Navigation Project Report

Introduction

During this project, a robot is trained to navigate and collect bananas in a large, square environment. Yellow bananas were rewarded with +1 points and purple bananas with -1 points. Therefore, our agent is tasked with collecting yellow bananas and avoiding blue bananas as much as possible.
There are 37 dimensions in the state space, which contains the agent's velocity and ray-based perceptions of objects around the agent's forward direction. The agent is assigned the task of selecting an action from the following four options:

- 0: move forward
- 1: move backwards
- 2: turn left
- 3: turn right

During the 100 consecutive episodes, the agent has to achieve a high average score (+13 or more) to succeed at the task.
Using the API of UnityAgents, we are able to simulate the environment. With the help of this API, we can simulate the environment, which saves us a significant amount of time.

Used Model

Our function approximator for the Q-function is a Deep Neural Network, and it is referred to as Deep Q-learning. In order to build the DNN, we select an architecture that consists of three linear layers. Initially, the input dimension of the first layer is 37, and the output dimension is 64. In the second layer, input and dimensions are 64 for both, and in the third layer, input dimensions are 64 and output dimensions are four, which indicates the number of actions to be performed. For each layer, a RELU function is used as the activation function. To update the model's weights, Gradient Descent is used in conjunction with the 'Adam' optimiser with a learning rate of 5e-4.
This project used a Q-learning agent with a 3-layered neural network as a function approximator. Q-learning is one of the most well-known reinforcement learning algorithms for learning action-value functions. As opposed to SARSA, q-learning directly learns the optimal q-value instead of switching between evaluation and improvement.
Instability is a problem associated with non-linear function approximators. Modifications made to improve the convergence are as follows:

Experience Replay:
We use replay buffer memory to store the experience tuple (consisting of state, action, reward and next state) to avoid learning experiences in sequence (correlated experiences). A random sample of experiences is taken from this buffer by the agent. In this way, we can learn from the same experience more than once. In certain rare cases, this comes in very handy.

Fixed Q-values:

Learning the network parameter 'w' determines the TD target. There is a risk of instability as a result. With a separate network with a similar architecture, we are able to remove this problem. Each update of the target network is accompanied by a gradual change in the hyperparameter TAU, while each update of the local network is accompanied by an aggressive change in the soft parameter.
When being trained, the learning algorithm uses an Epsilon-Greedy policy to select actions. The epsilon value specifies the probability of selecting a random action rather than following the "best action" in a given state (exploration-exploitation trade-off). Based on an action, the agent learns, updates the network every Update_every time step, and stores its experiences in memory using the agent class. In order to use experience replay, we need to define a class that contains memory buffers. ReplayBuffer objects initialize a deque to store experience tuples of maximum length Buffer_sizeE. With a batch size of Batch_size, it can store tuples and sample them.

DQN Algorithm

The DQN algorithm is included in the dqn method. Upon reaching a reward value of 13.0, it returns a list of scores for all 2000 episodes.

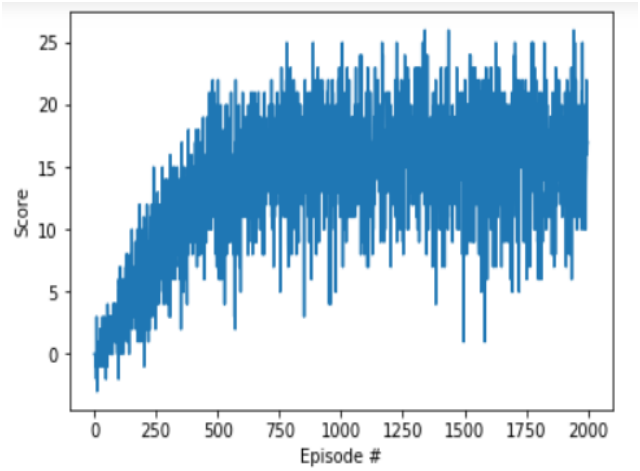
```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 64         # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR = 5e-4               # learning rate
UPDATE_EVERY = 4        # how often to update the network
```

```
def dqn(n_episodes=2000, max_t=1000, eps_start=1.0, eps_end=0.01, eps_decay=0.995):
    """Deep Q-Learning.

    Params
    ======
        n_episodes (int): maximum number of training episodes
        max_t (int): maximum number of timesteps per episode
        eps_start (float): starting value of epsilon, for epsilon-greedy action selection
        eps_end (float): minimum value of epsilon
        eps_decay (float): multiplicative factor (per episode) for decreasing epsilon
    """
    scores = []                        # list containing scores from each episode
    maxx = 0
    scores_window = deque(maxlen=100)  # last 100 scores
    eps = eps_start                    # initialize epsilon
    for i_episode in range(1, n_episodes+1):
        env_info = env.reset(train_mode=True)[brain_name]
        state = env_info.vector_observations[0]
        score = 0
        for t in range(max_t):
            action = (int)(agent.act(state, eps))
            env_info = env.step(action)[brain_name]
            next_state = env_info.vector_observations[0]
            reward = env_info.rewards[0]
```

```
            done = env_info.local_done[0]
            agent.step(state, action, reward, next_state, done)
            state = next_state
            score+=reward
            if done:
                break
        scores_window.append(score)       # save most recent score
        scores.append(score)              # save most recent score
        eps = max(eps_end, eps_decay*eps) # decrease epsilon
        print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.mean(scores_window)), end="")
        if i_episode % 100 == 0:
            print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.mean(scores_window)))
        if  np.mean(scores_window)>maxx and np.mean(scores_window)>=13:
            maxx = np.mean(scores_window)
            print('\nModel saved in {:d} episodes!\tAverage Score: {:.2f}'.format(i_episode,
np.mean(scores_window)))
            torch.save(agent.qnetwork_local.state_dict(), 'checkpoint.pth')

    return scores
```

Result

A best average score of 17.01 is achieved at 1354 episodes.



Future work to consider:

The following parts can be considered for future work:

- Dueling DQN
- Double DQN
- Work on Experienced Replay