Project 2 - Continuous Control

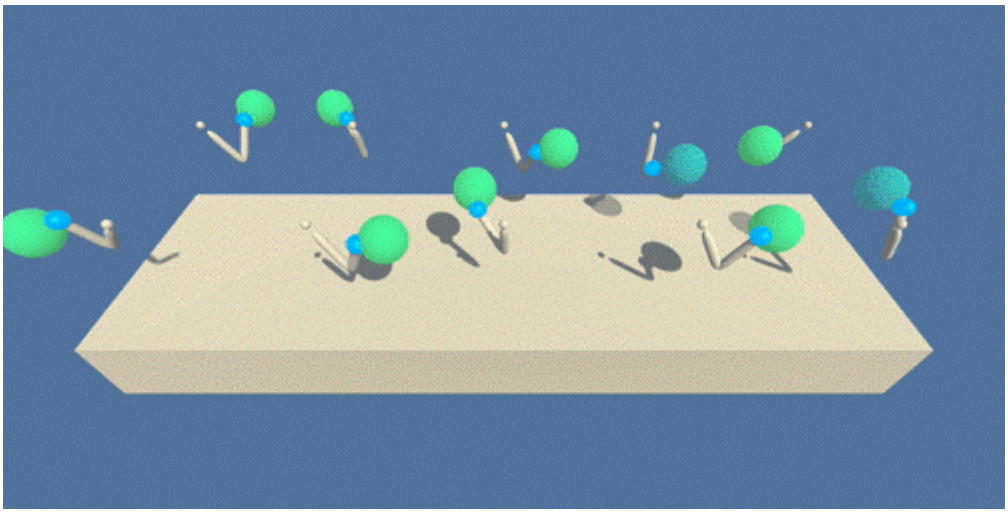
Deep Reinforcement Learning Nanodegree

M. Lippl

March 20, 2021

1 Introduction

This project illustrates the so-called Policy methods of reinforcement learning. The setting is a scenario where one (alternatively 20 simultaneous) robot arms with 4 degrees of freedom corresponding to two joints ('shoulder' and 'elbow') need to be trained to follow a sphere which orbits around their shoulder mount. Each arm is rewarded with 0.1 per time step points as long as the abstract "hand" stays in the target region. The following image illustrates this concept:



As indicated before, one may solve this challenge via two different paths: The first one involves training only one single arm, whereas the second one utilizes 20 independent arms simultaneously.

At the end of each episode, the score of each individual arm is calculated, and the final score of this episode is calculated as the average over these 20 scores (which me may call the 'episodic average'). The challenge is considered solved, if the average score of the episodic averages is greater than +30 over 100 consecutive episodes. In formulæ:

$$\mathrm{avg}_N(i) = \frac{1}{N}\sum_{j=0}^{N} \bar{s}_{i-j} \overset{!}{>} 30$$

where $\bar{s}_i$ is the mean score over all arms in episode i. Of course, in the case of one single arm,

the mean score over all arms is only the score itself, i.e. $\bar{s}_i = s_i$.

2 Learning Strategy

The challenge's desciption by Udacity suggests three different learning strategies. These are:

1. Proximal Policy Optimization (PPO) [1]
2. Asynchronous Advantage Actor-Critic (A3C) [2]
3. Distributed Distributional Deep Deterministic Policy Gradient (D4PG) [3]

Although we will finally settle with Distributed Distributional Deep Deterministic Policy Gradient (D4PG) [4], we will make a short digression and outline different concepts of policy gradient methods in the following section.

Recall that the utility function is defined as the expectation value:

$$U_\theta(\tau) = \mathbb{E}_\theta\left[\sum_{t=0}^{T} r(s_t, a_t)\right]$$

where $\tau = (s_0, a_0, s_1, a_1, \dots, s_T, a_T)$ is the path the agent has taken. It can be shown that

$$U_\theta = \mathbb{E}\big[\mathbb{E}\big[Q_{P_\theta}(s_0, a_0)\big]\big],$$

which means that, assuming that all initial states are of equal probability, maximizing the action-value function Q is equal to maximization of the utility function U.

Taking the sum over all possible paths τ, and incorporating the probability $P_\theta(\tau)$:

$$U_\theta = \sum_\tau \left(P_\theta(\tau) \left(\sum_{t=0}^{T} r(s_t, a_t) \right) \right)$$

Using the Reinforce-log-trick (cf. sec. 2.1), we arrive at:

$$\nabla_\theta U_\theta = \mathbb{E}\left[\sum_{t=0}^{T} \left(\nabla_\theta \log(P_\theta(a_t \mid s_t)) \left(\sum_{t=0}^{T} r(s_t, a_t) \right) \right)\right]$$

2.1 REINFORCE-Trick

The REINFORCE-trick is a simple substitution which is well-known from the field of statistical thermodynamics.

$$p(\tau; \theta, \phi) = \prod_{t=1}^{T} p(r_t, s_{t+1} \mid s_t, a_t; \phi)\pi(a_t \mid s_t; \theta)$$

Utility function:

$$\mathbb{E}_{p(\tau|\theta,\phi)}[R(\tau)]$$

We'd like to perform a gradient ascent into the direction of the utility function's gradient:

$$\hat{g} = \nabla_\theta \mathbb{E}_{p(\tau|\theta,\phi)}[R(\tau)]$$

Formally, this leads us to the update equation for the weights,

$$\theta \leftarrow \theta + \alpha \hat{g}.$$

The objective is to maximize this function which can be done by using gradient ascent. In order to evaluate this equation, we are taking a closer look at the gradient. First of all,

$$\hat{g} = \nabla_\theta \mathbb{E}_{p(\tau|\theta,\phi)}[R(\tau)] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta [R(\tau_i)]$$

Using the identity $\nabla_x \log x = 1/x$, we can write

$$\nabla_\theta p(z;\theta) = p(z;\theta) \nabla_\theta \log p(z;\theta).$$

$$\nabla_\theta \mathbb{E}_{p(\tau|\theta,\phi)}[R(\tau)] \approx \frac{1}{N} \sum_{i=1}^{N} [R(\tau_i) \nabla_\theta \log p(\tau_i;\theta,\phi]$$

Assuming a single sample trajectory:

$$\nabla_\theta \mathbb{E}_{p(\tau|\theta,\phi)}[R(\tau)] \approx R(\tau) \sum_{t=1}^{T} \nabla_\theta \log p(a_t \mid s_t;\theta)$$

Very high variance, so a baseline is needed.

3 The Actor-Critic Algorithm

Given a loss function $\mathcal{L}(\theta)$, the weights are updated as follows:

$$\theta \leftarrow \theta - \beta \nabla_\theta \mathcal{L}(\theta)$$

Using Q-Learning (SARSA-max), out TD-target is given by:

$$Q_\theta(s_t, a_t) \approx r(s_t, a_t) + \max_a \hat{Q}(s_{t+1}, a, \theta)$$

Hence, the gradient of the loss function is:

$$\nabla_\theta L(\theta) = -\Big(r(s_t, a_t) + \max \hat{Q}(s_{t+1}, a, \theta) - \hat{Q}(s_t, a_t, \theta) \Big) \nabla_w \hat{Q}(s_t, a_t, \theta)$$

The actor network is updated using the update equation, while the critic network is updated using the loss function.

4 Advantage-Actor-Critic

A second approach to minimizing the variance of the gain is to subtract a baseline b which is independent of both the action a and the parameter θ. If we consider the gradient of the utility function,

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=0}^{T}\left(\nabla_\theta \log(P_\theta(a_t \mid s_t))\left(\hat{Q}_{P_\theta}(s_t, a_t, \theta) - b\right)\right)\right)_i,$$

then we can easily see, that by decreasing the expression in the right pair of brackets, we lower the contribution of the gradient of the log-probability, $\nabla_\theta \log P_\theta$. A natural candidate for this baseline is the well-known state-value function V, i.e.

$$b_t = V(s_t).$$

This leads us to the definition of the Advantage function A,

$$A(s, a) = Q(s, a) - V(s).$$

In other words, we are taking the (state-)value-function $V(s)$ as the baseline. Substituting the bias in the above gradient equation yields Advantage-Actor-Critic:

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=0}^{m}\left(\sum_{t=0}^{T}\left(\nabla \log(P)\hat{A}(s_t, a_t, \theta)\right)\right)_i$$

Again, update is given by:

$$\theta \longleftarrow -\beta\nabla_\theta L(\theta)$$

Main downside of the REINFORCE algorithm is that the gradient estimator is a Monte-Carlo estimator

$$b = \hat{V}(s, \theta) = \mathbb{E}\left[\hat{Q}(s, a, \theta)\right]$$

$$\hat{A}(s, a, \theta) = \hat{Q}(s, a, \theta) - \hat{V}(s, \theta)$$

Approximate the state function via an ANN:

$$\hat{V}(s, \theta) \approx \hat{V}(s, \theta)$$

Advantage function can then be written as:

$$\hat{A}(s, a, \theta) \approx \hat{A}(s, a, \theta, \theta_v) = \hat{Q}(s, a, \theta) - \hat{V}(s, \theta_v)$$

The loss function's gradient is

$$\nabla_\theta \mathcal{L}(\theta) = -\hat{A}_{P_\theta}(s, a, \theta)\nabla_\theta \hat{V}(s, \theta)$$

and therefore, the update equation is given by:

$$\theta \longleftarrow \theta - \beta\nabla_\theta \mathcal{L}(\theta)$$

4.1 Deterministic Policy Gradient Algorithm

$$b_\theta(s) = \mu_\theta + \mathcal{N}$$

where $\mathcal{N}$ is a stochastic process. In our case, this can be either normal distributed noise or a Ornstein-Uhlenbeck-process which is described below.

4.1.1 Actor network

Let μ_θ be the target policy.

$$\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \hat{Q}_\theta(s_t, \mu_\theta(s_t), \theta)$$

Assuming a continuous action, and therefore being able to apply the chain rule yields:

$$\theta \leftarrow \theta + \alpha_\theta \nabla_a \hat{Q}_\theta(s_t, a, \theta) \nabla_\theta \mu_\theta(s_t)$$

4.1.2 Critic network

The TD-error is given by:

$$\delta_{\text{TD}} = r(s, a) + \hat{Q}_\theta(s', \mu_\theta(s'), \theta) - \hat{Q}_\theta(s, a \sim b(s), \theta) - \hat{Q}_\theta(s, a \sim b(s), \theta)$$

This error results in the weight update:

$$\theta \leftarrow \theta + \alpha_w \delta_{\text{TD}} \nabla_\theta \hat{Q}_\theta(s, a \sim b(s), \theta)$$

5 Deep Deterministic Policy Gradient Algorithm

The Deep Deterministic Policy Gradient method (DDPG) uses two different network-pairs, each pair consisting of a local and target-network, as familiar from DQL: one network-pair to learn the policy (the "Actor") and one pair to learn the value-function (the "Critic").

5.1 Ornstein-Uhlenbeck Process

The Ornstein-Uhlenbeck process is given by

$$dx_t = -\theta x_t dt + \sigma dW_t$$

where $\theta, \sigma > 0$, and W_t denotes the Wiener process.

6 Implementation and Results

6.1 Structure of the Project

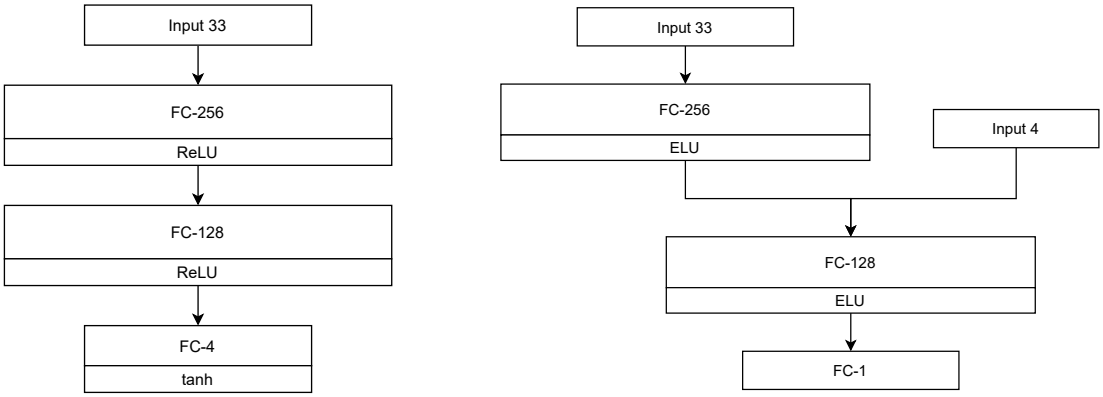
As the previous projects before, our project has the simple structure:

```
udacity-rl-p2/
├── checkpoints_torch/
├── Continuous_Control.ipynb
├── agent_torch.py
├── networks_torch.py
├── train.py
└── play.py
```

The Jupyter-notebook `Continuous_Control.ipynb` is self-contained and contains the results of the last experiment. The file `agent_torch.py` contains the definition of the MultiAgent- and Agent-classes. `networks_torch.py` - Definition of the actor- and critic-networks. `train.py` - Script containing setup and the main training loop `play.py` - Script containing the replay loop

6.2 Architecture

The network definition is given in file `networks.py`: The output layer consists of four neurons with a ‘tanh‘-activation layer.



(a) Actor (b) Critic

Figure 1: Network architectures used in the implementation

The architecture of the actor- and critic-networks is very simple. The actor (fig. 1a) basically only consists of two fully connected neural network layers, one with 256 neurons, and one with 128. In the critic (fig. 1b) however, the additional action inputs of the agent are concatenated with the state input which were already forwarded through the first fully connected layer of the network.

6.3 Hyperparameters

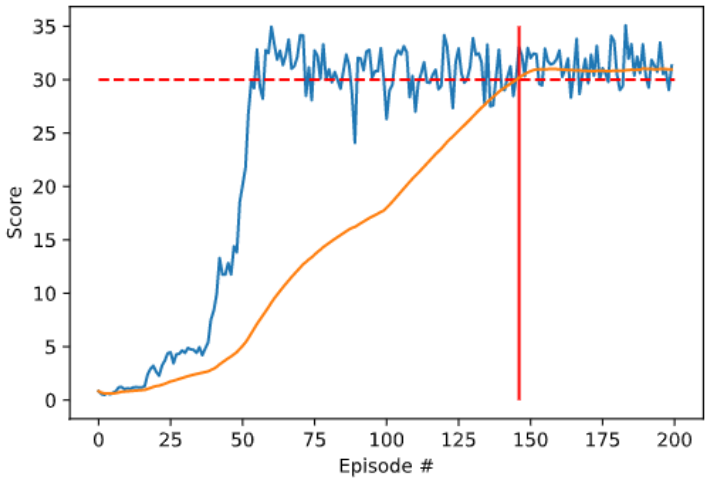
The hyperparameters were as follows:

Hyperparameter	Value	Description/Comment
`BUFFER_SIZE`	10,000	Total size of replay buffer
`BATCH_SIZE`	64	Size of training batches
`GAMMA`	0.98	Discounting factor
`TAU`	0.001	Soft updates
`learn_rate`	$5 \cdot 10^{-4}$	Learning rate of Adam-optimizer
`seed`	5	Manual seed for neural networks
`epsilon`	0.05	Final value during training

6.4 Results

The results of our experiments are illustrated in the shown in `https://github.com/asiopueo/udacity-rl-p2/Continuous_Control.ipynb`.

As one can see in the following graph and has been logged in the Jupyter-notebook, the game was solved after 146 episodes:



References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[2] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016.

[3] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, and T. Lillicrap, "Distributed distributional deterministic policy gradients," 2018.

[4] 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," 2019.