Navigation project (Bananas)

as part of UDACITY: Deep Reinforcement Learning Nanodegree Program

Ivan Masmitja

***Abstract*—This report describes the basic ideas behind the project number 1 (aka Bananas) conducted as part of the UDACITY nanodegree called: Deep Reinforcement Learning Nanodegree Program, were a Deep Q-Learning is used to train an agent in order to collect yellow bananas.**

Keywords—bananas, deep Q-Learning, reinforcement learning, deep neural network, artificial intelligence.

I. Introduction

For this project, we trained 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 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 task is episodic, and in order to solve the environment, your agent must get an average score of +13 over 100 consecutive episodes.

II. Implementation

The implementation is based with the work conducted in [1]. However, as it is known, the Deep Q-Learning algorithm represents the optimal action-value function q_* as a neural network (instead of a table).

Unfortunately, reinforcement learning is notoriously unstable [2] when neural networks are used to represent the action values. Here, the Deep Q-Learning algorithm implemented has addressed these instabilities by using two key features:

A. Experience Replay

When the agent interacts with the environment, the sequence of experience tuples can be highly correlated. The naive Q-learning algorithm that learns from each of these experience tuples in sequential order runs the risk of getting swayed by the effects of this correlation. By instead keeping track of a replay buffer and using experience replay to sample from the buffer at random, we can prevent action values from oscillating or diverging catastrophically.

The replay buffer contains a collection of experience tuples (S, A, R, S'). The tuples are gradually added to the buffer as we are interacting with the environment.

The act of sampling a small batch of tuples from the replay buffer in order to learn is known as experience replay. In addition to breaking harmful correlations, experience replay allows us to learn more from individual tuples multiple times, recall rare occurrences, and in general make better use of our experience.

B. Fixed Q-Targets

In Q-Learning, we update a guess with a guess, and this can potentially lead to harmful correlations. To avoid this, we can update the parameters w in the network $\hat{q}$ to better approximate the action value corresponding to state S and action A with the following update rule:

$$\Delta w = \alpha \overbrace{\underbrace{(R + \gamma \max_a \hat{q}(S', a, w^-)}_{TD\ target} - \underbrace{\hat{q}(S', A, w))}_{old\ value}}^{TD\ error} \nabla_w \hat{q}(S, A, w)$$

where w^- are the weights of a separate target network that are not changed during the learning step, and (S, A, R, S') is an experience tuple.

Moreover, three Deep Q-Network (DQN) improvements [3] has been utilized:

A. Double DQN (DDQN)

Deep Q-Learning tends to overestimate action values [2]. Double Q-Learning [4] has been shown to work well in practice to help with this.

B. Prioritized Experience Replay

Deep Q-Learning samples experience transitions *uniformly* from a replay memory. Prioritized Experienced Replay (PER) [5] is based on the idea that the agent can learn more effectively from some transitions than from others, and the more important transitions should be sampled with higher probability.

C. *Dueling DQN*

Currently, in order to determine which states are (or are not) valuable, we have to estimate the corresponding action values for each action. However, by replacing the traditional Deep Q-Network (DQN) architecture with a dueling architecture [6], we can assess the value of each state, without having to learn the effect of each action.

III. RESULTS

The above model proved to be very good at solving this problem. It achieved an average score above the score threshold of +13 over 100 consecutive episodes as can be observed in Fig 1, and summarized below.

Episode 100	Average Score: 0.52
Episode 200	Average Score: 3.82
Episode 300	Average Score: 6.49
Episode 400	Average Score: 10.57
Episode 500	Average Score: 14.66
Episode 600	Average Score: 15.30
Episode 700	Average Score: 16.16
Episode 800	Average Score: 16.93
Episode 900	Average Score: 16.41
Episode 1000	Average Score: 17.27
Episode 1100	Average Score: 17.10
Episode 1200	Average Score: 17.16
Episode 1300	Average Score: 17.46
Episode 1400	Average Score: 17.27
Episode 1500	Average Score: 16.87
Episode 1600	Average Score: 17.01
Episode 1700	Average Score: 17.34
Episode 1800	Average Score: 16.73
Episode 1900	Average Score: 17.25
Episode 2000	Average Score: 17.19

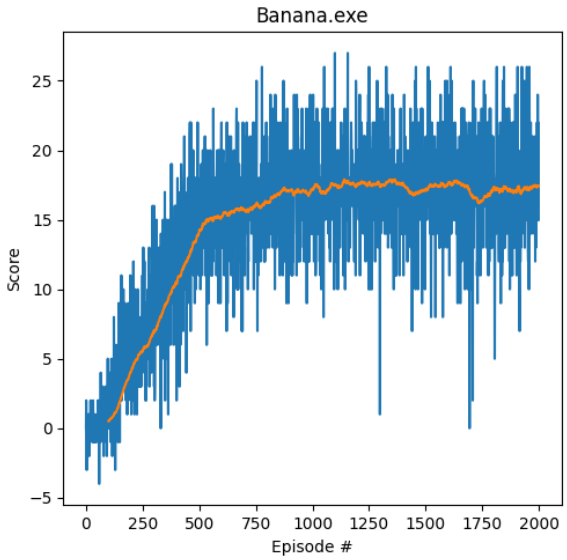

Fig. 1. Scores evolution over episode for Bananas project. Red line indicates the average values over 100 iterations.

IV. FUTURE WORK

As a future work, I will implement the full DQN Rainbow algorithm.

So far, we have implemented three extensions to the Deep Q-Networks (DQN) algorithm:

- Double DQN (DDQN)
- Prioritized Experience Replay
- Dueling DQN

But these aren't the only extensions to the DQN algorithm. Many more extensions have been proposed, including:

- Learning from multi-step bootstrap targets (as in A3C)
- Distributional DQN
- Noisy DQN

Each of the six extensions address a different issue with the original DQN algorithm.

Researchers at Google DeepMind recently tested the performance of an agent that incorporated all six of these modifications. The corresponding algorithm was termed Rainbow [3]. It outperforms each of the individual modifications and achieves state-of-the-art performance on Atari 2600 games.

Finally, this algorithms will be implemented to obtain the optimal trajectory for an underwater autonomous vehicle in order to localize and track an underwater target [7]–[9].

REFERENCES

[1] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, doi: 10.1038/nature14236.

[2] S. Thrun and A. Schwartz, “Issues in Using Function Approximation for Reinforcement Learning,” *Proc. 4th Connect. Model. Summer Sch. Hillsdale, NJ. Lawrence Erlbaum*, pp. 1–9, 1993.

[3] M. Hessel *et al.*, “Rainbow: Combining improvements in deep reinforcement learning,” *32nd AAAI Conf. Artif. Intell. AAAI 2018*, pp. 3215–3222, 2018.

[4] Y. Xiao, J. Hoffman, T. Xia, and C. Amato, “Learning Multi-Robot Decentralized Macro-Action-Based Policies via a Centralized Q-Net,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 10695–10701, 2020, doi: 10.1109/ICRA40945.2020.9196684.

[5] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, pp. 1–21, 2016.

[6] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Frcitas, “Dueling Network Architectures for Deep Reinforcement Learning,” *33rd Int. Conf. Mach. Learn. ICML 2016*, vol. 4, no. 9, pp. 2939–2947, 2016.

[7] I. Masmitja *et al.*, “Optimal path shape for range-only underwater target localization using a Wave Glider,” *Int. J. Rob. Res.*, vol. 37, no. 12, pp. 1447–1462, 2018, doi: 10.1177/0278364918802351.

[8] I. Masmitja *et al.*, “Range-Only Single-Beacon Tracking of Underwater Targets from an Autonomous Vehicle: From Theory to Practice,” *IEEE Access*, vol. 7, pp. 86946–86963, 2019, doi: 10.1109/ACCESS.2019.2924722.

[9] I. Masmitja *et al.*, “Mobile robotic platforms for the acoustic tracking of deep-sea demersal fishery resources,” *Sci. Robot.*, vol. 5, no. eabc3701, 2020.