Piotr Szpoton
pszpoton@gmail.com

Deep Reinforcement Learning Navigation

25th Sep 2018

OVERVIEW

This is a solution for Udacity Deep Reinforcement learning course to train an agent to navigate and collect yellow bananas in a large 3D scene. By picking yellow banana an agent gets +1 point on the contrary to a blue one when -1 points is applied. The project contains the code to train and evaluate the agent to score at least 13 points on average over 100 consecutive episodes.

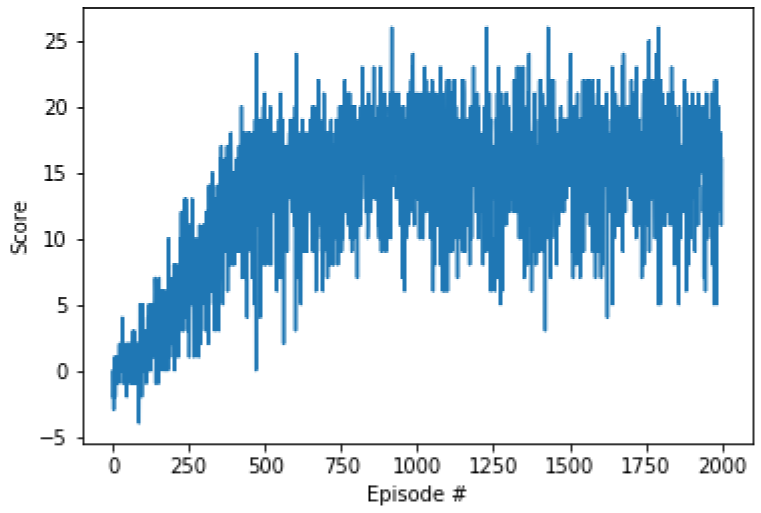
LEARNING ALGORITHM

I implemented Depp Q learning algorithm following the guidelines in DQN paper and reusing parts of the code from Udacity exercise implementation. To eliminate correlations between consecutive actions experience replay was implemented with buffer size of 10^5 elements and sampling batch size of 64. To improve stability of NN fixed Q-Target algorithm was used so there are two networks local and the target one. After every four steps training is applied to compute and backpropagate the gradients between two networks. Both nets have the same structure and were implemented in pytorch framework. They consists of three fully connected linear layers:

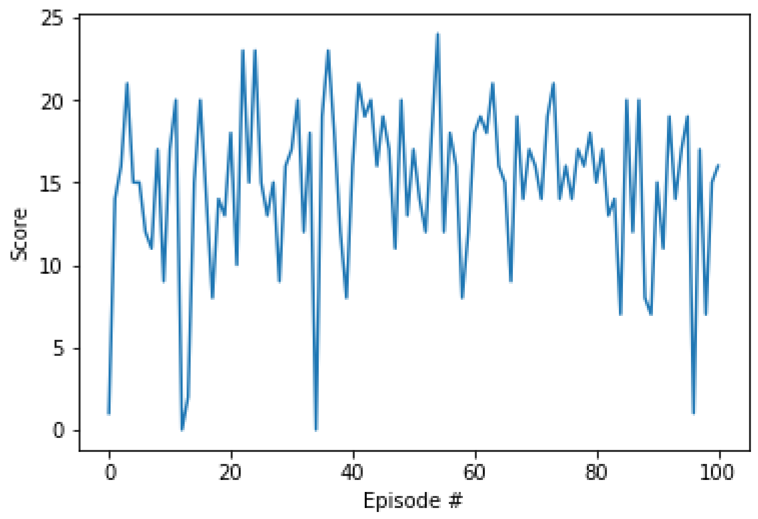
- Layer 1: input os state size of length 37 output 64
- Layer 2: Input 64 output 64
- Layer 3: Input 64 output of action set length 4

Learning algorithm computes Q target value from the current state by using gamma parameter of 0.99 and during update state weights from local network are taken with Tau value of 10^-3 and from target network 1-Tau. The learning session was performed on 2000 episodes each containing of 1000 actions steps. Exploration is regulated using epslion parameter that start from 1 with decay of 0.995 over each episode and with minimal value of 0.01

LEARNING CURVE



CHART



Episode 100	Average Score: 14.81

The trained agent was run over 100 episodes and received on average 14.81 score which is more than required 13 to complete the exercise.

CONCLUSIONS

The algorithm presented nice performance as a proof of work. The next steps would be to implement convolutional layers so that it can be trained to read state data directly from the screen in contrary to current tensor value. One can also observe that for some specific episodes the agent underperformed significantly by preferring to move along the walls. To improve the algorithm we could implement double DQN so that to avoid action overestimation. Also much longer training episodes would be preferable and implement Dueling DQN which also improve performance of the training over each episode. To increase even further we could prioritized experience replay by adding weights to each particular action in a buffer so that we may distinguish between valuable and irrelevant ones. One could also experiment with hyperparameters tuning to check how the performance is improved.