**Deep Reinforcement Learning Nanodegree**

**Project 1 – Navigation**

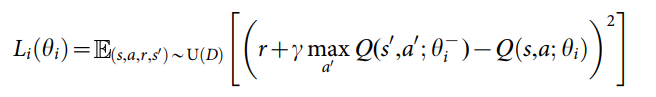
We implement DQN from Deepmind’s paper (<https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf>) to tackle this problem. We use a neural network as function approximator to estimate state action value. It is a rather simple network with following architecture where the input state has 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)

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