Deep Reinforcement Learning Nanodegree
Project 1 – Navigation

We implement DQN from Deepmind's paper (https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf) to tackle this problem. We use a neural network as function approximator to estimate state action value. It is a rather simple network with following architecture where the input state has dimension of 37 and output is dimension of 4 which is the size of action space:
- fully connected layer (37x64 weights, Relu)
- fully connected layer (64x64 weights, Relu)
- fully connected layer (64x4 weights)

Two identical neural networks are used with different weights where one lag behind another. We implement experience replay by storing the experience in buffer. During training, we retrive the experience randomly, and use the following loss function (from Deepmind's paper) to to perform gradient descent optimization (we use the Adam variant)

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

where r is the reward, gamma is discount factor, theta_i is the neural network weight while theta_i_minus is the target neural network weights that only updated after a number of steps. In other words, in each backprop, theta_i will be updated but not theta_i_minus.

We use epsilon-greedy to select the policy. In other words, for probablity of 1-epsilon, we select the action that has highest score from neural network, with probability of epsilon, we select action from the rest of the state with uniform random sampling.

The code are:
Navigation.ipynb – jupyter notebook to run the training
dqn_agent.py – DQN agent
model.py – contain the neural network definition
model.pth – saved neural network model