deep_reinforcement_learning_project1

Overview

This project trains an agent 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 your 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, corresponding to:

0 - move forward.

1 - move backward.

2 - turn left.

3 - turn right.

The task is episodic, my agent get an score of +15 over 100 consecutive episodes which is larger than +13 which the project requires.

Code Architecture

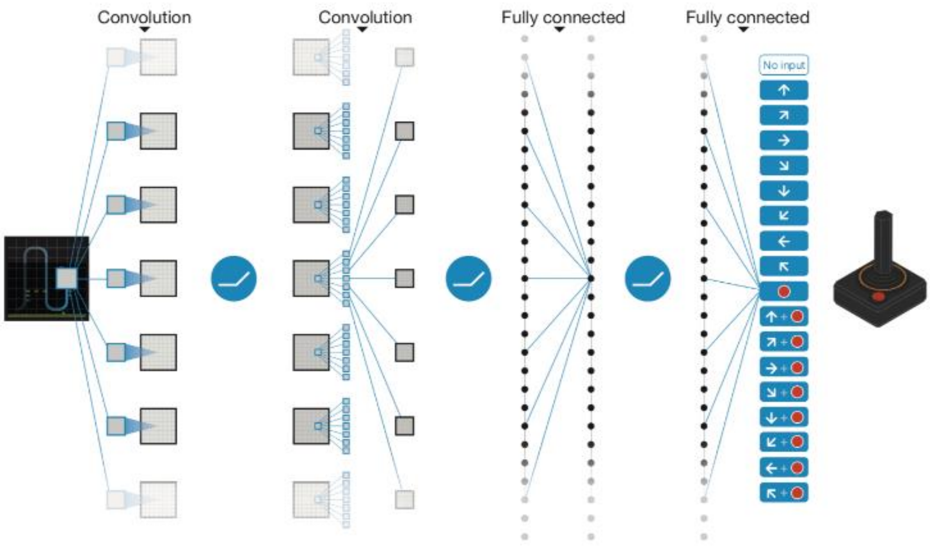
Navigation.ipynb: jupyter notebook based solution

dqn_agent.py: DQN agent code

model.py: Q-Network based model

checkpoint.pth: weights of the DQN model

DQN Architecture



The input to the neural network consists of an 84x84x4 image produced by the preprocessing map w. The first hidden layer convolves 32 filters of 8x8 with stride 4 with the input image and applies a rectifier nonlinearity31,32. The second hidden layer convolves 64 filters of 4x4 with stride 2, again followed by a rectifier nonlinearity. This is followed by a third convolutional layer that convolves 64filters of 3x3 with stride 1 followed by a rectifier. The final hidden layer is fully-connected and consists of 512 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action. The number of valid actions varied between 4 and 18 on the games we considered.

(*Refer to the paper “Human-level control through deep reinforcement learning” provided by Udacity)

DQN Algorithm

The optimal action-value function obeys an important identity known as the Bellman equation. This is based on the following intuition: if the optimal value $Q^*(s',a')$ of the sequence s' at the next time-step was known for all possible actions a', then the optimal strategy is to select the action a' maximizing the expected value of $r+\gamma Q^*(s',a')$:

$$Q^*(s,a) \quad = \mathbb{E}_{s'}\left[r+\gamma \max_{a'} Q^*(s',a')|s,a\right]$$

DQN with experience replay pseudocode:

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

(*The equation and pseudocode refers to the paper “Human-level control through deep reinforcement learning” provided by Udacity)

In the code delivered, the para setting is as below:

DQN is implemented as the deep reinforcement learning algorithm with parameter as below:
n_episodes=2000, max_t=1000, eps_start=1.0, eps_end=0.01, eps_decay=0.995

QNetwork and ReplayBuffer is used to build the agent class with the parameter as below.

```
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 64
GAMMA = 0.99
TAU = 1e-3
LR = 5e-4
UPDATE_EVERY = 4
```

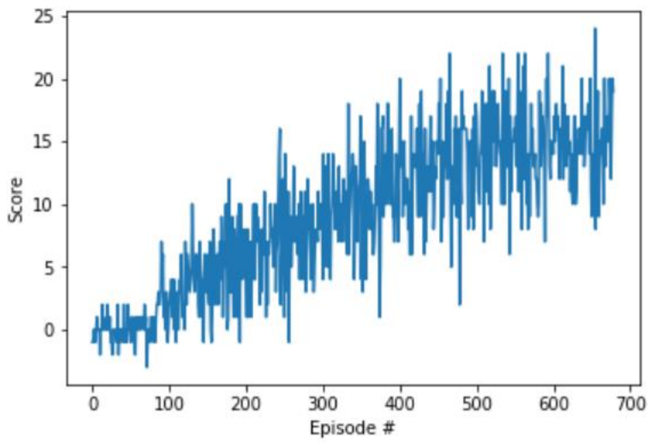
Plot the score

It takes 418 episodes to get an average score of 13.01

```
Requirement already satisfied: box2d in /opt/conda/lib/python3.6/site-packages (2.3.2)
Episode 100	Average Score: 1.04
Episode 200	Average Score: 4.68
Episode 300	Average Score: 7.43
Episode 400	Average Score: 9.37
Episode 500	Average Score: 12.59
Episode 518	Average Score: 13.01
Environment solved in 418 episodes!	Average Score: 13.01
```

It takes 579 episodes to get an average score of 15.00

```
Requirement already satisfied: box2d in /opt/conda/lib/python3.6/site-packages (2.3.2)
Episode 100     Average Score: 0.40
Episode 200     Average Score: 4.19
Episode 300     Average Score: 7.04
Episode 400     Average Score: 10.29
Episode 500     Average Score: 12.87
Episode 600     Average Score: 14.45
Episode 679     Average Score: 15.00
Environment solved in 579 episodes!     Average Score: 15.00
```



Ideas for future work

Double DQN and Dueling DQN could be further investigated if they have better results.