Navigation project with DQ Network: Bananas (Reinforcement Learning First Project)

By: Adan Yusseff Domínguez Ruiz

1. Description of the problem:

For this project, an environment is created based on Unity environment. It is required to train an agent to navigate in a large, square world and collect yellow bananas, while avoiding blue bananas.

a. State Space

Space consists in 37 dimensions, including agent's velocity and ray-based perception of objects around agent's forward direction.

b. Action Space

Action space consists in the movements that the agent can perform to navigate in this world:

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

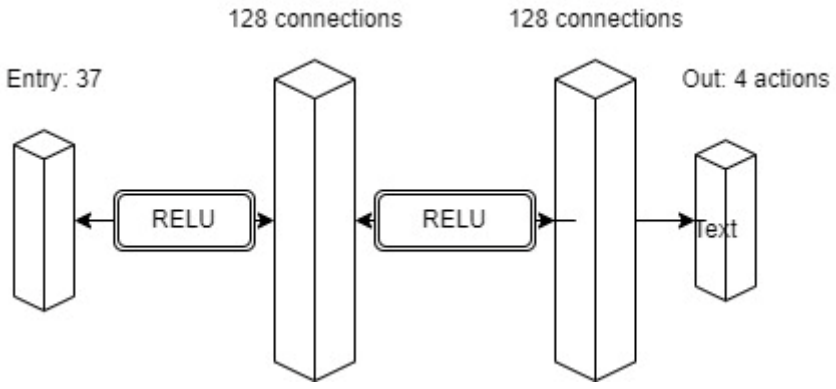
c. Rewards

Reward is simple, giving +1 each time a yellow banana is collected, while punishing with -1 when collecting a blue banana. The goal will be to collect as many yellow bananas as possible, while avoiding blue ones.

2. Algorithm Description

a. Neural Network Architecture

Based on the information reviewed during the first part of the course(Sutton and Barto, 1998), the neural network used consisted of only 3 fully connected layers. In this case, it was created with 128 connections each of them, as we can see in Figure 1.



3. Figure 1: QNetwork arrchitecture, entry = state_size, output= action_size

a. Learning Agent

For all Artificial Intelligence algorithms, an agent is required to interact with the environment, explore the options around and explode those actions that gives the most rewards.

Different hyperparameters need to be defined, these were chosen based on previous models shown by the course examples, they can be seen on Table 1.

Variable	Value
Buffer size of replay buffer	0.00001
Batch size for the sampling	64
Discount Factor (Gamma)	0.98
Soft update of parameters (TAU)	0.001
Learning rate	0.0001
Update step	4

Table 1: Hyperparameters

For the agent class, the following functions were defined:

i. Step function

The step function works as a method to keep experience from previous episodes on the training (via Replay buffer) and use them to learn better results.

ii. Act function

Act function performs the evaluation of the actual neural network architecture based on the environment observations that the agent is currently watching.

iii. Learn function

Learn function is one of the most important, since here is where the actual Q Learning process take effect. In the case for the current project, Double DQN is being used.

iv. Update function

This last function updates the values on the q-learning networks.

b. Double DQN

Double DQN improve the regular DQN algorithm on some weak points(van Hasselt, Guez and Silver, 2015). Values obtained by DQN expect that predicted values by the agents will always be better than the actual ones, leaving no room for improvement by the current actions gathered on the environment. Double DQN makes a copy of the model, to get the current best action based on the next state and use it to predict the next model and get the next Q values. These are then used on the Q Learning function already known:

$$Q(s,a) = r + gamma * Q(S_{+1}, a)$$

c. Replay Buffer

Replay buffer is used in the Q Learning process to store values from previous episodes as experiences used to learn in future steps(Mnih *et al.*, 2013).

d. Training Process

To train the neural network a set of steps are required:

i. Initialize scores
ii. Loop over the episodes from 1 to n+1

1. Reset environment
2. Get action based on current observations
3. Use the action to take a step and gather:
    a. Next state
    b. Reward
    c. Done value
4. Update the agent and take a step (including learning if the experience buffer has enough information)
5. Update state and score values
6. If average of score is higher than solution value (13), then the model is complete, otherwise, go to step 2.

3. Results

While attempting to use DQN algorithm, the reward values fluctuated from being close to 10 and going to negative values. Double DQN was used to speed up the learning process by using the 2 different network architectures to predict real reward values.

Model was trained in only 432 episodes as seen on Figure 2, however, when attempting to perform the actual testing with the prepared weights, the actions seen on the unity environment were mostly static, trying to get more than 13 points as score, but staying in same place to avoid hitting blue bananas.

```
Episode 100     Average Score: 0.98
Episode 200     Average Score: 4.35
Episode 300     Average Score: 7.69
Episode 400     Average Score: 10.19
Episode 500     Average Score: 12.64
Episode 532     Average Score: 13.00
Environment solved in 432 episodes!     Average Score: 13.00
```

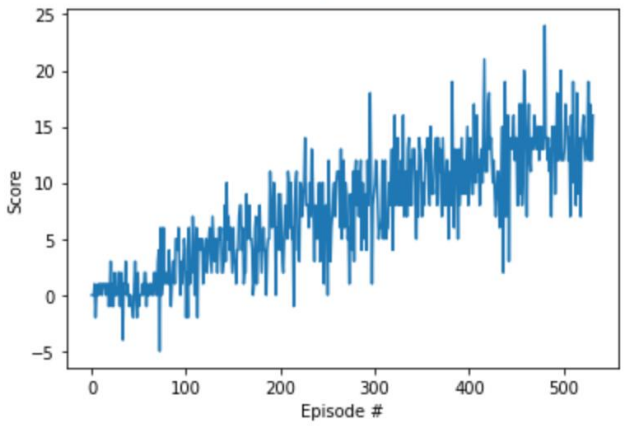


Figure 2: Results indicating the learning curve and number of episodes.

4. Future Work

As written in the results section, is possible to see on the unity testing environment that the agent tends to stay static to avoid picking up blue bananas, it will be good to modify values on the environment to add another penalty for time wasted and increase the necessity to get higher rewards.

An implementation of Dueling DQN and Prioritized Experience Replay could be an interesting point to see the improvement on the model.

5. Bibliography

van Hasselt, H., Guez, A. and Silver, D. (2015) 'Deep Reinforcement Learning with Double Q-learning', *arXiv:1509.06461 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1509.06461 (Accessed: 24 November 2021).

Mnih, V. *et al.* (2013) 'Playing Atari with Deep Reinforcement Learning', *arXiv:1312.5602 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1312.5602 (Accessed: 24 November 2021).

Sutton, R.S. and Barto, A.G. (1998) *Reinforcement learning: an introduction*. Cambridge, Mass: MIT Press (Adaptive computation and machine learning).