Udacity

Christian Leininger

Supervisor:
Advisor:

1 Introduction

This work is about solving the third project of the deep reinforcement learning course. In this environment the goal is to control two racket agents . If the agent hits the ball over the net he receives an +0.1 reward if the agnet let the ball hit the ground or out out the bound it recives an -0.01 reward. The general goal is to keep the ball in the game
The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

2 Environment

The Environment is given from Unity Machine Learning Agents (ML-Agents). The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction and the agent has 4 discrete actions forward, backward, left and right

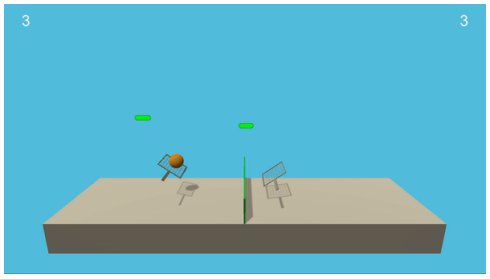

Figure 1: multi agent enviroment

3 Learning algorithm

3.1 Deep Deterministic Policy Gradient DDPG

Here the actor-critic is combined with the approach with the Deep Q Network.

It is not possible to straightforwardly apply Q-learning to continuous action spaces, because in continuous spaces finding the greedy policy requires an optimization of at at every timestep; this optimization is too slow to be practical with large, unconstrained function approximators and nontrivial action spaces. Instead, here we used an actor-critic approach based on the DPG algorithm (Silver et al., 2014) [2]
Proposed Algorithm in the research work

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=0

3.2 Networks

The implementation consist about to agents each of them as to networks a actor and a critic.
Critic
Critic estimates the V(s) function by using the TD Error. The uses the relu activation function and the Dropout technique to deactivate at random the nodes with a probability of 20 percent. The input (states) getting normalized with the BatchNorm1d function from pytorch. The network has 4 Layers the Input layer with 8 Nodes for the state space and 512 Nodes first hidden layer and 256 Nodes for the second hidden layer and The weights are initialized with uniform distributed random variables between $-0.003 and + 0.003$. The network uses

actions and states as Input to approximate the state action Function Q(s,a).
Actor
Actor maps a given state to an action. The policy of the actor will be improved by the critic. This network has no dropout but for the last layer the activation function is tanh to restrict the output from -1 to 1

4 Hyper-parameter

This algorithm trains the agents to solve the environment has many hyper-parameter. Those need to be optimized to solve it with a few episodes as as possible. For the network it is
Amount of layers = 2
Number of Nodes of layers 512 and 256
Type of layer: BatchNorm, fully connected
Activation Function: relu and tanh
Init. of Network weights: Uniform dist. -0.03 to 0.03
optimizer Adam first-order gradient-based optimization of stochastic objective functions [1]
learning-rate actor:0.0001 , learning-rate critic:0.0003 betas=(0.9, 0.999)
eps=1e-8 standard
weight-decay=0 standard
amsgrad=False standard
DDPG:
batch-size 512 amount of experience to compute the loss h
update-every 4 , the agents updates the weights after every 4 step (action)
tau = 0.001 indicate how much the weights from the online network transfer to the target network to do an soft update
memory-capacity = 100.000 samples of experince in the buffer is limited by your RAM

5 Results

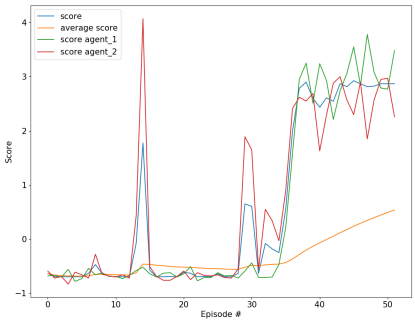

Figure 2: Scores during training

The environment was solved after 47 episodes with an everage score 0.54

6 Future ideas to improve the performance

Use PER instant of the ReplayBuffer. Try more hyper-parameter or a different network architecture. Create a model of the enviroment and combine it with a planning algorithm.

References

[1] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.