RL-Course: Final Project Report

Sam Tureski (5390674), Fabio Helle (3928325)

March 16, 2021

1 Introduction

This report details the implementation of two reinforcement learning algorithms, Deep Deterministic Policy Gradient and Dueling Deep Q Networks. Our first goal is to train our models to perform in simple environments within the Gym toolkit (`https://gym.openai.com/`) such as Pendulum and LunarLander, and then we move on to the more challenging Hockey environment, in which our agents are expected to hit a puck and defend a goal while playing hockey against an computer-generated or human opponent.
Before our agents can play hockey in "normal mode," however, they must be trained to both handle the puck (in `TRAIN_SHOOTING` mode) and prevent the opponent from scoring a goal (in `TRAIN_DEFEND` mode). The task is especially challenging because there are 16 data points associated with each observation, which include but are not limited to: the position and velocity for both players, the velocity and position of the puck, the angular velocity for both players, and the angle of both players. The provided Jupyter notebook for the Hockey environment demonstrates the environment's basic usage, and offers options for both continuous and discrete gameplay.
Another goal of this project is to enter into a hockey tournament on a remote server, so that our agents can play the game against agents that our classmates have implemented. The highest scorers of this tournament receive bonus points for the project.

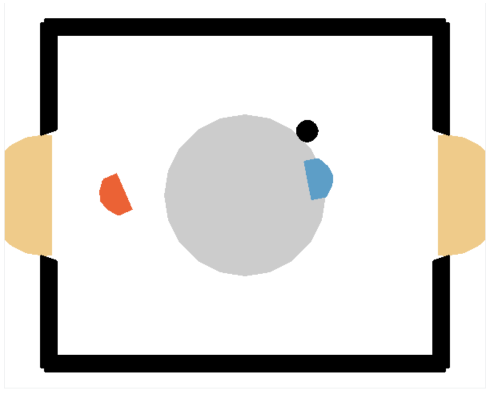

Figure 1: The Hockey environment

2 Method: Deep Deterministic Policy Gradient

Implemented by: Fabio Helle

A Deep Deterministic Policy Gradient model uses four neural networks: The actor network μ with weights θ^{μ}, which takes the current state of the environment as input and returns an action, thereby acting as a deterministic policy, the critic network Q with weights θ^{Q}, which takes a state-action pair as input and returns a value indicating how good the critic believed that action to be given the state, thereby modeling the Q function, as well as the target actor and the target critic networks μ' and Q' with weights $\theta^{\mu'}$ and $\theta^{Q'}$, which are slowly updated towards the actor and critic models, respectively, during each training step to provide more stability to the training process. In addition, the model has a replay buffer to store state transitions for training. Whenever a rollout is performed, each state transition (consisting of the previous state, the chosen action, the reward recieved and the next state) is stored in that replay buffer. To ensure a sufficient degree of exploration, we added both Gaussian noise with standard deviation = 0.05 as well as noise created by an Ornstein-Uhlenbeck process[1] to the action during training.

In our case, the actor network consists of an input layer with 18 neurons(the number of dimensions of states), a hidden layer with 100 neurons and an output layer with 3 neurons(number of dimensions of actions). The input and hidden layer use rectified linear unit as activation function, the output layer hyperbolic tangent to normalize the actions.
The critic network consists of an input layer with 21 neurons (state dimension + action dimension), a hidden layer with 150 neurons and an output layer with a single neuron. Like the actor network, the input and hidden layer used ReLU as activation function, but the output layer uses identity as no normalization is needed for the critic. As optimizer, we chose Adam with learning rate = 0.0002 for the actor and learning rate = 0.002 for the critic.
While those networks are rather small, this was necessitated by the limited amount of computation power available to us for training.

To train the actor and the critic networks, we first sample a batch of 256 transitions from the replay buffer. Due to this sample most likely consisting of non-consecutive state transitions from different training episodes, we get independently distributed data this way.
We update the actor network using the gradient calculated from that sample, as in [1]:

$$\nabla_{\theta^{\mu}} \approx \frac{1}{N}\sum_i \nabla_a Q(s, a \mid \theta^{Q}) \mid s = s_i, a = \mu(s_i) \nabla_{\theta^{\mu}} \mu(s \mid \theta^{\mu}) \mid s_i$$

The critic network is updated by minimizing the loss function

$$\frac{1}{N}\sum_i (r_i + \gamma Q'(\mu'(ns_i \mid \theta^{\mu'}) \mid \theta^{Q'}) - Q(s_i, a_i \mid \theta^{Q}))^2$$

with N being the size of the batch, in our case 256, and s_i, a_i r_i, ns_i respectively the state, action, reward and next state of the state transition i.

Both of the target networks are updated towards their respective network. For the target actor network:

$$\theta^{\mu'} \leftarrow 0.01 * \theta^{\mu} + 0.99 * \theta^{\mu'}$$

[1]https://github.com/vitchyr/rlkit/blob/master/rlkit/exploration_strategies/ou_strategy.py

And for the target critic network:

$$\theta^{Q'} \leftarrow 0.01 * \theta^{Q} + 0.99 * \theta^{Q}$$

During training, we used the provided rewards $reward\_closeness\_to\_puck$, $reward\_puck\_direction$ and $reward\_touch\_puck$ in addition to +10/-10 for goals as the reward function.

3 Performance: Deep Deterministic Policy Gradient

The performance of the Deep Deterministic Policy Gradient Model was limited by its rather small amount of neurons. Nevertheless, it can learn simple environments without issue. For instance, Gym's Pendulum environment (Pendulum-v0) is learned by the DDPG model after approximately 50 training episodes with a reward of around -250.

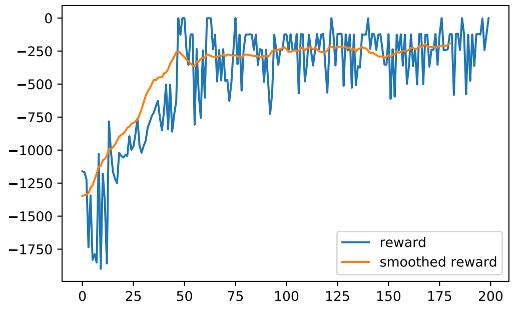

Figure 2: Reward for the Pendulum environment by episode

Running the Pendulum environment again for 100 episodes without training noise resulted in a mean error of -244.6 with empirical standard deviation 159.6
For the shooting mode of the hockey environment, training provides a slight but noticeable improvement, as seen in Figure 3.

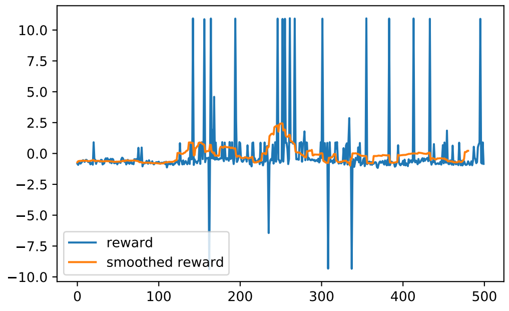

Figure 3: Reward for the shooting mode of the Hockey environment by episode

For the defense and playing modes, 500 training episodes proved insufficient for noticeable improvement.

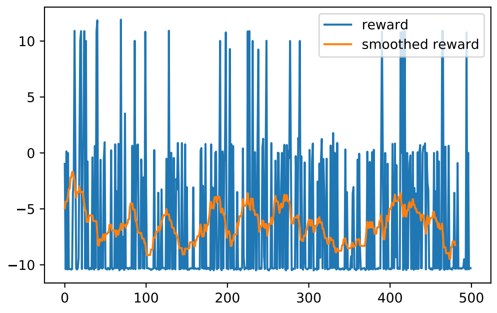

Figure 4: Reward for the defense mode of the Hockey environment by episode

4 Method: Dueling Deep Q Networks

Implemented by: Sam Tureski

As laid out by [3], given an observation of a state from an environment at a given timestep t, our agent is tasked with choosing from a set of possible actions an action a which results in a reward r. The agent seeks to maximize its total return over episodes R_t, which includes in its formula a discounting factor γ to account for the trade-off between immediate and future rewards.
In the vanilla Deep Q Learning algorithm as implemented first by [2], given an agent's policy π, we use neural networks to approximate the maximum Q-value for a state-action pair (s, a) from the Q-value function:

$$Q^\pi(s,a) = \mathbb{E}_\pi[R_t \mid s_t = s, a_t = a, \pi] \tag{1}$$

However, also defined within Reinforcement learning texts is a Value function V at state s:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi}[Q^\pi(s,a)] \tag{2}$$

And an Advantage function A, calculated by subtracting the Value Function from the Q-value:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{3}$$

Put plainly, the Value function measures the benefit of being in a specific state, the Q function shows how good it is to take a specific action within a state, and the Advantage function shows how good it is to take a specific action in a state, relative to all other actions.
In contrast with the typical Deep Q learning architecture, which seeks to approximate Q-values directly, dueling Deep Q learning explicitly separates state-value and advantage calculations into two separate streams (See the architecture in Figure 5). These streams are then aggregated to produce a final Q-value.

In order to be able to recover V and A uniquely, [4] note that the values cannot be aggregated by simply adding the values. Instead, one of the formulas they propose for generating the Q-value is:

$$Q(s,a;\theta,\alpha,\beta) = V(s,a) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a';\theta,\alpha) \right) \tag{4}$$

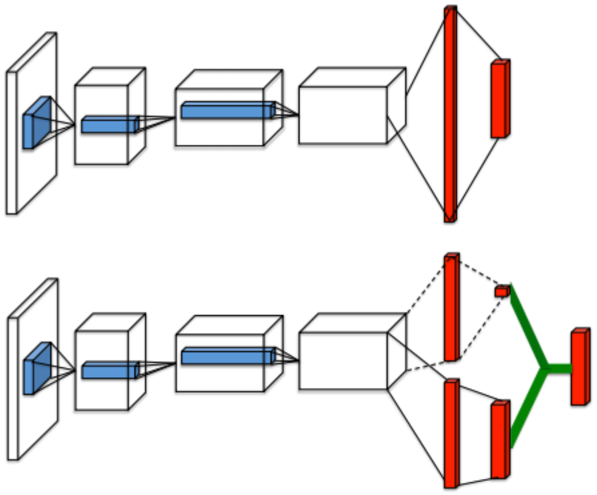

Figure 5: The architecture of a vanilla Deep Q Learning network (above) and a dueling Deep Q network (below), as presented in [4]. Note the two separate streams (state-value and advantage) that are aggregated to a final output layer.

Where θ is the parameters of the input layers, and α and β are "the parameters of the two streams of fully-connected layers" [4]. The intuition behind dueling Deep Q Networks is that, by separately estimating the state value function and advantage function, some states can be found to be more "valuable" than others: there may be states in which no action significantly effects the overall reward. In a racing-style game, for example, only states containing other cars may be important for the agent to learn actions for, so as to avoid collisions.

In our Pytorch implementation of Dueling Deep Q Networks, we follow in the steps of [4] by implementing an initial feature layer of size `[input_size, 512]` to collect the observations. Both the separate value and advantage streams are implemented as a single fully-connected layer with 512 units. The value stream has an output of size one, whereas the advantage stream has as many outputs as actions that can be taken. The two streams are then combined with the formula described in Equation 4.

We adopt an Adam optimizer with a learning rate of 0.0002 and a discount factor $\gamma = 0.95$.

As in vanilla Deep Q Learning, our implementation also contains a replay buffer to store/sample past experiences in/from memory, as well as a target network with the same architecture that independently predicts the Q-value for the timestep $t + 1$, which helps to maintain stability. The target network is updated every 20 iterations.

5 Performance: Dueling Deep Q Network

Our implementation of the dueling Deep Q Network was trained to perform as an agent in a number of environments, with varying levels of success.

The first checkpoint for this project was to get the model up and running in a simple environment. Given the nature of the algorithm, we felt it might be interesting to choose an environment in which not all states would be valuable, in the sense that the agent could learn an optimal action in those states. Thus, we train our model within the LunarLander-v2 environment, in which a small spacecraft must be guided to land between two flags. States after the rover has touched down on the "moon," for example, may not be so important: it is very difficult to move once we have landed. It was hypothesized that the dueling DQN would learn more quickly than a vanilla DQN, because the split learning of V and A values could make training more efficient.
We present a comparison of the two models' rewards for the LunarLander task over 1000 episodes in Figures 5 and 6. Here, the dueling DQN learns much more quickly, with its smoothed reward rising almost immediately over -200, while the vanilla DQN requires about 100 episodes for this. However, the dueling DQN remains at this level for much longer, due potentially to the fact that it is learning to undervalue certain states, where in reality all states before the lander hits the ground should be considered.
Overall, our dueling Deep Q Network agent learned to avoid strong negative rewards quickly, but had difficulty maintaining an average reward much higher than 0, most likely because it did converge to a hovering behavior for some of the episodes. The next part of the project involved a simulated hockey game against an opponent in the LaserHockey environment. Before playing a game, our agent was trained to shoot the puck towards the opponent's goal and to defend its own goal. In general, our agent learned to shoot well, with reward rising dramatically after about 200 training episodes (see Figure 7). However, reward also sharply decreased for some episodes after this point, potentially because shots were ricocheting off of the sides of the field and into our own goal. Therefore, the mean reward never breached a score of 1000, despite frequent scores of over 4000 points. Our agent was also trained to defend the goal. Unfortunately, our agent could not seem to learn defense even over 1000 episodes of 500 steps each (see Figure 8), which may have been due to either lack of sufficient training iterations or issues with the reward function.
In regards to being able to successfully play hockey against a preprogrammed virtual opponent, our dueling DQN-based agent scored around 10% of the time, though many of these "shots" were seemingly the puck bouncing off of our agent from out opponent's shot.

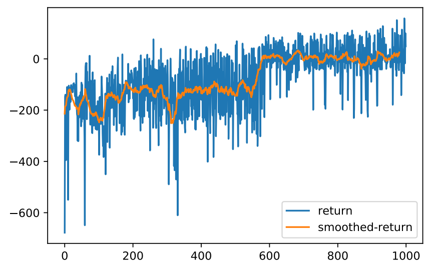

Figure 6: The dueling DQN agent in the LunarLander environment, reward over episodes

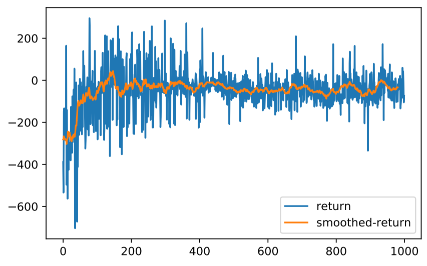

Figure 7: The vanilla DQN agent in the LunarLander environment, reward over episodes

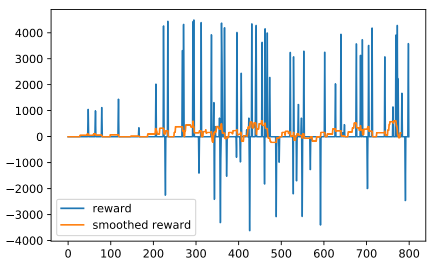

Figure 8: The dueling DQN agent in the LaserHockey environment shooting mode, reward over episodes

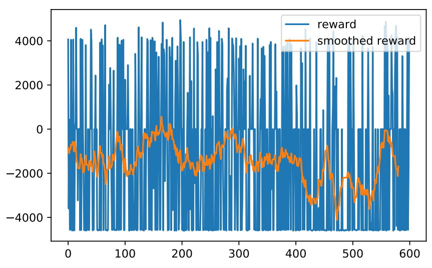

Figure 9: The dueling DQN agent in the LaserHockey environment defense mode, reward over episodes

6 Discussion

In the end, both of the algorithms successfully learned simple environments and had some success learning how to shoot in the Hockey environment, though both struggled to learn defense and play well against an opponent.
Both the DDPG agent and the Dueling DQN agents were ultimately not able to learn the Hockey environment. Since they were able to learn a simple like the Gym's Pendulum-v0 or the LunarLander-v2 without noticeable issues, this was most likely not caused by a flaw in the algorithm or the implementation itself, but by a combination of the small size of the model's neural networks and a lack of training. According to their presentations, most of the models that did well in the tournament required 30000+ training episodes, which was not possible for us to do in any reasonable amount of time. This lack of computational power (models were trained with a 1.6 GHz Intel Core i5 processor or similar) also made searching for good hyperparameters difficult.

An environment like the Hockey one requires a rather large amount of training, among other reasons, because without using the proxy rewards from the info-dictionary provided by the environment, the critic network at first has no way of telling whether any given action is good or bad unless the goal is hit by coincidence, which requires a lot of exploration and therefore a lot of training episodes. While using the proxy rewards somewhat alleviates this issue, it encourages counterproductive behavior in the long run, i.e. the agent blocking its own shot because it wants the reward for touching the puck.
Another issue we ran into during training in shooting mode were episodes in which the agent shot the puck at the goal, barely missed and had the puck ricochet into its own goal, thus getting a strong negative reward for actions that were very close to the optimal ones. If such an episode occurred near the beginning of training, we usually hat to restart, since the agent would learn to stay away from the puck, the exact opposite of what it is supposed to do during shooting.
We predict that, given suitable computational power to train and tune hyperparameters, both of our models would have successfully learned to play hockey against an opponent. The DDPG algorithm would be preferred over the dueling DQN, because it is specifically optimized to perform in continuous spaces like the Hockey environment. Furthermore, dueling DQN is a better choice when an environment has some states where actions can be ignored, though it hockey it may be best to always actively move towards the puck or move defensively.

References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[3] R. S. Sutton, A. G. Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[4] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.