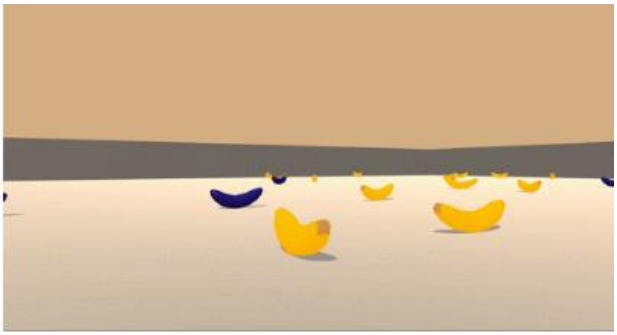
Deep Reinforcement Learning Nanodegree

Project 1 – Navigation

1 Introduction

For this project, an agent is trained to navigate (and collect bananas!) in a large, square world.



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 the 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 agent's forward direction. Given this information, the agent has to learn how to best select actions. Four discrete actions are available, corresponding to:

- 0 – move forward.
- 1 – move 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.

2 Learning Algorithm

The agent outputs an action and the environment returns an observation or, the state of the system and a reward. The goal of the agent is to select the actions that maximizes cumulative expected rewards which corresponds to the maximum overall or total rewards at each time step discounted by a discount factor γ.

2.1 Deep Q-Learning Algorithm

The Deep Q-Learning algorithm is an off-policy learning algorithm where the policy being learned is different from the policy on which the agent is evaluated. The idea of Q-Learning algorithm is to learn the optimal action-value function denoted by $Q^*(s,a)$ where s is the state and a is the action. The update at each step is as follows:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$$

where r_t is the reward at the current time-step t, γ is the discount factor, under policy $\pi = P(a_t | s_t)$ after observing state (s_{t+1}) by performing action a_t on state s_t.

The above state works fine when the state and action space is limited. It gets complex when the actions are continuous. Hence a function approximator is introduced which uses a deep neural network.

The deep neural network, depicted in Figure 2-1, comprises of three layers:

- an input layer with 37 neurons corresponding to 37 possible observation states
- two hidden layers with 64 neurons activated by a RELU function in each layer
- an output layer with four neurons corresponding to four possible actions, namely, forward, backward, left and right.

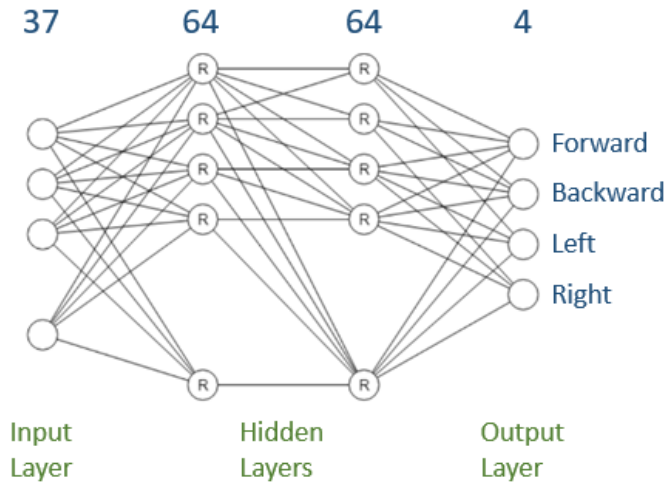


Figure 2-1 Neural Network with 37 input states mapped to 4 possible actions

There are situations where the network weights can oscillate or diverge due to high correlation between actions and states. The result is a very unstable and ineffective policy. Two techniques are used to stablize training, namely, experience replay and fixed Q-targets.

2.1.1 Experience Replay

During training, the sequence of observations can be highly correlated. A (circular) replay buffer is used to store each experience tuple (S, A, R, S') as the agent interacts with the environment. Using experience replay to sample a small batch of tuples from the replay buffer randomly out-of-order breaks the correlation between consecutive experience tuples. This allows the agent to learn more from individual tuples multiple times, recall rare occurrences, make better use of experiences and stabilize the action space - prevent action values from oscillating or diverging catastrophically.

2.1.2 Fixed Q-Targets

In Q-Learning, the parameters of the network is shifted based on a constantly moving target, and this can potentially lead to harmful correlations. Two separate networks with identical architectures are used, one with fixed targets and another learns the q value. The target network weights are updated less often (or more slowly) than the online network. This decouples the target from the parameters, making the learning algorithm more stable, and less likely to diverge or fall into oscillations.

2.2 Hyper-parameters

The hyper-parameters used in the Deep Q-Learning algorithm are shown in Table 1.

Hyper-parameter	Name in code	Value
Discount factor, γ	GAMMA	0.99
Replay buffer size	BUFFER_SIZE	10^5
Minibatch size	BATCH_SIZE	64
Soft update parameter, τ	TAU	10^{-3}
Learning rate	LR	$5 \cdot 10^{-4}$
Target update rate	UPDATE_EVERY	4
Initial epsilon, ε_0	eps_start	1.0
Final epsilon, ε_n	eps_end	0.01
Epsilon decay, δ	eps_decay	0.995

Table 1 Hyper-parameters

3 Results

The model training completes once the score – averaged over the past 100 episodes – exceeds 13.0. The model training completes in 415 episodes as shown in Figure 3-1.

```
Episode 100     Average Score: 1.34
Episode 200     Average Score: 4.70
Episode 300     Average Score: 8.03
Episode 400     Average Score: 9.65
Episode 500     Average Score: 12.74
Episode 515     Average Score: 13.01
Environment solved in 415 episodes!     Average Score: 13.01
```

Figure 3-1 Training Average Score Over 100 Episodes

Figure 3-2 depicts the model scores by episode.

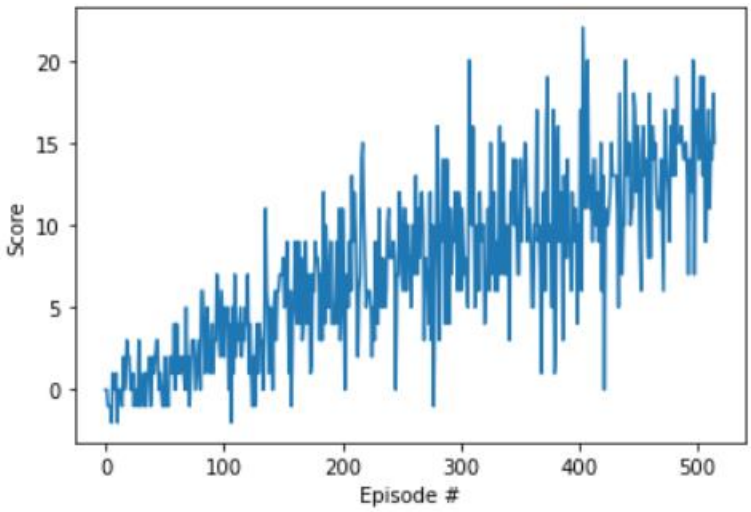


Figure 3-2 Scores by Episode

4 Potential Improvements

- Investigate prioritized experienced replay
- Investigate alternative network architectures: double DQN, dueling DQN