Project Navigation

Introduction

This is a summary of my implementation of a Deep Reinforcement learning agent to solve the continuous control problem which is a task of the Udacity Deep Reinforcement learning Nanodegree program. The task is to control a robot arm and to teach the robot to follow a moving location, which is highlighted by a green sphere (see Figure 1). Further information about the environment and how to use the code is described in the readme file of this repository.

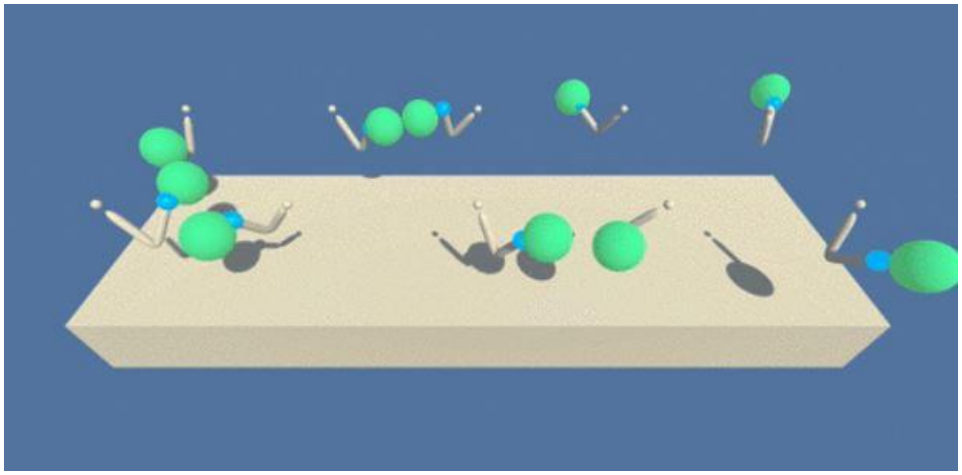

Figure 1: Example screen of the environment

Actor Critic Reinforcement learning algorithm

Algorithm and Extensions

Actor Critic Reinforcement learning algorithms are a category of Reinforcement learning (RL) methods that try to combine the advantages of policy and value based approaches. Policy based approaches are trying to estimate the best policy directly while value based methods provide a value estimate that is used to determine and select the best action. Policy based algorithms have less bias but a larger variance than value based algorithms. The goal of the combined RL approach is to reduce variance by still having a low bias. This results in more stable agents that converge faster than a policy-only agent.

Reinforcement learning implementation

As Reinforcement learning algorithm, I have applied a Deep Deterministic Policy Gradient (DDPG)[1] to solve the given task. This algorithm consists an Actor and a Critic whereof the Actor determines a policy deterministically. In the next step, the Critic takes the action determined by the Actor as input to evaluate the decision of the Actor (see Figure 2). That way the Agent training is more stable at the Agent is also able to provide a continuous action value prediction.

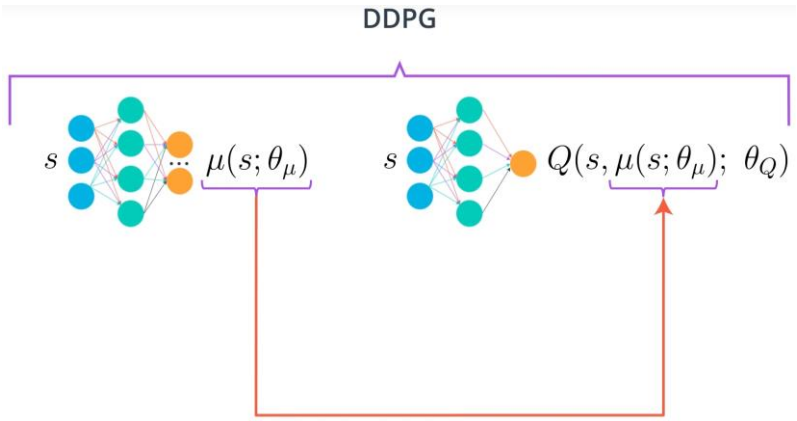

Figure 2: Sketech of DDPG algorithm

The implementation of DDPG does also contain a Replay Buffer to sample a batch of training data from a memory of past experiences. Additionally, Actor and Critic do have a local and target network to reduce correlation during the training. The target network receives a soft update of the local network parameters after a fixed amount of steps. Randomness for exploration is added with additional noise generated by an Ornstein-Uhlenbeck process. The noise implementation can be adjusted to the problem, by modification of parameters. This is an overview of the parameters used in my implementation:

Parameter type	Parameter	Value
OU noise parameter	Mu	0.00
OU noise parameter	Theta	0.15
OU noise parameter	Sigma	0.05
Neural network setup	Number of hidden units 1	400
Neural network setup	Number of hidden units 2	300
Neural network setup	L2 weight decay critic	0
Memory buffer setup	Memory buffer size	10000
Memory buffer setup	Batch size for learning	1024
Memory buffer setup	Number of steps between learning	20
Learning speed factor	Discount factor	0.99
Learning speed factor	Network learning rate actor	1e-3

Learning speed factor	Network learning rate critic	1e-3
Learning speed factor	Target network soft update factor	1e-2

Results

Evaluation methods

The total score achieved in one episode measures the agent's performance. The goal is to achieve a score larger than 30 within a minimum training time. However, there is randomness in the setup and the training process, which results in score fluctuations. Hence, an average score over the last 100 episodes is used to measure success. The environment is considered to be solved if the agent has reached an average score larger than 30.

Agent scores

The score distribution of the final learning agent is presented in Figure 3. The blue line shows the individual scores in each episode and the orange line indicates the target score, namely the average score over the last 100 episodes. After an exploration phase until approximately episode 50, the agent begins to learn quickly until episode 250. After that, the learning slows down and the improvements are less significant until the environment is finally solved in episode 534.

Plot of rewards

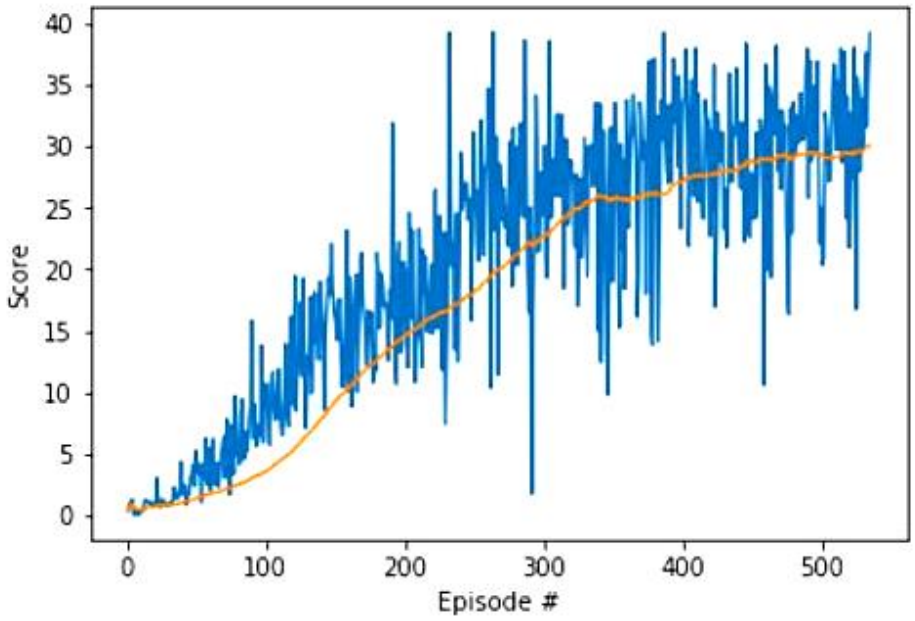

Figure 3: Training scores of the final agent (blue = 1 episode, orange = avg. 100)

Extensions for performance improvements

Parameter optimization

A variation of some parameters was tested during the training, but a grid search still offers potential improvement in the learning rate. I have realized that parameters that balance the exploration and exploitation of the agent have a strong impact on the learning rate. This could be even further improved by testing a decaying noise values that decreases with training time.

Further approaches for parallelized learning

The Reacher environment has a second version that allows training up to 20 agents in parallel. This is beneficial for algorithms like the Asynchronous Advantage Actor-Critic (A3C)[2] algorithm which is suited for parallel training.

1) https://arxiv.org/abs/1509.02971

2) https://arxiv.org/abs/1602.01783