Project 1: Deep-Q-Learning-for-Navigation

Udacity Deep Reinforcement Learning Nanodegree Program

Bob Flagg [*]

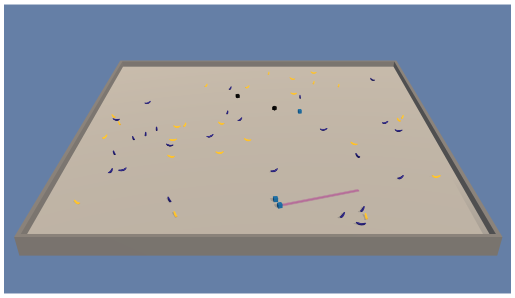

Figure 1: *Unity ML-Agent: Banana Collector*

Keywords: Reinforcement Learning, Deep Learning

1 Introduction

In this project I'll implement some solutions to the Unity ML-Agent Banana Collector environment. In this environment an agent observes a 37 dimensional vector containing the agent's velocity, along with ray-based perception of objects around the agent's forward direction and tries to learn how to best select one of the following actions:

1. move forward,
2. move backward,
3. turn left, and
4. turn right.

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 the agent is to collect as many yellow bananas as possible while avoiding blue bananas.

I'll give the following three solutions:

1. *Deep Q-Learning* [Mnih et al. 2015],
2. *Double Deep Q-Learning* [van Hasselt et al. 2015], and
3. *Dueling Deep Q-Learning* [Wang et al. 2015].

Source code in Python, using PyTorch, is available on github in the repo Deep-Q-Learning-for-Navigation.

2 Background

The Unity ML-Agent Banana Collector is a *sequential decision making problem*, in which an agent interacts with an environment over discrete time steps and tries to find a *policy* to maximize the expected *discounted return*:

$$G_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} R_\tau,$$

where $\gamma \in [0,1]$ is a discount factor that trades-off the importance of immediate and future rewards. See [Sutton and Barto 1998] for a general discussion of this sort of problem.

In this project I'll focus on a particular class of algorithms for solving sequential decision making problems called *Q-Learning*. The main idea behind Q-Learning is that if we had a function

$$Q^* : \mathcal{S} \times \mathcal{A} :\rightarrow \mathbb{R},$$

where $\mathcal{S}$ is the set of states and $\mathcal{A}$ is the set of possible actions, that could tell us what our return would be, if we were to take an action in a given state, then we could easily construct a policy that maximizes our rewards:

$$\pi^*(s) = \arg\max_a Q^*(s,a).$$

Unfortunately we don't know enough about the environment to compute $Q^*(s,a)$. One approach to overcome this problem is *Q-learning* [Watkins 1989], an early breakthrough in reinforcement learning. Q-learning approximates the optimal action-value function with a learned action-value function $Q(s,a)$, which is initialized randomly and then updated incrementally according to the formula

$$Q(S_t, A_t) \leftarrow (1-\alpha)\cdot Q(S_t, A_t) + \alpha\cdot \big[R_{t+1} + \gamma\cdot \max_a Q(S_{t+1}, a)\big].$$

In the standard Q-learning implementation Q-values are stored in a table. One cell is required per combination of state and action. This implementation is not amenable to continuous state problems like the Unity ML-Agent Banana Collector. The simplest way to get around this is to apply discretization but that scales poorly. As the number of state and action variables increase, the size of the table used to store Q-values grows exponentially. The next section describes an alternative to discretized Q-learning, which scales well.

3 Deep Q-Learning for Navigation

Deep Q-learning is an alternative to discretized Q-learning, which addresses the scaling problem by approximating $Q^*(s,a)$ with a *deep Q-network*, $Q(s,a|\theta)$, and tuning the parameters by optimizing the following of loss function:

$$L(\theta) = \mathbb{E}\big[\big(y^{DQN} - Q(S_t, A_t|\theta)\big)^2\big],$$

[*] Deep-Q-Learning-for-Navigation

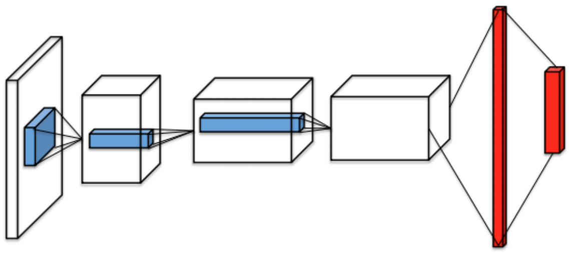

Figure 2: *Deep Q-Networt [Wang et al. 2015]*

with

$$y^{DQN} = R_t + \gamma \cdot \max_{a'} Q(S_{t+1}, a'|\theta^-),$$

where θ^- representes the parameters of a fixed and separate *target network*.

I've implemented deep Q-learning for the Unity ML-Agent Banana Collector in the notebook Deep-Q-Learning-for-Navigation. The Q-network has two fully connected linear layers each with 64 hidden units. I trained for 500 episodes with a batch size of 64, discount factor of $\gamma = 0.99$, soft update weight of $\tau = 0.001$ and learning rate of 0.0005. The score plot is below and shows the evironment was solved in about 429 episodes.

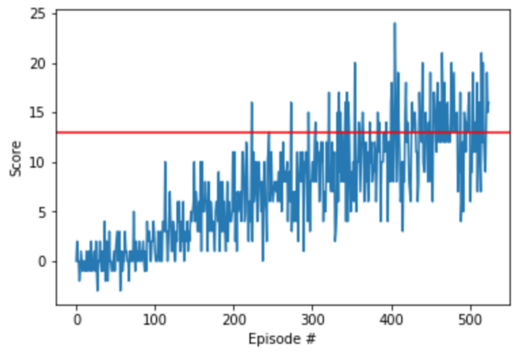

Figure 3: *Scores for Deep Q-Learning*

4 Double Deep Q-Learning for Navigation

In Q-learning and deep Q-learning, the max operator in the loss function uses the same values to both select and evaluate an action. This can lead to over optimistic value estimates. *Double deep Q-learning* mitigates this problem by using a different target:

$$y^{DDQN} = R_t + \gamma \cdot Q(S_{t+1}, \arg\max_{a'} Q(S_{t+1}, a'|\theta)|\theta^-).$$

I've implemented double Q-learning for the Unity ML-Agent Banana Collector in the notebook Double-Deep-Q-Learning-for-Navigation. I used a Q-network with two fully connected linear layers each with 64 hidden units. I trained for 500 episodes with a batch size of 64, discount factor of $\gamma = 0.99$, soft update weight of $\tau = 0.001$ and learning rate of 0.0005. The score plot is below and shows the evironment was solved in about 427 episodes.

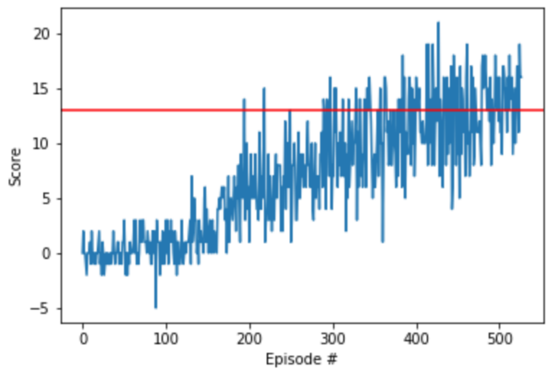

Figure 4: *Scores for Double Deep Q-Learning*

5 Dueling Deep Q-Learning for Navigation

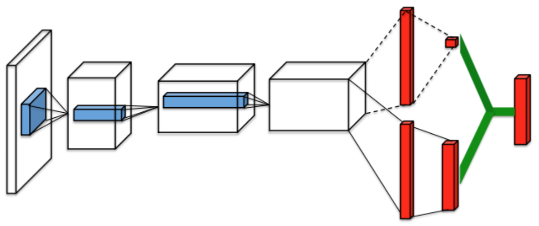

Figure 5: *Dueling Deep Q-Network [Wang et al. 2015]*

The architecture of the Q-network in dueling deep Q-learning has two streams, one, with paraameters θ, α, estimates the state-value function:

$$V(s) = \mathbb{E}\big[Q(s, A_t)\big]$$

and the other, with paraameters θ, β, estimates the expression

$$A(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a'),$$

where

$$A(s,a) = Q(s,a) - V(s).$$

is the *advantage function*, which provides a relative measure of the importance of each action for a given state.

The estimate of the action-value function is then

$$Q(s,a|\theta,\alpha,\beta) = V(s|\theta,\beta) - \big(A(s,a|\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a'|\theta,\alpha)\big).$$

As Figure 5 illustrates, the two streams share some weights, θ, but have separate heads with independent parameters, α and β. I don't use convolution in the network for the Unity ML-Agent Banana Collector so the shared θ parameters in this example are for a couple of simple fully connected layers.

Since the output of the dueling network is a Q function, it can be trained with the many existing algorithms. I tried Deep Q-Learning and Double Deep Q-Learning. In both experiments I used a Dueling Q-Network with a shared fully connected linear layer with 64 hidden units, a fully connected linear layer with 64 hidden units for the advantage head and a fully connected linear layer with 64 hidden units for the value head. I trained for 500 episodes with a batch size of 64, discount factor of $\gamma = 0.99$, soft update weight of $\tau = 0.001$ and learning rate of 0.0005. The score plots are below. Dueling Deep Q-Learning solved the environment in after 339 episodes.

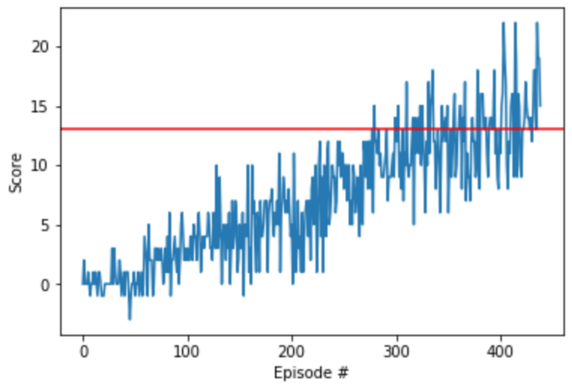

Figure 6: *Scores for Dueling Deep Q-Learning*

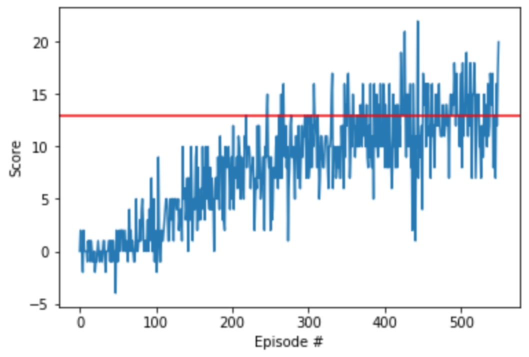

Figure 7: *Scores for Dueling Double Deep Q-Learning*

6 Improving Performance

Dueling Deep Q-Learning performed best in the experiments I did. To improve performance futher, I would do grid search on the hyper-parameters and network architecture for this approach. Another interesting direction would be *prioritized replay* [Schaul et al. 2015], which is built on top of DDQN. Their key idea is to increase the replay probability of experience tuples that have a high expected learning progress (as measured via the proxy of absolute TD-error). This has led to both faster learning and to better final policy quality across many environments.

References

MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M. A., FIDJELAND, A., OSTROVSKI, G., PETERSEN, S., BEATTIE, C., SADIK, A., ANTONOGLOU, I., KING, H., KUMARAN, D., WIERSTRA, D., LEGG, S., AND HASSABIS, D. 2015. Human-level control through deep reinforcement learning. *Nature 518*, 7540, 529–533.

SCHAUL, T., QUAN, J., ANTONOGLOU, I., AND SILVER, D. 2015. Prioritized experience replay. *CoRR abs/1511.05952*.

SUTTON, R. S., AND BARTO, A. G. 1998. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press.

VAN HASSELT, H., GUEZ, A., AND SILVER, D. 2015. Deep reinforcement learning with double q-learning. *CoRR abs/1509.06461*.

WANG, Z., DE FREITAS, N., AND LANCTOT, M. 2015. Dueling network architectures for deep reinforcement learning. *CoRR abs/1511.06581*.

WATKINS, C. J. C. H. 1989. *Learning from Delayed Rewards*. University of Cambridge.