# Project 1: Navigation

This is for the vectorized input problem.

## Learning Algorithm

I have taken the previous exercise as a skeleton for this problem.

You have the same structure a dqn\_agent.py file that contains the learn and acting part. And a model.py file that contains the Deep Neural Network, use to evaluate the Q function.

This implementation is allowing Deep Reinforcement learning with 3 optional features

* Double DQN
* Prioritized Experience Replay
* Dueling

You can activate and deactivate using the Boolean in the hyperparameters cell.

I have also added tools, in the ./tools folder, like the prioritized replay buffer. To manage performance of big buffer, I had to implement Binary Tree sampling like proposed in the DeepMind paper.

Hyperparameters for the DQN:

All hyperparameters are manage from this cell, and are passed a argument dictionary to the relevant function.

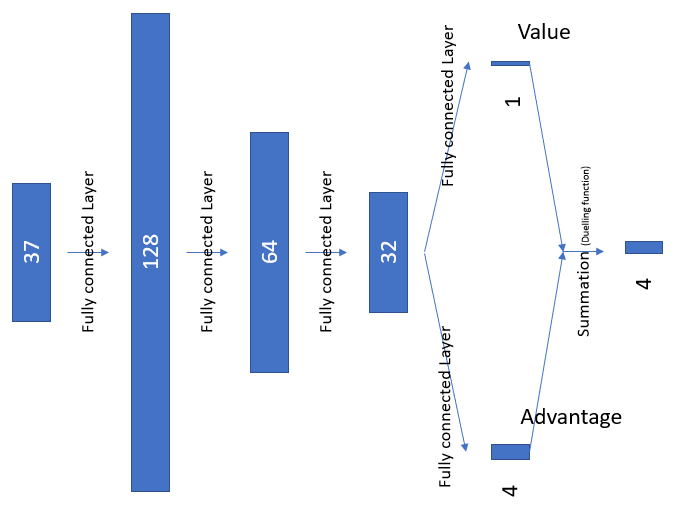


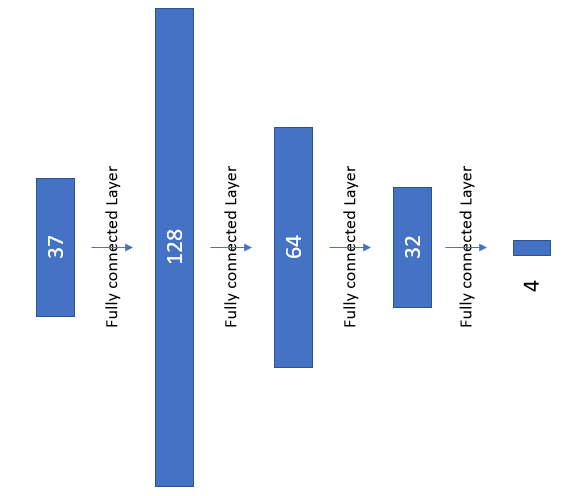
After some long trial and error, I have settled on those parameters:

* GPU active for performance.
* Double DQN active,
* Prioritized replay inactive, I have better performance without, I tried introducing clipping of TD Error, changed Alpha and Beta, etc… But still better without.
* Dueling active
* Replay buffer at 10000 like previously
* Batch size for learning at 64
* Gamma at 0.99, very standard, just not to loop.
* Learning Rate 0.05
* Trigger learning every 4 episodes
* And change the target network every 20 episodes
* Alpha, TD\_Error clipping and beta not used as PER inactive
* Tau at 0.05 as the rate to move from current to target network.

For the Neural Network

I have used a simple structure for the vectorized problem,

Without Dueling With Dueling

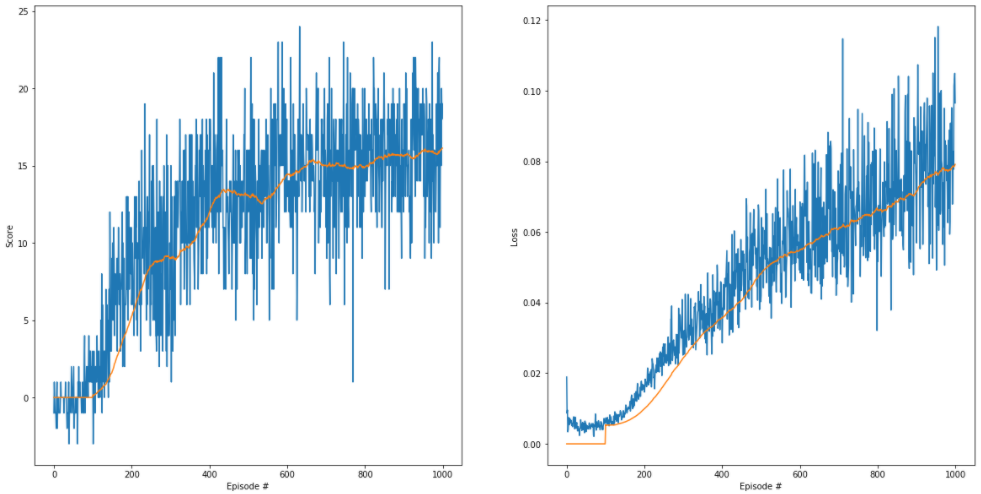


## Plot of Rewards

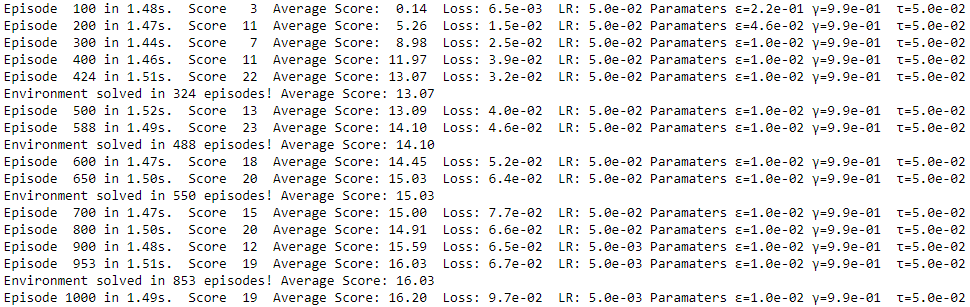
When using the “Navigation.ipynb” you have the following result in 2 plots.

1. score and average score over the last 100 episodes per episode of training.
2. Loss function and average loss over the last 100 episodes per episode of training.

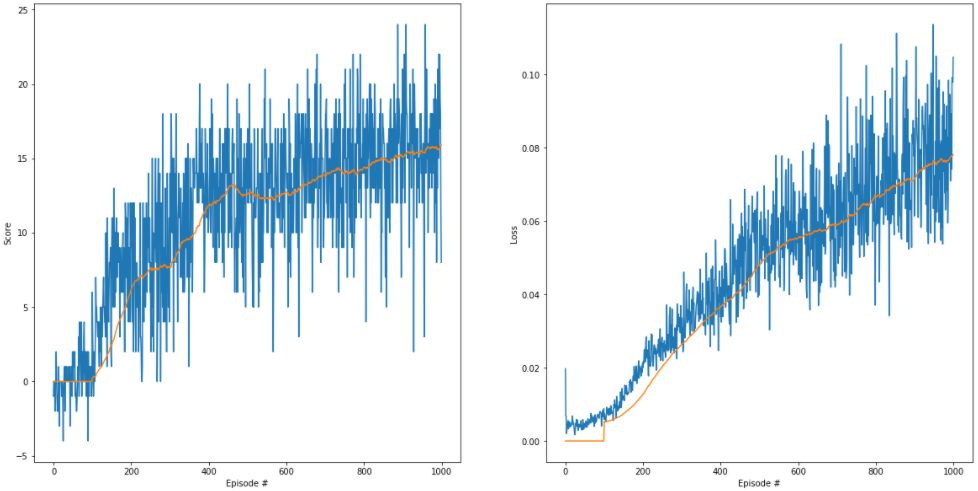
With Dueling and DDQN activate



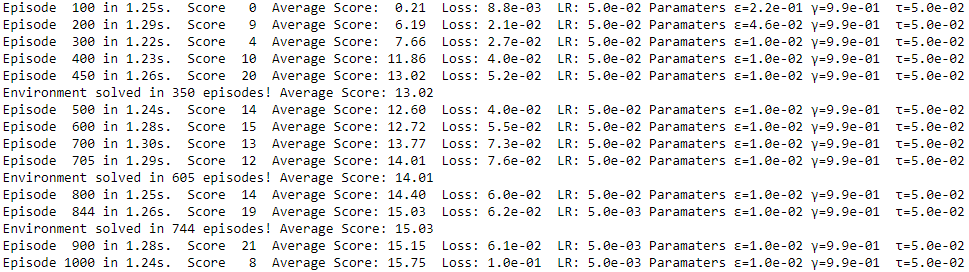
The average score over 100 episodes has reached 13 on episodes 424, as the instructor way of counting, this implies that the score of 13 was reached 100 episodes before. So 324



Without any extra feature, just DQN



The average score over 100 episodes has reached 13 on episodes 450, as the instructor way of counting, this implies that the score of 13 was reached 100 episodes before. So 350.

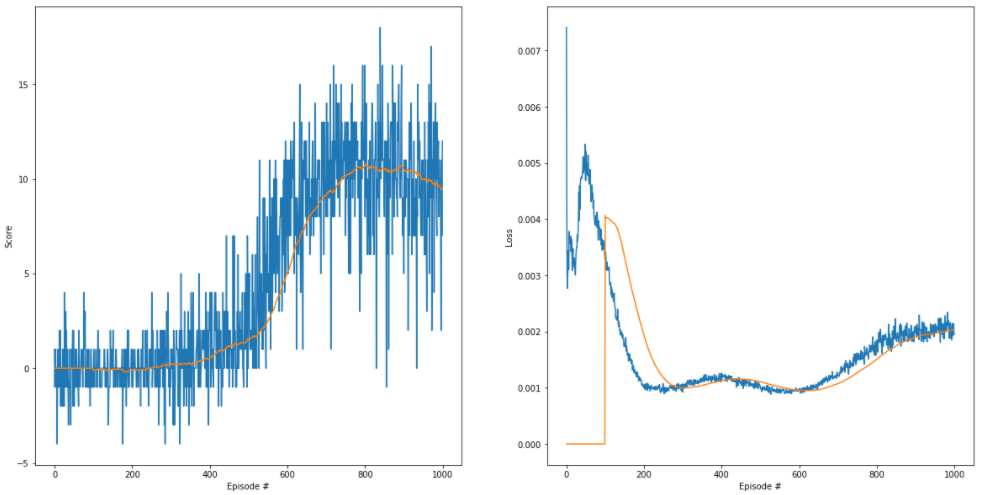


My conclusion is that without extra features it takes more time to learn and you get a lower best score. Probably due to the difficultly of randomly walking into a yellow banana, it seems to take 100 episodes to understand that grabbing yellow bananas is better than wandering around. But even more than the extra features, I fear that hyperparameters right picking is also very influential on the end result.

## Ideas for Future Work

On the vector problem:

* Find a better quicker/less complex NN structure that converge and learn faster.
* Find better alpha and beta parameter that make PER more beneficial.
* Better understand the link between network structure and convergence.
* Understand why PER is bringing so much overfitting



On the vision problem:

* Change the input processing of the input image. Stop using RGB and create 1 frame for Yellow, one for Blue and one for background, on 4 frames, so 12 input channels to the convolutional network.
* Understand how long a network need to converge. (not like Deepmind brute force)
* Better understand the link between structure and convergence.
* Better understand impact/benefits of more depth or more convolutions.
* Better understand where to put residual layers.

I have spent some time on the code to make it nice, it is more now the fine tuning of parameters the next and perpetual challenge.