Multi Agent – Work description

Learning Algorithm

The learning algorithm is the Deep Deterministic Policy Gradient with two Agents and shared memory.

DDPG – this algorithm is used for continues state / action space. It uses Actor – Critic technic for differentiable during continues learning.

I try only simple Replay Memory buffer

Neural network – I used two neural networks, local and target with Polyak method of coping the weights. This is the partial coping from local to target network. Where the target network is used to predict the action from next state. The network contains two hidden layers with 512, 256 respectively. I also try the one size hidden layer but no success. I try different setup of different size of network. This just shows that 2:1 is the best I found.

Hyper param

The hyper-parameters

I choose
BUFFER_SIZE = 100000 # replay buffer size
BATCH_SIZE = 512 # minibatch size
GAMMA = 0.991 # discount factor
TAU = 1e-3 # for soft update of target parameters
LR = 1e-4 for Agent and 3e-4 for Critic optimizer # learning rate
UPDATE_EVERY = 3 # how often to update the network

It was really challenging to resolve such un unstable environment. I test the warmup methods they just get sucked at some random number a round 0. I either try algorithm TD3 – Twin Delay DDPG, this algorithm used second Critic and should perform more stable. But either with this I was not success.

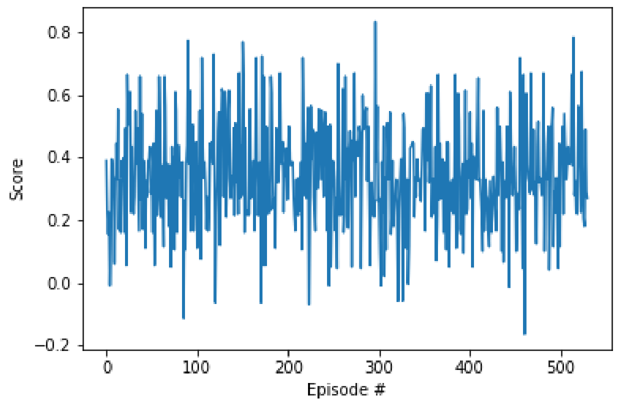

Figure 1TD3 with WarmUp

I try even bigger network size with 1024 to 512. This was slower in learning but get either unstable with no learning.

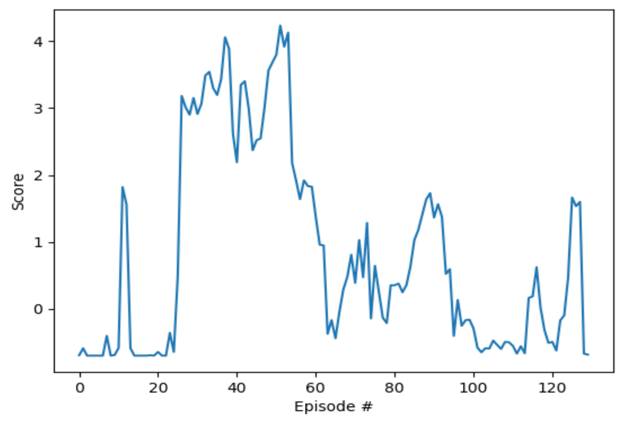

Figure 2Newtork size of 1024 + 512

The key point was to set proper initiation of weights. The default Pytorch initiation use the uniform distribution based on number of incoming nerons. The values of borders for distribution is calctulated as 1/ square root of number of incoming neurons. If I compare the range between pytorch default values and values made from outgoing numbers they made difference : 0.0416 compare to 0.0019. This is very noticeable in environment like this which is very unstable.

I change the default init weights and change size(1) to border = 1. / math.sqrt(layer.weight.size(0))

I also use the Ornstein–Uhlenbeck, which describe the Brownian move and fits better for this kind of actions than regular normal distribution.

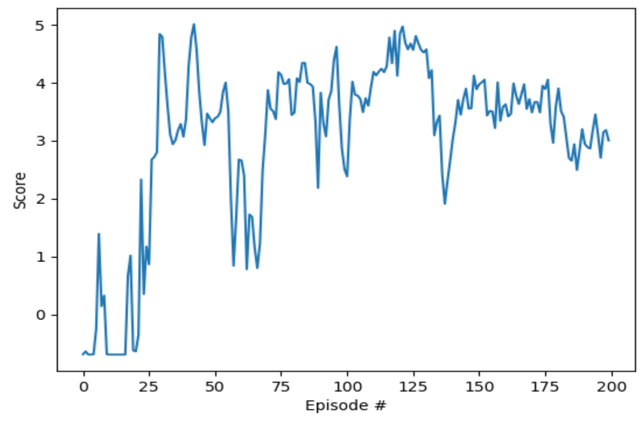

Figure 3 Size of 512 + 256 and init weights

I actually never try this kind of improvement. This was very new to me and I was surprise how unstable and hard can be to estimate such a simple think, when it is about two unstable Agents, which influence each other. This was the biggest surprise, and either biggest knowledge for me over all course. Before I start, I thought it will be pretty easy to resolve it. But it was not.

Ideas of future work.

Is hard to make some improvement, is it too unstable. I think the way to improve is to make the action little smoother, because the ball is to sensitive. After certain experiences the noise can be decreasing, the exploration in this case no mater, and it rapidly change the move of ball. This might be done be decay of theta param in Ornstein-Uhlenbeck process, in cooperation with reward.

The overall problem with this approach is that the agent does not know he does the best action, and still trying to explore.

The action of one agent influence the other. The agent does not react to action but instead the state of ball. The actions between agents are not connected. Maybe to become with inter action exchange between agents, brings better results. This is highly seen when agent is close to net, all time when the ball is on other side. This is actually very bad strategy; he sometimes loses to catch the ball after environment reset. One agent does not count on what happened on other side of net. The agent does not recognize a lot of missed state. It surprises me how less it does learn. It is stable only in certain positions.