Navigation Using Deep Reinforcement Learning

October 2018

1 Introduction

Reinforcement learning is computational approach to learning from interaction with environment in order to maximizing a numerical reward signal [3].

For this project, I trained an agent to navigate and collect bananas in a large, square world, using Unity Machine Learning Agents Toolkit[1], which is an open-source Unity plugin that enables games and simulations to serve as environments for training intelligent agents. I used a Deep Q-learning and Double Deep Q-learning algorithms.

In this specific environment an agent should collect as many yellow bananas as possible while avoiding blue bananas. For collecting a yellow banana is provided a reward of +1 reward, for collecting a blue banana is provided a reward of -1.

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, the agent must get an average score of +13 over 100 consecutive episodes.

The first algorithm used was Deep Q-Learning (DQN) [2], which is combines Q-learning algorithm [5] with a deep neural network, that method was tested on a varied and large set of deterministic Atari 2600 games, reaching human-level performance on many games. This algorithm use a neural network to approximate the optimal action-value function and learn successful policies directly from high-dimensional sensory inputs using end-to-end reinforcement learning.

[1] `https://github.com/Unity-Technologies/ml-agents`

The standard Q-learning update for the parameters after taking action A_t in state S_t and observing the immediate reward R_{t+1} and resulting state S_{t+1} is then

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(Y_t^Q - Q(S_t, A_t; \boldsymbol{\theta}_t)\nabla Q(S_t, A_t; \boldsymbol{\theta}_t) \tag{1}$$

where α is a scalar step size, $Q(s, a; \theta_t)$ is value function, and the target Y_t^Q is defined as

$$Y_t^Q = R_{t+1} + \gamma \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t) \tag{2}$$

The authors proposed two modifications of standard Q-learning: experience replay and target network. The experience replay, observed transitions are stored for some time and sampled uniformly from this memory bank to update the network. For the target network, with parameters $\boldsymbol{\theta}^-$, is the same as the online network except that its parameters are copied every τ steps from the online network, so that then $\boldsymbol{\theta}_t^- = \boldsymbol{\theta}_t$, and kept fixed on all other steps. The target used by DQN is then

$$Y_t^{DQN} = R_{t+1} + \gamma \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t^-) \tag{3}$$

Also I used a Double Deep Q-Learning (Double DQN), the authors proposed to use the online network to select the actions and the target network to evaluate, this solution handles with the problem of the overestimation of Q-values, this problem is caused because the Deep Q-Learning algorithm uses the same network to select and to evaluate an action.

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}_t^-) \tag{4}$$

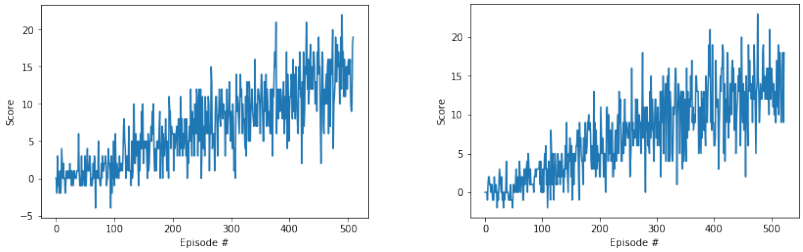
All tests in this project I used a buffer size of 100,000 experiences, a mini-batch size of 64, $\lambda = 0.99$, the learning rate of 0.0005 and updated the Q-networks every four steps. Both local and target Q-networks have same architecture: a input layer which receives a state vector $s \in \mathbb{R}^{37x1}$ produced by the environment, two fully-connected hidden layers, each with 64 neurons and using relu activation, and a fully-connected linear layer as output layer.

The Figure 1 presents the results of the simulation using DQN and Double DQN algorithms, surprisingly a DQN algorithm was able to solve the environment in less episodes than Double DQN algorithm.

There are some possibles extensions: an exploration of the various hyperparameters, Dueling network [4], Prioritized Experience Replay, and Rainbow [1], that combines different improvements in Deep Reinforcement Learning, including the models presented in this project.

References

[1] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David



(a) Learning Curve using DQN (b) Double DQN

Figure 1: 2 Figures side by side

Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[3] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

[4] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[5] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.