

Project's goal

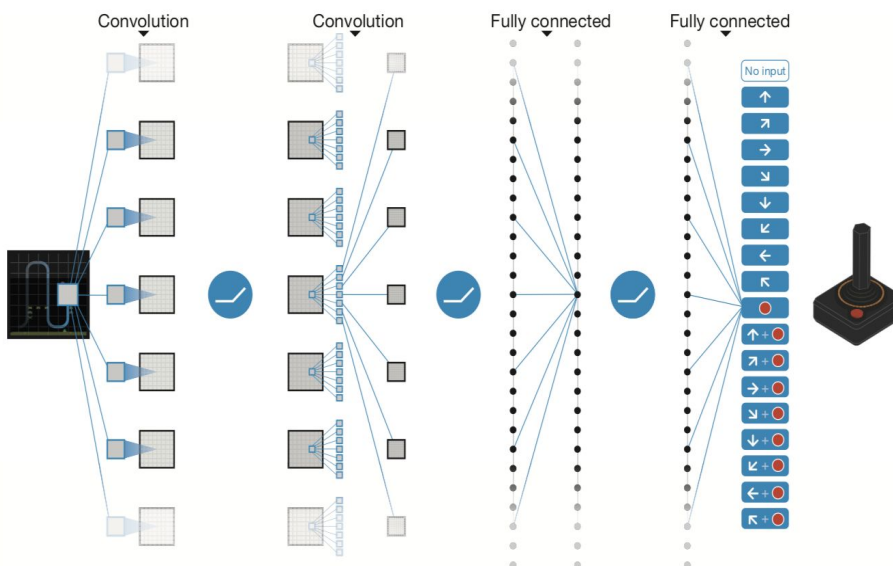
This project is for train an agent to collect yellow bananas while avoiding blue bananas in a virtual environment.

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.

Learning Algorithm and Agent Implementation

Deep Q-Networks(DQN) is used for the train an agent.

Illustration of the DQN:



Algorithm of the Deep Q-Learning:

Algorithm

Algorithm: Deep Q-Learning

- Initialize replay memory D with capacity N
- Initialize action-value function \hat{q} with random weights \mathbf{w}
- Initialize target action-value weights $\mathbf{w}^- \leftarrow \mathbf{w}$
- **for** the episode $e \leftarrow 1$ to M :
 - Initial input frame x_1
 - Prepare initial state: $S \leftarrow \phi(\langle x_1 \rangle)$
 - **for** time step $t \leftarrow 1$ to T :
 - Choose action A from state S using policy $\pi \leftarrow \epsilon\text{-Greedy}(\hat{q}(S, A, \mathbf{w}))$
 - Take action A , observe reward R , and next input frame x_{t+1}
 - Prepare next state: $S' \leftarrow \phi(\langle x_{t-2}, x_{t-1}, x_t, x_{t+1} \rangle)$
 - Store experience tuple (S, A, R, S') in replay memory D
 - $S \leftarrow S'$
 - Obtain random minibatch of tuples (s_j, a_j, r_j, s_{j+1}) from D
 - Set target $y_j = r_j + \gamma \max_a \hat{q}(s_{j+1}, a, \mathbf{w}^-)$
 - Update: $\Delta \mathbf{w} = \alpha (y_j - \hat{q}(s_j, a_j, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(s_j, a_j, \mathbf{w})$
 - Every C steps, reset: $\mathbf{w}^- \leftarrow \mathbf{w}$

SAMPLE

LEARN

DQN parameters and results:

```
BUFFER_SIZE = int(1e5) # replay buffer size
BATCH_SIZE = 64 # minibatch size (Initially 64)
GAMMA = 0.995 # discount factor (Initially 0.99)
TAU = 1e-3 # for soft update of target parameters
LR = 5e-4 # learning rate (Initially 5e-4)
UPDATE_EVERY = 4 # how often to update the network
```

```
agent = Agent(state_size=37, action_size=4, seed=42)
```

```
start_time = time.time() # Monitor training time
scores = dqn(n_episodes=2000, max_t=1000, eps_start=1.0, eps_end=0.01, eps_decay=0.995)
```

Unity brain name: BananaBrain

Number of Visual Observations (per agent): 0

Vector Observation space type: continuous

Vector Observation space size (per agent): 37

Number of stacked Vector Observation: 1

Vector Action space type: discrete

Vector Action space size (per agent): 4

Vector Action descriptions: , , ,

Number of agents: 1

Number of actions: 4

States look like: [1. 0. 0. 0. 0.84408134 0.
0. 1. 0. 0.0748472 0. 1.
0. 0. 0.25755 1. 0. 0.
0. 0.74177343 0. 1. 0. 0.
0.25854847 0. 0. 1. 0. 0.09355672
0. 1. 0. 0. 0.31969345 0.
0.]

States have length: 37

Score: 0.0

Episode 100 Average Score: 1.29

Episode 200 Average Score: 4.60

Episode 300 Average Score: 7.46

Episode 400 Average Score: 8.81

Episode 500 Average Score: 10.39

Episode 600 Average Score: 11.80

Episode 700 Average Score: 11.93

Episode 800 Average Score: 12.07

Episode 900 Average Score: 11.58

Episode 1000 Average Score: 12.17

Episode 1100 Average Score: 12.42

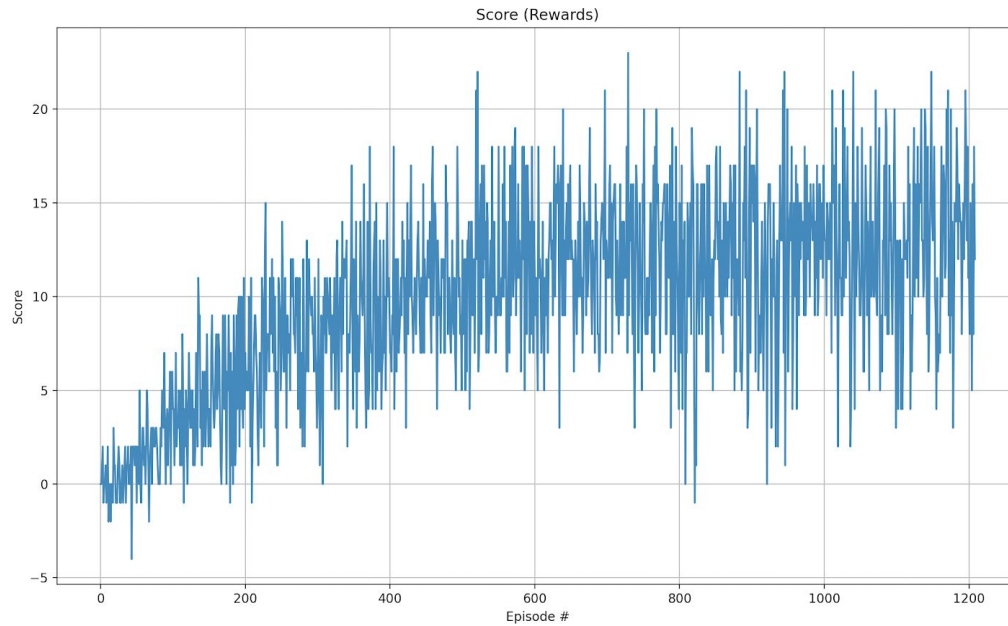
Episode 1200 Average Score: 12.81

Episode 1209 Average Score: 13.01

Environment solved in 1109 episodes! Average Score: 13.01

Total Training time = 57.5 min

Plot of Rewards



Conclusion and Ideas for Future Work:

Trained agent with DQN, better than agent which decide randomly

For the future work several improvements to the original Deep Q-Learning algorithm have been suggested in the classroom. These are:

1.Double DQN

Deep Q-Learning tends to overestimate action values. Double Q-Learning has been shown to work well in practice to help with this.

2.Prioritized Experience Replay

Deep Q-Learning samples experience transitions *uniformly* from a replay memory. Prioritized experience replay is based on the idea that the agent can learn more effectively from some transitions than from others, and the more important transitions should be sampled with higher probability.

3. Dueling DQN

Currently, in order to determine which states are (or are not) valuable, we have to estimate the corresponding action values *for each action*. However, by replacing the traditional Deep Q-Network (DQN) architecture with a dueling architecture, we can assess the value of each state, without having to learn the effect of each action.