report.md

Leticia Lorena Rodríguez - Decembre 2018 - Deep Reinforcement Learning Nanodegree - Udacity

Report: Continous Control Project

Introduction

This project solves the Unity3D environment Rearcher for 20 agents.

Solution

It's solve the environment using a DDPG Agent. The agent uses Actor and Critic networks defined in the following way:

	Actor			
Layer		Input size	Output size	Activation
1	Linear	33	512	
	Batch Normalization	512	512	Relu
2	Linear	512	512	
	Batch Normalization	512	512	Relu
3	Linear	512	512	
	Batch Normalization	512	512	Relu
4	Linear	512	512	
	Batch Normalization	512	512	Relu
5	Linear	512	512	
	Batch Normalization	512	512	Relu
6	Linear	512	512	
	Batch Normalization	512	512	Relu
7	Linear	512	4	Tanh

	Critic			
Layer		Input size	Output size	Activation
1	Linear	33	512	Relu
2	Linear	512+4	512	Relu
3	Linear	512	512	Relu
4	Linear	512	512	Relu
5	Linear	512	512	Relu
6	Linear	512	512	Relu
7	Linear	512	1	Linear

The DDPG Algorithm is defined:

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

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

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

The parameters used are:

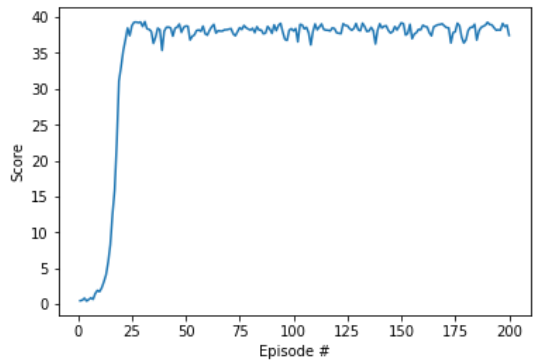
```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 128        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 1e-5         # learning rate of the actor
LR_CRITIC = 1e-4        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay
```

I've tried with different learning rate but the training just went slow without any increase.

Also, I've used 2048 as input and output size in the network without great results.

I've noticed that learning rate has a strong impact in learning.

The plot of the rewards is:



It's showing that after around 20 episodes the average reward is more than 30 as expected.

The project also includes when the solution goes above 30 for the last 100 episodes:

Average score in the latest 100 episodes: 32.36344927662052

After 101 episodes.

Future work

Some ideas for future work are:

- Try solving one agent with DDPG
- Try implementing PPO, A3C, and D4PG
- Tunning network training: changing learning rates, optimizers, etc.