Deep Learning Lab

Assignment 4

Università della Svizzera Italiana

Caselli Alex

December 2019

Indice

1 Report

1.1 Wrappers

Wrappers were very useful to interact with the Gym environment without having to pre-process the data every time, in particular i used:

1. **FrameStack** This wrapper is used to stack k frames retrieved from the environment and wrap them into a LazyFrame object, LazyFrames are much more memory efficient than normal 3-D numpy array. In fact, a very big Replay Buffer could be problematic without them, but they have to be converted to numpy arrays to be used by the network. In our case k is equal to 4.

2. **ScaledFloatFrame** This wrapper is used to normalize observations, observations are scaled from values between 0 and 255 down to values in [0, 1]. This value range is more comfortable to the network.

3. **MaxAndSkipEnv** This wrapper allow us to skip n frames (n=4) and repeat the same action for each of them. The 4 observations are buffered and then the max is taken over them and returned as a single observation.

4. **ClipRewardEnv** Probably the most easy wrapper, the scope of this wrapper is just clip the rewards value between -1 and 1, this clipping is useful during training.

1.2 Target Network

The target Q-networks has been introduced by the author to **generate the targets for** the training phase of **the online Q-network**, this usage of two networks should make the **algorithm more stable** avoiding the online network going crazy with huge oscillations by **creating a delay** between an update of the Q is made and when this change will influence also the targets. This could happen because when we change $Q(s_t, a)$ it could change also $Q(s_{t+1}, a)$.

1.3 ϵ-greedy policy

We use an ϵ-greedy policy to be able to **escape local optima** and **find new actions** that can lead the agent **towards better rewards** and score, we decrease the probability of taking a random action over time because the network should become more and more good in choosing the action to take estimating the values of Q and there should not be many more new actions to discover.

1.4 Training

Using these hyper-parameters:

- Learning Rate: $1e-4 = 10^{-4}$
- Batch size: 32

- Buffer Min and Max size: 10.000
- Number of steps: 2.000.000
- Update target Q-network every: 10.000 steps
- Train online Q-network every: 4 steps
- $\epsilon \in [0.1, 1.0]$ Starting at 1.0 and decreasing linearly for 1.000.000 steps

(A) **Return during training**

The rewards obtained from each episode of the training has been averaged over the last 30 episodes to smooth the values:

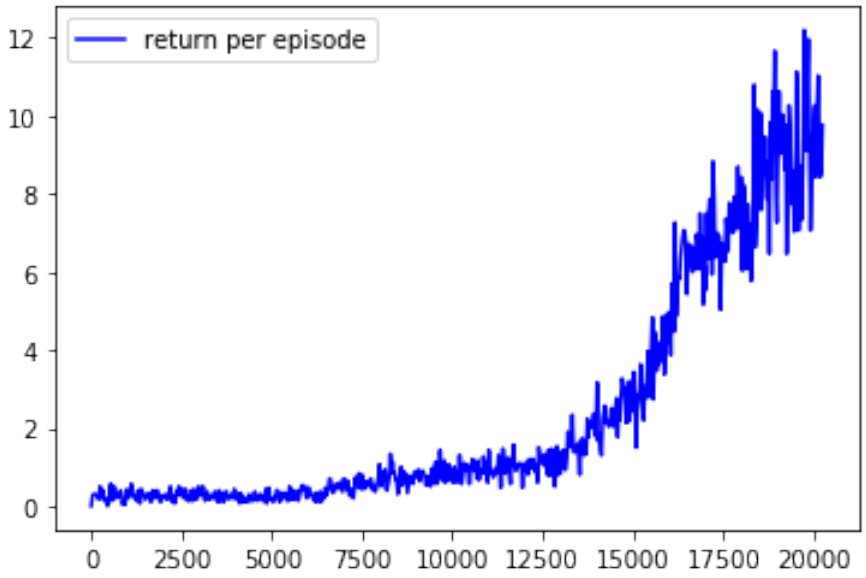

Figura 1: Return per episode averaged over the last 30 episodes

As we can see, even averaging over 30 episodes the return per episode is very noisy, but the trend is very clear anyway with a **return that increase gradually for the first 12.500 episodes and then increase its growing speed**. We can also notice as the **variance of the returns is very little at the beginning** and then becomes **very high at the end of the training**, this could be **caused by the very high rewards obtainable at the end of the training** when the agent is often able to take many steps and obtain big sequences of rewards.

(B) **Return during evaluation**

Every 100.000 steps we compute the evaluation of the Q-network, in this phase we evaluate an ϵ-greedy policy with ϵ= 0.001 and the rewards are not clipped. To evaluate the model we sum up the returns obtained in 5 episodes (called a Play) obtaining a Score; we make 20 Plays and we average their Scores to obtain the Score of the Evaluation:

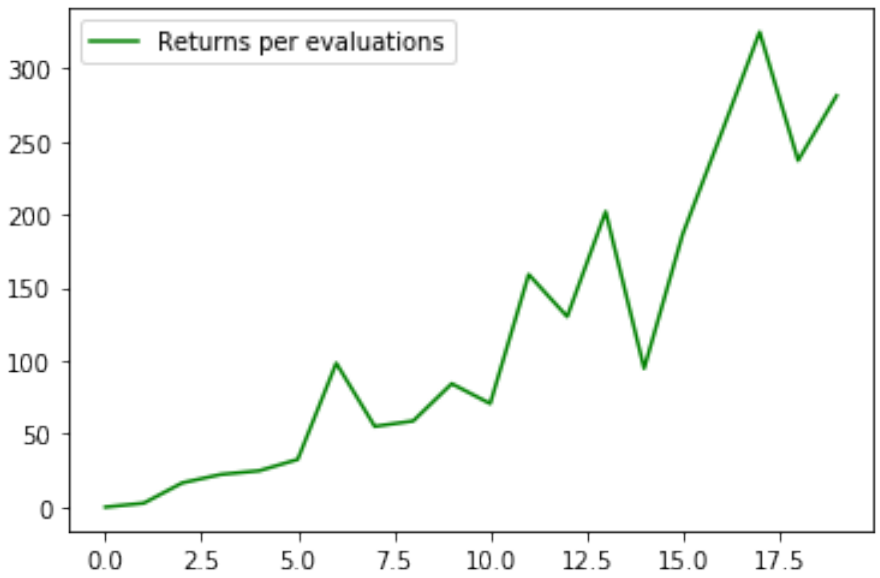

Figura 2: Return during evaluation of the 20 evaluations executed during the training

As we can see also the returns from the evaluations **did not grow linearly**, often the return at time t is worst than the return at time t-3, but globally the return grown up over time reaching it **maximum** at the 18Th evaluation where the agent get a score equal to **324.7**, then ending later with a **final return value of 281.2**

Evaluation n.	**Return Score**
20	281.2
19	236.9
18	324.7
17	255.0
16	186.4
15	94.7
14	201.9
13	130.1
12	159
...	...
2	2.7
1	0.0

(C) **Temporal-difference error** $L(\theta)$

Every 4 steps we take the update step in which we train our Online Q-network and compute the temporal difference loss using the target Q-network to create targets (if the transition is not terminal).

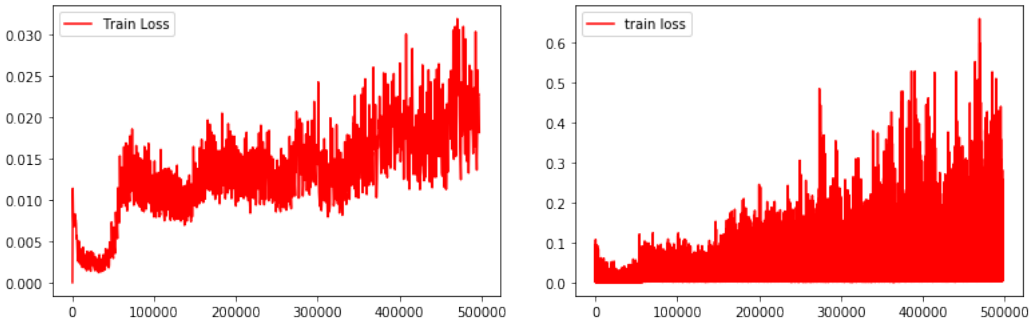

Figura 3: Training loss averaged over last 500 update steps (moving average) to improve readability and without averaging

The Temporal-difference error seems to decrease at the beginning but then start growing at each step and become also more and more noisy, this behavior is quite counterintuitive because we would expect that the loss decrease while the return increase but in this case they both grow over time, this **because the target Qs are just an approximation** of the real ones made by an old version of the online network itself, **the loss would be become more stable** and decrease **once the online network** will be able to play the game good enough to **obtain good results more and more frequently**. **Without the target Q-network the loss would have been probably even more unstable.**

1.5 Monitor

Having saved the model i can test it and see how good it is in playing using the Monitor Wrapper, you can find the **video of the episode in this directory**.

Figura 4: A frame of episode I capture using the monitor wrapper

1.6 Change target Q-net update frequency

What if Instead of updating the the target Q-network every 10.000 steps we do it every 50.000?

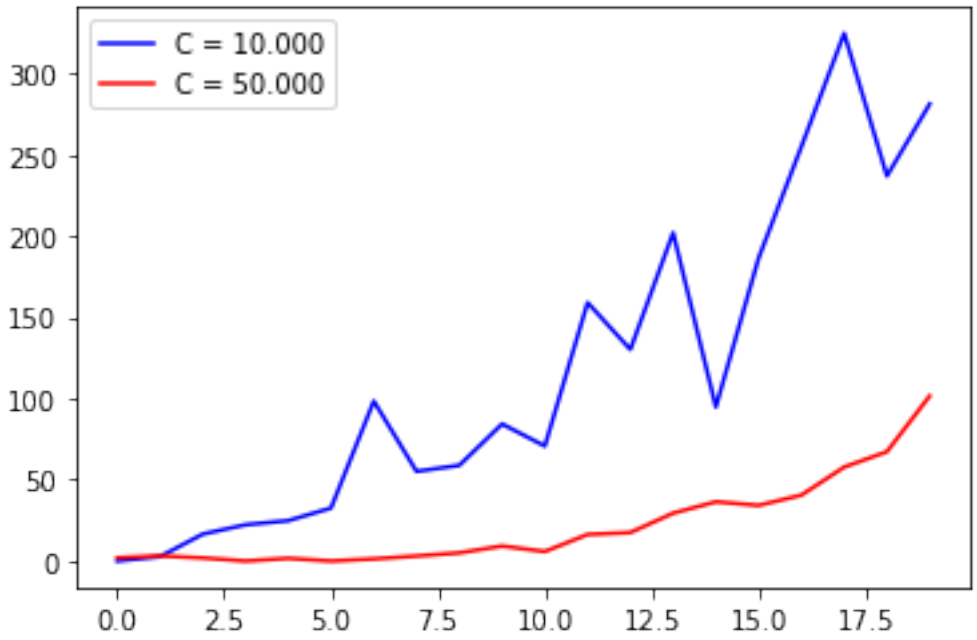

Figura 5: Comparison of the Return per evaluations

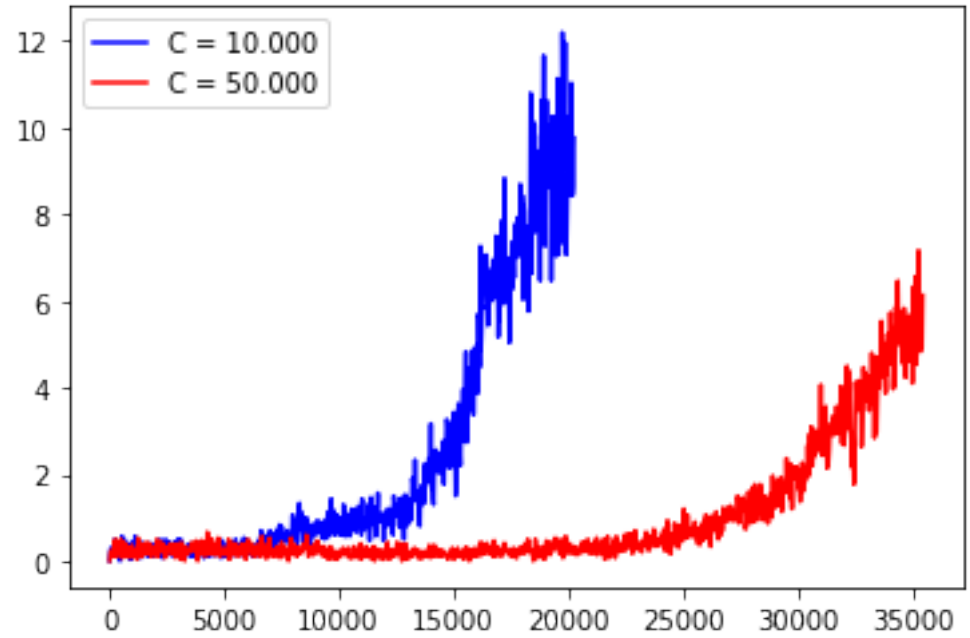

Figura 6: Comparison of the Return per episodes (moving average)

From the graphs we can notice that:

- The **trend of the Return per evaluation is more linear and less noisy** in the model with C = 50.000, **but** it also **perform much worse in terms or return** with a maximum score equal to **101 vs** the **324** points of the base model.

- Also **looking at the return per episodes** we can see that the **number of episodes** during the 2.000.000 steps **is almost double** and the return value is smaller, this is because **this model was not good enough to create episodes of a large number of steps** before loosing, while in the basic model the **mean length of an episode was 100** steps while i**n this model it was around 54** steps.

This difference in terms of performance it's probably caused by the **bigger delay created between the two networks**, 50.000 steps is a very big number and if **the online network has to learn from targets generated from a quite different version of itself that performs quite worse** than itself (we can suppose that after 50.000 steps the online network would become better than the target one) i**t cannot learn enough because of the poor quality targets**. Also the **target network should stabilize the online network**, but with this big number of steps it **is not able to do it in the optimal way**.
Training this model **required almost half the time** respect to training the base model, this **because the time require for the evaluation steps** was smaller (shorter episodes require less steps and time) and also because the **update operation of the target network** was **5 times less frequent**.

1.7 Bonus - Assault game

This time i decided to train the model on the game Assault, expecially i used the environment "AssaultNoFrameskip-v4". This game is quite similar to Space Invaders but it seems also a little more complex because of the enemies. As for BreakOut i trained for 2.000.000 steps obtaining these results:

(A) **Return during training**

The rewards obtained from each episode of the training has been averaged over the last 30 episodes to smooth the values:

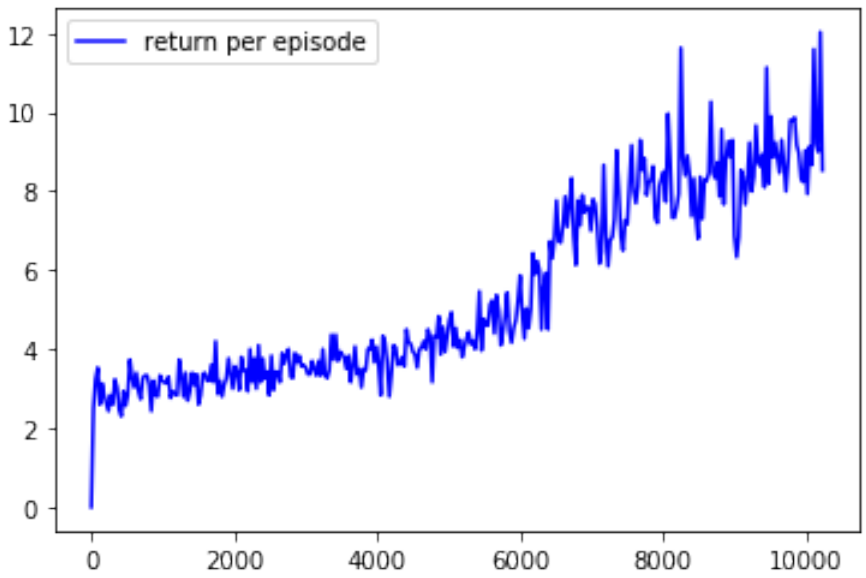

Figura 7: Return per episode averaged over the last 30 episodes

In this game **we got half the number of episodes** for the same number of steps, this mean that the game is slower and **any episode requires almost two times** the number of steps than in BreakOut.

(B) **Return during evaluation**

Every 100.000 steps we compute the evaluation of the Q-network, in this phase we evaluate an ϵ-greedy policy with ϵ= 0.001 and the rewards are not clipped. To evaluate the model we sum up the returns obtained in 5 episodes (called a Play) obtaining a Score; we make 20 Plays and we average their Scores to obtain the Score of the Evaluation:

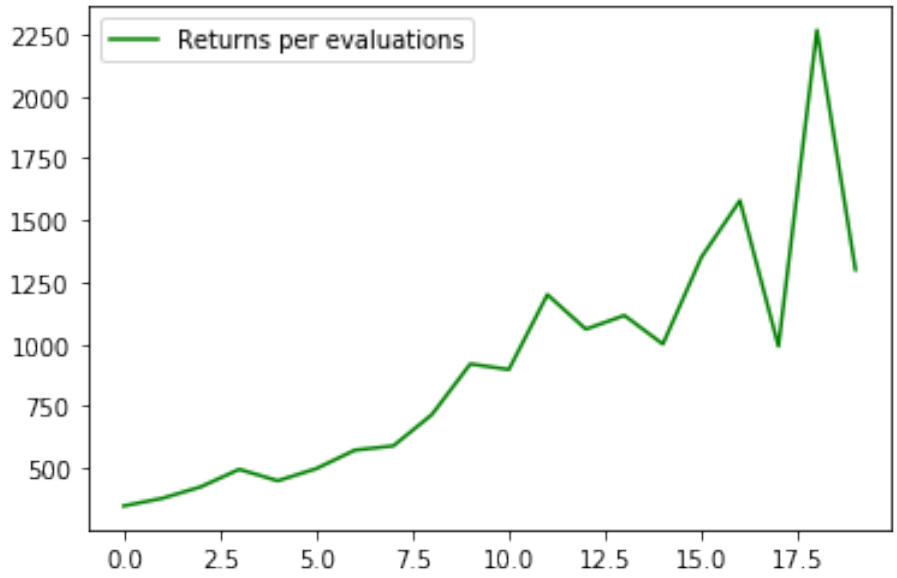

Figura 8: Return during evaluation of the 20 evaluations executed during the training

In this game the **rewards are quite higher** than in the previous one but in both the games **the model seems to learn in the same way**, the two **plots are very similar**.

(C) **Temporal-difference error** $L(\theta)$

Every 4 steps we take the update step in which we train our Online Q-network and compute the temporal difference loss using the target Q-network to create targets (if the transition is not terminal).

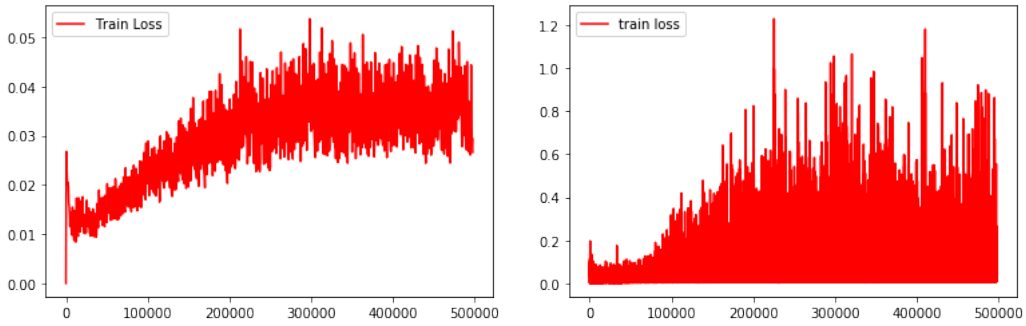

Figura 9: Training loss averaged over last 500 update steps (moving average) to improve readability and without averaging

Also the loss seems to increase in the same way, **this learning algorithm is robust enough to be used with different games** obtaining quite similar performances.

Monitor - Gameplay You can find the **video of the episode in this directory** under the name **"Bonus.mpg4"**.

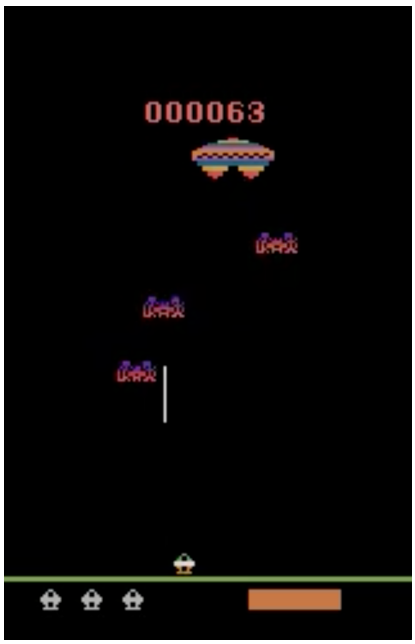

Figura 10: A frame of Assault I capture using the monitor wrapper