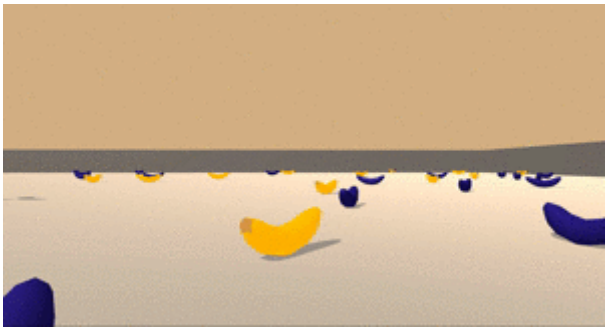
Banana collector project report

Daniel Thell, August 2020

The problem[1]

For this project, the aim is to train an agent to navigate (and collect bananas) in a large, square world.



A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of your agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around agent's forward direction. Given this information, the agent has to learn how to best select actions. Four discrete actions are available, corresponding to:

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

The task is episodic, and in order to solve the environment, the agent must get an average score of +13 over 100 consecutive episodes.

The learning algorithm

The agent uses a Deep Neural Network to model the Q-value function. It learns from a number of random episodes, from the environment set in training mode, and choses actions at each step using and epsilon greedy policy.

It starts with uniformly random actions and learns from their effect on the environment. At start the epsilon parameter is 1, indicating the actions are sampled using a uniform random

[1] This first section “The Problem” is taken from Udacity Navigation Project from the Deep Reinforcement Learning Nanodegree

distribution. At every episode, epsilon is reduced gradually to leverage on the accumulated knowledge and refine what seems like the best actions (exploitation) while gradually reducing the degree of exploration. Further details on the hyperparameters chosen are given in the next section. The value of epsilon is kept constant for an entire episode.

At each step within an episode, once the action is chosen, the agent acts using this action and the environment gives a reward value as a feedback (which could be +1, -1 or 0 if no banana was collected) and the next state value.

At every step, the current state, the action chosen, the next state, the reward and an indication whether the episode is ended are stored. Once a sufficient number of experiences are achieved, they are used to learn from the problem from time to time through experience replay. Every N step, the agent uses a replay buffer to sample from the stored experiences a number of past experiences to learn from (which constitute a mini-batch) and update its policy.

The learning is done by applying gradient decent on the deep neural network describing the q-value function, using a given learning rate. Actually 2 identical deep neural networks are used to avoid correlated updates and the quick loss of useful and rare events, this is described in the next section.

The score achieved (the number of yellow bananas collected minus the number of blue bananas collected) is reported and this score should reach at least +13 during 100 consecutive episodes for the environment to be considered solved.

Once this is reached, the parameters of the DQN are stored in a file.

The solutions investigated

The investigation started with the following:

An agent with 2 QNetworks, a target and a local QNetworks, to avoid the common DQN issues: strongly correlated updates that breaks the independent and identically distributed assumption of the gradient decent algorithm and the quick forgetting of useful and rare experience. One DQN is called target network, it is the one used to get the target next state q-value, kept stable for a while. The other DQN is called local network, it is the one where gradient decent is applied. The target network is gradually updated only every 4 steps using a given percentage (governed by Tau) of the local network. This is however not a double DQN, it is a vanilla DQN algorithm as the the action and its q-value for the target q-values are obtained using the same QNetwork.

I started with vanilla DQN and only 2 hidden layers with ReLU activation fonctions (except after the last layer which was always plainly linear).

The hyperparameters were:

- Epsilon: starting at 1 and multiplied by a decay rate of 0.995 at each episode with a minimum epsilon value of 0.02
- Learning rate: 5e-4
- Update of the target QNetwork: every 4 steps

- Gamma: 0.99 (discount factor)
- Tau: 1e-3, soft update parameter to update the target QNetwork after a learning step
- Mini-batch size: 64, number of experienced sampled from the replay buffer used for learning
- Optimizer: Adam optimizer

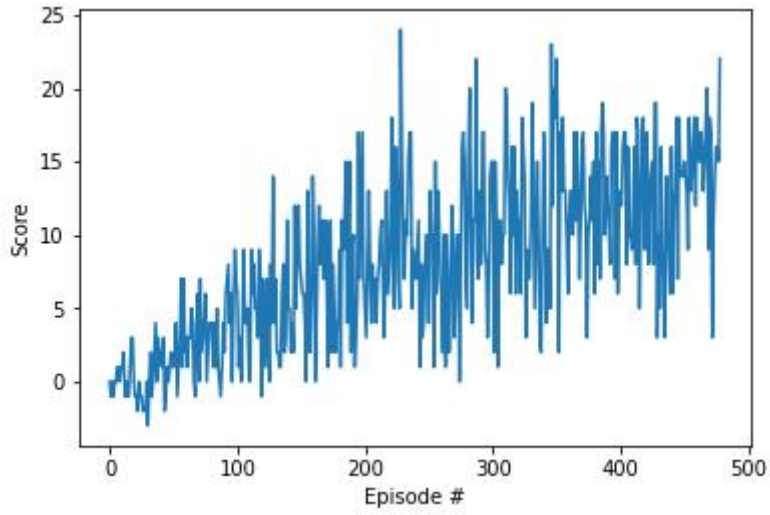
2 hidden layers vanilla DQN would not solve the algorithm in less than 500 episodes and the average score over 100 consecutives episodes reaches between 12.29 and 12.65 after 500 episodes. This is close to the target and letting the agent train longer could help reach the 13 mark but the aim at this stage is to optimize the parameters.

Increasing to 3 hidden layers helped just a bit.

Number of hidden layers	Layers sizes	Average score after 500 episodes
2	64 x 64	12,64
2	64 x 128	12,29
2	128 x 64	12,56
3	64 x 128 x 32	12,83
3	64 x 64 x 32	12,55

Going to 4 hidden layers improves the average score to over 13 in less than 500 episodes as with 64 x 128 x 128 x 64 networks the agent reaches average score of 13 over the last 100 episodes on episode 459 on the first training and 499 on a second training attempt. The agent solved hence the environment in 359 episodes on the first training and 399 episodes on the second training.

The following graph shows the evolution of the scores with the number of episodes for this vanilla DQN.



This 4 hidden layer architecture has been retained for the rest of the project.

Optimizing the hyper parameters

To speed up the performance, the epsilon decay rate, the minimum epsilon level, the sizes of the 4 hidden layers, the frequency of parameters update and the learning rate have been tuned.

Starting with the layers size, complexifying the network to get potentially more accurate or complex decisions did not lead to an improvement. The agent was achieving lower scores after 500 episodes with a more complex network architecture. Repeating the training a second time to see whether it was only bad luck with the training set did not change the conclusion. The initial architecture (64x128x128x64) was retained. Note that for these the epsilon decay rate was set to 0.99.

Hidden layers sizes				Average score after # episodes					
1st	2nd	3rd	4th	100	200	300	400	500	Comment
64	128	128	-64	1,14	6,38	9,53	13,15	14,21	1st training attempt
64	128	128	64	1,9	6,98	10,79	13,02	15,22	2nd training attempt
128	256	128	64	1,56	6,86	10,21	12,45	13,12	1st training attempt
128	256	128	64	1,92	6,71	9,55	12,89	13,98	2nd training attempt

An epsilon decay rate of 0.99 instead of 0.995 seemed to work better. Using the 4 hidden layers DQN identified above (hidden layer sizes: 64x128x128x64), the target average score of 13 was reached faster with this value, before episode 400 compared to between episode 459 and 499 with a 0.995 decay rate and the exact same network architecture.

The minimum epsilon level of 0.01 instead of 0.02 was barely making a difference. The score obtained at a high number of episodes, meaning when the minimum epsilon was reached, was just slightly higher with the lower minimum epsilon value.

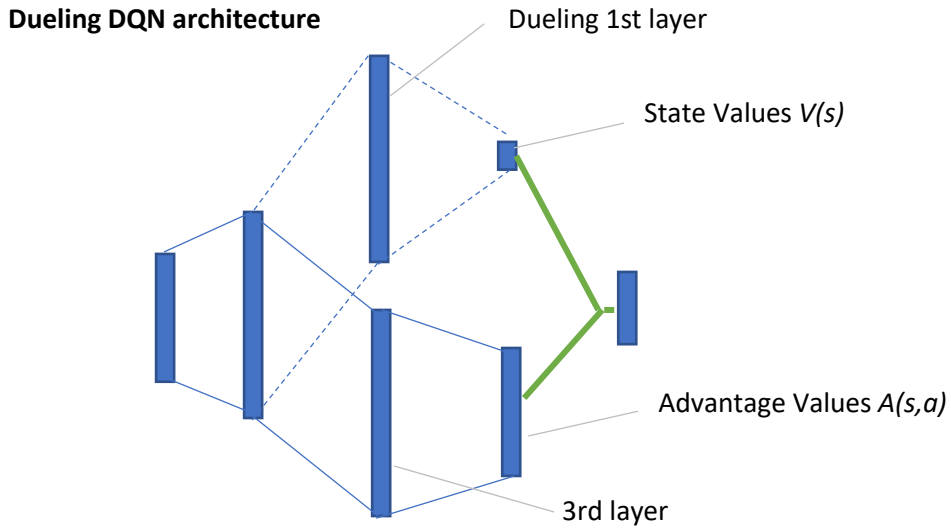
In Deep Reinforcement Learning, we use 2 QNetworks to avoid the the issues discussed at the start of the investigation section. To update the target QNetwork, an hyperparameter is defined to specify how often the target QNetwork parameters are learned. 2 parameters have been tested: 4 and 10. The best results, meaning the best average score over the past 100 episodes for various number of episodes, have been achieved with an update every 4 steps. This was observed early, in the middle or late in the training consistently. The parameter 4 was retained for the future of the investigations.

	Average score after # episodes (DQN 4 hidden layers 64x128x128x64)				
Update frequency	100	200	300	400	500
Every 4 steps	1,8	5,91	10,81	13,3	14,84
Every 10 steps	0,74	2,79	5,04	8,95	11,69

Last the learning rate has been tested over 2 parameters 5e-4 and 1e-3. The results were similar, sometimes better at some stages sometimes a bit worse with the later parameter but depending in the end on the training attempt and the final achieved score after 500 or even 800 episodes was very similar.

Dueling DQN

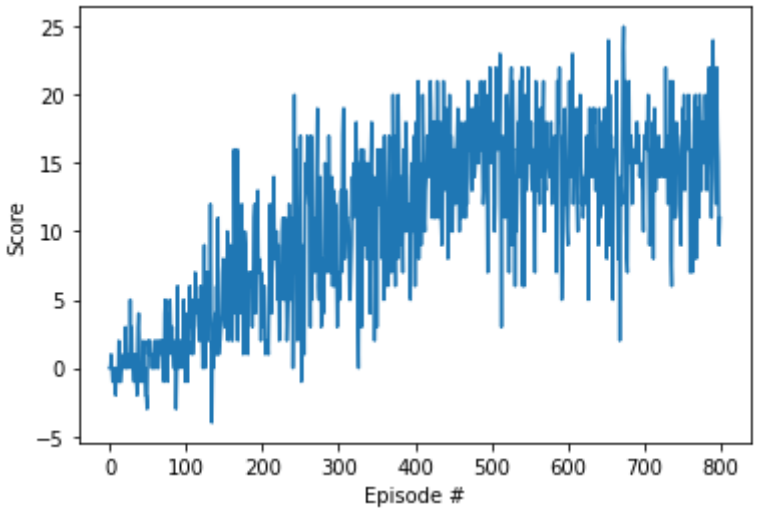
To further optimize the agent, I tried a dueling network with sizes for the 1st hidden layer of the V(s) branch of the dueling network, the only layer where the size can be changed. The following notations are used in the table below:



The following results were achieved, showing a more complex network does not really improves the performances, it would on the contrary be slower to learn. On the other side, a too small network with only 64 nodes on the 3th layer of the q-values hammers the learning rate.

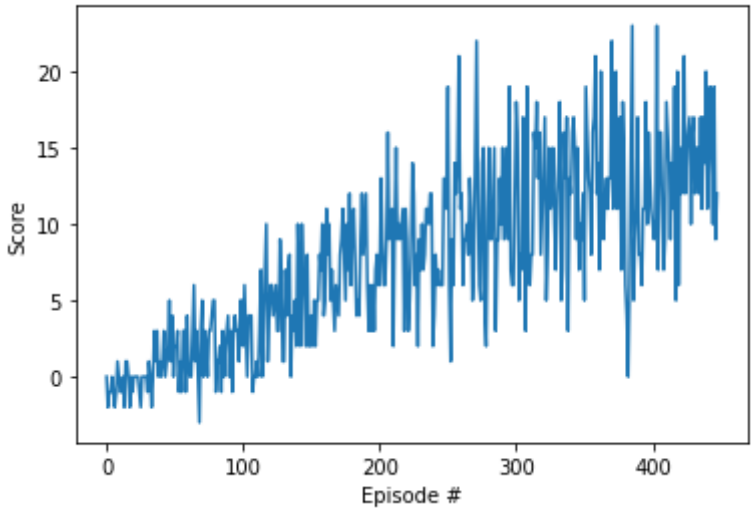
Hidden layers sizes					Average score after # episodes							
1st	2nd	3rd	4th	Dueling 1st	100	200	300	400	500	600	700	800
64	128	128	64	128	1,32	5,61	8,55	11,9	14,8	14	13,4	15
64	128	128	64	32	1,02	5,05	8,72	10,7	15	15,1	14,4	14,6
64	128	256	4	256	0,21	1,34	1,79	4,12	6,51	7,53	8,4	9,02
64	128	64	4	32	0,25	2,15	2,63	4,79	8,17	10,2	11,1	12,6
64	128	128	4	32	0,33	2,1	6,59	8,85	11,3	13,8	14,7	15,2

The evolution of the score over the training episodes for the first Dueling DQN of the table above is shown below



The Dueling DQN solved the problem here in 359 episodes. The average score over the past 100 episodes was 13.01 on the 459th episode. We also notice that the algorithm seems to have a plateau around a score of 15 beyond 500 episodes.

For the 2nd Dueling DQN architecture, very similar results were achieved, with an environment solved in 348 episodes with an average score over episodes 348 to 448 of 13.01.



Even though the performance is quite good for multiple architectures after 800 episodes, only 2 architectures achieve in less than 500 episode a score of at least 13, the first 2. The second architecture being simpler than the 1st one (only 32 nodes in the 1st layer of the State Values branch versus 128 for the other architecture), it is the one retained.

Nonetheless, the performance of this dueling architecture is not better than the one of the vanilla DQN considered above. As after 500 episodes, the vanilla DQN achieves 14.84 of average score vs 14.8 for the dueling DQN. It may not be worth having a more complex model for this environment.

Prioritized experience replay

Looking at the performance of the agent over various episodes, it looks like some states causes the agent to oscillate without knowing what to do. This is the case when it is surrounded by blue bananas close to it and the yellow bananas are far away and uniformly distributed. The agent is not able to either turn around completely as seek for an easier state or to choose a way to some of the yellow bananas turning around the blue ones.

To try to solve for this problem, I implemented a prioritized experience replay.

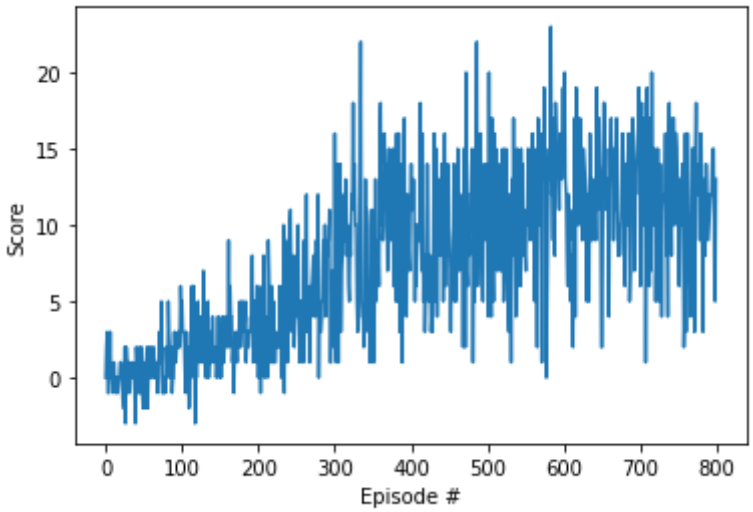
The learning time is visibly increased with this refinement. It is due to the TD error computation at every step (which needs to be stored in the experience replay buffer) on top of the same TD error computation done once every 4 steps for the learning part only without the prioritized experience replay.

The prioritized experience replay is controlled with an alpha parameter which governs the weighting given to the most promising experiences. An alpha of 0 is a pure uniform random sampling of experiences while an alpha of 1 means a random sampling weighted by the probabilities driven by the TD error of each experience. A beta parameter controls the importance sampling weighting. I started with a fixed beta equal to 0.

Also another epsilon parameter, different from the epsilon greedy policy one, is added to the probabilities derived from the TD error to avoid discarding completely experiences with no TD error. This epsilon was fixed to 1e-5.

The results with an alpha parameter at 0.5 and a beta of 0 show a much slower learning rate. After 500 episodes, the average score is only 9.42 compared to about 15 with the vanilla DQN and the Dueling DQN. The learning continues beyond but cannot reach 13 within 800 episodes (11.42 achieved over the last 100 episodes).

The evolution of the score is displayed below:



Further investigations

After these trainings, the agent collects efficiently bananas most of the time but some situations still cause issues as the agent does not seem able to choose what to do and ends up in an oscillation state which does not seem to end.

3 refinements could be then tested:

- Fine tune the alpha, beta and prioritized learning epsilon parameters to optimize the prioritized experience replay and potentially achieve a faster learning
- Fine tune the discount factor to see if it can push the agent to collect bananas that are close by first before leaving a location
- Implement a double DQN to test whether this further refinement toward the rainbow algorithm would make any significant change on this environment

Also a training of the agent on GPUs could certainly make sense as the learning process is taking more time as complexities are added to the algorithm.