Deep Q-Learning on Unity's Banana Environment

Christopher J. Miles

December 21, 2018

1 Introduction

Deep reinforcement learning (in particular, the DQN algorithm) is applied to the Unity Banana environment [1]. The environment is a virtual 3D space where an agent is confined to a square region. The space is filled with randomly placed blue and yellow bananas. The objective is for the agent to collect as many yellow bananas as possible in the allotted time of 300 time steps. The agent receives a $+1$ reward for every yellow banana collected and a -1 for every blue banana collected.

To solve this problem, variants of the Deep Q-Learning algorithm are employed [5, 3, 2]. The details of each algorithms are discussed in the next section.

2 DQN Learning Algorithms

A 'vanilla' DQN [2, 4] is implemented as follows:

Initialize replay memory D with capacity N
Initialize action function $\hat{q}$ with random weights w
Initialize target action-value weights $w^- \leftarrow w$
for the episode $e \leftarrow 1$ to M **do**
    Initial input state x_1
    Prepare initial state: $S \leftarrow \phi(x_1)$
    for time step $t \leftarrow 1$ to T **do**
        Choose action A from state S using policy $\pi \leftarrow \epsilon{-}Greedy(\hat{q}(S, A, w))$
        Take action A, observe reward R, and next input frame x_{t+1}
        Prepare initial state: $S' \leftarrow \phi(x_{t+1})$
        Store experience (S, A, R, S') in replay memory D
        $S \leftarrow S'$
        Obtain minibatch of tuples (s_j, a_j, r_j, s_{j+1}) from D of size K.
        Set target $y_j = r_j + \gamma \max_a \hat{q}(s_{j+1}, a, w^-)$
        Update w: $\Delta w = \alpha(y_j - \hat{q}(s_j, a_j, w))\nabla_w \hat{q}(s_j, a_j, w)$
        Soft update w^-: $w^- \leftarrow (1 - \tau)w^- + \tau w$

The function $\phi(x)$ is a feature mapping from the raw input x to a vector with 37 dimensions representing ray-based perception of objects in the agent's forward direction. The hyperparameters were set to the following: buffer size $N = 10^5$, discount factor $\gamma = 0.99$, batch size $K = 64$, soft update parameter $\tau = 10^{-3}$, learning rate $\alpha = 5 \times 10^{-4}$. The network used to represent $\hat{q}$ is a fully connected neural network with an input layer of 37 nodes, a hidden layer of 64 nodes, a second hidden layer with 64 nodes, and the final output layer with 4 nodes (one for each action value). The activation functions were all ReLU functions with the exception of the final layer which is simply a linear combination without the addition of nonlinearity from an activation function.

The dueling DQN algorithm is identical to the algorithm above except that the neural network architecture is slightly different. The final stages of the neural network branch off to calculate the advantage of each action and state-value which is then used to determine the over all action-state values.

The double DQN algorithm is also a slight variation from the original DQN algorithm. the target $y_j = r_j + \gamma \max_a \hat{q}(s_{j+1}, a, w^-)$ is modified to $y_j = r_j + \gamma \hat{q}(s_{j+1}, a^*, w^-)$ where $a^* = argmax_a \hat{q}(s_{j+1}, a, w)$. This change improves stability of the algorithm.

3 Results and Plot of Rewards

The model discussed in the previous section was implemented and resulted in the following plot shown in Figure 1. In addition, variations of the the vanilla DQN were also implemented such as double DQN and dueling DQN. These results of these implementations are shown in figures 2 and 3. From these single trial runs, there isn't a noticable differences in the length of time to achieve the solved criteria: average reward of 13 over 100 consecutive episodes. DQN solved this criteria in just over 350 episodes, Double DQN solved it in about 275 episodes, Dueling DQN solved it in about 325 episodes.

4 Ideas for Future Work

In future work, I would like to implement prioritized experience replay. I would also like to try learn directly from raw pixel input data rather then imposing the ray-based feature map. If I had more computational resources, I would also like to run many trials of the DQN algorithms discussed here to provide statistical averages and variances to better compare these algorithms in the statistical sense. It's hard to say much from a few single trials of a stochastic process.

References

[1] Unity: https://unity3d.com/.

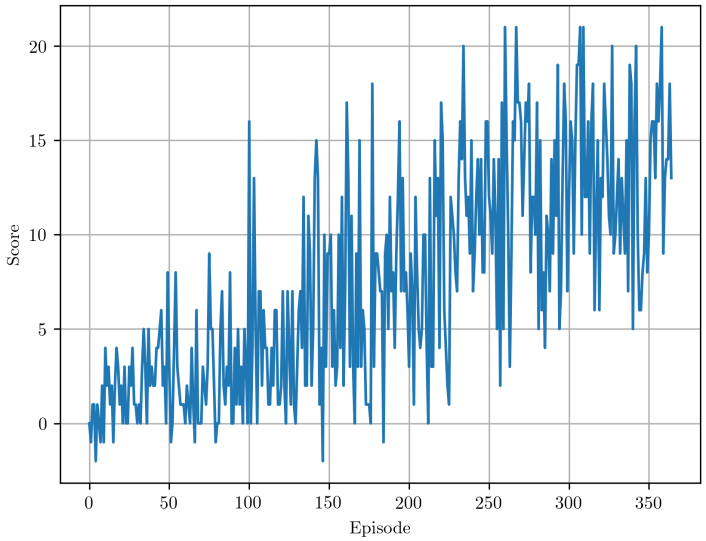

Figure 1: Scores over episodes during DQN training.

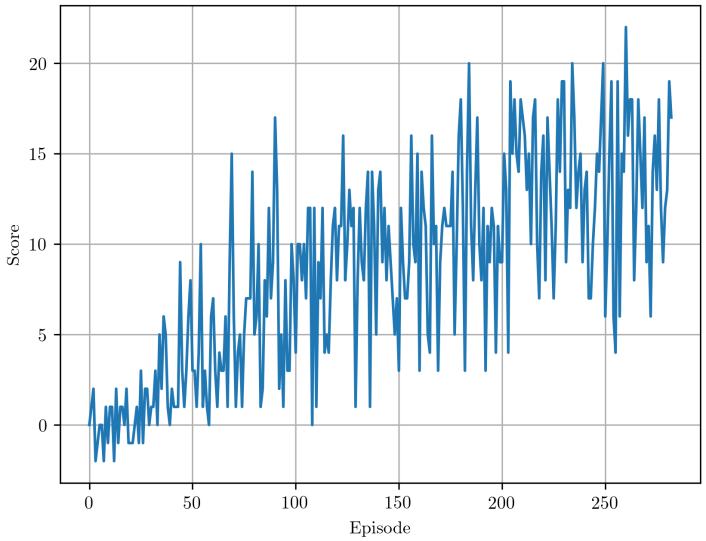

Figure 2: Scores over episodes during Double DQN training.

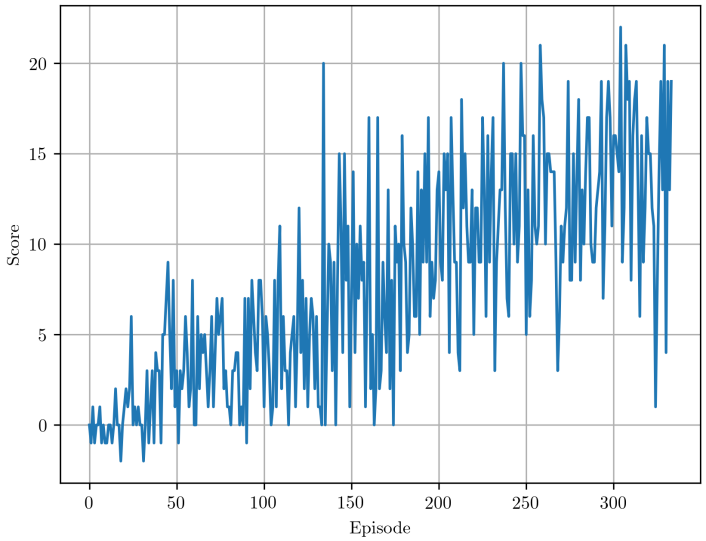

Figure 3: Scores over episodes during Dueling DQN training.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. *ICLR 2016*, pages 1–21, 2016.

[4] Udacity. Deep Reinforcement Learning Nanodegree Course Material, 2018.

[5] Ziyu Wang, Tome Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. *arXiv:1511.06581v3*, 2016.