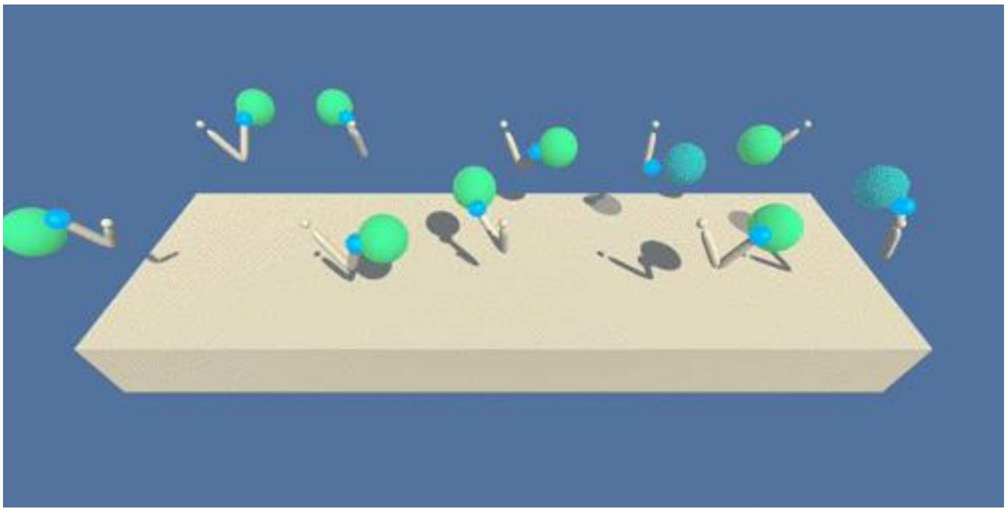
Reacher project report

Daniel Thell, September 2020

The Environment

For this project is based on the Reacher environment.



Unity ML-Agents Reacher Environment

In this environment, a double-jointed arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of the agent is to maintain its position at the target location for as many time steps as possible.

The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

Distributed Training

For this project, two separate versions of the Unity environment are provided:

- The first version contains a single agent.
- The second version contains 20 identical agents, each with its own copy of the environment.

The 1st version has been solved in this report.

The learning algorithm

The agent implemented the DDPG algorithm. It has hence 2 actor networks and 2 critic networks. The 2 actor networks are identical in shape, the 2 critic networks also but they are different from the actor networks.

The agent learns from a number of random episodes. It tries out some actions, the actor network estimates the best action to take and the critic networks gives an estimate of the q-value given the action chosen and the current state. The networks are both updated through gradient descent using an Adam optimizer. As the actor aims at maximizing the q-value estimate,

There are 2 actor networks (a local and a target) and 2 critic networks (a local and a target). The gradient descent is applied on the local networks. The target networks are only soft updated using a Tau parameter to include Tau amount of the local network parameters and keep (1-Tau) of the old target network parameters. As Tau is small, of the order of 1e-3, the target networks get updated slowly, ensuring some stability to the algorithm.

The learning is done from sampled episodes stored in a replay buffer. The learning cannot start for the very first time steps as the replay buffer is too small. We set the size of the batch to 256 so learning car start only after having done at least 256 timesteps. Having tested a batch size of 512, the learning was slower in time by construction and also for a given number of episodes.

There is a trade-off between stability and training speed on this environment. A basic implementation of the DDPG algorithm would lead to a quick training although unstable after a given number of episodes and finally an agent that would not learn.

To overcome this issue, some changes compared to a vanilla DDPG algorithm have been implemented:

- Learning is not performed at every timestep but only every T timesteps. In the learning loop, the learning is done N times in a row. After testing a few options, both N and T have been set to 10.
- The gradients of the critic network are clipped between -1 and 1
- A mix of soft and hard update have been tested as described in the paper *"Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor"* by Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine. The target networks are hard updated every 2000 timesteps (so about every 2 episodes), while the paper indicated every 1000 timesteps but it meant here every episode.

The score achieved (proportional to the time the end of the robotic arm spent in the turquoise sphere) is reported and this score should reach at least +30 during 100 consecutive episodes for the environment to be considered solved.

The parameters of the networks are stored in a file every 25 episodes to allow replay or to resume training from the achieved level.

The solution implemented

The solution implemented used the following parameters

Agent hyperparameters:

- BUFFER_SIZE = int(1e6) # replay buffer size
- BATCH_SIZE = 256 # minibatch size
- GAMMA = 0.99 # discount factor
- TAU = 1e-3 # for soft update of target parameters
- LR_ACTOR = 1e-4 # learning rate of the actor
- LR_CRITIC = 3e-4 # learning rate of the critic
- WEIGHT_DECAY = 0 # L2 weight decay
- NUMBER_UPDATE_PASSES = 10 # number of learning passes every time the learning from the replay buffer is called
- NOISE_DECAY = 0.99999 # speed decay of the OU noise
- GAE_LAMBDA = 0.0 # GAE lambda parameter
- learning_frequency = 10 # Number of timesteps between 2 learning calls

The networks used for the actor and the critic are the following:

Actor:

- Batch Norm
- FC1: Fully connected layer (128 neurons) with ReLU activation function
- FC2: Fully connected layer (128 neurons) with ReLU activation function
- FC3: Fully connected layer (4 neurons) with tanh activation function : it is the output layer

Critic

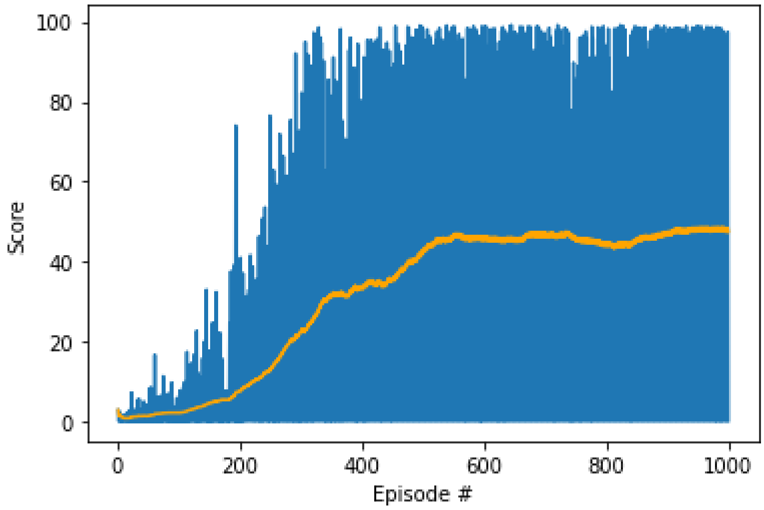
- Batch Norm
- FC1C: Fully connected layer (256 neurons) with ReLU activation function
- FC2C: Fully connected layer (256 neurons) with ReLU activation function: on this layer the input includes the output of the previous layer (FC1C) and the actor's output (4 values)
- FC3: Fully connected layer (1 neuron): it is the output layer giving the q-value estimate

A test of the Generalized Advantage Estimation was done, based on a 20 steps bootstrapping. With this implementation, the GAE is accurate for lambda not too high as for a lambda of 0.8 or less, over 99% of the lambda return is located on the first 20 time-steps. It has been tested with lambda=0 (as with no GAE), lambda=0.3, lambda=0.5 and lambda=0.8 but the higher the lambda and the lower the learning efficiency so it has been deactivated in the final solution.

Alternate actor and critic models were tested (the actor network with 1 single hidden layer with 256 neurons, the critic network with 300 and 200 neurons for the 2 hidden layers) with worse results so not kept.

The results

The environment was solved in 334 episodes with an average score over the episodes 235 to 334 being +30.05. The graph of learning is shown below.

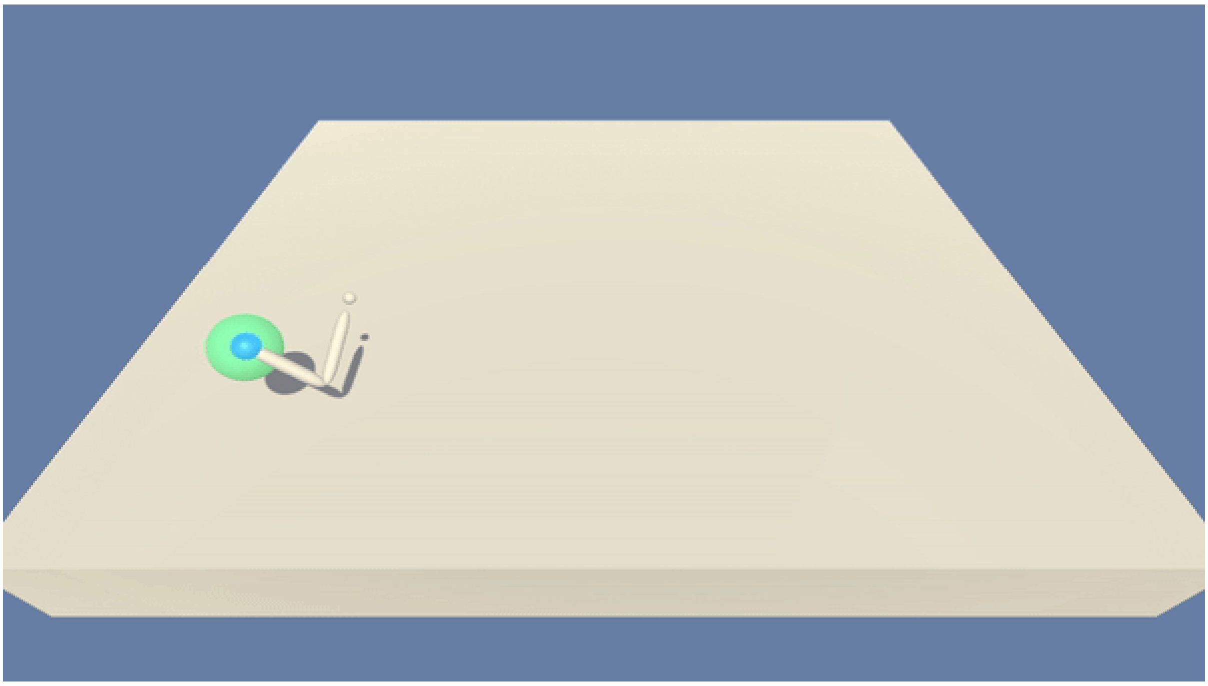


The learning seems to plateau around 47-48 although max scores can reach almost +100 on a single episode.

The files with the saved weights are:

- For the actor: checkpoint_actor_Model_actor128-128_critic256-256_f10_cLR3e-4_aLR1e-4_GAE-0.0.pth
- For the critic: checkpoint_critic_Model_actor128-128_critic256-256_f10_cLR3e-4_aLR1e-4_GAE-0.0.pth

The trained agent behavior on a given scenario is displayed below.



Further improvements

This is the single agent version of the reacher environment. It would be great to implement and test the 20 agents version and see how these can share experience and speed up training. It is likely that the number of episodes to reach the +30 average score would be lower with 20 parallel agents.

Another axis for improvement would be to test importance sampling especially after the first learning phase (so after about 500 episodes training) to see whether this could help figuring out the changes needed on the episodes that yield to lower scores when the learning seems to plateau.