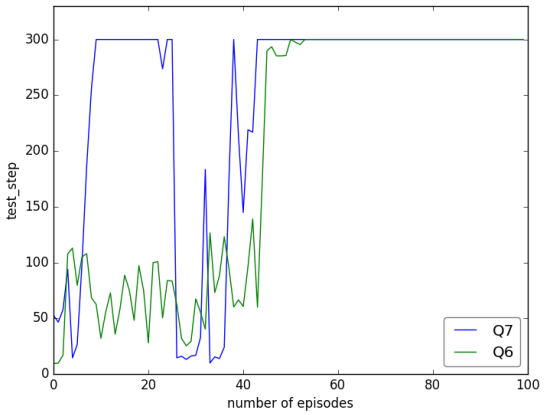

Figure 19: test performance

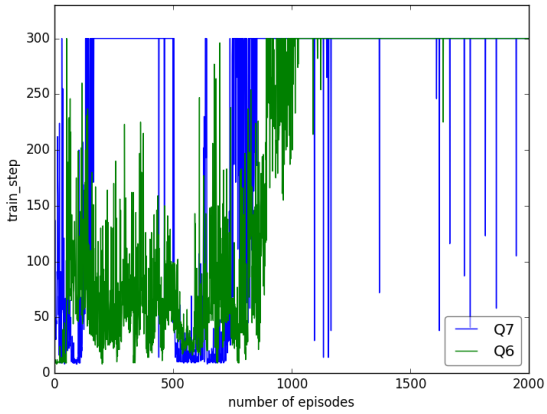

Figure 20: train performance

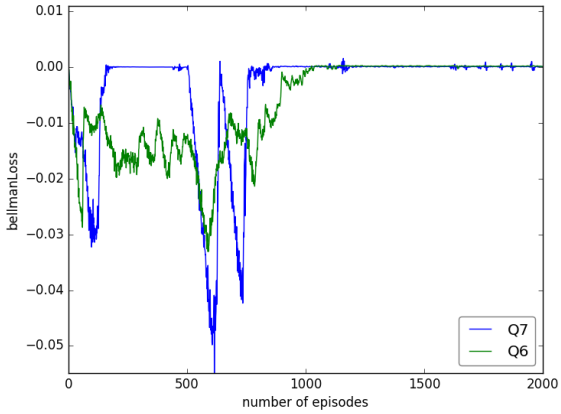

Figure 21: bellman residual

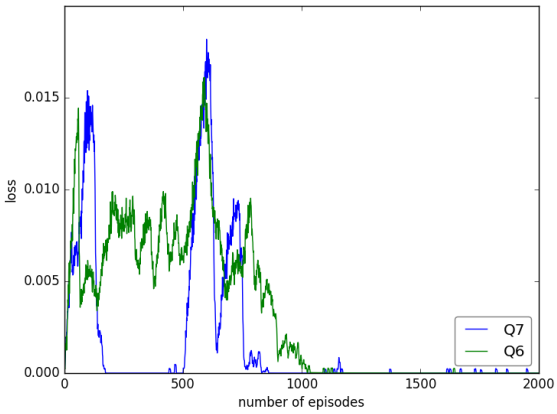

Figure 22: Training Loss

In order to continue improving the model, we modified the neural network implementation to add a target network that was updated every **5 episodes** (this was done by using the *tensorflow assign operation*) and combined with the buffer replay implemented before. This model was trained using the parameters presented in question 6, the sole difference being the **the learning rate that was set to 0.00001**. The results for this model were plotted against the previous model *(question 6)* in order to be able to clearly see the improvements. Indeed, this model seems to be even better than what was implemented above as it **converges in less than 800 episodes**.

Question 8

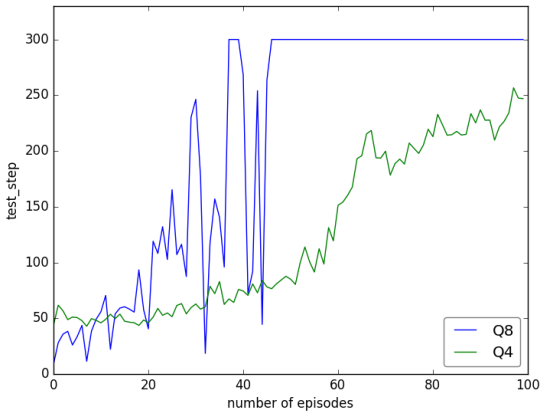

Figure 23: test performance

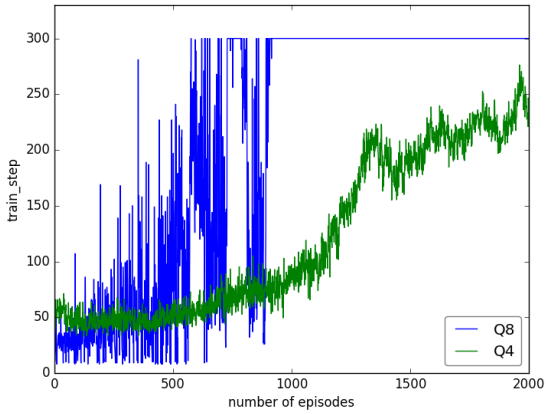

Figure 24: train performance

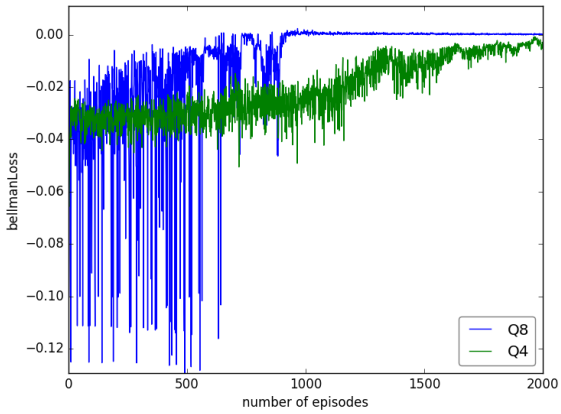

Figure 25: bellman residual

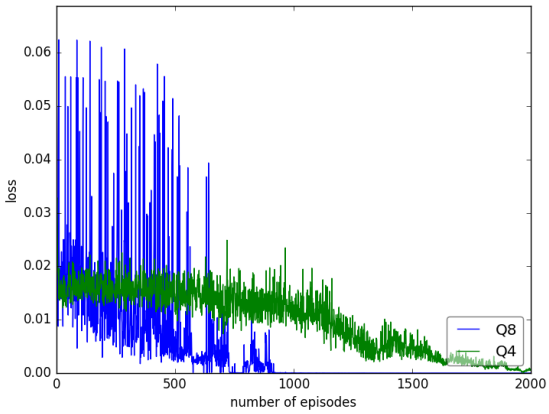

Figure 26: Training Loss

Finally, we implemented a Double-Q learning Agent in order to compare its performance with a simple Q-learning agent. In order to be able to properly compare this model with the model implemented in question 4, it was not combined with a replay buffer. The learning rate chosen for this model was **0.035**. The performance reached by this algorithm is comparable to the ones reached by the model combining a target network and a replay buffer. It converges to the maximum threshold in **less than 1000 episodes**.

II - Problem B

Question 1

bellow are the mean and standard deviation for both the cumulative discounted score as well as the frame count under a **random policy** for each game over 100 episodes .

	Ms Pacman		Pong		Boxer	
	Score	Frame count	Score	Frame count	Score	Frame count
mean	2.56	670.3	-0.84	1019.43	-0.24	2378.14
standard deviation	0.74	105.5	0.36	9.377	1.13	12.73

Table 3: mean and standard deviation for score and frame count averaged over 100 episodes for all games obtained following a random policy

Question 2

bellow are the mean and standard deviation for both the cumulative discounted score as well as the frame count under a **greedy policy using an untrained model** for each game over 100 episodes .

Table 4: mean and standard deviation for score and frame count averaged over 100 episodes for all games obtained following an untrained neural network

	Ms Pacman		Pong		Boxer	
	Score	Frame count	Score	Frame count	Score	Frame count
mean	2.50	627.73	-1.14	1019.36	-0.38	2379.62
standard deviation	0.07	6.34	0.03	9.68	0.075	13.55

Two main differences exists between the results obtained here and the results obtained above:

- **the standard deviation** is consistently smaller in the second case
- **the score**: is consistently worse for the untrained model than for a random policy

This is due to the fact that while **a random policy will always randomly pick an action to take**, an **untrained model will nearly always take the same action** as it is a deterministic process. Indeed, when the convultion layers are not trained, the untrained model is metaphorically *blind* as it cannot identify any interesting feature, it is then quite likely always take the same action which would mean obtaining the same (likely bad) outcome everytime hence the smaller standard deviation and cumulative score.

Question 3

After initially training the model under the parameters specified in the assignment, we tried lowering the learning rate to 0.0001 as the system was not performing well for the initial parameters. Furthermore, in order to improve the performance, we also tried increasing the preprocessed image size to **60x60**. Bellow are presented the results for all attempts made.

Boxing

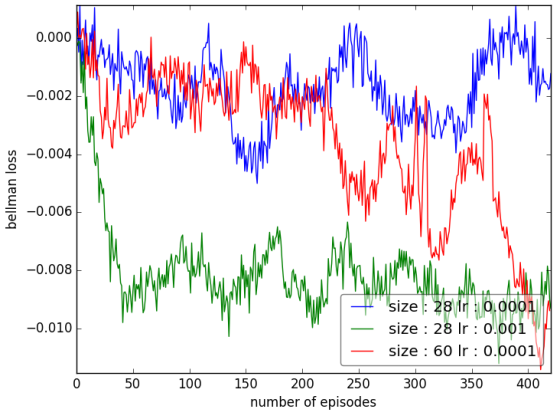

Figure 27: bellman residual

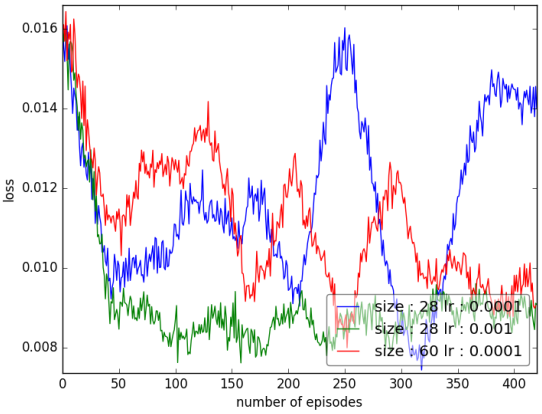

Figure 28: Training Loss

Ms Pacman

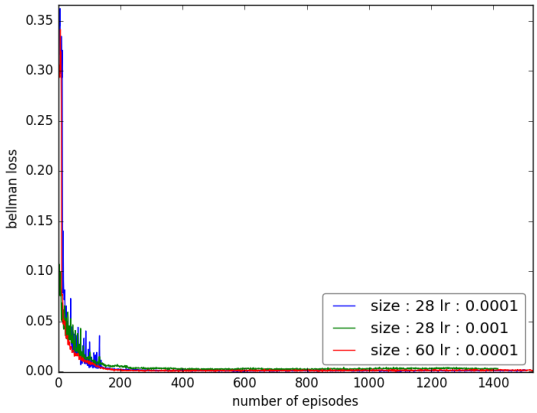

Figure 29: bellman residual

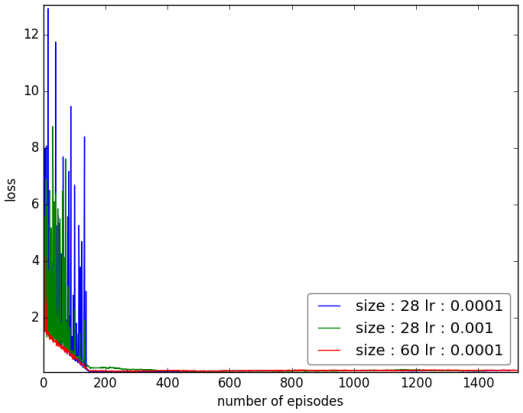

Figure 30: Training Loss

Pong

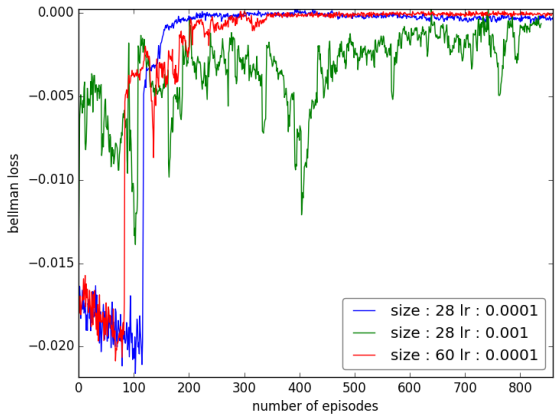

Figure 31: bellman residual

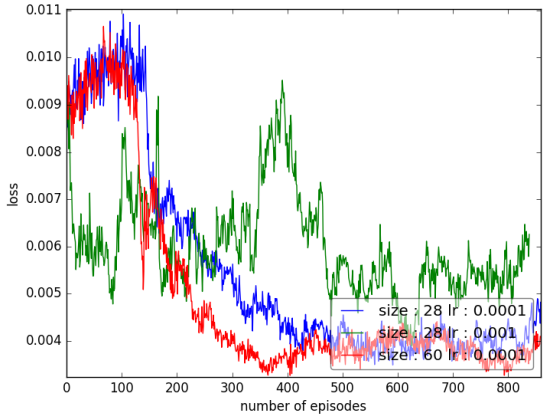

Figure 32: Training Loss

Overall, it seems that the **loss decreases** and **the bellman loss** gets closer to zero for a smaller learning rate of 0.0001 with the exception of **boxer** game where the loss decreases for a higher learning rate of 0.001. However, these plots differ from a supervised learning case because, as opposed to the supervised learning where the loss would correlated with whether or not the agent took the right action, in this case the loss can decrease even though the system is not learning the targeted behaviour. as it aims to accurately predict its cumulative reward and not maximize it.

Question 4

Boxing

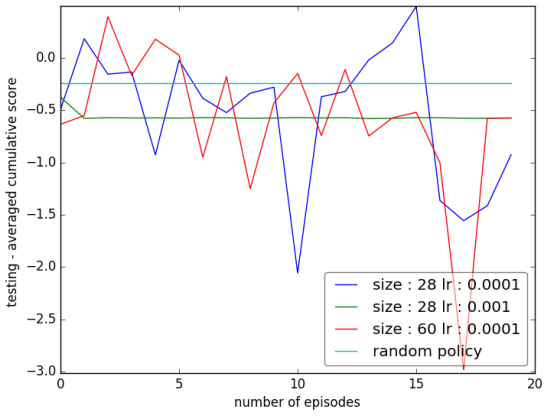

Figure 33: test performance

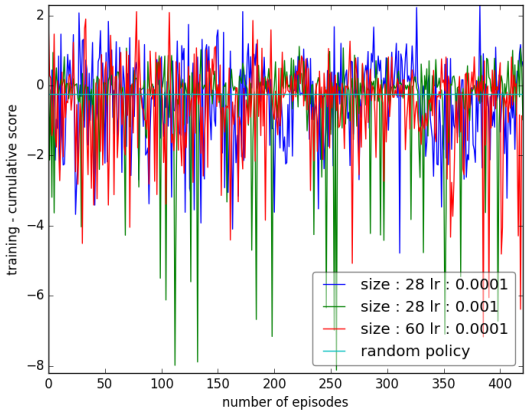

Figure 34: train performance

Ms Pacman

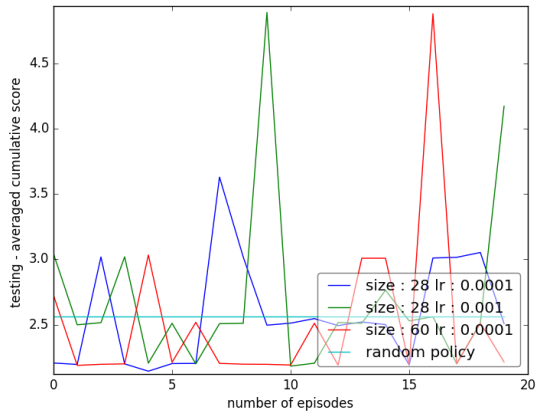

Figure 35: test performance

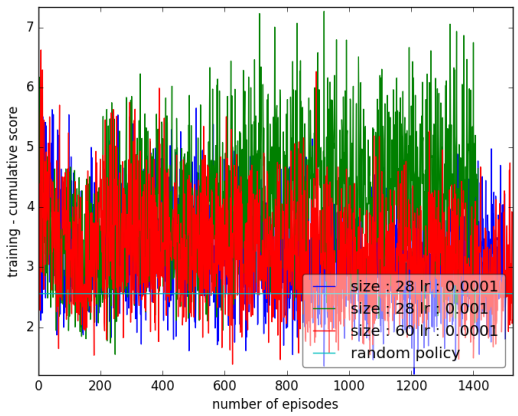

Figure 36: train performance

Pong

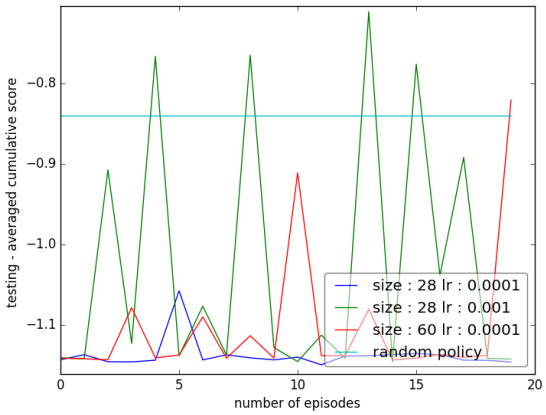

Figure 37: test performance

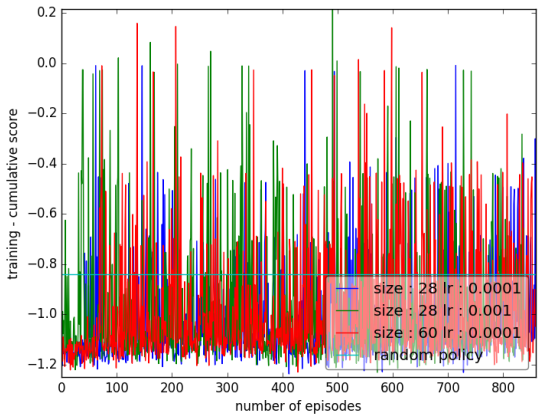

Figure 38: train performance

here are the final cumulative discounted scores (tested on the best model saved) obtained for each attempt:

settings	Ms Pacman	Pong	Boxer
size : 28, lr : 0.0001	3.61	-1.08	**0.66**
size : 28, lr : 0.001	4.87	**-0.81**	-0.37
size : 60, lr : 0.0001	**4.944**	-1.14	0.51
random policy	*2.56*	*-0.84*	*-0.24*

Table 5: final cumulative scores averaged over 100 test epsiodes and compared with the results for a random policy

Overall, it seems that increasing the preprocessed image size does seem to increase performance. As it can be seen above, the optimal settings for each games are the following :

- **Ms PacMan** : size 60 and learning rate : 0.0001
- **Pong** : size 28 and learning rate : 0.001
- **Boxer** : size 28 and learning rate : 0.0001

Overall, it seems that **increasing the preprocessing image size** helps with the performance, as both MsPacman and Boxer have a performance that is better than a random policy for a size of 60. However, more parameter tuning is required in order to get a proper learning curve where the cumulative reward is consistently increase and converge to a high value. **cropping useless information from the images such as the score or the number of lives** as well as trying a bigger range of learning rates might be good methods to improve the results.