Deep Reinforcement Learning Robot Navigation

Introduction

This project aims to train a robot or agent to navigate its environment while collecting the maximum number of yellow bananas. 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 agent aims to collect as many yellow bananas as possible while avoiding blue bananas.

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

1. 0 - Walk forward
2. 1 - Walk backward
3. 2 - Turn left
4. 3 - Turn right

The task is episodic, and to solve the environment, the agent must get an average score of +13 over 100 consecutive episodes. This problem will be solved using the Deep Q-Network Algorithm using PyTorch and Python.

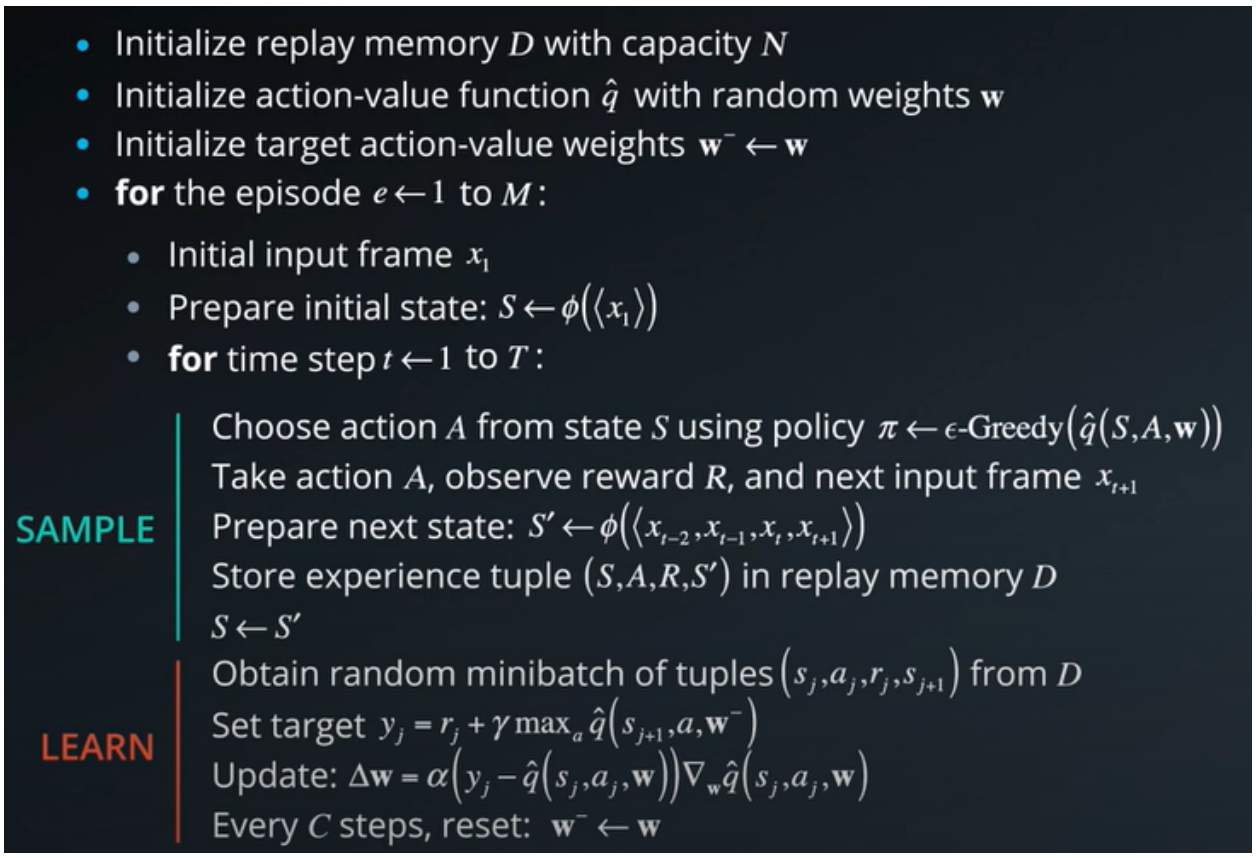
Methodology

When compared with traditional Reinforcement Learning, Deep Reinforcement Learning (DRL) combines RL with deep learning techniques. DRL uses nonlinear function approximators to calculate the value function or policy based directly on observation from the environment. To optimally find the parameters of the function approximator, Deep learning methods are used. When neural networks are used as function approximators to estimate Q-values, the algorithm is called **Deep Q-Network (DQN).**

A deep neural network that acts as a function approximator produces a vector of action values with the max value indicating the action to take. The Neural network will produce Q values for every possible action in a single pass. This project uses a Q-network neural network architecture with two hidden layers and an output layer. The ReLu activation function was used for both layers.

A few modifications are required for the deep reinforcement learning algorithm to be stable, which are also applied in this project; they are as follows:-

1. **Experience Replay:** It is used to avoid oscillations and inefficient policies. In this, we store each state reward state tuple in a replay buffer instead of trashing them. Then, a sample for the buffer is given to the agent to learn from it. This makes the RL problem into a supervised learning problem. It helps in breaking the correlation between consecutive experiences and improves learning stability.
2. **Fixed Q Target:** Q learning is a TD learning method, and the main idea is to reduce the TD error. However, the TD target is dependent on the function approximation parameter w. It is like chasing a moving Target. So, we must decouple the target's position from the agent's actions, giving a more stable environment. Finally, we use w^{-}. This does not change every step, but after a few iterations.



The detailed algorithm can be seen in the figure above, but in summary, the DQN algorithm has two main processes:-

1. Sample the environment by performing actions and store the experience tuples.

2. Select a random small batch of tuples from the memory and learn from it using the gradient decent update step.

The DQN Algorithm's update rule is shown in the equation below.

$$\Delta w = \alpha \cdot \overbrace{(\underbrace{R + \gamma \max_a \hat{q}(S', a, w^-)}_{\text{TD target}} - \underbrace{\hat{q}(S, A, w)}_{\text{old value}})}^{\text{TD error}} \nabla_w \hat{q}(S, A, w)$$

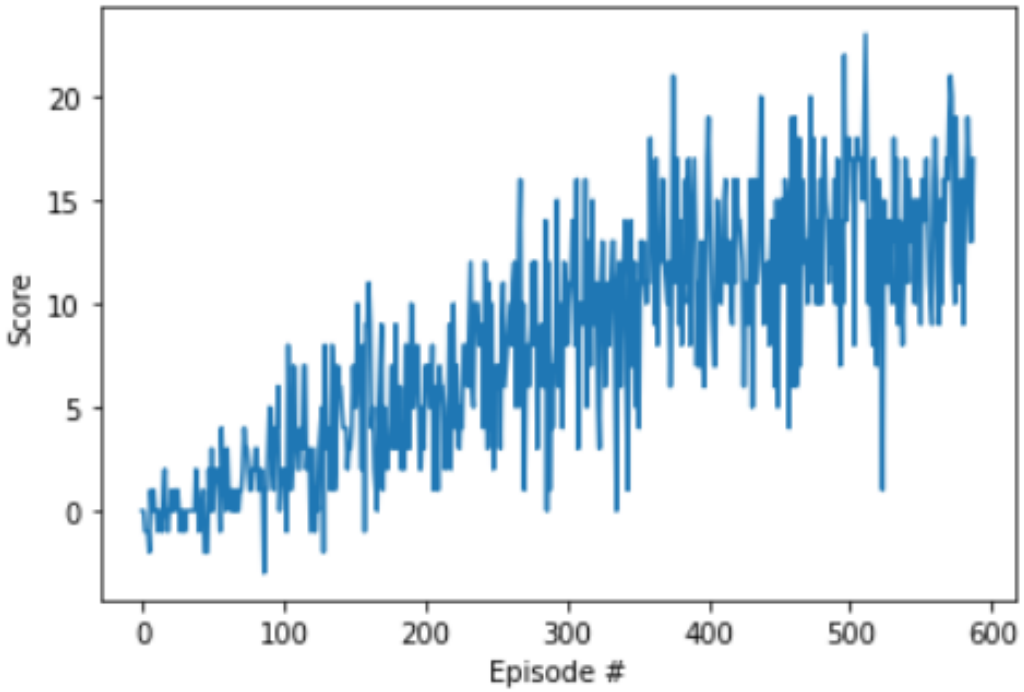
Hyperparameters

For the training, the following hyperparameters were used.

Parameter	Value
Number of Episodes	2000
Maximum Time steps	1000
Epsilon start	1.0
Epsilon Minimum	0.01
Epsilon Decay	0.995
Replay buffer size	int(1e5)
Batch Size	64
Discount Factor	0.99
Soft update of target Parameters	1e-3
Learning Rate	5e-4
Update the Network after every	4

Results

The following reward plot was obtained when the agent achieved an average reward of 13.9 for 100 continuous cycles. The environment was solved in 488 episodes.



Future Work

In the future, I intend to do the following:-

1. Implement Double DQN (DDQN), Prioritized experience replay, and Dueling DQN to improve the results obtained in this project.
2. To complete the 'Learning from Pixels' challenge, in which the agent would learn directly from pixels.