**DRLND - P1 - Navigation : Report**

In this report, I am presenting the environment and the algorithms that I used to solve the navigation problem where the agent is to collect yellow bananas.

**Environment**

The project uses an environment built in Unity. The environment contains ***brains*** which are responsible for deciding the actions of their associated agents. Below are the characteristics of the environment.

* Unity brain name: BananaBrain
* Vector Observation space type: continuous
* Vector Observation space size (per agent): 37
* Number of stacked Vector Observation: 1
* Vector Action space type: discrete
* Vector Action space size (per agent): 4

**Learning Algorithm**

**DQN**

To solve this reinforcement learning problem, I am using a Deep Q Network (DQN) as has been taught in the course. The agent uses replay buffer of length 10000 and a simple neural network. The agent uses **discount factor 0.99** and **learning rate 0.0005**

**Neural Network**

To approximate the value function a simple neural network with three fully connected layers with ReLU activation function, is used.

**Network Architecture**

State --> 64 --> ReLU --> 32 --> ReLU --> action

**Pytorch Implementation**

self.fc1 = nn.Linear(state\_size, 64)

self.fc2 = nn.Linear(64,32)

self.fc3 = nn.Linear(32, action\_size)

...

x = F.relu(self.fc1(state))

x = F.relu(self.fc2(x))

x = self.fc3(x)

**Experience Replay**

We store the last **10,000** experience tuples (S, A, R, S′) into a data container called **replay buffer** from which we sample **a mini-batch of 64** experiences. This batch ensures that the experiences are not highly correlated/independet and stable enough to train the network.

BUFFER\_SIZE = int(1e4) # replay buffer size

BATCH\_SIZE = 64 # minibatch size

**Epsilon Greedy**

In order to select the next action, the agent uses epsilon-greedy policy. The agent selects an action from action space randomly with a probability of **epsilon**, and is reduced gradually with decay rate **0.99** with the number of iterations till it reaches **0.01**

eps\_start=1.0, eps\_end=0.01, eps\_decay=0.99

**Parameters**

BUFFER\_SIZE = int(1e4) # replay buffer size

BATCH\_SIZE = 64 # minibatch 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

**Plot of Rewards**

If the agent achieves the score over 13, the problem is deemed to be solved. After tuning of the parameters, I could solve the problem in **307 episodes**. The plot below shows the rewards per episode and moving average over last 100 episodes.



**Ideas for Future Work**

* To improve the performance, it’s possible to implement other DQN alternatives, such as Double DQN, Prioritized replay, or Dueling DQN.