Deep Q Network Variants in Unit Environment

Environment:

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

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 must 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, and in order to solve the environment, the agent must get an average score of +13 over 100 consecutive episodes.

Methods:

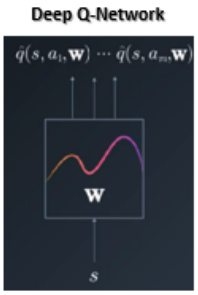
I experimented with different deep reinforcement learning architectures such as Deep Q-Network (DQN) [1], Double Deep-Q-Learning (DDQN) [2], and Dueling Network [3]. DQN is a simple deep neural network architecture that approximates the Q-value function for a given state. It tends to overestimate the Q-values and can be difficult to train. Therefore, several techniques were developed. This project employs experience replay, where state transitions, actions, and rewards are all saved in a buffer, and uniformly sampled in batches during training. Additionally, a separate network that is updated less frequently was used as a target in calculating the loss during backprop to mitigate oscillations during training. The main difference between DQN and DDQN is the choice of target Q values during training. In DDQN, the greedy policy is evaluated according to the online network, but its value is estimated using the target network. Dueling Network, on the other hand, uses the DDQN target Q-function, but differs from the other two in the representation of the network architecture, which separates the representation of state values and (state-dependent) action advantages. For more details, see [1-3].

Deep Q-Network (DQN):

$$\Delta w = \alpha \left(Q_{target} - Q(s,a;w)\right) \nabla_w Q(s,a;w)$$

$$^{DQN}Q_{target} = R + \gamma \, max_a \, Q(s',a';w^-)$$

(w^- updated less frequently)

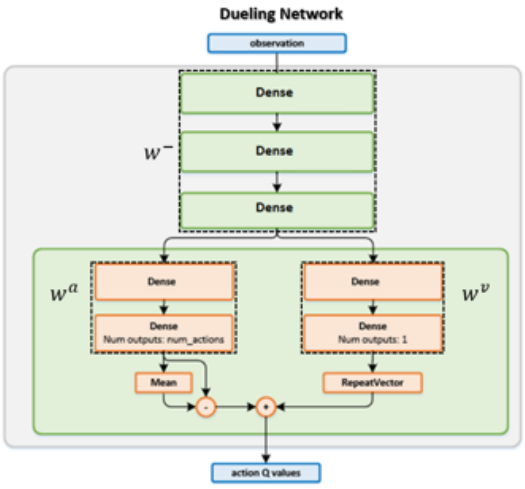


Double DQN (DDQN):

$$^{DDQN}Q_{target} = R + \gamma Q(s', \, max_a Q(s',a;w);w^-)$$

Dueling DDQN:

$$^{Dueling}Q_{target} = V(s;w^-,w^v) + \left(A(s,a;w^-,w^a) - \frac{1}{|A|}\sum_{a'} A(s,a';w^-,w^a)\right)$$



Architecture:

DQN (w):

- Fully Connected (input: state_size, output:256)
- RELU
- Fully Connected (input: 256, output:128)
- RELU
- Fully Connected (input: 128, output:64)
- RELU
- Fully Connected (input: 64, output: action_size)

Dueling-Network:

- **Feature modules (w^{-}):**
  - Fully Connected (input: state_size, output:256)
  - RELU
  - Fully Connected (input: 256, output:128)
  - RELU
  - Fully Connected (input: 128, output:64)
- **Value modules (w^{v}):**
  - Fully Connected (input: 64, output:32)
  - RELU
  - Fully Connected (input: 32, output:1)
- **Action modules (w^{a}):**
  - Fully Connected (input: 64, output:32)
  - RELU
  - Fully Connected (input: 32, output: action_size)

Parameters:

BUFFER_SIZE = int(1e4) # replay buffer size

BATCH_SIZE = 64 # minibatch size

GAMMA = 0.99 # discount factor

TAU = 1e-3 # for soft update of target parameters

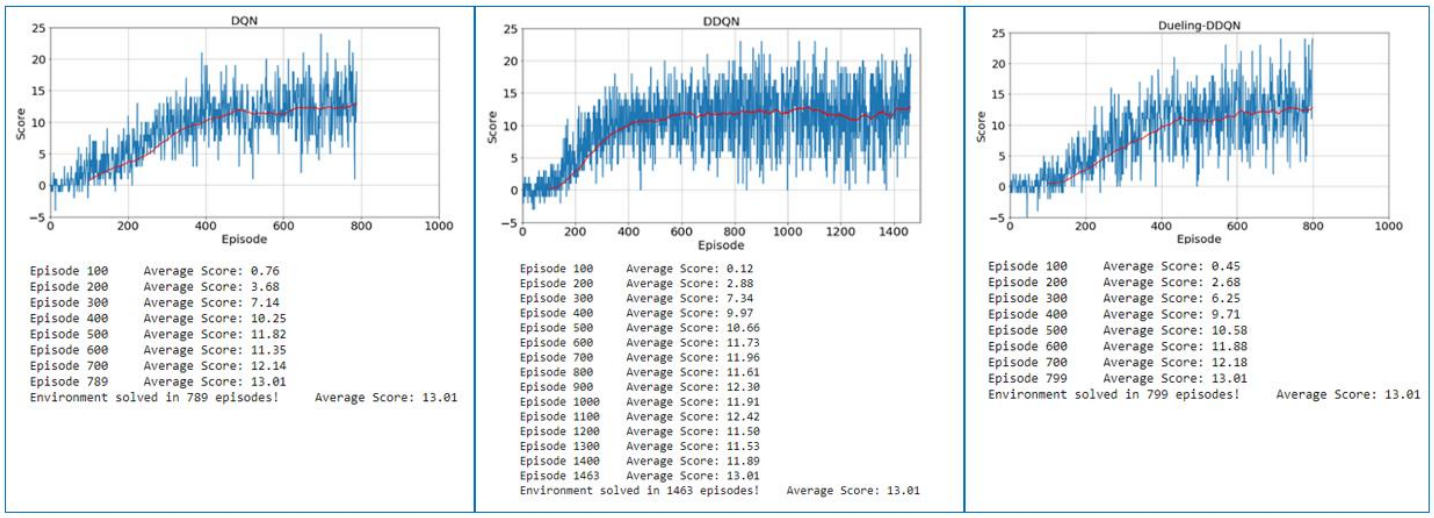
LR = 5e-4 # learning rate

UPDATE_EVERY = 4 # how often to update the target network

Deep Q-Learning Network

Results: By the 400th hundred episodes, all learning algorithms reached above a score of 10. DDQN took the longest to train. Although they all successfully complete the training by hitting the score 13, the learned behaviors are quite different. DQN has frequent glitches where at certain states it keeps oscillating between two states and the frames look like agent is paralyzed. Dueling Network (***Results/trained_dueling.gif*** shown on ReadMe.md) makes strategic moves and maximizes the return. However, Dueling Network did not learn to utilize the action “move backward” in the runs that I examined. Double DQN on the other hand mastered the “move backward” action and makes more risky maneuvers.

Please see the gif under ***Results/ dqn_backing_up_move.gif***. This can be interpreted as the network that the search space longer, discovers new tricks. Ending training prematurely based on a score may not be the best strategy.



Future Work:

There are three logical next steps to improve performance.

- ***Prioritized experience replay***: Experiences can be sampled based on their priority rather than sampling uniformly. It's common to define the priority as a function of $TD\ error = Q_{target} - Q(s, a; w)$ in order to learn more from states where the learning error is large.
- ***Learning from pixels***: In this project, the agent learned its velocity and ray-based perception of objects. We can instead learn from pixels as in the original DQN paper [1] by stacking multiple consecutive images as the state.
- **State of the art networks:** There are further extensions to the DQN algorithm. Rainbow [4] incorporates most of them and outperforms on Atari 2600 games.

References:

[1] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

[2] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." Thirtieth AAAI conference on artificial intelligence. 2016.

[3] Wang, Ziyu, et al. "Dueling network architectures for deep reinforcement learning." arXiv preprint arXiv:1511.06581 (2015).

[4] Hessel, Matteo, et al. "Rainbow: Combining improvements in deep reinforcement learning." *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

Referenced in the code:

[4]OrderedDict: https://towardsdatascience.com/three-ways-to-build-a-neural-network-in-pytorch-8cea49f9a61a

[5]Dueling Network: https://lilianweng.github.io/lil-log/2018/05/05/implementing-deep-reinforcement-learning-models.html