Banana Navigator with Deep Reinforcement Learning

Ismail Geles

April 2020

Abstract

To solve the first project of the Udacity Deep Reinforcement Learning Nanodegree, a Deep Q-Network with the details in this report was used to let an agent navigate in an Unity environment.

1 The Environment

The agent to train navigates in an Unity environment consisting of a large square world with the goal to collect as many yellow bananas as possible whilst avoiding the blue bananas.

1.1 States and Actions

The state space consists of 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.
The four discrete actions available to the agent are:

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

The task is episodic and is solved when the agent reaches a score of +13 over 100 consecutive episodes.

2 Implementation of the Deep Q-Network

The code for the Deep Q-Network (DQN) was modified from the first DQN task of the Nanodegree. The agent was trained using a neural network as value function approximator and a target network for the fixed Q-Values as described in the paper of Google-Deepmind by Mnih et al (2015).
The expected reward was discounted and the actions taken were based on a epsilon-greedy policy. A experience buffer was used to sample batches and learn from past experiences.

2.1 Network Architecture

The model, used to solve the environment, has two fully connected hidden layers with 64 units each.
So to summarize the layer structure (in `model.py`):

37 units (input) - 64 units (1st hidden) - 64 units (2nd hidden) - 4 units (output)

2.2 Network Hyper-parameters

Following hyper-parameters were used for training:

Parameter	Value
buffer size	1e5
batch size	64
gamma (discount factor)	0.99
tau (soft update interpolation)	1e-3
learning rate	5e-4
update rate	4
episodes	1000
max timesteps per episode	1000
epsilon start	1.0
epsilon end	0.01
epsilon decay	0.95

3 Training Results

The rewards over the number of episodes can be found below. Notice that the environment was solved around 400 episodes (solved means a score of +13 over 100 consecutive episodes). However for the sake of performance improvement, the agent was trained until 1000 episodes, which led to significant improvements.

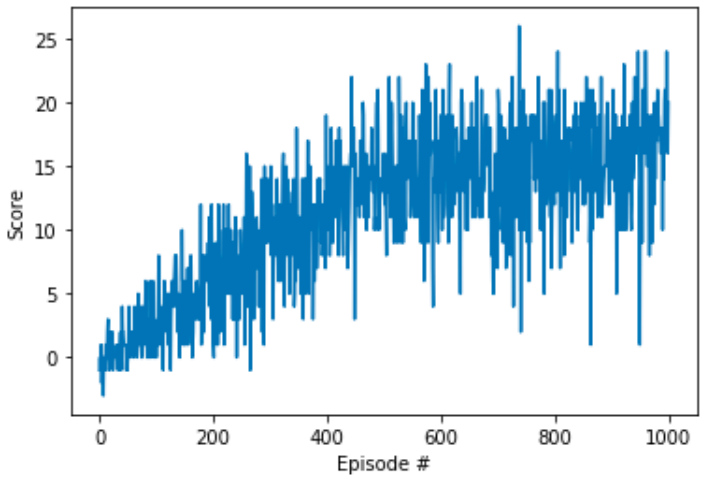

Figure 1: Scores over episodes during training.

4 Ideas for Future Work

Another approach to tackle this problem would be learning directly from the pixels of the environment. This would need some convolutional layers as described in the initial paper of DQN (by Mnih et al (2015)). Moreover, it is an important aspect to further tune the hyper-parameters to reach better performance. One could also aim for improved performance by using optimized variants of Deep Q-Networks (for example, Dueling-DQN, Double-DQN, Noisy-DQN,...).