# Navigation project report Deep Q Learning

In the Jupyter Notebook Navigation.ipynb you can see how an agent learns to maximize a reward in the environment of the Navigation project of the Udacity Deep Reinforcement Nanodegree program.

In the notebook you find implementations of agents based on a Vanilla Deep Q Network (Vanilla DQN or simply DQN) and of a Dueling Deep Q Network (Dueling DQN). Dueling DQN is one of the proposed improvements of the Vanilla DQN algorithm (Wang et al. 2016).

Q learning is a form of Temporal Difference learning (TD), which uses the following control algorithm:

$Q(S_t, A_t) <- Q(S_t, A_t) + \alpha[R_{t+1} + \gamma maxQ(S_{t+1}, a) – Q(S_t, A_t)]$

In this formula Q is the action-value function, which is updated after every step of the agent. Q is a table with the possible states in the rows and the actions in the columns (or the other way around). If the state space becomes too big, such as is the case in the Bananas environment, it becomes impossible to use a Q-table for learning, because the number of states is too high. A solution for this problem is to use a neural network which approximates the Q-function (Mnih et al. 2015).

The Vanilla DQN network is implemented in the class QNetwork. It makes use of the Pytorch framework. The neural networks are initialized in the classes QNetwork and duelingQNetwork. The structure of both classes are similar. In the __init__ function the layers of the network are defined.
The QNetwork consists of three fully connected layers: an input layer (fc1), a hidden layer (fc2) and an output layer (fc3). In the forward function a forward pass through the network is made, using the ReLu activation function in the first two layers.

An way to stabilize learning is applied in Dueling DQN, of which you can find the implementation in the class duelingQNetwork in the notebook. On the Atari benchmarks, the dueling network brought more training stability, faster convergence and better results than the Vanilla DQN. In Dueling DQN Q(s, a) is decomposed in:

V(s), which is the Value of being in a certain state

and

A(s, a), the Advantage of taking a certain action in a state.

In the duelingQNetwork there is an input layer (fc1), which splits in two branches, the Advantage branch and the Value branch. Each branch consists of two layers. Finally, the output is calculated by adding the value and the advantage. Also, the mean of the advantage is subtracted, because we want the mean of the value of the advantage to be zero.
The given implementation does not limit the maximum number of steps in one episode, because the maximum number of steps is limited to 300 in the environment itself, so there is no risk of episodes that take forever to finish.

In DQN in which the agent learns from the raw pixels, in general the first layers of the network consist of

a number of convolutional layers, after which the network is split in a value and an advantage branch. In our case, the input of the model consists of 37 values on each time step, so the convolutional layers do not make sense here.

# Recommendations

Both agents solve the environment in about 500 episodes, but there is substantial variation between different runs of the function dqn(). Based on the results of Wang et al. (2016) one would expect that the Dueling DQN solves the environment faster than the Vanilla DQN network, but this does not happen. If we look at the plots in the notebook showing the scores of all the episodes of both networks, the Dueling DQN does not even learn in a more stable way than the Vanilla DQN network.
Which improvements can lead to solving the environment faster?
In the implementation I have chosen to use, the hyperparameters of both agents are identical. Of course, an improvement can be to tweak the hyperparameters of the agents (buffer_size, batch_size, the discount factor gamma, tau, the learning rate, and the frequency of updating the network). Also, the structure of the networks can be tweaked. One can increase the number of layers, or change the number of neurons in a layer.
One can also try to increase the speed with which the environment is solved by applying another extension of the Vanilla DQN, such as Double DQN or Experienced Replay.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Humanlevel control through deep reinforcement learning. Nature, 518 (7540):529–533, 2015.
Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M.,and De Freitas, N., Dueling Network Architectures for Deep Reinforcement Learning, Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.