Solving the Banana Navigation Problem through Deep Reinforcement Learning

Alessandro Leite

1 Context

An important concept in machine learning comprises the tasks of deciding from experience the sequence of action to perform in an uncertain environment to achieve some objective [1]. Inspired by behavioral psychology [2], reinforcement learning (RL) proposes a formal framework in which an agent learns by iterating with an environment. Based on experiences gathered by this iteration, the agent learns to optimize some objectives represented in the form of cumulative rewards. Thus, reinforcement learning focuses on the problem of learning a behavioral policy, a mapping from states to actions, which maximizes cumulative long-term rewards [2]. A policy can be represented as a lookup table, listing the appropriate action from any state. Therefore, in a rich environment, this approach is infeasible, and the policy must be encoded as a parametrized function.

Over the last years, RL has achieved some breakthroughs by integrating deep learning techniques. This combination, named deep reinforcement learning (DRL) is appropriate for problems with high dimensional spaces. As a result, deep reinforcement learning can learn different levels of abstractions from the data, which helps it to be successfully employed in complicated tasks with lower prior knowledge.

2 Goal

This project aims to design and implement an intelligent agent that can learn to navigate in a continuous environment, named Banana collector. Likewise, it must collect as much as possible yellow bananas, while avoiding the blue ones. The state space comprises 37 dimensions, agent's velocity along with the ray-based perception of objects around its forward direction. The discrete actions are move forward, move backward, turn left and right. The task is episodic and in a successful solution, the agent must score at least +13 points over 100 consecutive episodes. In this context, the agent must learn how to select the best actions with a reward of +1 for collecting a yellow banana and -1 for the blue ones.

3 Method

The proposed solution relies on Deep Q-Network (DQN) algorithm proposed by Mnih et al. [3]. DQN is a value-based temporal difference (TD) algorithm that approximates the Q-function. The learned Q-function is the used by the agent to select the actions. Thus, the goal of the agent comprises in selecting actions in a way that maximizes cumulative future reward. Formally, DQN uses convolutional neural networks to approximate the optimal action-value function [3]:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi],$$

which represents the maximum sum of rewards r_t discounted by γ at each time step t, achievable by a behavior policy $\pi = P(a|s)$, after observed (s) and took the action (a). DQN uses experience replay and an iterative action-values (Q) updates to handle the instability of reinforcement learning when a neural network is used to represent the action-value function. Experience replay randomizes the data to remove correlations in a observed sequence and to smooth the changes in the data distribution. The the value function $Q(s, a; \theta_i)$ is parametrized by the weights θ_i of a convolutional neural network. Thus, for each time step t the agent's experience (e_t) is stored in a fixed replay memory $D_t = \{e_1, \dots, e_t\}$. During learning, Q-learning updates mini-batches of experiences $(s, a, r, s') \sim U(D)$ drawn uniformly at random from the set of stores examples. Equation (1) shows the loss function used by the Q-learning at iteration i.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)}[(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a, \theta_i))^2] \quad (1)$$

where γ is the discount factor determining how the agent's horizon, θ_i are the parameters of the Q-network at iteration i, and θ_i^- are the network parameters used to compute the target at iteration i.

Deep Q-learning tends to overestimate the action values [4] as it *max* operator uses the same value to select and evaluate an action. Double DQN (DDQN) [4] handles this issue by decomposing the action selection and the action evaluation by using the target network to evaluate the action chosen by the local network as depicted in Equation (2).

$$Y_t = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-) \quad (2)$$

where, Y_t is the target value. This approach is similar to have two separates function approximators that have to agree on the best action. Moreover, it prevents the algorithm for propagating incidental high-rewards that may have been obtained by change and that don't reflect long-terms rewards.

4 Results

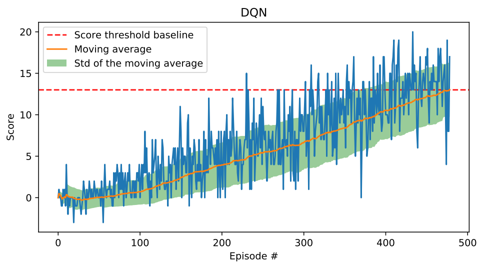
The architecture of the deep Q-network comprises an input vector of dimension 37 x 1 produced by the environment, a fully-connected linear input layer with 64 units and ReLU activation function, a hidden layer with the same characteristics of the input layer, and an fully-connected linear output layer with a single output for each action. Different from the original implementation that clones the current network to the target one, this work employs a "soft-update" to make the target network closer to the $Q(s, a; \theta)$ using $\theta^- = \tau\theta + (1 - \tau)\theta^-$. This change guarantees that the target network is always different from the local one. Table 1 illustrates the values of the parameters of the network.

Figure 1 shows the performance of DQN and DDQN algorithms when solving the navigator banana collector task. The Double DQN (Figure 1b) algorithm required 488 episodes to solve the task, whereas DQN (Figure 1a) required 479 episodes.

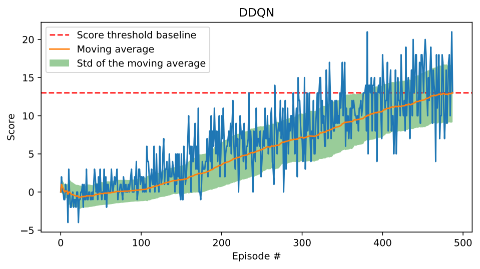
A future improvement includes the use of prioritized experience replay [5] as not all the stored experiences have the same importance. Likewise, important experiences might rarely occur, and thus, they have small chances to be selected. Furthermore, since buffers have limited capacity, important experiences may get lost.

Table 1: Hyperparameters' values

Interpolation	**buffer size**	**mini-batch size**	**discount factor**	**learning rate**
$\tau = 0.001$	100,000	64	$\gamma = 0.99$	$\alpha = 0.0005$



(a) DQN



(b) DDQN

Figure 1: Learning curves of DQN and DDQN algorithms

References

[1] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Foundations and Trends in Machine Learning*, vol. 11, no. 3–4, pp. 219–354, 2018.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 2nd ed. MIT press, 2018.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb 2015.

[4] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[5] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint*, 2015.