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 $\phi : S \times A \rightarrow \mathbb{R}^{|S||A|}$, 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 ".

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 plots of this network in Figs 2-2.

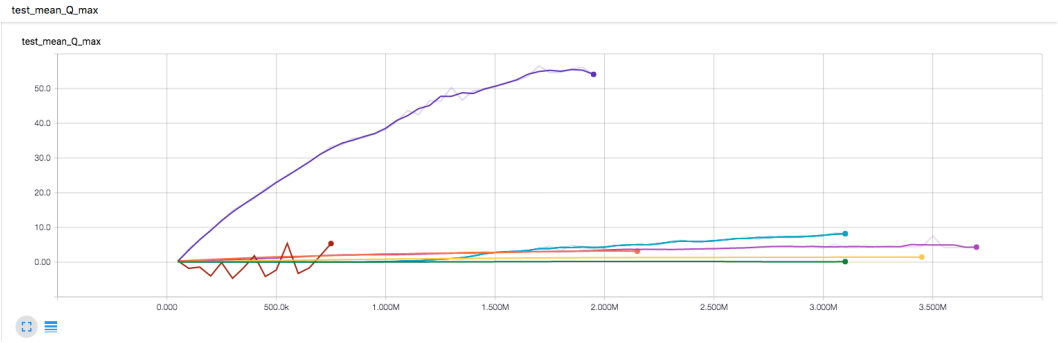

Figure 1: Mean Q per step plot for the case of linear network without target fixing and without experience replay

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:

- Bla
- Bla

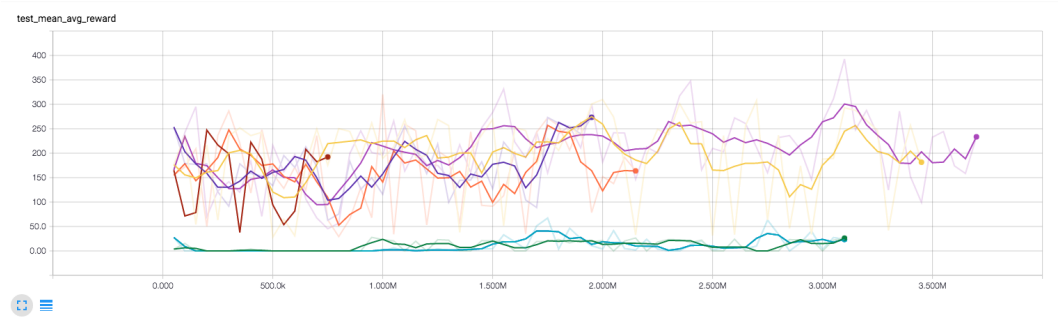

Figure 2: Mean reward per episode plot for the case of linear network without target fixing and without experience replay

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, and to run the training process, one needs to run the command "python dqn.py –modes ".

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 Figs **??-??**.

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:

- Bla
- Bla

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 ".

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 Figs **??-??**.

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:

- Bla
- Bla

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

We implemented a deep Q-network. We tested the performance of this network with different games, and to run them, one can use the following commands:

- Space invaders: "python dqn.py –modes "
- Enduro: "python dqn.py –modes "
- Breakout: "python dqn.py –modes "

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 for different games in Figs **??**-**??**.

Using the *Monitor* wrapper of the gym environment, we generated videos of the behavior of the agent across different stages of training, for the 3 games considered:

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

For Enduro:

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

For Breakout:

- 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:

- Bla
- Bla

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 ".

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 Figs **??**-**??**.

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:

- Bla
- Bla

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 ".

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 Figs **??**-**??**.

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:

- Bla
- Bla

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:

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	50 ± 5
Linear, with target fix, with exp replay	Space Invaders	50 ± 5
Double Linear	Space Invaders	50 ± 5
Deep Q	Space Invaders	50 ± 5
Deep Q	Enduro	50 ± 5
Deep Q	Breakout	50 ± 5
Double Deep Q	Space Invaders	50 ± 5
Dueling Deep Q	Space Invaders	50 ± 5

Here are some comments about the results in the table:

- Bla
- Bla

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.