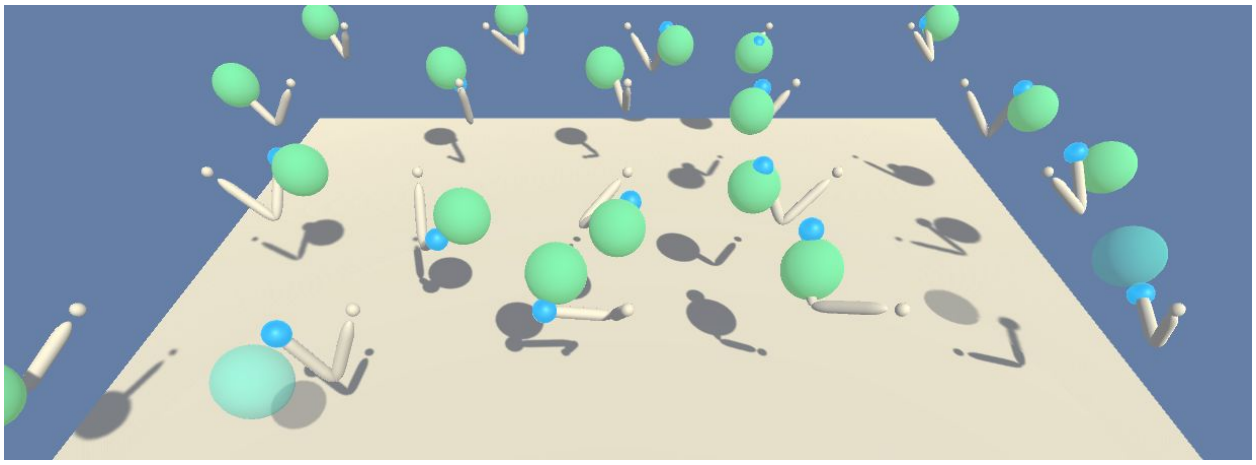


REACHER DDPG AGENT

Overview

This project is an assignment from the Udacity Deep Reinforcement Learning course. [p2_continuous-control] found here. The objective is to train the Unity reacher agent to keep its arm in a rotating target. The environment consists of a state space of 33 (continues) and action space of 4 (continues). There are two variants, one with 1 agent and one with 20 agents. The environment is considered solved if the agents achieve an average score of 30+ over 100 episodes. To solve the environment the DDPG algorithm was used to train the agent.

Implementation

1. agent.py
Contains the agent that is trained, it's learning code and the experience replay.

2. model.py
Contains a pytorch model wich is trained in Agent.py

3. Training notebook.ipynb
Contains the code to train an agent and save its model

4. Evaluation notebook.ipynb
Contains code to load the model evaluate its performance

Learning algorithm and model architecture

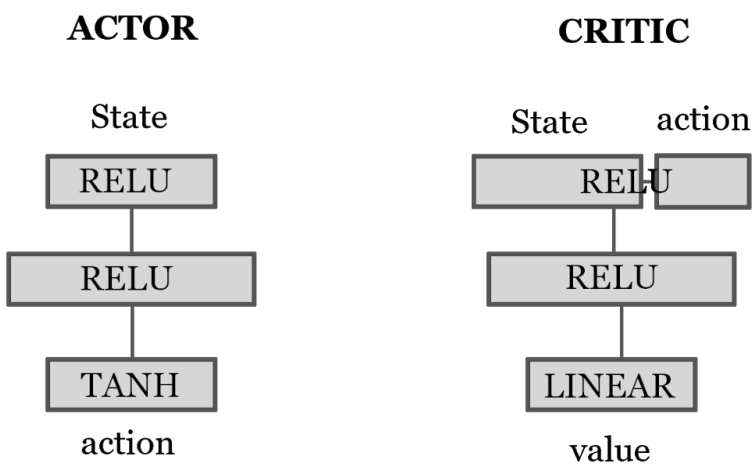
The model consists of an actor and critic network trained using the DDPG algorithm.
Both the state and last state are passed into the network doubling the input space, which is done to improve performance.

The learning algorithm used is DDPG. Which trains two networks, one actor and one critic. Where the actor network learns an estimated optimal action to take by estimating an optimizer over the value function. And the critic which tries to estimate the value function over the current estimated best action.

Learning is done every step and uses a replay buffer and soft update to a target model.

The hyper parameters used are:

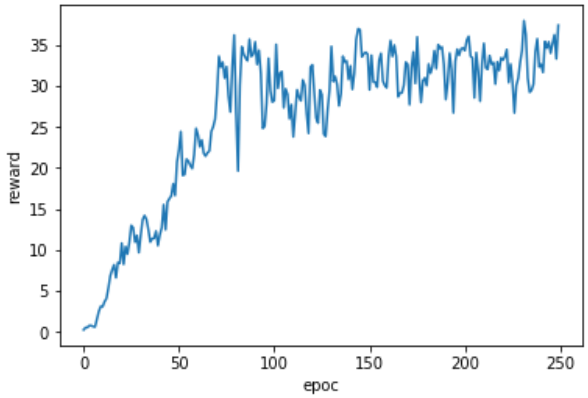
```
ACTOR_LEARNING_RATE = 1e-4
CRITIC_LEARNING_RATE = 1e-3
DISCOUNT_RATE = 0.99
TAU = 0.01
```



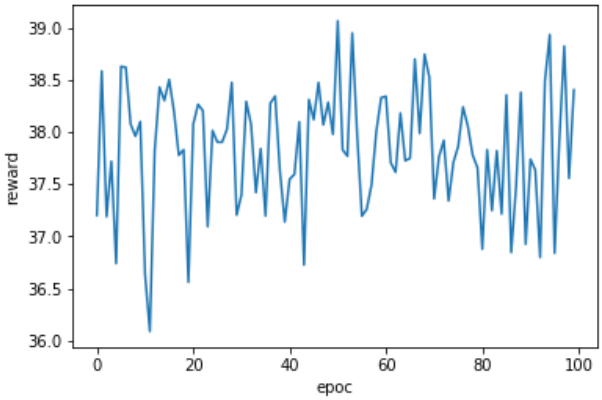
Performance

Training epochs are cut short at 1000 steps, whereas the testing runs until the end of an episode. the average score over 100 episodes is: **37.84** which is achieved at around 100 training epochs.

1.



2.



(1. Training performance, 2. Testing performance)

Improvements

Improvements can probably be made in both model performance as well as training. Since the problem is highly temporally correlated an RNN or other architecture aimed at dealing with time series data would likely improve performance. As for training the model occasionally gets stuck so a form of noise to aid exploration could guarantee convergence.