10-703 - Homework 2: Playing Atari With Deep Reinforcement Learning

Rogerio Bonatti
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
rbonatti@andrew.cmu.edu

Ratnesh Madaan
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
ratneshm@andrew.cmu.edu

Abstract

In this assignment we implemented Q-learning using deep learning function approximators for the Space Invaders game in the OpenAI Gym environment. We implemented the following variations of Q-learning: linear network without and with experience replay and target fixing, linear double Q-network with experience replay and target fixing, and dueling deep Q-learning.

1 [5pts] Show that update 1 and update 2 are the same when the functions in Q are of the form $Q_w(s,a) = w^T\phi(s,a)$, with $w \in \mathbb{R}^{|S||A|}$ and $\phi : S \times A \to \mathbb{R}^{|S||A|}$, where the feature function ϕ is of the form $\phi(s,a)_{s',a'} = \mathbb{1}[s' = s, a' = a]$

Updates:

$$Q(s,a) := Q(s,a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a) \right) \tag{1}$$

$$w := w + \alpha \left(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a) \right) \nabla_w Q_w(s,a) \tag{2}$$

Solution:

We begin with Eq 2, substituting the derivative with respect to w, given that $Q(s,a) = w^T\phi(s,a)$:

$$w := w + \alpha \left(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a) \right) \phi(s,a) \tag{3}$$

Now we transpose both sides of the equation, and multiply both sides by $\phi(s,a)$:

$$w^T\phi(s,a) := w^T\phi(s,a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a) \right) \phi^T(s,a)\phi(s,a) \tag{4}$$

Now we can again use the fact that $Q(s,a) = w^T\phi(s,a)$:

$$Q(s,a) := Q(s,a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a) \right) \phi^T(s,a)\phi(s,a) \tag{5}$$

10807 Class Project (2016).

Lastly, since $\phi(s,a)_{s',a'} = \mathbb{1}[s' = s, a' = a]$, the norm of the dot product will equal to 1, resulting in:

$$Q(s,a) := Q(s,a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a) \right) \tag{6}$$

And this we proved that Eq 2 is the same as Eq 1.

2 [5pts] Implement a linear Q-network (no experience replay or target fixing). Use the experimental setup of [1, 2] to the extent possible

We implemented a linear Q-network, and to run the training process, one needs to run the command "python dqn.py –modes='q2' ".

We used the following hyper-parameters for this network:

- Discount factor $\gamma = 0.99$
- Learning rate $\alpha = 0.0001$
- Exploration probability $\epsilon = 0.05$, decreasing from 1 to 0.05 in a linear fashion during training process
- Number of iterations with environment: 5,000,000
- Number of frames to feed to the Q-network: 4
- Input image resizing: 84×84
- Steps between evaluations of network: 10,000
- Steps for "burn in" (random actions in the beginning of training process): 50,000
- Maximum episode length: 100,000 steps (basically we chose to allow any game size)

We plotted the performance plot of this network in Fig 1.

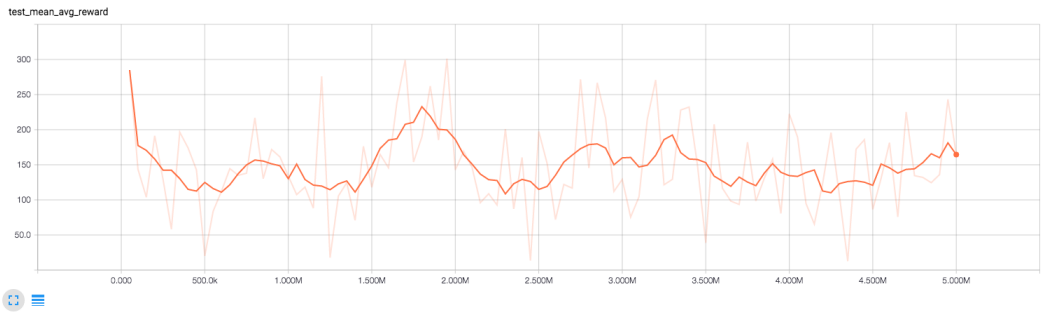

Figure 1: Mean reward per episode plot for the case of linear network without target fixing and without experience replay

Using the *Monitor* wrapper of the gym environment, we generated videos of the behavior of the agent across different stages of training:

- 0/3 of training: Youtube video
- 1/3 of training: Youtube video
- 2/3 of training: Youtube video
- 3/3 of training: Youtube video

Here are also some comments about the behavior and training of this specific network:

- In terms of coding, this was the last network we implemented in this assignment, and in order to remove target fixing and experience replay (which were implemented first) we simply set the target network to in training to be the current network, and set the size of the batch from experience replay to be 1 (we only look at the last sample added to the memory), and train at every step the agent gives
- This network's performance, if filtering for noise, was practically stable during the 5M iterations, being the same as a random policy. Therefore we can assume that training in this condition was unstable
- It was already expected that using a linear network we would not obtain a good testing performance, because we are not able to learn all the game's complexity, specially given the unstable training conditions

3 [10pts] Implement a linear Q-network with experience replay and target fixing. Use the experimental setup of [1, 2] to the extent possible

We implemented a linear Q-network with experience replay and target fixing, and to run the training process, one needs to run the command "python dqn.py –modes='q3' ".

We used the following hyper-parameters for this network:

- Discount factor $\gamma = 0.99$
- Learning rate $\alpha = 0.0001$
- Exploration probability $\epsilon = 0.05$, decreasing from 1 to 0.05 in a linear fashion during training process
- Number of iterations with environment: 5,000,000
- Number of frames to feed to the Q-network: 4
- Input image resizing: 84×84
- Replay buffer size: 1,000,000
- Target Q-network reset interval: 10,000
- Batch size: 32
- Steps between evaluations of network: 10,000
- Steps for "burn in" (random actions in the beginning of training process): 50,000
- Maximum episode length: 100,000 steps (basically we chose to allow any game size)

We plotted the performance plot of this network in Fig 2.

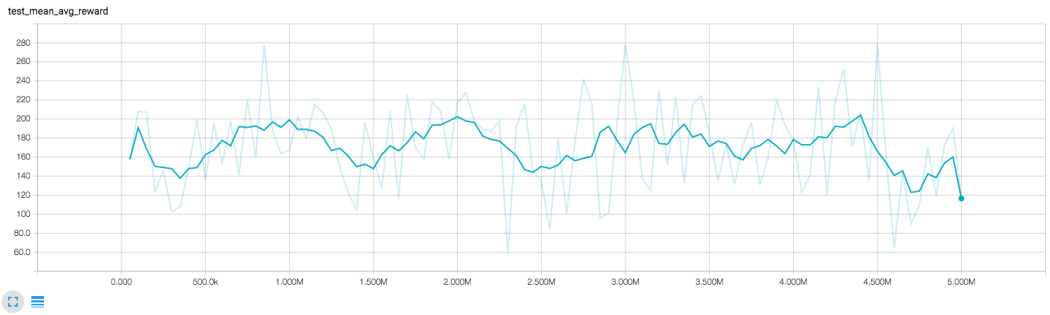

Figure 2: Mean reward per episode plot for the case of linear network with target fixing and with experience replay

Using the *Monitor* wrapper of the gym environment, we generated videos of the behavior of the agent across different stages of training:

- 0/3 of training: Youtube video

- 1/3 of training: Youtube video
- 2/3 of training: Youtube video
- 3/3 of training: Youtube video

Here are also some comments about the behavior and training of this specific network:

- Similar to the case of the linear network without target fixing and experience replay, even after introducing target fixing and experience replay we obtained a resulting linear network that did not show improvement in training in comparison with a random policy. The reward / episode plot in Fig 2 shows a practically stable value close to 170 mean reward/episode during testing conditions.
- It was already expected that using a linear network we would not obtain a good testing performance, because we are not able to learn all the game's complexity

4 [5pts] Implement a linear double Q-network. Use the the experimental setup of [1, 2] to the extent possible.

We implemented a double linear Q-network, and to run the training process, one needs to run the command "python dqn.py –modes='q4' ".

We used the following hyper-parameters for this network:

- Discount factor $\gamma = 0.99$
- Learning rate $\alpha = 0.0001$
- Exploration probability $\epsilon = 0.05$, decreasing from 1 to 0.05 in a linear fashion during training process
- Number of iterations with environment: 5,000,000
- Number of frames to feed to the Q-network: 4
- Input image resizing: 84×84
- Replay buffer size: 1,000,000
- Target Q-network reset interval: 10,000
- Batch size: 32
- Steps between evaluations of network: 10,000
- Steps for "burn in" (random actions in the beginning of training process): 50,000
- Maximum episode length: 100,000 steps (basically we chose to allow any game size)

We plotted the performance plot of this network in Fig 4.

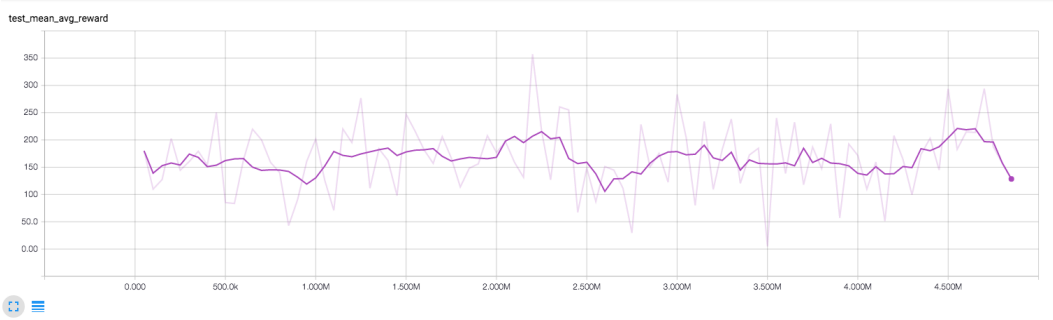

Figure 3: Mean reward per episode plot for the case of double linear network with target fixing and with experience replay

Using the *Monitor* wrapper of the gym environment, we generated videos of the behavior of the agent across different stages of training:

- 0/3 of training: Youtube video
- 1/3 of training: Youtube video
- 2/3 of training: Youtube video
- 3/3 of training: Youtube video

Here are also some comments about the behavior and training of this specific network:

- Similar to the other two cases of linear networks, even after introducing target fixing and experience replay we obtained a resulting linear network that did not show improvement in training in comparison with a random policy. The reward / episode plot in Fig 2 shows a practically stable value close to 150 mean reward/episode during testing conditions.
- It was already expected that using a linear network we would not obtain a good testing performance, because we are not able to learn all the game's complexity

5 [35pts] Implement the deep Q-network as described in [1, 2]

We implemented a linear Q-network with experience replay and target fixing, and to run the training process, one needs to run the command "python dqn.py –modes='deep' –question='vanilla' ".

We used the following hyper-parameters for this network:

- Discount factor $\gamma = 0.99$
- Learning rate $\alpha = 0.0001$
- Exploration probability $\epsilon = 0.05$, decreasing from 1 to 0.05 in a linear fashion during training process
- Number of iterations with environment: 5,000,000
- Number of frames to feed to the Q-network: 4
- Input image resizing: 84×84
- Replay buffer size: 1,000,000
- Target Q-network reset interval: 10,000
- Batch size: 32
- Steps between evaluations of network: 10,000
- Steps for "burn in" (random actions in the beginning of training process): 50,000
- Maximum episode length: 100,000 steps (basically we chose to allow any game size)

We plotted the performance plots of this network in Fig 5.

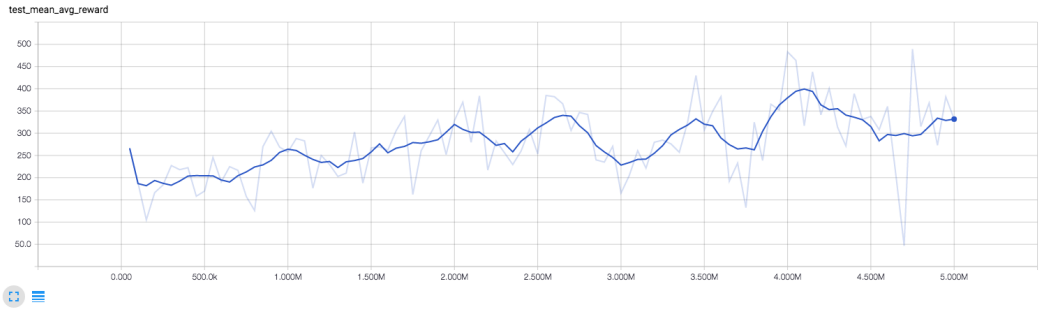

Figure 4: Mean reward per episode plot for Space Invaders for the case of deep Q network with target fixing and with experience replay

Using the *Monitor* wrapper of the gym environment, we generated videos of the behavior of the agent across different stages of training:

For Space invaders:

- 0/3 of training: Youtube video
- 1/3 of training: Youtube video
- 2/3 of training: Youtube video
- 3/3 of training: Youtube video

Here are also some comments about the behavior and training of this specific network:

- We implemented a deep Q network equal to the setup described in [2], using 3 convolutional layers
- Using the deep Q network we achieved performance significantly higher than the random policy. While the random policy presents mean reward/episode around 160 or 170, with at the end of training we obtained mean reward/episode of about 350 points
- It was interesting to notice a improvement rate of reward/episode which was, if you consider a highly smoothed curve, almost linear with respect to the number of iterations. Since we did not observe any significant plateuing of this curve during the 5M iterations, we can suppose that if we allowed the system to run for longer we could observe even better performances

6 [20pts] Implement the double deep Q-network as described in [3]

We implemented a double deep Q-network, and to run the training process, one needs to run the command "python dqn.py –modes='deep' –question='double' ".

We used the following hyper-parameters for this network:

- Discount factor $\gamma = 0.99$
- Learning rate $\alpha = 0.0001$
- Exploration probability $\epsilon = 0.05$, decreasing from 1 to 0.05 in a linear fashion during training process
- Number of iterations with environment: 5,000,000
- Number of frames to feed to the Q-network: 4
- Input image resizing: 84×84
- Replay buffer size: 1,000,000
- Target Q-network reset interval: 10,000
- Batch size: 32
- Steps between evaluations of network: 10,000
- Steps for "burn in" (random actions in the beginning of training process): 50,000
- Maximum episode length: 100,000 steps (basically we chose to allow any game size)

We plotted the performance plots of this network for Space Invaders in Fig 6.

Using the *Monitor* wrapper of the gym environment, we generated videos of the behavior of the agent across different stages of training:

- 0/3 of training: Youtube video
- 1/3 of training: Youtube video
- 2/3 of training: Youtube video
- 3/3 of training: Youtube video

Here are also some comments about the behavior and training of this specific network:

- Using the double deep Q network we did not observe a significant improvement over the "single" deep Q network, both in terms of training time nor increase in mean reward/episode
- Perhaps a significant difference of the use of the double deep Q network can be noted for a larger number of iterations

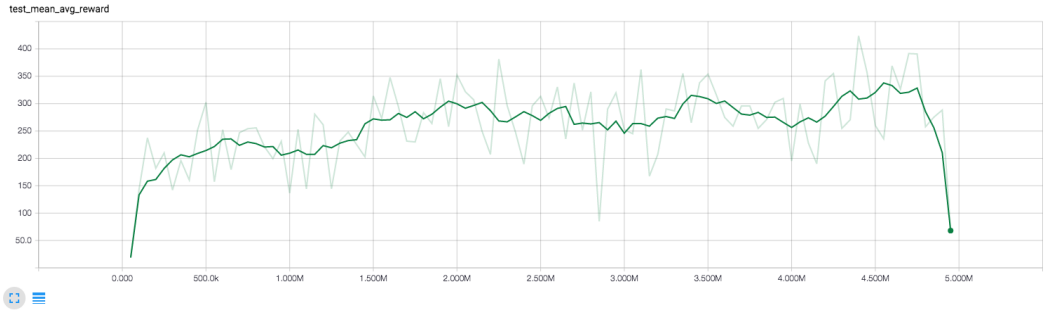

Figure 5: Mean reward per episode plot for the case of double deep network with target fixing and with experience replay

7 [20pts] Implement the dueling deep Q-network as described in [4]

We implemented a dueling deep Q-network, and to run the training process, one needs to run the command "python dqn.py –modes='q7' ".

We used the following hyper-parameters for this network:

- Discount factor $\gamma = 0.99$
- Learning rate $\alpha = 0.0001$
- Exploration probability $\epsilon = 0.05$, decreasing from 1 to 0.05 in a linear fashion during training process
- Number of iterations with environment: 5,000,000
- Number of frames to feed to the Q-network: 4
- Input image resizing: 84×84
- Replay buffer size: 1,000,000
- Target Q-network reset interval: 10,000
- Batch size: 32
- Steps between evaluations of network: 10,000
- Steps for "burn in" (random actions in the beginning of training process): 50,000
- Maximum episode length: 100,000 steps (basically we chose to allow any game size)

We plotted the performance plot of this network in Fig 7.

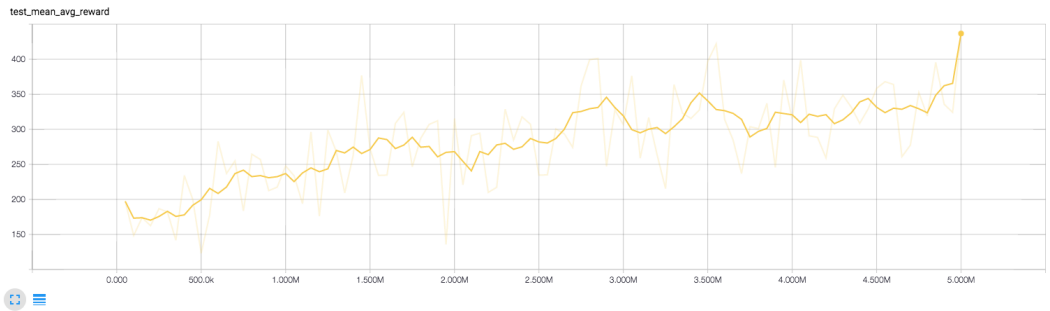

Figure 6: Mean reward per episode plot for the case of dueling deep network with target fixing and with experience replay

Using the *Monitor* wrapper of the gym environment, we generated videos of the behavior of the agent across different stages of training:

- 0/3 of training: Youtube video
- 1/3 of training: Youtube video
- 2/3 of training: Youtube video
- 3/3 of training: Youtube video

Here are also some comments about the behavior and training of this specific network:

- As stated in [4], we achieved faster training and higher rewards with the use of the dueling network architecture. which performed equal or better than the other deep Q-network architectures
- We also expect results to be significantly better for the dueling architecture if we allowed it to run for more iterations with the environment

8 Table comparing rewards for each fully trained model

We constructed a table comparing the average total reward found in 100 episodes for each fully trained model we implemented. To generate the table one has to run the command "python dqn.py –modes='deep' –question='double' "

Table 1: Avg reward per episode for 100 episodes in implemented networks

Model	Game	Avg Reward 100 episodes
Linear, no target fix, no exp replay	Space Invaders	146.3 ± 76.2
Linear, with target fix, with exp replay	Space Invaders	138.1 ± 100.9
Double Linear	Space Invaders	184.5 ± 144.1
Deep Q	Space Invaders	319.7 ± 148.7
Double Deep Q	Space Invaders	315.9 ± 154.2
Dueling Deep Q	Space Invaders	343.5 ± 135.6

Here are some comments about the results in the table:

- Both linear networks (with and without target fixing) had very similar results with low performance
- The double linear network performed, on average, a bit better than the other linear networks, but the difference was not very significant, and includes with much higher variance
- Both deep Q networks had very similar performances, well above the linear cases, and similar variances
- The dueling deep Q network obtained the best performance of all cases, and in addition had a variance slightly smaller than the other deep networks

9 Conclusion and overall comparison of architectures

In this assignment we compared the performance of different neural network architectures for the function-approximation task in the reinforcement learning problem, trying to learn how to play Atari games, more specifically Space Invaders. We obtained significantly better results using deep architectures rather than linear networks. Linear networks did now show great improvement in comparison with a purely random policy.

We plotted all performance curves together in Fig 9 for comparison:

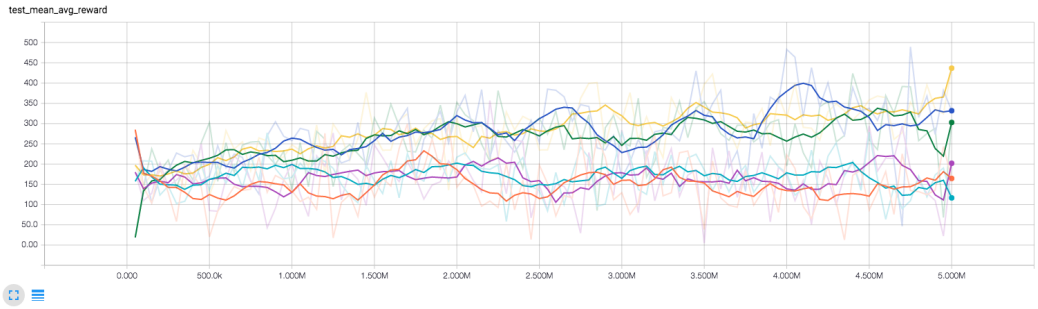

Figure 7: Mean reward per episode plot for all architectures. Orange: linear without target fixing and replay. Cyan: linear without target fixing and replay. Purple: double linear. Blue: deep network. Green: double-deep network. Yellow: dueling deep network

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[4] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.