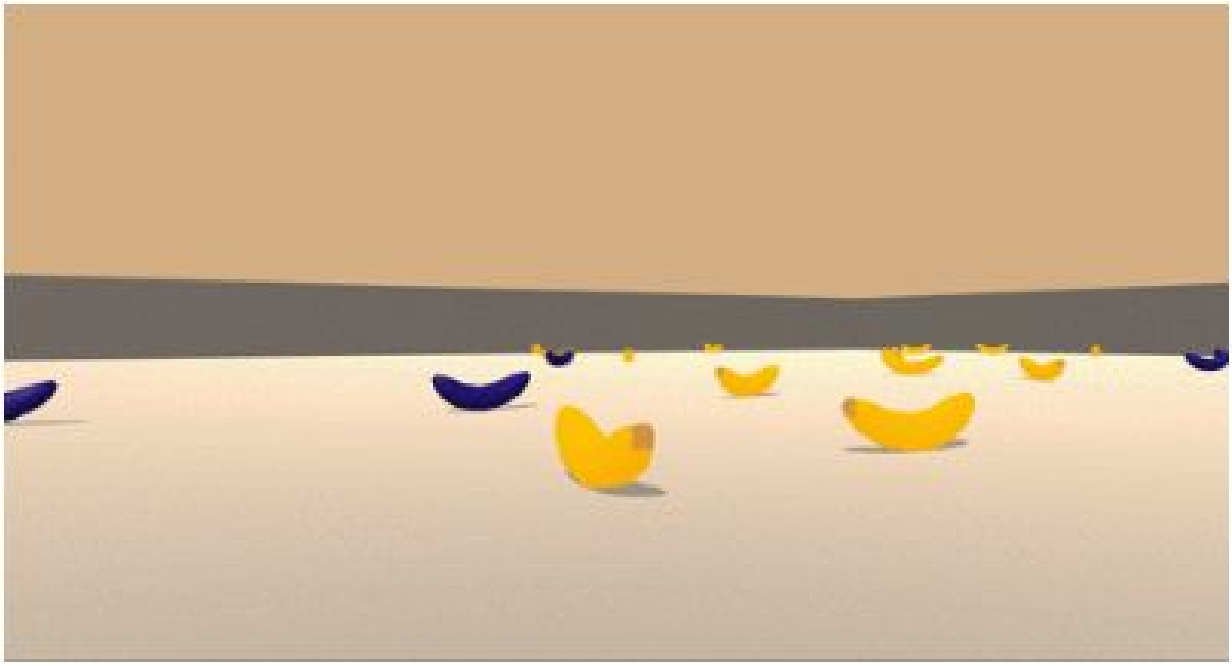
Udacity Deep Reinforcement Learning Nanodegree Program

Navigation Project

Kellin Bershinsky



Introduction

This project is intended to introduce students to Unity's ML-Agents machine learning toolset and provide an opportunity to apply their knowledge of Deep Q-Learning networks to train a Deep Reinforcement Learning agent. The environment used for this project includes 37 discrete states and 4 actions which include moving the agent forward, backward, left, and right to collect yellow bananas while avoiding blue bananas. The state space includes an array of ray casts that provide the agent information to determine banana color, direction, and distance. To solve the episodic task, the agent must get an average score of +13 over 100 consecutive episodes. This report discusses the training methodology used to create the RL agent, a plot of the results of the training, and provides insight into future improvements.

Learning Algorithm Used - DQN with Improvements

Q-Learning or Sarsamax

A Deep Q-Learning agent was trained to complete this task. Q-Learning is a temporal difference method that uses an epsilon greedy policy to predict the best action for the next state.

Algorithm 14: Sarsamax (Q-Learning)

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
    Take action A_t and observe R_{t+1}, S_{t+1}
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q

Figure 1: Q-Learning Algorithm[1]

Deep Q-Networks

Deep Q-Networks build on Q-Learning by making use of Neural Networks that allow them to make use of hidden layers to identify features that make predicting best actions for unknown states possible. For more complicated environments, the state space can become massive, and with enough variation in possible actions, traditional RL methods are insufficient.

There are two main processes which include sampling the environment and learning from a minibatch of tuples using a gradient descent update step.

[1] "Reinforcement Learning - Richard S. Sutton." 5 Nov. 2017, http://incompleteideas.net/book/bookdraft2017nov5.pdf. Accessed 16 Jul. 2019.

Improvements Added:

Experience Replay

Experience replay uses a replay buffer to sample previous experience tuples (S, A, R, S') in an effort to overcome correlation bias.

Fixed Q-Targets

A fixed Q-Target uses a separate target network that doesn't charge during the learning step to decouple target from the TD error thereby stabilizing the update function.

Hyperparameters Used

Experience Replay Buffer Size = 1e5

Minibatch Size = 64

Discount Factor Gamma = 0.99

Soft Update Target Parameter Tau = 1e-3

Learning Rate Alpha = 5e-4

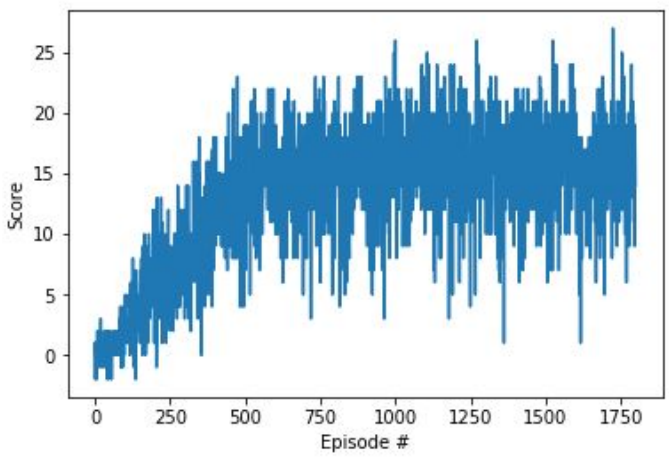
Fixed Q-Target Update Every 4 Episodes

Epsilon Start = 1.0

Epsilon End = 0.01

Eps Decay = 0.995

Results - Plot of Rewards




```
Episode 100     Average Score: 0.69
Episode 200     Average Score: 4.13
Episode 300     Average Score: 6.98
Episode 400     Average Score: 9.99
Episode 500     Average Score: 12.73
Episode 600     Average Score: 14.91
Episode 700     Average Score: 14.54
Episode 800     Average Score: 15.08
Episode 900     Average Score: 15.62
Episode 1000    Average Score: 15.50
Episode 1100    Average Score: 16.10
Episode 1200    Average Score: 16.33
Episode 1300    Average Score: 15.59
Episode 1400    Average Score: 15.73
Episode 1500    Average Score: 15.82
Episode 1600    Average Score: 16.53
Episode 1700    Average Score: 15.03
Episode 1800    Average Score: 15.75
```

The figure above shows the agents average score throughout the training process which included 1800 episodes. The table to the right of the plot shows the average score over 100 hundred episodes for every 100 episodes throughout the training session. Based on these results, it appears the environment was solved in approximately 600 episodes.

Ideas for Future Work

Double DQN

The DQN algorithm, which combines Q-learning with a deep neural network, suffers from substantial overestimations. Double DQN can not only reduces overestimation but can also improve performance.[2]

Prioritized Experience Replay

Experience replay allows reinforcement agents to reuse previous experiences but are uniformly sampled from memory. Prioritized experience replay allows memorized experience tuples to be prioritized based on temporal difference error and the sampling

[2] "Deep Reinforcement Learning with Double Q-learning." https://arxiv.org/abs/1509.06461. Accessed 16 Jul. 2019.

probability to weight each memory based on the calculated priority. Prioritized experience replay can significantly improve learning efficiency and performance.[3]

Dueling DQN

This neural network architecture includes two separate estimators one for the state value function and one for the state-dependent action advantage function. This allows generalized learning across actions without changing the underlying reinforcement algorithm while. A dueling DQN architecture can greatly enhance learning for environments that include similar-valued actions and lead to increased performance compared to a traditional DQN.[4]

Google Dopamine Framework

Google's Dopamine framework includes implementations of the Rainbow and variant C51 Rainbow algorithm. The Rainbow algorithm includes improvements to DQN which include Double DQN, Prioritized Experience Replay, Dueling DQN, Multi-step bootstrap targets, Distribution DQN, and Noisy DQN.[5] I considered included this framework which can be downloaded from the Dopamine git repository, but felt it's simplified integration may take away from some of the lessons intended for students completing this project.

[3] "Prioritized Experience Replay." https://arxiv.org/abs/1511.05952. Accessed 16 Jul. 2019.
[4] "Dueling Network Architectures for Deep Reinforcement Learning." 20 Nov. 2015, https://arxiv.org/abs/1511.06581. Accessed 16 Jul. 2019.
[5] "ml-agents/README.md at master · Unity-Technologies/ml ... - GitHub." https://github.com/Unity-Technologies/ml-agents/blob/master/gym-unity/README.md. Accessed 16 Jul. 2019.

Works Cited

Hasselt, van, et al. “Deep Reinforcement Learning with Double Q-Learning.” *ArXiv.org*, 8 Dec. 2015, arxiv.org/abs/1509.06461.

Schaul, et al. “Prioritized Experience Replay.” *ArXiv.org*, 25 Feb. 2016, arxiv.org/abs/1511.05952.

Sutton, Richard S. *Reinforcement Learning: an Introduction*. MIT Press, 2018.

Wang, et al. “Dueling Network Architectures for Deep Reinforcement Learning.” *ArXiv.org*, 5 Apr. 2016, arxiv.org/abs/1511.06581.