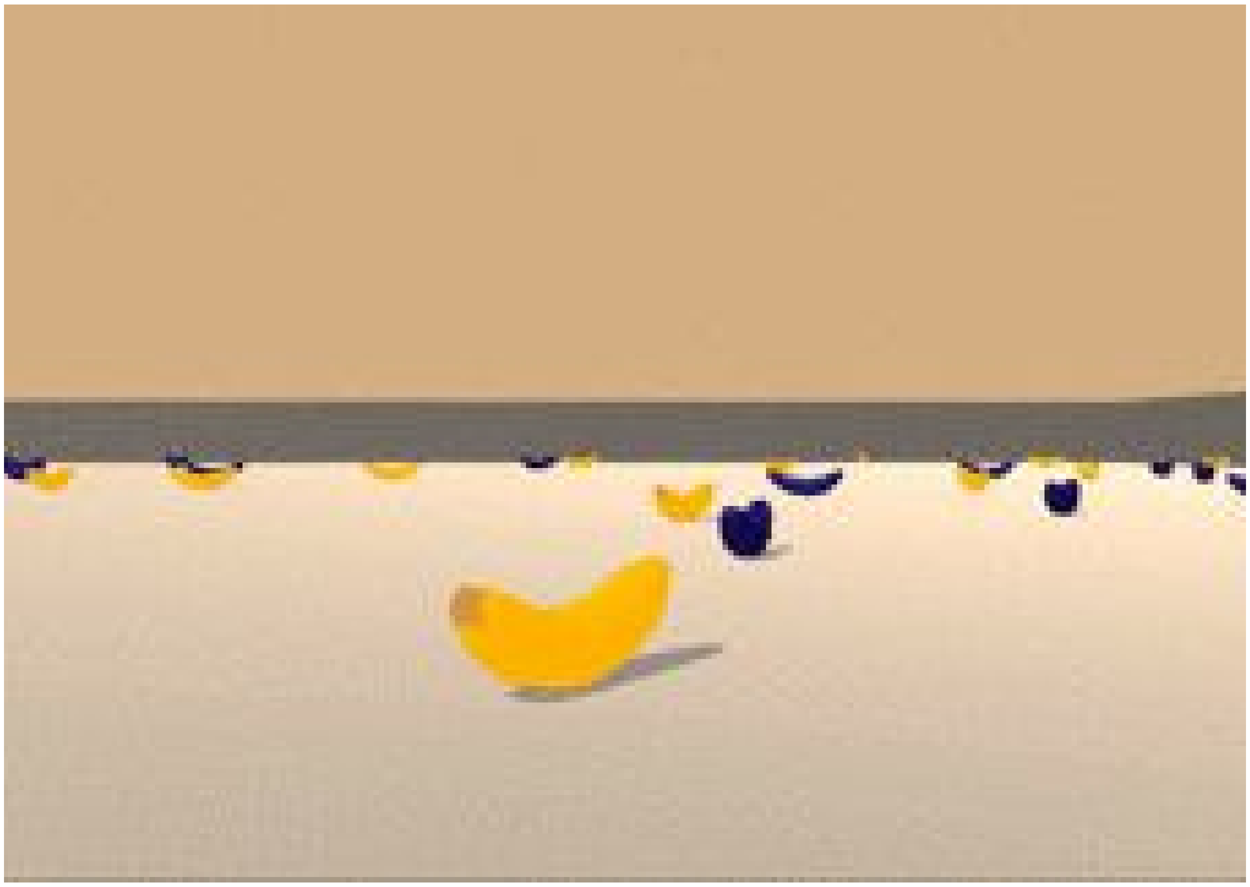
Navigation using DQN



Matthew Hayes

09.08.2019

Udacity Deep Reinforcement Learning Project 1

INTRODUCTION

The aim of this project is to navigate the agent and collect 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 goal of your agent is to collect as many yellow bananas as possible while avoiding blue bananas. To achieve this an agent is trained to output 4 actions: Up, Down, Left or Right. An average score of >= 13 must be achieved over the last 100 episodes. The input state is 37

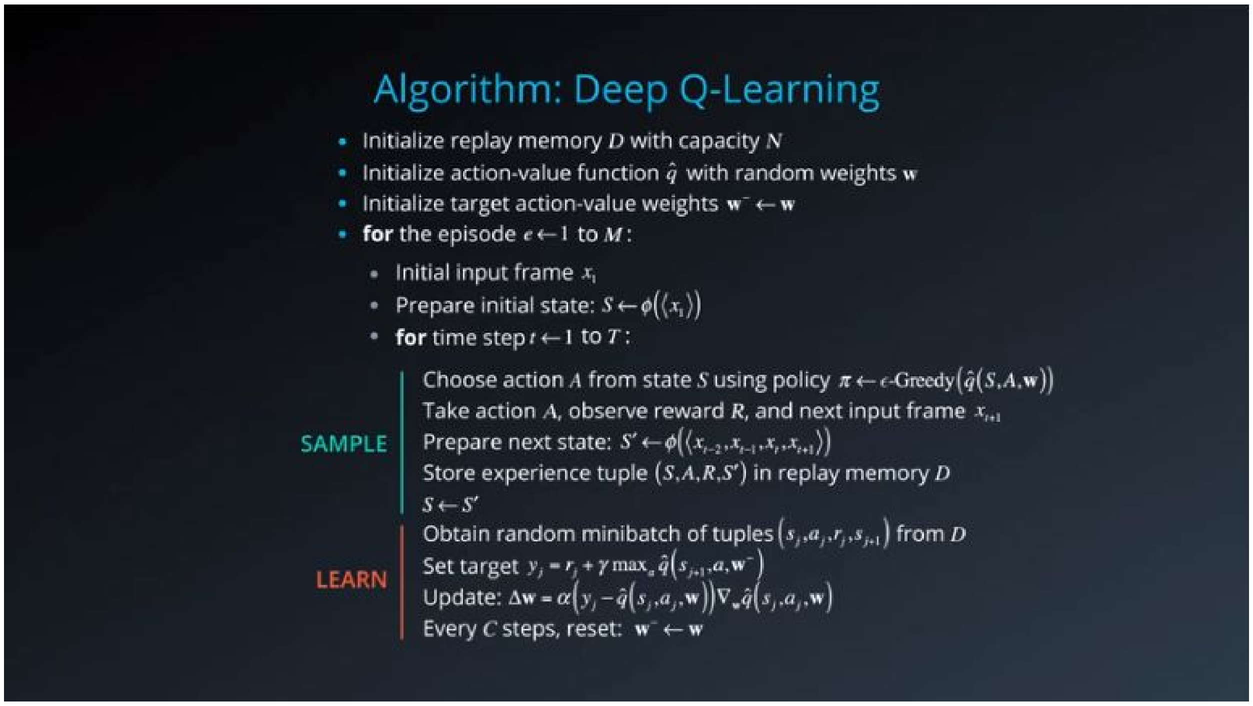
DQN Overview

The Deep Q-Network (DQN) was the first deep reinforcement learning method proposed by DeepMind, the paper was published on Nature in 2015.

DQN has four important features:

- For continuous or large state spaces a table approach is not usable so a function approximator is needed. In DQN DNN are used to perform the “Q-table: **function approximation**.
- DQN follows an extension of SARSA called **Q-Learning**. SARSA (State-Action-Reward-State-Action) is an on-policy TD control method. Q-Learning is an off-policy TD control policy. It’s exactly like SARSA with the only difference being — it doesn’t follow a policy to find the next action A’ but rather chooses the action in a greedy fashion. Similar to SARSA its aim is to evaluate the Q values.
- **Experience Replay**: experience replay that randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution.
- **Target Network**: In TD error calculation, target function is changed frequently with DNN. Unstable target function makes training difficult. So Target Network technique fixes parameters of target function and replaces them with the latest network every X steps.

Algorithm Pseudo Code



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 $T$:
SAMPLE    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 $D$
          $S \leftarrow S'$
LEARN     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: $\mathbf{w}^- \leftarrow \mathbf{w}$
```

*source: Udacity Deep Reinforcement Learning

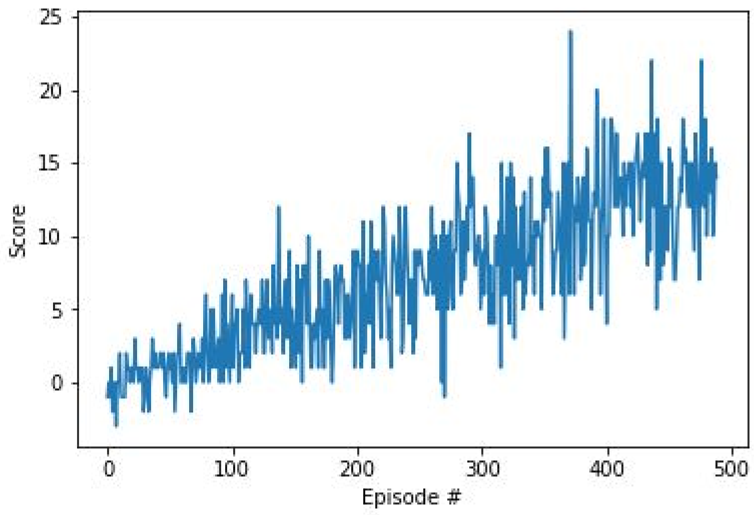
Code Overview

1. Deep_Q_Network_Solution.ipynb: iPython Notebook driving the DQN training:
    a. Imports packages
    b. Examine environment
    c. Demonstrate how to take random actions in env.
    d. Train DQN enabled agent.
    e. Plot results.
2. Dqn_agent.py: Implements DQN.
    a. Step: Save experience in replay memory.
    b. Act: Returns actions for given state as per current policy.
    c. Learn: Update value parameters using given batch of experience tuples.
    d. Update: Soft update model parameters.
3. Model.py:
    a. Forward: Build a network that maps state -> action values.

Results

Project's requirements satisfied as the agent is able to receive an average reward over 100 episodes of at least +13 in 389 episodes.

```
Episode 100     Average Score: 1.08
Episode 200     Average Score: 4.49
Episode 300     Average Score: 7.73
Episode 400     Average Score: 10.06
Episode 489     Average Score: 13.07
Environment solved in 389 episodes!     Average Score: 13.07
```



REFERENCES

1. Nature publication : "Human-level control through deep reinforcement learning (2015)"
2. https://www.udacity.com/course/deep-reinforcement-learning-nanodegree--nd893
3. Double DQN