Deep Reinforcement Learning Nanodegree

Project 1 Report

Introduction

This project is the first one in Udacity's Deep Reinforcement Learning Nanodegree. In this project, the task is to train an agent to navigate and collect yellow bananas and avoid blue bananas in a Unity ML-Agents environment.

A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction. Given this information, the agent has to learn how to best select actions. Four discrete actions are available:

- `0` - walk forward
- `1` - walk backward
- `2` - turn left
- `3` - turn right

The task is episodic, and in order to solve the environment, the agent must get an average score of +13 over 100 consecutive episodes.

Learning Algorithm

A Double Deep Q Networks (DQN) algorithm has been used to solve this project. DQN is a famous type of Temporal Difference (TD) learning that is implemented using deep learning. TD learning is a category of Reinforcement Learning where the action value function of each state and action Q(s,a) gets updated by using only knowledge from the current and next state. A Double DQN is made of two identical deep neural networks, one acting as an action value estimator and the other is a replica that gets updated with the first network weights every specific number of episodes to assist the training of the first network in an approach called **Fixed Targets**. To improve training convergence of the networks and prevent them from learning the correlation between consecutive time frames, an approach called **Experience Replay** is being used where we use a buffer memory called replay buffer to store the experience tuple (consisting of state, action, reward and next state). We allow agent to randomly sample experiences from this buffer. This will allow us to learn from same experience multiple times. This is very helpful if we encounter certain rare experience

Hyperparameters Tuning:

After careful monitoring of the training process using multiple sets of hyperparameters. The best set that was found was as follows:

Variable Name	Hyperparameters Description	Chosen Value
n_episodes	how many episodes to train for	2000
eps_start	Starting value of epsilon factor	1
eps_end	Minimum value of epsilon factor	0.01
eps_decay	Rate of decay of epsilon factor	0.995
BUFFER_SIZE	replay buffer size	1e5
BATCH_SIZE	minibatch size	64
GAMMA	discount factor	0.99
TAU	Target network soft update factor	1e-3
LR	Main network learning rate	5e-4
UPDATE_EVERY	how often to update the target network get updated	4

Network Architecture

Both the main and target networks were composed of 5 fully connected (dense) layers as follows:

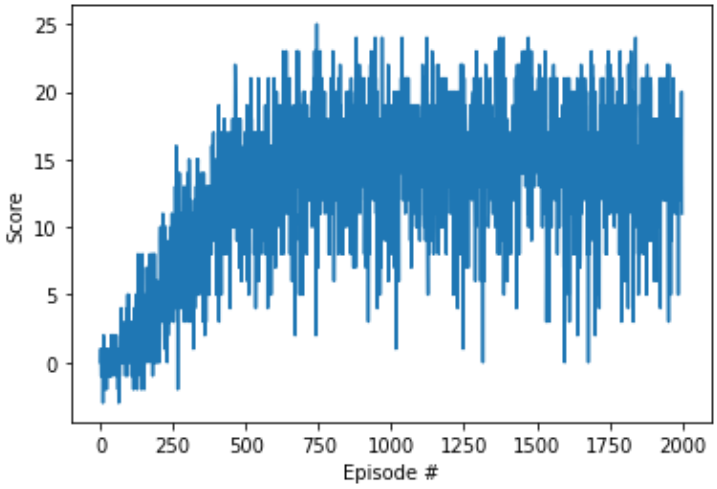
```python
class QNetwork(nn.Module):
    """Actor (Policy) Model."""

    def __init__(self, state_size, action_size, seed):
        """Initialize parameters and build model.
        Params
        ======
            state_size (int): Dimension of each state
            action_size (int): Dimension of each action
            seed (int): Random seed
        """
        super(QNetwork, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(state_size, state_size*4)
        self.fc2 = nn.Linear(state_size*4, state_size*8)
        self.fc3 = nn.Linear(state_size*8, action_size*8)
        self.fc4 = nn.Linear(action_size*8, action_size*4)
        self.fc5 = nn.Linear(action_size*4, action_size)

    def forward(self, state):
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = self.fc5(x)
        return x
```

Resulted Network:

```
QNetwork(
  (fc1): Linear(in_features=37, out_features=148, bias=True)
  (fc2): Linear(in_features=148, out_features=296, bias=True)
  (fc3): Linear(in_features=296, out_features=32, bias=True)
  (fc4): Linear(in_features=32, out_features=16, bias=True)
  (fc5): Linear(in_features=16, out_features=4, bias=True)
)
```

Results

The training process lasted for 2000 episodes, where each episode final scores has been recorded and is shown in the figure below:



The maximum mean score of every 100 consecutive episodes was recorded as 17.24 after from episode number 1432 to 1532 as shown in the screenshot below from the notebook file:

```
Episode 1532    Average Score: 17.24
Environment solved in 1432 episodes!    Average Score: 17.24
```

The model weights at this point have been saved in the *checkpoint.pth* file in the root of the GitHub repository so it can be retrieved again after the end of the 2000 episodes.

Future work to consider:

To improve on the current implementation, it is recommended to add following features to the above learning algorithm :

- Dueling DQN
- Prioritized Experienced Replay