Report

In this report I discuss the results of my work on solving the Reacher environment with the DDPG algorithm.

In the Navigation project the DQN was used. With DQN, the neural network approximates the action-value function Q(s, a). DQN is a so-called value-based method. From the action-value function, the optimal policy is derived, by taking the action with the highest value for a certain action. Instead of finding out the policy from the action-value function, one can try to get the policy directly with a Policy Gradient (PG) approach. PG is a whole family of algorithms in which the policy is obtained directly. Next to being more direct, PG is also more appropriate for continuous action spaces, like the Reacher environment, which is solved in this project.

A straightforward PG approach is the REINFORCE algorithm, but it has various problems, such as that is needs complete episodes before it can start learning. Also, convergence and speed of learning are a problem for REINFORCE. An important family of PG algorithms that improve on REINFORCE are actor critic algorithms. These algorithms contain an actor network, which return the policy, and a critic. This critic network shows how good actions were.

The DDPG algorithm also has an Actor and a Critic network, but is also differs from “ordinary” actor critic algorithms. Actor Critic algorithms like A2C learn the policy as a probability distribution of the actions for a state. The action is simply sampled from that distribution. DDPG has in common with A2C that is learns using an actor and a critic, but the policy is deterministic, hence the name Deterministic Deep Policy Gradient.

An important advantage of DDPG over other Actor Critic algorithms is that it can learn offline. Just like in DQN learning, the DDPG algorithm works with a replay buffer. In this replay buffer past experiences are collected, from which the agent can sample randomly and learn like in supervised learning. So, from this perspective one can call DDPG a kind of extension of DQN for continuous action spaces.

Contrary to other policy gradient algorithms like A2C, the DDPG policy is deterministic (therefore its name). However, the environment should be explored somehow, so some noise is added to the actions, using an Ornstein-Uhlenbeck process (Lapan, 2018: 411).

My implementation is based on this implementation, which is based on the implementation described in paper in which DDPG was proposed (Lillicrap et al. 2015).

The goal of the Reacher environment is to have an average reward of +30 over a series of 100 episodes, so the whole game takes at least 100 episodes. In the ideal situation, the environment is solved 100 episodes. With the hyperparameters as used by Lilicrap et al. (2015: 11), the environment is solved in about 140 episodes. I have tweaked the hyperparameters in such a way that the environment was solved after the first 100 episodes. Of course, the goal of the Lilicrap paper was not to optimize the algorithm for just this environment, but to be able to be able to play a series of environments with

The hyperparameters are:

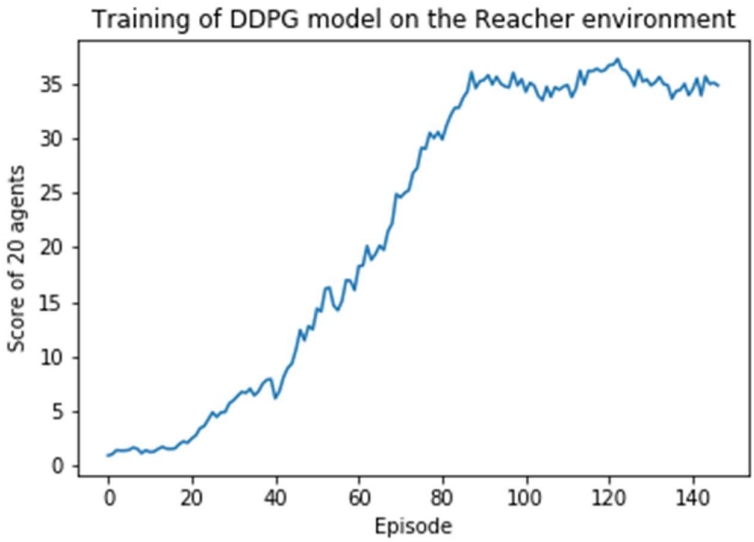
BUFFER_SIZE = int(1e6)
BATCH_SIZE = 512

GAMMA = 0.99
TAU = 1e-3
LR_ACTOR = 2e-4
LR_CRITIC = 2e-3
WEIGHT_DECAY = 0
UPDATE_EVERY = 2

In DDPG the network weights are updated after a fixed number of time steps. In the present implementation this number is 2, so the networks are updated after every second step.

The actor is a feed-forward network with two hidden layers with 128 neurons each. The critic has a slightly different structure. It has two inputs. The first input consists of the observations. The second input, the actions, are concatenated with the processed observations in the second layer. This layer has 68 neurons (64 + action space), the third and fourth layer consist of respectively 64 and 32 neurons. The critic has a single output, the Q value.

The environment was solved by the agent in 147 episodes. See the figure below. The training progress looks pretty stable, but I think it can be done a bit faster.



Recommendations

I have played a bit with the learning rates of the Actor and the Critic, but it is difficult to improve the speed of the learning process consistently. Maybe the learning rates are a bit too high now. Possible improvements of the present result are trying different algorithms, like PPO, reducing the learning rates, or making the UPDATE_EVERY value higher.

Literature

Lapan, M., Deep Reinforcement Learning Hands-On, Birmingham: Packt, 2018.
Lillicrap, TP, Hunt, JJ , Pritzel, A, Heess, N, Erez, T, Tassa, Y, Silver, D & Wierstra, D., Continuous Control With Deep Reinforcement Learning, 2015, arXiv:1509.02971.