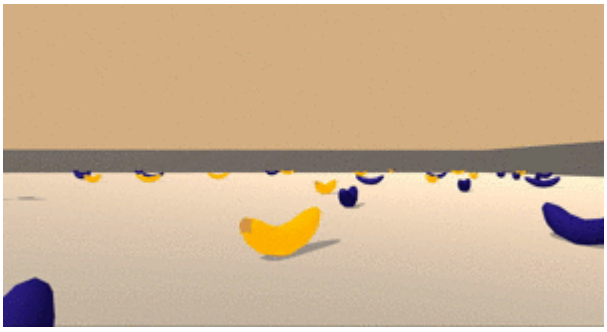
Navigation

In this project, we train an agent to navigate (and collect bananas!) in a large, square world.



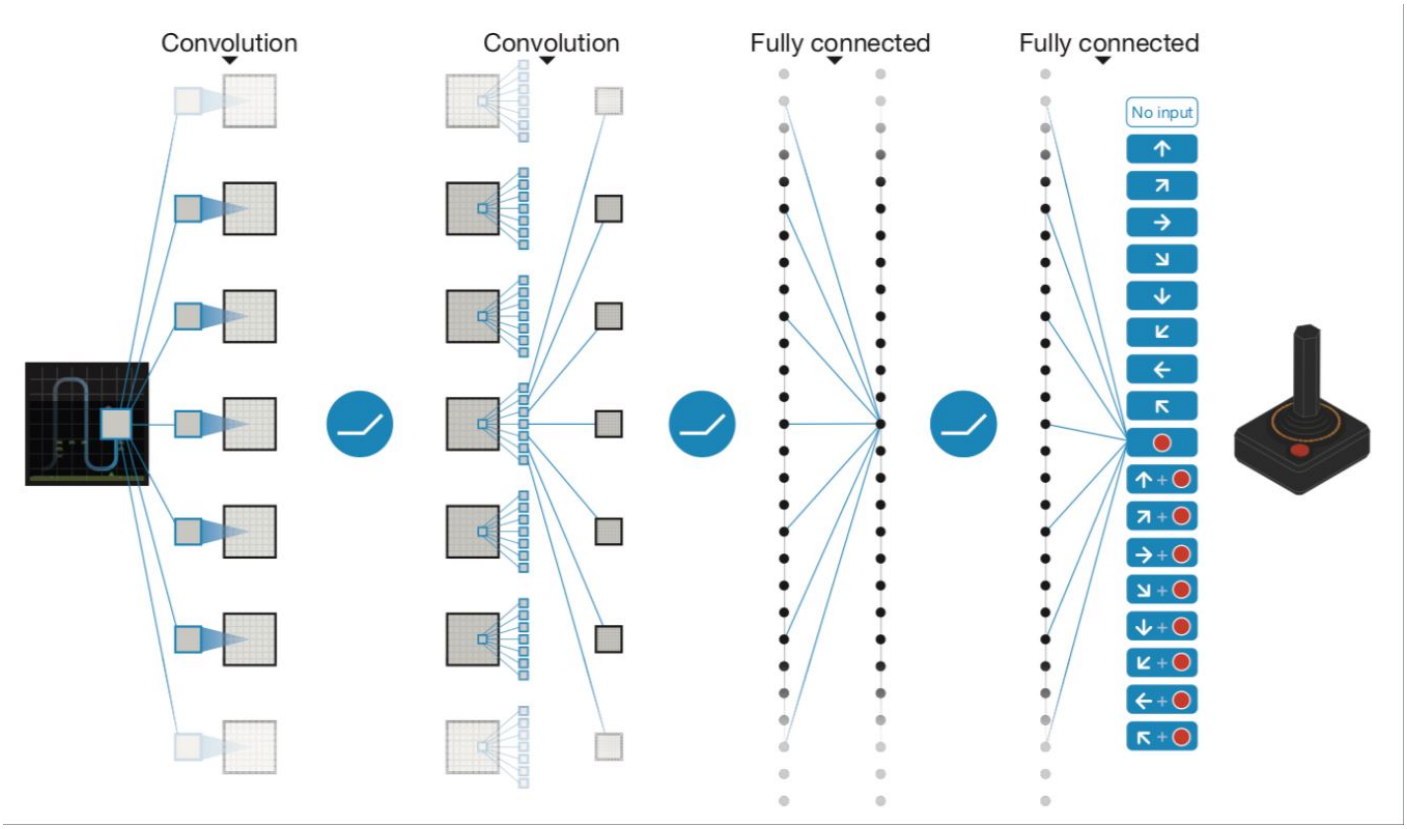
A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of your agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around 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.

Agent implementation

Deep Q-Networks

The DQN architecture is composed of a couple of convolutional layers and no fully-connected layers, which takes the state as input, and returns the corresponding predicted action values for each possible game action.



Experience Replay

When the agent interacts with the environment, the sequence of experience tuples can be highly correlated. The naive Q-learning algorithm that learns from each of these experience tuples in sequential order runs the risk of getting swayed by the effects of this correlation. By instead keeping track of a replay buffer and using experience replay to sample from the buffer at random, we can prevent action values from oscillating or diverging catastrophically.

The replay buffer contains a collection of experience tuples (S, A, R, S'). The tuples are gradually added to the buffer as we are interacting with the environment.

The act of sampling a small batch of tuples from the replay buffer in order to learn is known as experience replay. In addition to breaking harmful correlations, experience replay allows us to learn more from individual tuples multiple times, recall rare occurrences, and in general make better use of our experience.

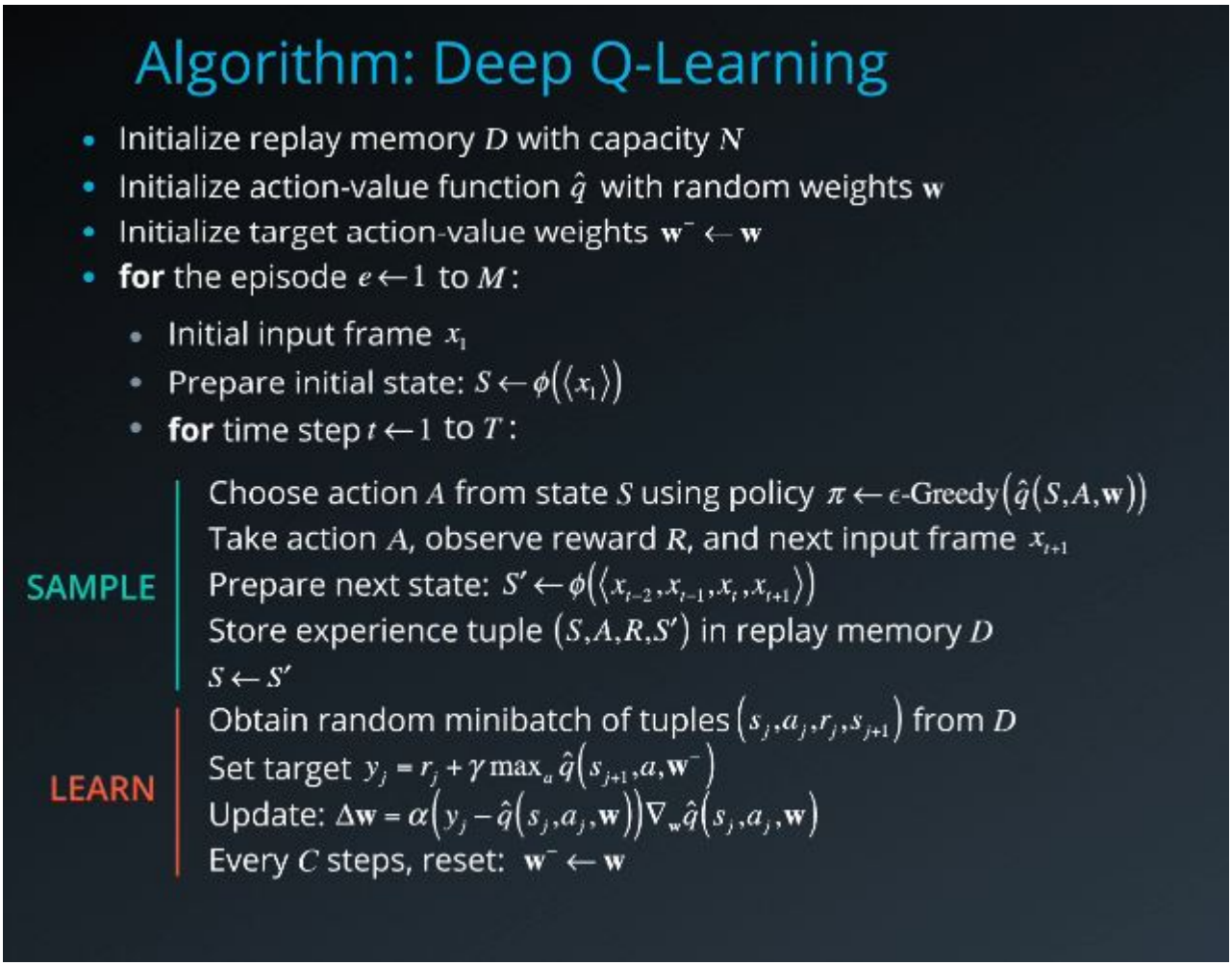
Fixed Q-Targets

In Q-Learning, we update a guess with a guess, and this can potentially lead to harmful correlations. To avoid this, we can update the parameters ww in the network q to better approximate the action value corresponding to state S and action A with the following update rule:

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

Enjoy first-class Markdown support with easy access to Markdown syntax and convenient keyboard shortcuts.

Algorithm



Code

Code consists of `Navigation.ipynb`

QNetwork class

This is fully connected Deep Neural Network implemented using PyTorch. This network predicts the action to perform depending on the observed state.

- It consists of 2 hidden fully connected layers of 1024 cells each

DQN Agent

- initialize:
  - Replay buffer
  - *Local* Q-Network and *target* Q-Network
- step:
  - Save experience tuple (S, A, R, S', done) in replay memory
  - Update the target network, every 4 steps, with the weights from local network
- act:
  - Returns actions for given state as per current policy (action selected using epsilon-Greedy policy)
- learn:
  - Update value parameters using given batch of experience tuples (S, A, R, S', done)
- soft update:
  - updates the value of target network with values from local network

Replay Buffer

- add:
  - Add a new experience to replay memory
- sample:
  - Randomly sample a batch of experiences from memory

DQN parameters

Neural network uses Adam optimizer with following parameters

```
BUFFER_SIZE = int(1e5) # replay buffer size
```

```
BATCH_SIZE = 64 # minibatch size
```

```
GAMMA = 0.995 # 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

```
Episode 100     Average Score: 0.88
Episode 200     Average Score: 2.69
Episode 300     Average Score: 5.90
Episode 400     Average Score: 8.38
Episode 500     Average Score: 10.17
Episode 600     Average Score: 11.00
Episode 700     Average Score: 10.92
Episode 800     Average Score: 12.30
Episode 832     Average Score: 13.04
Environment solved in 732 episodes!     Average Score: 13.04

Total Training time = 19.1 min
```