Report: Banana-Picker

Description

Project goal.

In this project, the goal is to train an agent to navigate a virtual world and **collect as many yellow bananas** as possible **while avoiding blue bananas.**

Environment details.

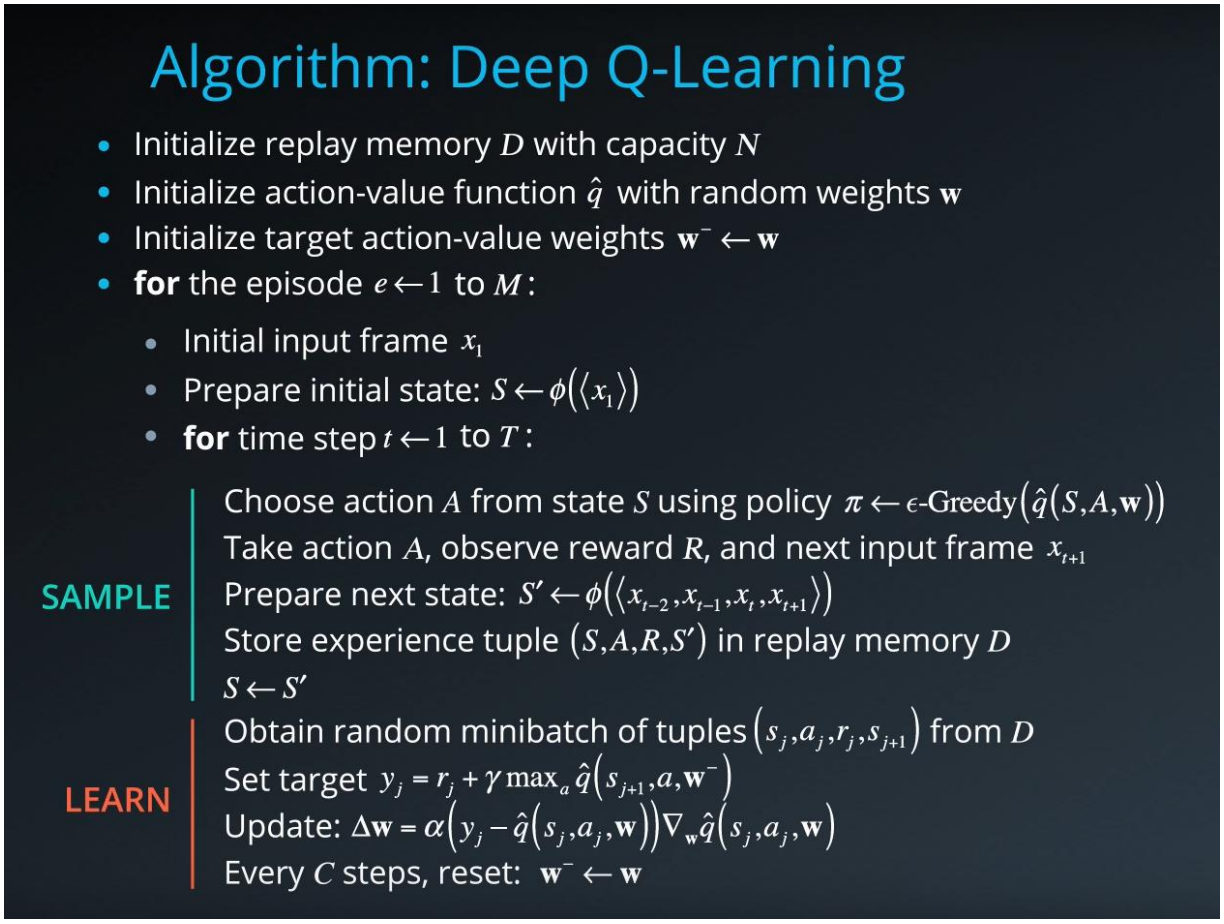
All the details of setting up the environment as well as the description of **state space, action space** and **reward structure** are described in the Readme file.

Agent's implementation.

The working of agent is based on Deep Q Learning Algorithm with **Experience Replay** and **Fixed Targets**. Both of these approaches are described in detail in the following paper. A brief description is given below.

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) function20. This instability has several causes: the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between the action-values and the target values. We address these instabilities with a novel variant of Q-learning, which uses two key ideas. First, we used a biologically inspired mechanism termed experience replay that randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution. Second, we used an iterative update that adjusts the action-values towards target values that are only periodically updated, thereby reducing correlations with the target.

Algorithm used:



This algorithm screenshot is taken from the Deep Reinforcement Learning Nanodegree course

Code description.

The code consist of :

- **model.py** : In this python file, a PyTorch QNetwork class is implemented. This is a regular fully connected Deep Neural Network using the PyTorch Framework. This network will be trained to predict the action to perform depending on the environment observed states. This Neural Network is used by the DQN agent and is composed of :
  - the **input layer** which size depends of the *state_size* parameter passed in the constructor
  - **3 hidden fully connected layers** of 512, 256 & 128 cells respectively.
  - the **output layer** which size depends of the *action_size* parameter passed in the constructor.
  - There is a **batch normalization (1d) layer** implemented atop input layer. This is to normalize the variance in input *state space*.

- **dqn_agent.py** : In this python file, a DQN agent and a Replay Buffer memory used by the DQN agent are defined.

  The DQN **Agent class** is implemented, as described in the Deep Q-Learning algorithm. It provides several methods:

  - *__init__():*
    - Initialize the memory buffer **(*Replay Buffer*)**
    - Initialize 2 instance of the Neural Network: the ***target*** network and the ***local*** network
  - *step*():
    - Allows to store a step taken by the agent (state, action, reward, next_state, done) in the Replay Buffer/Memory
    - **Every 4 steps** (and if their are enough samples available in the Replay Buffer), **update** the *target* network weights with the current weight values from the *local* network (That's part of the Fixed Q Targets technique)
  - *act*()
    - Which returns actions for the given state as per current policy (Note : The action selection use an **Epsilon-greedy** selection so that to balance between ***exploration*** and ***exploitation*** for the Q Learning)
  - *learn():*
    - which update the Neural Network value parameters using given batch of **experiences** from the Replay Buffer.
  - *soft_update*():
    - is called by learn() to softly updates the value from the *target* Neural Network from the *local* network weights (That's part of the **Fixed Q Targets** technique)

  The **ReplayBuffer class** implements a fixed-size buffer to **store experience tuples** (state, action, reward, next_state, done)

  - *add()*: allows to add an experience step to the memory.
  - *sample*(): allows to randomly sample a batch of experience steps for the learning.

- **DQN_Banana_Navigation.ipynb** : This Jupyter notebooks allows to train the agent. More in details it allows to :

- Import the Necessary Packages
- Examine the State and Action Spaces
- Take Random Actions in the Environment (No display)
- Train an agent using DQN algorithm described above.
- Plot the scores.
- Test run the trained agent on the environment.

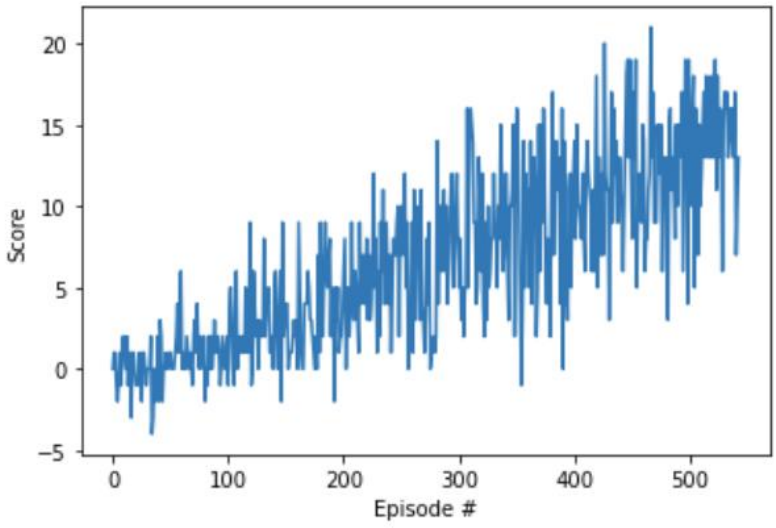
Hyperparameters used

```
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
```

Results

Achieving an average score greater than equal to +13 is considered solving the environment. This code was able to do it in 443 episodes only.

```
Episode 100     Average Score: 0.49
Episode 200     Average Score: 3.00
Episode 300     Average Score: 6.04
Episode 400     Average Score: 8.76
Episode 500     Average Score: 11.61
Episode 543     Average Score: 13.01
Environment solved in 443 episodes!     Average Score: 13.01
```



Ideas for future work

1. Using **Convolution Neural Network** to decipher the agent's field of view and take action according to it.
2. Using **Prioritised Replay** to focus on those actions more, that yield better expected cumulative reward/ score.
3. Using **Double DQN** to counter the problem of Overestimation of action values.
4. Using **Dueling DQN**. It represents two separate estimators: one for the state value function and one for the state-dependent action advantage function. The main benefit of this factoring is to generalize learning across actions without imposing any change to the underlying reinforcement learning algorithm. The results show that this architecture leads to better policy evaluation in the presence of many similar-valued actions.
5. I also wanted to apply **Rainbow algorithm** but maybe on a more popular task (Atari-2600 domain) to compare with research.