Project Report: Project 1 - Navigation

Christian Michler

February 26, 2019

1 Introduction

Reinforcement Learning deals with sequential decision making in an environment of which the model is unknown; cf. the text book of [1]. At each time step, an agent observes the environment's state and takes an action based on the observation. In response to its action, the environment provides the agent with a numerical reward. By interacting with the environment, the agent has to learn the optimal policy which tells the agent what action to take under what circumstances such as to maximize anticipated future reward. In [2], a deep Q-network (DQN) has been introduced, approximating the Q-function by a deep neural network. Several modifications of DQN have been proposed such as Double DQN [3], Dueling DQN [4] and Prioritized Experience Replay [5].

The purpose of this project is to train and compare different Reinforcement Learning Agents using the Banana Collector Environment. The goal of the exercise is to train an agent via Reinforcement Learning to navigate a large, square world and collect yellow bananas while avoiding blue ones. This project is part of Udacity's Nanodegree on Deep Reinforcement Learning [6].

The exercise uses the Unity ML Agents Toolkit [7] to train and evaluate various agents. In particular, I will assess Agents based on DQN [2], Double DQN [3] and Dueling DQN [4].

The outline of this report is as follows: Section 2 describes the methods / agents that were assessed. Section 3 first describes the problem setting of the banana collector environment in more detail. This section also contains the numerical results obtained using this environment and discusses the findings. Section 4 discusses some improvements and future work.

2 Methods

The Agents considered in this study all use the following architecture of the deep Q network: All layers of the network are fully connected linear layers and use a ReLU. The input consists of a vector of 37 observations given by the environment. The first hidden layer has 64 nodes and is followed by a second hidden layer of also 64 nodes. The output layer has four nodes, one node for each of the four actions. This architecture is used for both target and local Q networks.

I used a replay buffer size of 10^5, a training mini-batch size of 64, a discount factor $\gamma = 0.99$ a $\tau = 10^{-3}$ in the soft update of target parameters, a learning rate of $5 \cdot 10^{-4}$ and updated the Q-network every 4 steps.

Below I briefly explain the DQN, Double DQN and Dueling DQN Agents that were used subsequently in this study.

2.1 DQN

Deep Q-Learning in the form of a Deep Q-network (DQN) was introduced in [2]. In DQN, a deep neural network is used to approximate the action-value function Q. To this end, the network weights are determined such that the neural network best approximates the Q function, effectively learning the optimal policy that maximizes expected future reward. In this project, the neural network is implemented in `model.py`.

The DQN implementation uses Experience Replay, i.e. rather than learning from sequential batches of experience tuples directly, these batches are stored in a replay buffer from which the agent then randomly samples. This mitigates the risk of potential correlations in a sequence of experience tuples and, moreover, enables the agent to learn from individual tuples multiple times, recall rare occurrences, and in general make better use of the generated samples. Experience Replay is implemented in `dqn_agent.py`.

2.2 Double DQN

Deep Q-Learning has a tendency to overestimate the action values, see [8] for details, since it uses the same Q values to select and evaluate an action. An elegant way of addressing this issue is by means of Double Q-Learning proposed in [3] in which the target network evaluates the action chosen by the local network. This is implemented in the `learn` function of `dqn_agent.py`.

2.3 Dueling DQN

The motivation behind Dueling DQNs [4] is that for many states it is not necessary to estimate the value of each action choice. Therefore, Dueling DQNs decompose action-value function Q as the sum of the value of being at that state plus the advantage of taking an action at that state. When subsequently combining these two streams the advantage mean needs to be subtracted to maintain identifiability; for details see [4]. This is implemented in the `forward` function of `model.py`.

3 Results and Discussion

In the Banana Collector environment, 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 the 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: move forward, move backward, turn left, 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 (moving average).

Figs. 1-3 display the training of the DQN, Double DQN and Dueling DQN agents, respectively, and Fig. 4 compares their performance. While it appears that Double DQN performs slightly better, one needs to keep in mind that the number of episodes required to attain an average score of +13 can vary more over consecutive runs than the difference between the required number of episodes of the different agents. Also it is worth mentioning that, although Double DQN does not always improve performance, it substantially benefits the stability of learning. Note also the change in slope of the curves, in particular of the Double DQN curve. While in the first couple of hundred episodes it slightly underperforms the other two agents, it subsequently accelerates and outperforms them towards the end; see Fig. 4. This behaviour, however, has not been seen in general when repeating those runs.

I have also undertaken some parameter studies to change / increase the size of the neural network by adding more nodes to each layer, e.g. increasing the number of nodes from 64, 64, 4 in the respective layers to 128, 128, 4. However, I found that the results were inconclusive in that subsequent runs with the same parameters yield a different number of episodes required for completing the task, and I therefore defer this to future investigations, Section 4.

Agent	Number of episodes required
DQN	408
Double DQN	378
Dueling DQN	404

Table 1: Comparison of the different agents: Number of episodes -100 required for trailing-score-window mean to achieve +13.0

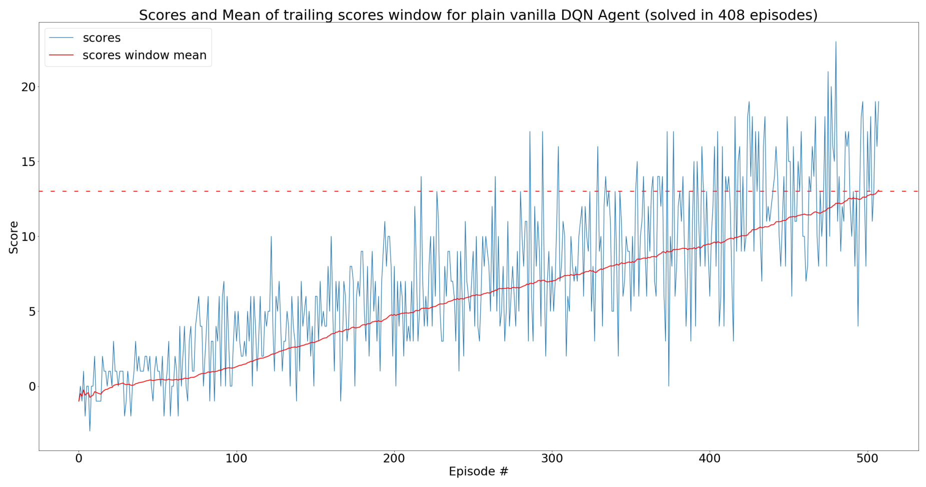

Figure 1: Scores and mean of trailing scores window for plain vanilla DQN Agent

4 Ideas for Future Work

Below I discuss some ideas for future work. Re-running the same agent with the same parameter settings may give slightly different results in terms of number of episodes required to reach an average score of +13 due to the randomness inherent in the environment and observations provided and also due to the agent's random sampling of experience tuples from the replay buffer. To properly compare the different agents and assess variations in different hyperparameters settings (such as the size of the fully connected layers, the learning rate and the epsilon for epsilon-greedy action selection) one should repeat the runs sufficiently many times and average the number of episodes required.

Next, I would try further improvements based on Prioritized Experience Replay [5] which samples the experiences in the replay buffer based on priorities related to the magnitude of the error. This may help with sampling rare but important experiences and has been shown to enhance performance; cf. [5]. Other possible improvements I would consider are NoisyNets [9] and Rainbow [10].

Finally, I would redo this exercise using pixels as input.

References

[1] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, Second Edition, MIT Press, Cambridge, MA, 2018.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg and Demis Hassabis, Human-level control through deep reinforcement learning, Nature (518), 529–533, 2015. `https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf`

[3] Hado van Hasselt, Arthur Guez and David Silver, Deep Reinforcement Learning with Double Q-learning, 2015. `https://arxiv.org/pdf/1509.06461.pdf`

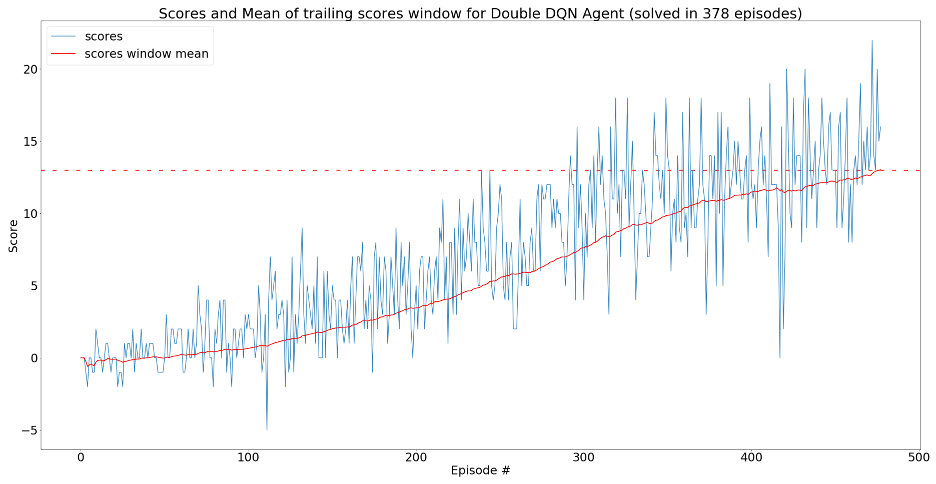

Figure 2: Scores and mean of trailing scores window for Double DQN Agent

[4] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot and Nando de Freitas, Dueling Network Architectures for Deep Reinforcement Learning, 2016. https://arxiv.org/pdf/1511.06581.pdf

[5] Tom Schaul, John Quan, Ioannis Antonoglou and David Silver, Prioritized Experience Replay, ICLR Conference paper (2016). https://arxiv.org/pdf/1511.05952.pdf

[6] https://eu.udacity.com/course/deep-reinforcement-learning-nanodegree--nd893

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

[8] Sebastian Thrun and Anton Schwartz, Issues in Using Function Approximation for Reinforcement Learning, Proceedings of the Fourth Connectionist Models Summer School (1993).

[9] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell and Shane Legg, Noisy Networks for Exploration, ICLR conference paper (2018). https://arxiv.org/pdf/1706.10295.pdf

[10] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar and David Silver, Rainbow: Combining Improvements in Deep Reinforcement Learning, 2017. https://arxiv.org/pdf/1710.02298.pdf

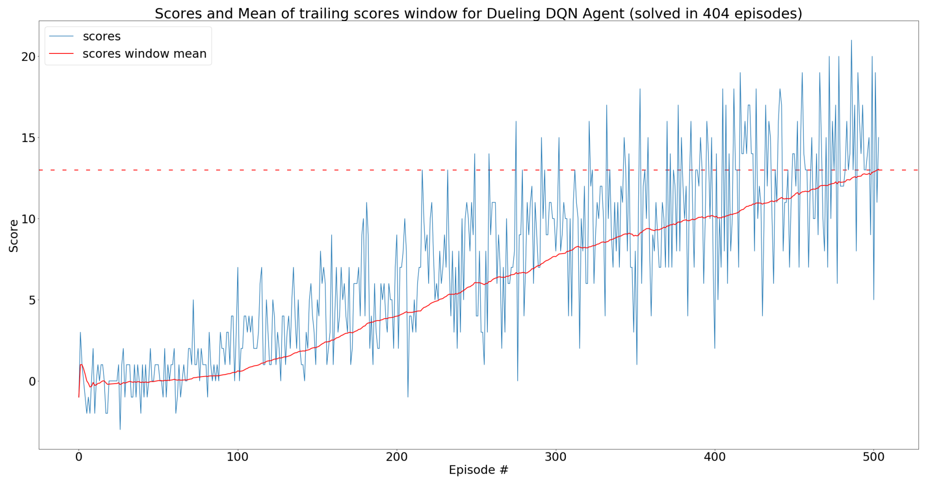

Figure 3: Scores and mean of trailing scores window for Dueling DQN Agent

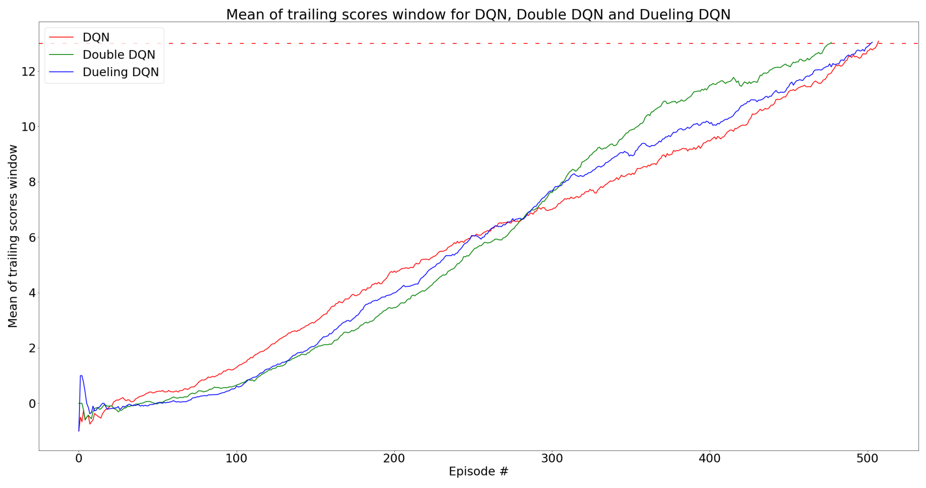

Figure 4: Comparison of DQN, Double DQN and Dueling DQN in terms of mean of trailing scores window