DRL - Project 1 - Navigation

Christian Eberhardt

November 2019

1 Introduction

The goal of this project is to train an agent to collect yellow bananas and avoid blue bananas in a large square world. The environment is a Unity-environment. See the file **README.md** for a description of the environment and the state and action space.
The solution to the project has been achieved by using the solution for the DQN exercise in the Udacity Deep Reinforcement Learning Nanodegree with tweaks to the function that trains the agent in the environment, so the function is compatible with the Unity environment.

2 Learning Algorithm

Deep Q-Learning has been used to solve this task. This learning algorithm was chosen in order to accommodate the continuous state space. In a continuous state space the amount of states are infinite and the process does therefore not have the Markov property. As with normal Q-learning it is not possible to construct a Q-table that has a Q-value for every state-action pair, when there is infinite states or actions. Therefore the Q-value for each of the available actions is approximated using a function-approximator. The function approximator in this case is a neural network.
To mitigate the non-linearity of the function approximator which can cause the algorithm to oscillate, two techniques have been implemented: Experience replay and fixed Q-targets.
With experience replay the algorithm trains for a while in this case for 4 time-steps without learning. However at each time-step, an experience tuple (state, action, reward, next_state) is stored in a memory buffer. When the agent learns it samples a small batch of random experience tuples from the buffer and learns from these. This removes the correlation that can be in a consecutive sequence of experience tuples minimising oscillations.
With fixed Q-targets the target Q-networks weights is updated less regularly than the primary Q-network weights. This method helps minimise oscillations as the loss is not computed based on a constantly moving target.

2.1 Neural Networks

The agent uses two neural networks. One network is for predicting the best action given a state input. The other network is a target network that is used to update the weights of the first network based on a state-action-reward-pair at each time-step in the episode.

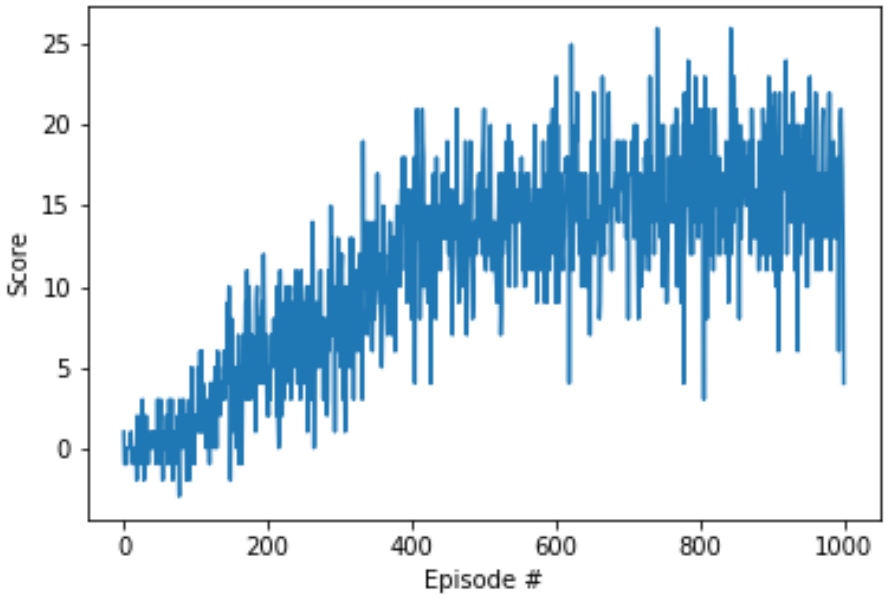

Figure 1: Plot of scores for every episode the agent spent training in the environment.

The two neural networks have identical architectures. The architecture consists of an input layer consisting of 37 inputs and 50 outputs corresponding to the state space of 37 dimensions and hidden layer with 50 inputs and outputs and an output layer with 50 inputs and 4 outputs corresponding to the discrete action space.
The activation function used in all the perceptrons in the different layers are ReLU (Linear Rectified Units) activation functions.

3 Plot of Rewards

On figure 1 the score for every episode has been plotted. As can be seen on the figure, the score for the agent is different from episode to episode but the underlying trend is that the score increases, the more time the agent has spent training. By around episode 470-520 the agent has achieved an average score of at least 13.0 points for the last 100 consecutive episodes. Therefore it is assumed that the agent accomplished the goal by episode 370-420. However as can be seen on the figure the agent improves its performance even more and seems to converge towards the optimal policy around the 800th episode mark

4 Ideas for Future Work

By tweaking the current hyperparameters, hidden layer size, amount of hidden layers, learning-rate, experience batch size, etc it might be possible to reduce the agents training time even further.
Another idea is to adapt the code to a gazebo-environment where a differential drive robot with a control abstraction is implemented (so has same controls; up, down, left and right). In this gazebo-environment the task is to collect marbles that are randomly distributed in a mazelike-map, see

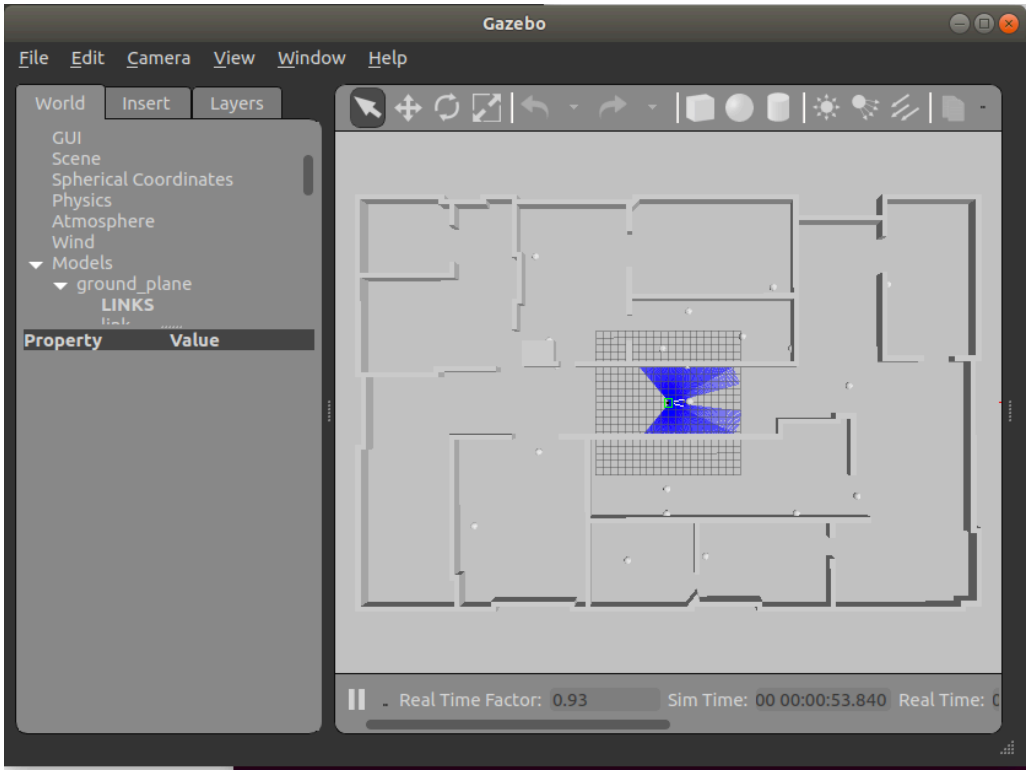

Figure 2: Gazebo-environment where a model of a differential-drive robot roams a map separated into rooms by walls searching for marbles. The environment has many similarities to the Unity-environment. The robot has an abstraction with the same controls; up, down, left, and right as well as ray-based information from a LIDAR and knowledge about its position on the map. However the environment is significantly larger than the Unity-environment and the marbles are not as densely packed as the bananas in the Unity-environment, meaning less opportunity for reward, so it would be interesting to see how well the agent would perform in this environment.

figure 2.