Navigation with Deep Reinforcement Learning

This report presents the description of the implementation of a deep reinforcement learning agent that solves an environment similar to the Unity's Banana Collector from the on the Unity ML-Agents GitHub page. The implementation uses Deep Q-Networks with Double DQN and Dueling DQN.

The goal of the agent is to collect as many yellow bananas as possible while avoiding blue bananas. A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the 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 environment is considered solved when the average score over 100 episodes overreaches 13.

1 Deep Reinforcement Learning Algorithm

The navigation with deep reinforcement project initially uses the basic deep q-networks algorithm.

The idea is to approximate the Q function with a deep neural network F with weights w:

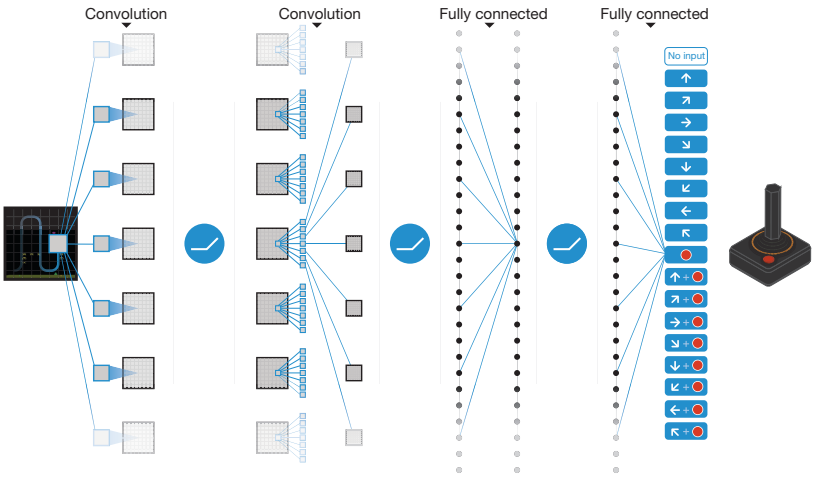
$$F(s,a;w) \approx Q(s,a).$$

Deep q-network algorithm extracted from Human-level control trough deep reinforcement learning paper.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

The architecture of the neural network used by the deep q-network algorithm is showed in the next graph.

Neural network architecture used for deep-networks, extracted from Human-level control trough deep reinforcement learning paper.



1.1 Improvements to the basic q-network algorithm

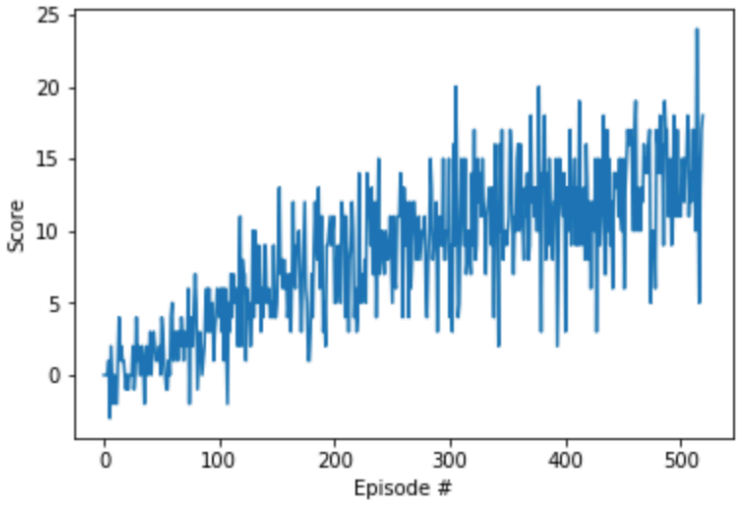
The first improvement adopted by the project is the Double Q-Learning algorithm. The basic q-network algorithm tends to overestimate the Q function, one way to prevent that is decoupling the selection from the evaluation; for each update, one set of weights is used to determine the greedy policy and the other to determine its value.

The second improvement adopted by the project is the Dueling DQN, this approach consists on the modification of the structure of the neural networks. Instead of simply stacking together fully connected layers, adds a fork as the penultimate layer. This fork contains two layers, one having an output of dimension one and the other a number of dimensions corresponding to the dimension of the action space. The output of the neural networks is the sum of these two layers.

2 Plot of Rewards

In this project, the agent received and average of 13 rewards for 420 episodes using the following parameters (eps_start=1.0, eps_end=0.05, eps_decay=0.995).

The next graph presents the evolution of the scores to solve the problem. Both double DQN and dueling DQN algorithm are implemented in the solution.



3 Ideas for Future Work

Some improvements to be done to the actual project are the following.

- Fine tuning of the model hyperparameters.
- Try other neural network architectures and optimizers.
- Implement prioritized memory replay algorithm to permit to the agent learn more effectively from some transitions than from others where the more important transitions should be sampled with higher probability.