"TENNIS COMPETITION"

by

Ruben Kapp

A project submitted in partial fulfillment
of the requirements for the degree of

Deep Reinforcement Learning
Nanodegree

UDACITY

2021

UDACITY

ABSTRACT

Navigation Agent

by Ruben Kapp

A report about the project how to use reinforcement learning to train an agent to play tennis. A multi agent training approach was used to increase training speed. The continuous action-space was solved using a Deep Deterministic Policy Gradient (DDPG) implementation. The agent learned by playing against himself.

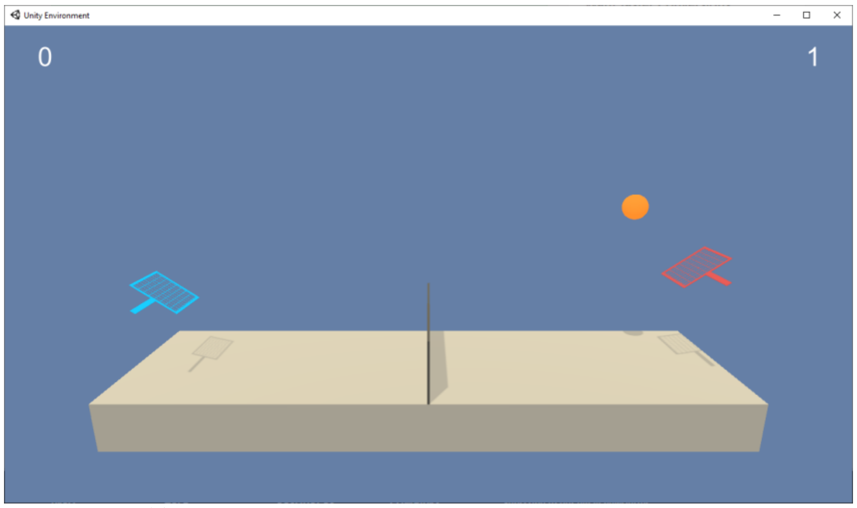

Figure 1. Tennis environment. Self-trained agent in competition environment.

TABLE OF CONTENTS

Content

LIST OF FIGURES

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to UDACITY and the whole AI community. In addition, special thanks to https://stackoverflow.com/ and all open sources which have been very helpful during the early programming phase and understanding of this undertaking.

CONTINUOUS CONTROL PROBLEM AND CHALLENGES

Continuous Control Problem:

As in contrast to desecrate action spaces like where you can have a limited variation and possibilities of possible actions to select on, the continuous control problem cannot be limited in this regard. Thus, the possible action space is infinite or at least, if you try to discretize, would lead to an action space to large to be handled with a Q-table based solution approach. As *Timothy P. Lillicrap et.Al 2019* pointed out in their paper about continuous control *"… while DQN solves problems with high-dimensional observation spaces, it can only handle discrete and low-dimensional action spaces".* Hence a DQN algorithm is not suitable for this problem. The solution they presented is an "*model-free, off-policy actor-critic algorithm using deep function approximators that can learn policies in high-dimensional, continuous action spaces*". This solution approach was used within this project to solve the continuous control environment.

Goal – Tennis Environment

In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play.

The task is episodic, and in order to solve the environment, your agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically,

- After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. We then take the maximum of these 2 scores.

- This yields a single score for each episode.

The environment is considered solved, when the average (over 100 episodes) of those scores is at least +0.5.

METHODS

DDPG - Deep Deterministic Policy Gradient Agent

In DDPG, two deep neural networks are used, one is called the actor and the other the critic. The actor is used to approximate the optimal policy deterministically. That means the actors goal is to always output the best believed action for any given state which leads to a deterministic policy. The actor is basically learning the **argmax** Q(State, action) function, hence which is the **best** action. The critic learns to evaluate the **optimal action value function** by using the actors best believed action. The actor, which is an approximate maximizer, is therefore used to calculate a new target value for training the action value function, hence the critic.

DDPG uses a replay buffer and a soft update strategy to the target networks. In DDPG, there are is a copy for each trained network. Therefore, a regular for the actor and the critic, and a target for the actor the critic. The target networks are now updated using a soft update strategy. This means to slowly blending the regular network weights into the target network weights. Therefore, every time step the target network will only update a 0.01 percent of the regular network weights. So the regular network weights will slowly merged into the target network weights.

DDPG architecture

The actor and critic structures are listed below:

- state_size = 33
- fc1_units = 128
- fc2_units = 128
- action_size = 4

```
actor (
            self.fc1 = nn.Linear(state_size, fc1_units)
            self.fc2 = nn.Linear(fc1_units, fc2_units)
            self.fc3 = nn.Linear(fc2_units, action_size)
      )

critic(
      self.fcs1 = nn.Linear(state_size, fcs1_units)
      self.fc2 = nn.Linear(fcs1_units+action_size, fc2_units)
      self.fc3 = nn.Linear(fc2_units, 1)
```

As activation function a LeakyReLu was used.

Hyperparameters

Replay buffer size:

- BUFFER_SIZE = int(1e6)

Minibatch size:

- BATCH_SIZE = 128

Discount factor for update rule:

- GAMMA = 0.99

For soft update of target parameters:

- TAU = 1e-3

Noise Parameter long-running mean (Ornstein Uhlenbeck):

- MU = 0.0

Noise Parameter the speed of mean reversion (Ornstein Uhlenbeck):

- SIGMA = 0.2

Noise Parameter the volatility parameter (Ornstein Uhlenbeck):

- THETA = 0.15

Noise added to act step:

- NOISE_FACTOR = 1.0

Decay rate for noise process:

- NOISE_DECAY = 1e-6

As optimization algorithm we use the pytorch implementation of the adam optizer [ADAM].

L2 weight decay:

- WEIGHT_DECAY = 0

Learning timestep interval:

- LEARN_EVERY = 20

How often to execute the learn-function each EARN_EVERY steps:

- LEARN_NB = 10

Learning rate for ADAM optimizer:

- LR_ACTOR = 1e-3
- LR_CRITIC = 1e-3

How often to update the main-network:

- UPDATE_EVERY = 4

How often to update the target-network:

- UPDATE_TARGET_EVERY = 110

RESULTS

The goal was to reach an mean score over +0.5 over the last 100 episodes. Using the given model and hyperparameter the mean average over all agents reached the

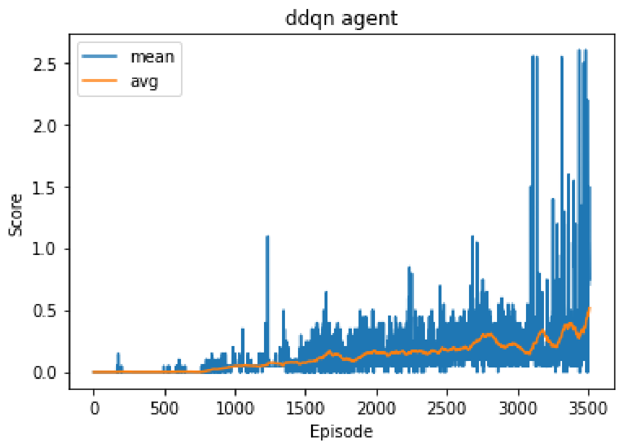

Figure 2. Mean and average score over episodes

target score after 3500 training episodes (see Figure 2. Mean and average score over episodes).

IMPROVEMENTS

IMPROVEMENT 1

For possible improvement a cross-normalization could be used. Cross-normalization introduced by *Aditya Bhatt et.Al. 2019*, stabilizes the training and improves the results in terms of the reward achieved. Moreover, it increased stability sufficiently to enable training without the need for target networks.

Using normal Gaussian Noise to test if *Ornstein Uhlenbeck* noise has a positive influence or not. As *Scott Fujimoto. 2018* pointed out "*Afterwards, we use an off-policy exploration strategy, adding Gaussian noise N (0, 0.1) to each action. Unlike the original implementation of DDPG, we used uncorrelated noise for exploration as we found noise drawn from the Ornstein-Uhlenbeck process offered no performance benefits.*"

SOURCES & LITERATURE

[Timothy P. Lillicrap et.Al, 2019]
Timothy P. Lillicrap∗, Jonathan J. Hunt∗, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver & Daan, 2019
Google, Continuous Control with deep reinforcement learning https://arxiv.org/pdf/1509.02971.pdf

[Scott Fujimoto, 2018]
Scott Fujimoto, Herke van Hoof, David Meger, 2018.
Addressing Function Approximation Error in Actor-Critic Methods
https://arxiv.org/pdf/1802.09477.pdf

[Aditya Bhatt et.Al. 2019]
Aditya Bhatt, Max Argus, Artemij Amiranashvili, Thomas Brox, 2019, CrossNorm: On Normalization for Off-Policy TDReinforcement Learning. https://arxiv.org/pdf/1902.05605.pdf

[ADAM]
ADAM torch.optim, https://pytorch.org/docs/stable/optim.html