**Deep Reinforcement Learning Nanodegree**

**Project 1 – Navigation**

**Description**

We implement DQN from Deepmind’s paper (<https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf>) to tackle this problem. In essence, we use Q-learning to perform online reinforcement learning and use a neural network as function approximator to estimate state action value. In Q-learning, instead of waiting for entire episode to finish to get the total reward hence the Q value, we calculate the Q value using the estimation of Q-value from next state.

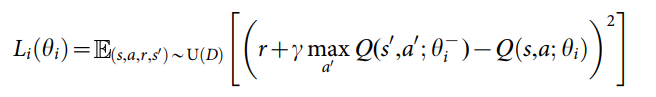
The neural network used in this project is rather simple network with following architecture where the input state has dimension of 37 and output is dimension of 4 which is the size of action space:

- 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.

**Hyper parameters**

Hyper parameters include neural network layers and neuron. The more layers and neurons then the more accurate is the prediction but will result in lower inference and training speed. The following are other hyperparmaters for the neural networks

BUFFER\_SIZE = int(1e5) # replay buffer size

BATCH\_SIZE = 64 # minibatch size, the bigger the better, limited by GPU size

GAMMA = 0.99 # discount factor

TAU = 1e-3 # for soft update of target parameters

LR = 5e-4 # learning rate

UPDATE\_EVERY = 4 # how often to update the network

For Q learning

max\_t = 1000: maximum number of timesteps per episode

eps\_start=1.0: starting value of epsilon, for epsilon-greedy action selection

eps\_end =0.01: minimum value of epsilon

eps\_decay=0.995 multiplicative factor (per episode) for decreasing epsilon

The code are:

Navigation.ipynb – jupyter notebook to run the training

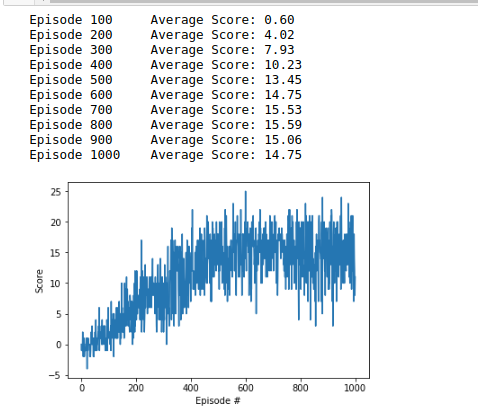
dqn\_agent.py – DQN agent

model.py – contain the neural network definition

model.pth – saved neural network model

**Result**

The desired score of +13 is achieved after 500 episode. The chart is shown in jupyter notebook and below.



**Ideas to improving agent’s performance**

Some approaches:

1. Double DQN to solve overestimation of Q-value by using two set of network weights that must agree on the best action.

2. Priotized Experience Replay. Instead of sampling experience replay randomly, we assign higher priority to experience that we want our algorithm to learn more from.

3. Dueling DQN. The network estimates two things – state values and advantage value which are combined to provude Q-values.