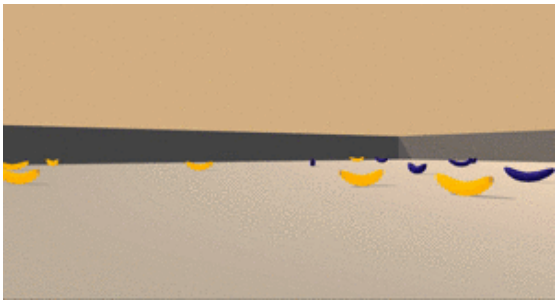
Banana NavigationProject Report

Deep Q Network Introdution

There are 8 details of the paper [1] worth to pay great attention to.

1. Pre-processing: Extracting the Y channel, also known as luminance, from the RGB frame and rescale it to 84 by 84.
2. Frames Stacking : To encode a single frame the authors take the maximum value for each pixel colour value over the frame being encoded and the previous frame. This was necessary to remove flickering that is present in games where some objects appear only in even frames while other objects appear only in odd frames, an artefact caused by the limited number of sprites Atari 2600 can display at once.
3. Frame-Skipping Technique: More precisely, the agent sees and selects actions on every kth frame instead of every frame, and its last action is repeated on skipped frames. Because running the emulator forward for one step requires much less computation than having the agent select an action, this technique allows the agent to play roughly k times more games without significantly increasing the runtime. We use k=4 for all games.
4. Experience Replay： This approach has several advantages over standard online Q-learning. First, each step of experience is potentially used in many weight updates, which allows for greater data efficiency. Second, learning directly from consecutive samples is inefficient, owing to the strong correlations between the samples; randomizing the samples breaks these correla- tions and therefore reduces the variance of the updates. Third, when learning on- policy the current parameters determine the next data sample that the parameters are trained on. For example, if the maximizing action is to move left then the train- ing samples will be dominated by samples from the left-hand side; if the maximiz- ing action then switches to the right then the training distribution will also switch. It is easy to see how unwanted feedback loops may arise and the parameters could get stuck in a poor local minimum, or even diverge catastrophically.
5. Fixed Q Target：
6. Reward Clipped: As the scale of scores varies greatly from game to game, we clipped all positive rewards at 1 and all negative rewards at -1, leaving 0 rewards unchanged. Clipping the rewards in this manner limits the scale of the error derivatives and makes it easier to use the same learning rate across multiple games. At the same time, it could affect the performance of our agent since it cannot differentiate between rewards of different magnitude. For games where there is a life counter, the Atari 2600 emulator also sends the number of lives left in the game, which is then used to mark the end of an episode during training.
7. Error Clipped
8. Target Networks Soft Update

Details on Navigation Project



Navigation Setting

State Space

The observations are in a 37-dimensional continuous space corresponding to 35 dimensions of ray-based perception of objects around the agent's forward direction and 2 dimensions of velocity. The 35 dimensions of ray perception are broken down as: 7 rays projecting from the agent at the following angles (and returned back in the same order): [20, 90, 160, 45, 135, 70, 110] where 90 is directly in front of the agent. Each ray is 5 dimensional and it projected onto the scene. If it encounters one of four detectable objects (i.e. yellow banana, wall, blue banana, agent), the value at that position in the array is set to 1. Finally there is a distance measure which is a fraction of the ray length. Each ray is [Yellow Banana, Wall, Blue Banana, Agent, Distance].

For example, [0,1,1,0,0,0.2] means that there is a blue banana detected 20% of the distance along the ray with a wall behind it. The velocity of the agent is two dimensional: left/right velocity (usually near 0) and forward/backward velocity (0 to 11.2). Given this information, the agent has to learn how to best select actions.

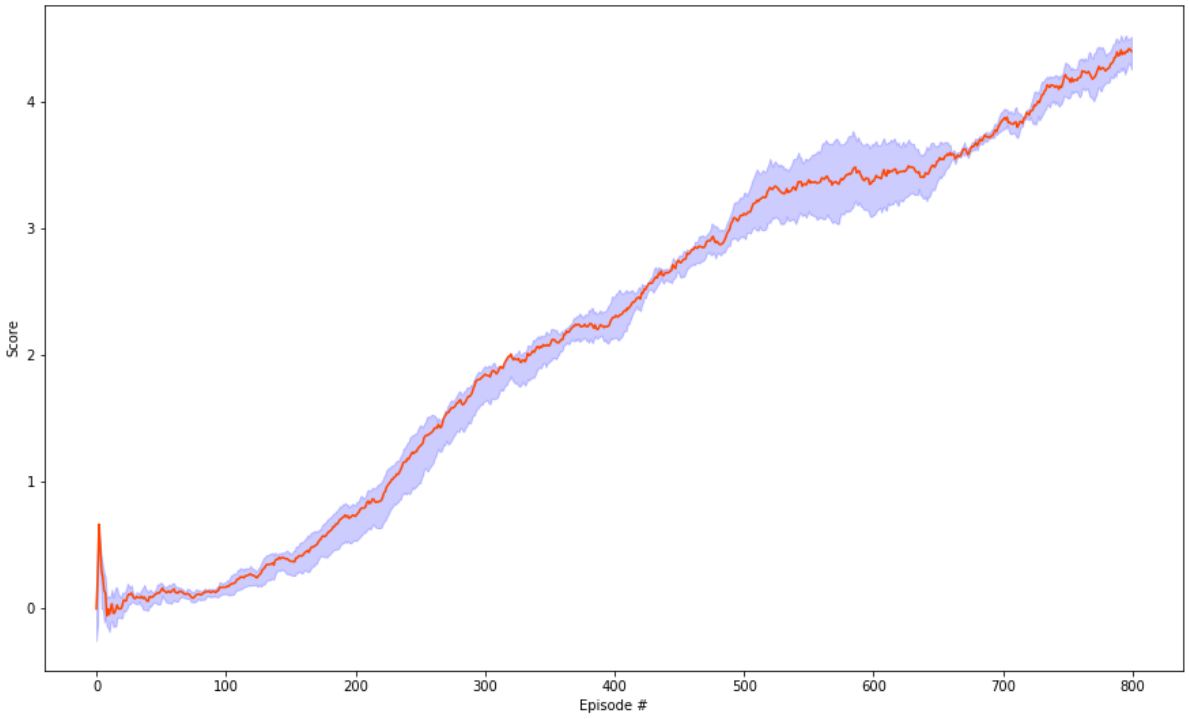
Action Space

Four discrete actions are available:

- 0: move forward.
- 1: move backward.
- 2: turn left.
- 3: turn right.

Experiment Setting

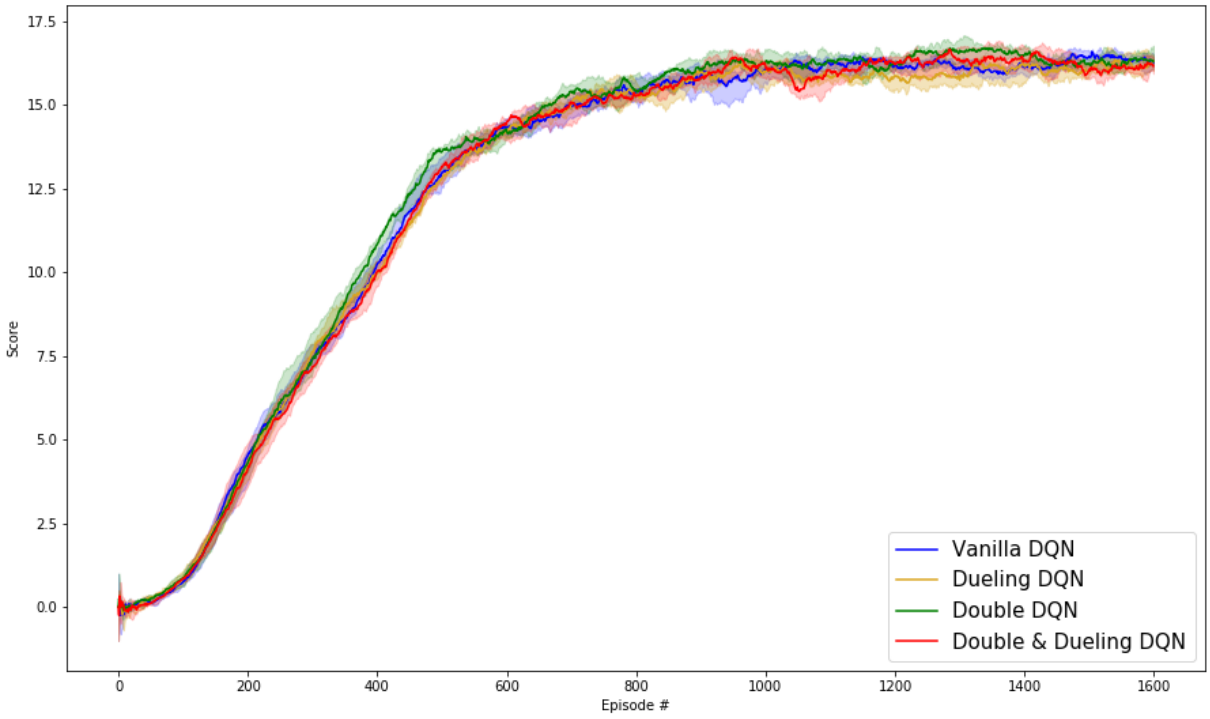
In these experiments, we used the A algorithm with minibatches of size 32. The behaviour policy during training was ϵ-greedy with ϵ annealed linearly from 1.0 to 0.01 with the decay rate 0.995, and fixed at lowest value 0.01. I trained for a total of 1600 episodes, which maximum time step is 1000. Used a replay memory of 1 million most recent frames, and samples uniformly at random from replay buffer when performing updates. Learning rate α is 0.0005, and discount factor γ is 0.99. And I set Frame-Skipping parameter as 4, that means he agent sees and selects actions on every 4th frame instead of every frame.



The solid lines are the median scores, and the shaded area denotes the interquartile range across 8 random initializations. While the variability between runs is substantial, there are significant differences in final achieved score, and also in learning speed.

Advanced DQN

Two improvements had been done in this navigation project. Detailed learning curves for vanilla (red), double Q learning (blue), dueling Q network (green) and Double Q learning based on dueling network. Again, the solid lines are the median scores, and the shaded area denotes the interquartile range across 8 random initializations. While the variability between runs is substantial, there are significant differences in final achieved score, and also in learning speed.

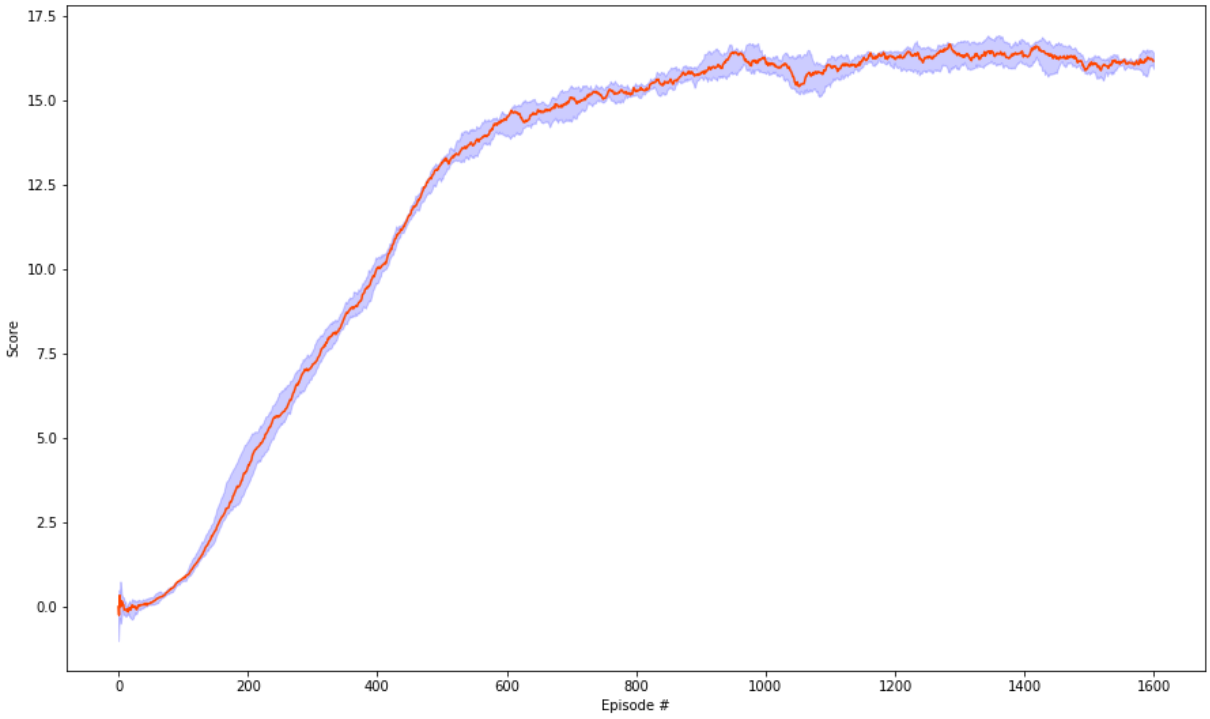


In the following I will introduce the relative advanced DQN improvement as compared to the vanilla one.

Double Q Learning

In order to achieve better training performance, here I applied the double Q learning method to improve navigation DQN.

The double Q learning can reduce the learning signal noise. To be more specifically, the vanilla DQN tends to exaggerate the estimated Q value: when you take the max, you're going to exaggerate this error where the Q function is bigger than it should be, which leads to overestimation.



The fig. shows that the double Q learning is not only more stable than the vanilla one, but also more quickly converge and achieve higher scores, which shows the efficiency of reducing the learning noise.

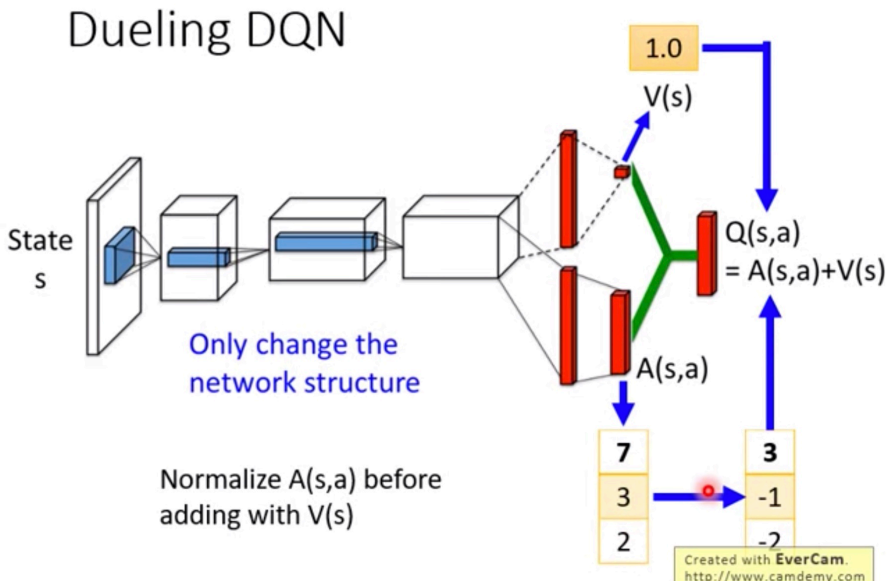
Dueling DQN

Dueling DQN is just modified the final second layer of the vanilla one, which have two separate fully connected layers output stream, namely V and A, referred to Value and Advantage respectively.

Why this works? When we have V output, which stands for the expected value of the Q value, it improves the performance of the network generalization and data efficiency. Because in this situation the V can efficiently update all the same state Q value, with fewer single update of each Q value.

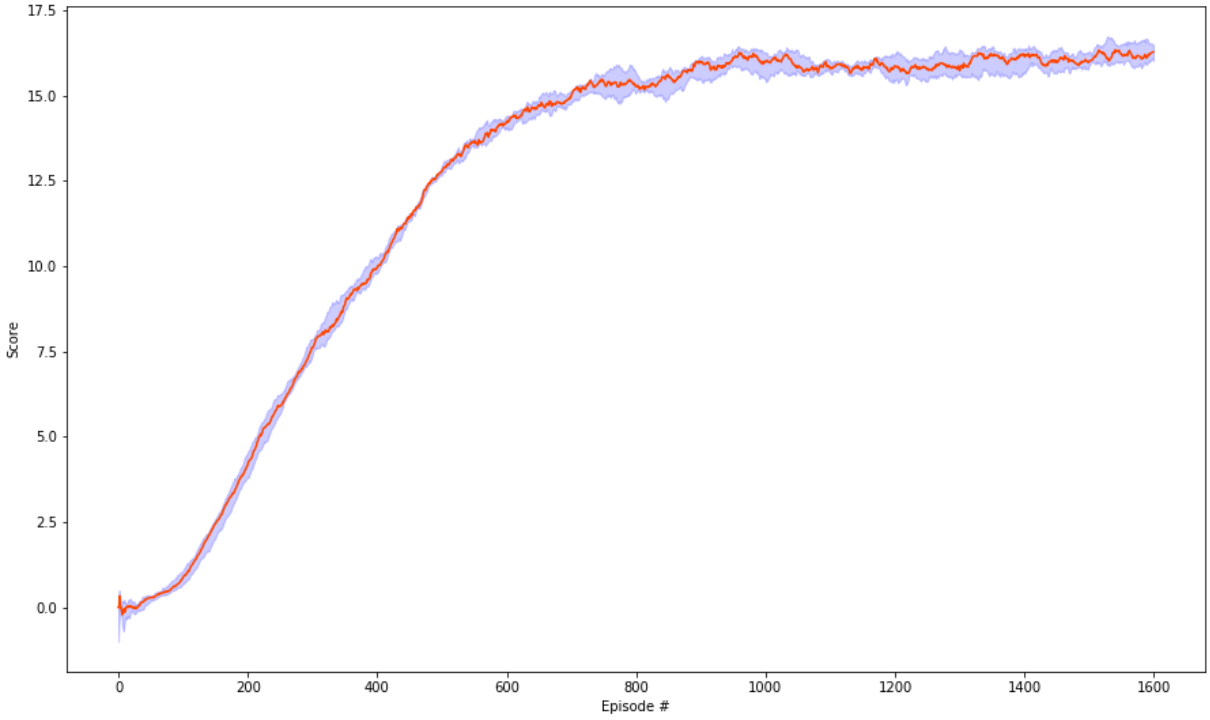
To achieve this approach, A constraint must be considered.

- Each state's Advantages summation so equals to zero, in order to force the network to update more the V instead of A.
- The normalization of Advantage: minus all the A value with the mean of all A value, so that the normalized A summation



equals to one.

Here is the experiment result, as we can see there is no obvious different with the above two, that is because the input state is simple and easy to learn. But if we input the raw RGB pixal image, we would see the algorithm different performance.



Further Improvement

1. Prioritized Buffer Replay:
   In this project, the navigation agent samples uniformly at random from D when performing updates. This approach is in some respects limited because the memory buffer does not differentiate important transitions and always overwrites with recent transitions owing to the finite memory size N. Similarly, the uniform sampling gives equal importance to all transitions in the replay memory. A more sophisticated sampling strategy might emphasize transitions from which we can

learn the most, similar to prioritized sweeping.

2. Dueling DQN
3. Multi-step Return
4. Noise Net
5. Distributtional Q Learning
6. Rainbow

Resource

The relative code is provided:

https://github.com/YijiongLin/Udacity_DRL_NanoDegree

Youtube:

https://www.youtube.com/watch?v=R5-DxW0c0uU&feature=youtu.be

Bilibili:

References

[1] Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540):529-533.