Navigation Report

Haoxiong Zhang

July 2020

1 Introduction

In this project, we will be training an agent to navigate and collect designated objectives (bananas) in a a large, square environment using Deep Q-Network (DQN). The project environment is similar to, but not identical to the Banana Collector environment on the Unity ML-Agents GitHub page. If you would like to know more about the environment, there is a detailed description in the README.md file of this project. For an agent to be considered successful in solving the environment, it must achieve an average reward (score) of 13 for 100 episodes.

2 Training the Agent

For this project, we will be reusing large portions of the solution to OpenAI Gym's LunarLander environment exercise with minor modifications to accommodate the Unity ML-Agents.

2.1 Learning algorithm and model architecture

The project will implement a simple model consisting of two fully-connected hidden layers (fc1 and fc2) with ReLU[Aga19] activation and an output layer (fc3). Each hidden layer contains 64 nodes and the output layer contains 4 nodes corresponding to the size of the action space of the environment.

2.2 Hyperparameters

Initially, the project uses the same hyperparameter values as the solution to the LunarLander environment. After successfully solving the environment, we attempt to optimize the model with minor modifications to the values of certain hyperparameters and observe the performance of the agent after these modifications. Some of these modifications include changing: the number of fully-connected hidden layers, the number of nodes in the hidden layers, the discount factor γ, the learning rate of the model, and the decay of ϵ in the epsilon-greedy

policy. However, we find that results of these modifications show no statistically significant improvement on the learning ability of the agent. Thus the final values of the hyperparameters are the same as their initial values.

2.3 List of hyperparameters and their values

Hyperparameter	**Value**	**Description**
BUFFER_SIZE	100,000	The size of the replay buffer in experience replay
BATCH_SIZE	100	Size of the mini-batch
GAMMA (γ)	0.99	Discount factor
TAU (τ)	0.003	Soft update of target parameters
LR	0.0005	Learning rate
UPDATE_EVERY	4	Frequency of network updates
eps_start	1.0	Starting value for ϵ in the epsilon-greedy policy
eps_end	0.01	Terminal value for ϵ in the epsilon-greedy policy
eps_decay	0.995	Rate of decay (exponential) for ϵ in the epsilon-greedy policy
num_layers	2	Number of hidden layers in the neural network
fc1_units	64	Number of nodes for the 1st hidden layer in the neural network
fc2_units	64	Number of nodes for the 2nd hidden layer in the neural network

2.4 Plot of Rewards per episode

After successfully training an agent to solve the environment using the model architecture and hyperparameters above, we can then plot the rewards obtained against the number of episodes.

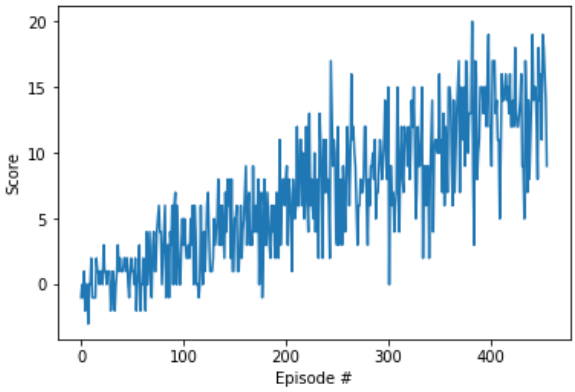

Figure 1: Plot of rewards (scores) per episode of a successful agent

As you can see in the figure 1, the agent is able to achieve an average reward (score) of 13 at around 400 episodes. As a matter of fact the agent achieved the average reward of 13 per 100 episode required to successfully solve the episode at 456 episodes, a satisfactory performance for a 'Vanilla' Deep Q Network.

3 Ideas for Future Work

The learning algorithm implemented in this project is a so-called 'Vanilla' Deep Q-Network, the original model proposed by 'Human-level control through deep reinforcement learning'[Mni+15] in 2015. Since the inception of the deep Q-network, there have been many studies that suggests modification to the 'Vanilla' deep Q-network to address certain problems in the original model. The 'Double DQN'[HGS15] addresses the 'The Overestimation Phenomenon' in deep Q-networks, a term attributed to the tendency to overestimate action values in Deep Q-Learning[TS99]. The 'Double DQN' selects the best action using one set of parameters but evaluate it using a different set of parameters, in the long run, this prevents the algorithm from propagating incidental high rewards that may have been obtained by chance, and don't reflect long-term returns. Several other suggested modifications to the 'Vanilla' DQN have proven to be able to significantly improve the learning ability of an agent in practice, modifications including 'Prioritized Experience Replay'[Sch+16], 'Duel DQN'[Wan+16], 'A3C'[Mni+16], 'Distributional DQN'[BDM17], and 'Noisy DQN'[For+19]. Each of these improved algorithms address a separate issue in the original deep Q-network model. Furthermore, in 2017, researches at Google DeepMind combined all six of the improvements above, the resulting learning algorithm was termed 'Rainbow' and it outperforms each of the individual modifications and achieves state-of-the-art performance on Atari 2600 games[Hes+17]. In addition to the robust Rainbow algorithm, principles of Transfer Learning can also be applied to further improve the Rainbow algorithm. All of the above modifications can

be implemented to the current model to improve the learning ability of the agent.

References

[TS99] Sebastian Thrun and Anton Schwartz. "Issues in Using Function Approximation for Reinforcement Learning". In: 1999. URL: https://www.ri.cmu.edu/pub_files/pub1/thrun_sebastian_1993_1/thrun_sebastian_1993_1.pdf (visited on 07/16/2020).

[HGS15] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". In: *arXiv:1509.06461 [cs]* (Dec. 2015). arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461 (visited on 07/16/2020).

[Mni+15] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". en. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14236. URL: http://www.nature.com/articles/nature14236 (visited on 07/16/2020).

[Mni+16] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *arXiv:1602.01783 [cs]* (June 2016). arXiv: 1602.01783. URL: http://arxiv.org/abs/1602.01783 (visited on 08/06/2020).

[Sch+16] Tom Schaul et al. "Prioritized Experience Replay". In: *arXiv:1511.05952 [cs]* (Feb. 2016). arXiv: 1511.05952. URL: http://arxiv.org/abs/1511.05952 (visited on 07/16/2020).

[Wan+16] Ziyu Wang et al. "Dueling Network Architectures for Deep Reinforcement Learning". In: *arXiv:1511.06581 [cs]* (Apr. 2016). arXiv: 1511.06581. URL: http://arxiv.org/abs/1511.06581 (visited on 07/16/2020).

[BDM17] Marc G. Bellemare, Will Dabney, and Rémi Munos. "A Distributional Perspective on Reinforcement Learning". In: *arXiv:1707.06887 [cs, stat]* (July 2017). arXiv: 1707.06887. URL: http://arxiv.org/abs/1707.06887 (visited on 08/06/2020).

[Hes+17] Matteo Hessel et al. "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *arXiv:1710.02298 [cs]* (Oct. 2017). arXiv: 1710.02298. URL: http://arxiv.org/abs/1710.02298 (visited on 07/16/2020).

[Aga19] Abien Fred Agarap. "Deep Learning using Rectified Linear Units (ReLU)". In: *arXiv:1803.08375 [cs, stat]* (Feb. 2019). arXiv: 1803.08375. URL: http://arxiv.org/abs/1803.08375 (visited on 08/06/2020).

[For+19] Meire Fortunato et al. “Noisy Networks for Exploration”. In: *arXiv:1706.10295 [cs, stat]* (July 2019). arXiv: 1706.10295. URL: http://arxiv.org/abs/1706.10295 (visited on 08/06/2020).