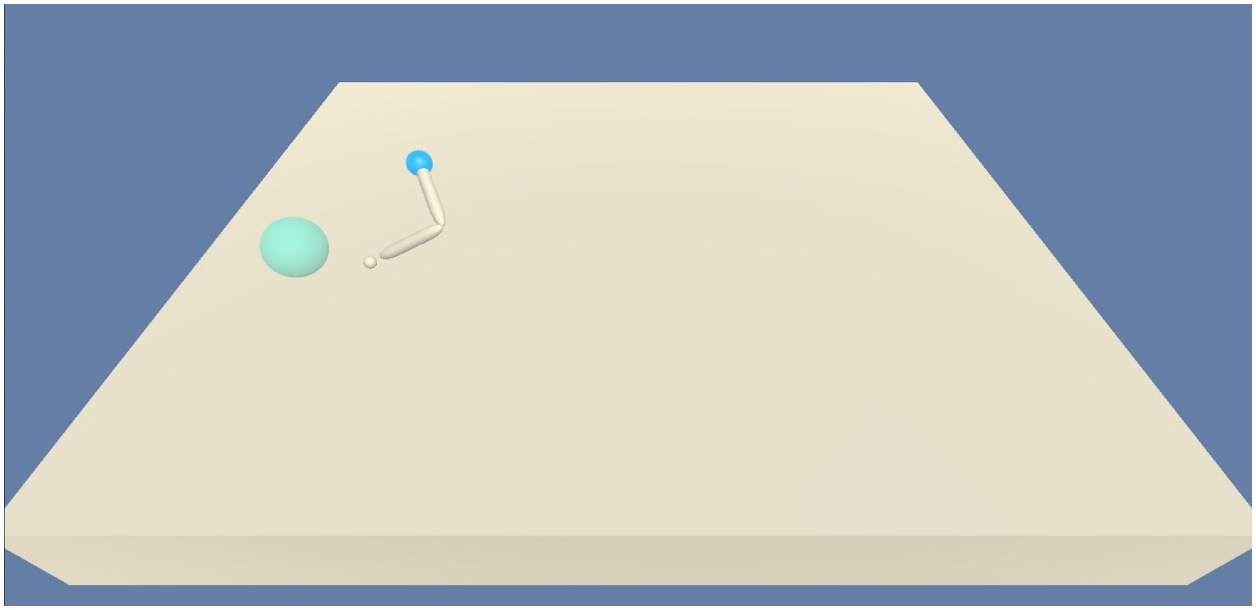


Train Report

Continuous Control with DDPG

REDZHEP MEHMEDOV REDZHEBOV

22/04/2020

Introduction

In this project, D3PG(Delayed Deep Deterministic Policy Gradient) algorithm is used to solve the Reacher environment. DDPG is an Actor-Critic algorithm for continuous action spaces. The model training strategy is similar to the classical Deep Q-Learning algorithm but it depends on quick updates instead of waiting for very long observation periods. Both actor and critic model implementations are inspired by classic DQN implementation. Thus, every actor and critic modules have one local and one target neural network.

Implementation Details

Target models are updated once in every “3” step. During model initialization, the initialized local model’s weights are copied into the target model’s weights to prevent initialization based on high model difference problems. Critic-Networks have a Dropout layer before the last fully connected layer as regularizer.
Gradient clipping is applied to the local-critic network during training to avoid exploding gradient problems.

Neural Networks :
Actor Networks : State_Size -> 256 -> 128 -> action_size
Output Gate of Actor Network : Tanh [-1, 1]
Learning Rate : 1e-4

Critic Networks : State_size -> 256 -> 256 + action_size -> 128 -> 1(value)
Output Gate of Critic Network : Linear f(x) = x
Learning Rate : 2e-4
Dropout : 0.2

Batch Size : 256
Buffer Type : Classic Experience Replay Buffer
Buffer Size : 1e6

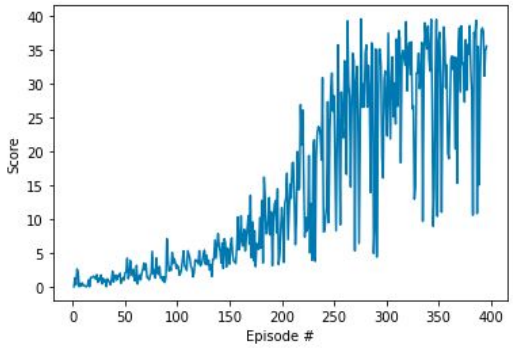
Model Update Rate (Tau) : 1e-3
Weighted Decay : 0
Discount Rate : 0.99

SEED : 1337(LEET FTW!)

Results

Environment is solved in 396 epochs and the targeted average score was 30.0.

```
Episode 100     Average Score: 1.69     Score: 3.19
Episode 200     Average Score: 5.95     Score: 11.67
Episode 300     Average Score: 20.68    Score: 25.24
Episode 396     Average Score: 30.05    Score: 35.49
Environment solved in 396 episodes!     Average Score: 30.05
```



Further Work

- Distributed Training to collect different observations faster. It will allow the agent to learn faster.
- Hyper-parameter Optimization: Better initial hyper-parameters can lead the agent to learn faster.
- Prioritized Replay: Instead of sampling a chunk of observations randomly, we can sample them based on the observation's quality. Observation's quality can force the agent to focus on the cases that it failed very badly. It is kinda adding extra intuition into the model