Continuous Control Project

Project in Deep reinforcement learning as part of a Nano degree.

22/12/2019
Jonas le Fevre Sejersen
Jonas.le.fevre@gmail.com

1 The scope of the project

For this project, we will train an agent to control an arm to keep the hand within a target sphere. The rules are quite simple, the sphere is moving around the arm and a reward of +0.1 is provided for each step the agent's hand is within the sphere. Thus, the goal of our agent is to maintain its position at the sphere location for as many time steps as possible.

The state space has 33 dimensions and contains the position, rotation, velocity, and angular velocities of the arm. Given this information, the agent has to learn how to best select actions. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Unsure of the exact torque applicable, my guess an action could be:

- **action[0]** - joint 1 rotation velocity
- **action[1]** - joint 1 up and down velocity
- **action[2]** - joint 2 rotation velocity
- **action[3]** - joint 1 up and down velocity

This is not for sure, since i have not found any source being specific in the actions, but every entry in the action vector is a number between -1 and 1, which makes it a continuous action space.

In this project i will solve Reacher with use of multiple agents gathering experiences but only training one policy to rule them all. This version contains 20 identical agents, each with its own copy of the environment. The barrier for solving this version of the environment is slightly different, to take into account the presence of many agents. In particular, our agents must get an average score of +30 (over 100 consecutive episodes, and over all agents). Specifically,

- After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 20 (potentially different) scores. We then take the average of these 20 scores.
- This yields an **average score** for each episode (where the average is over all 20 agents).

As an example, consider fig: 1, where we have plotted the average score (over all 20 agents) obtained with each episode.

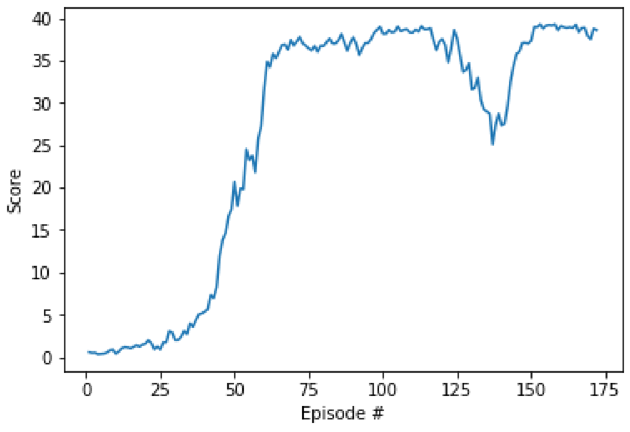

Fig. 1: Plot of average scores (over all agents) with each episode. (taken from udacity example)

The environment is considered solved, when the average (over 100 episodes) of those average scores is at least +30. In the case of the plot above, the environment was solved at episode 63, since the average of the average scores from episodes 64 to 163 (inclusive) was greater than +30.

In this project, I chose to implement the Twin Delayed Deep Deterministic (TD3) Policy Gradient network to give myself a challenge and to learn the different aspects of state-of-the-art. In the next couple of section, we will go briefly through what TD3 is and how it tries to improve some of the pitfalls of the standard DDPG.

2 Twin Delayed Deep Deterministic Policy Gradient

TD3 is introduced in the paper: Addressing Function Approximation Error in Actor-Critic Methods, and is based around the classic Deep Deterministic Policy Gradient(DDPG) network, which is categorised as a Actor-critic method. Actor-critic methods evolve around have two separate networks helping each other completing the task given. The **Actor** is used to approximate the optimal policy deterministically, meaning we want it to output the best possible action for a given state. This is unlike a stochastic policies in which the goal is to learn the probability distribution over the actions. So in DDPG we want the **Actor** to learn $\arg\max_a Q(s, a)$. The **Critic** is used to evaluate the optimal action value function given by the **Actor**. The training loop is then given:
The **Actor** is updated based on how well it can chose the yields the best result from the **critic**. While the **critic** is updated based on how well it can minimise the loss between the target network and the local network.

So some of the know problems of DDPG is that the learned Q-function begins to dramatically overestimate Q-values, which then leads to the policy breaking, because it exploits the errors in the Q-function [1].

[1] This is based on following site: openai

The TD3 method is addressing these issues by introducing following three tricks:

- **Trick One: Clipped double-Q learning**
- **Trick two: Delayed Policy updates**
- **Trick three: Target Policy Smoothing**

In the following sections we are going more into details of what these tricks are and how they improve the quality of the network.

2.1 Clipped double-Q learning

TD3 keeps two **Critic's** (hence the twin in the title), to learn the Q-function. The clipped double-Q learning is then taking the lowest value of the two **Critic's** and use that as the Q-Value. As stated in the TD3 paper, this method favours underestimation of Q values, which is less of a problem as low bias values do not tend to propagated through the algorithm during learning, unlike overestimate values. This provides a more stable approximation, thus improving the stability of the entire algorithm.

2.2 Delayed Policy Updates

To create stabilisation while learning we are using target networks as we do in DQN, however, because of the interaction going on between **Actor** and **Critic** networks the training might diverge when a poor policy is overestimated. The agents policy will continue to get worse and worse as it updates on states with a lot of errors. This problem is simple solved by delaying the update on the **Actor** network. This allows the **Critic** networks to become more stable and reduce errors before it is used to update the **Actor**. These less frequent policy updates will result in value estimate with lower variance and therefore should result in a better policy.

2.3 Target Policy smoothing

The final trick of TD3 is to smooth the target policy. A concern with Deterministic policy methods is they tend to produce target values with high variance when updating the **Critic**, hereby overfitting to narrow peaks in the value estimate. This induced variance can be be reduced through regularisation. What TD3 does is reducing the variance by adding a small amount of clipped noise to the target and then averaging over the training batches. By applying the additional noise to the value estimate, policies tend to be more stable as the target value returns a higher value for actions that are more robust to noise and interference[2].

[2] These tricks are descriped well in https://towardsdatascience.com/td3-learning-to-run-with-ai-40dfc512f93

3 Hyper-parameters

This section will lead up to the result section by explaining some of the hyper-parameters used to produce the final result. We will go through each hyper-parameter and shortly describe how it affects the model.

3.1 General info

The general info is info about the game and what seed have been used:

```
game: Reacher.exe
seed: 0
state_size: 33
action_size: 4 # Continuous space between -1 and 1
slow_and_pretty=False # If it should display in real-time or training time
result_folder='TD3_continues_control\\results' # Where to save result
```

Not much to note here since it is the basic game data and the only real changeable parameter is the seed.

3.2 Model info

The model info is the hyper-parameters for the model structure. We used two kind of models a **Actor** and a **Twin Critic** which is a single network but runs two separate streams, one for each approximated Q-function.

I have chosen to only make the seed a changeable parameter and let the structure of the model (the number of layers and neuron) be constant.

Twin Critics:
In the twin critic model i use two fully connected hidden layers and a batch normalisation layer for each stream. We take in the state and action so the input layer is of size state_size + action_size, then the first hidden layer contains 400 neurons and second hidden layer 300 neurons. As we want a single output which is the best action, we have a single output neuron.

Actor:
is very similar as we have two hidden layers with same amount of neurons. The different here is the input layer takes in the state and the output layer gives the action values.

3.3 Agent info

The agent info is the hyper-parameters used to configure the TD3 agent.

```
discount: 0.99              # Discount value
TAU: 0.001                  # Amount we update the target model each update
lr_actor: 4e-4              # The learning rate of the actor
lr_critic: 4e-4             # The learning rate of the critic
weight_decay: 1e-6          # The decaying on critic’s learning rate
steps_before_train: 10      # The number of steps between model updates
train_delay=2               # Policy delay
train_iterations=1          # How many batches we train on each train call
policy_noise=0.2            # noise to smoothing the policy
noise_clip=0.5              # How big the noise is for smoothing
exploration_noise=0.1       # Noise added for exploring new policies
```

I found that using Gaussian noise instead of OUNoise improved the agent’s exploration vs exploitation by a lot. Another stupid mistake I did was setting the TAU value to high (0.01) which lead to overfitting. Since I didn’t see this mistake I had to reduce the overfitting by other means, hence reducing the model size and variating the noise. But After correcting the TAU to 0.001 the overfitting disappeared.

3.4 Replay buffer

The replay buffer info telling us something about high long we store experiences and how many experiences we use each training step.

```
BUFFER_SIZE: 2**20              # The space we use to store Experiences.
BATCH_SIZE:512                  # Amount of replays we train on each update.
```

I tend to use a large Buffer size, since this way we do not tend to forget less experienced replays. The batch size is also rather big, as we like to generalise the updates done to the network.

Training info

Lastly is the training info, which is simply how many episodes we want to train for, how long an episode is and if we want to load an old model or evaluate.

```
episodes: 1000          # Max 1000 episodes to complete the task.
max_timesteps=1000      # How many steps each episode contains.
load_model_path: ’’     # If we want to train on a old model.
eval=False              # If we should just evaluate.
```

In the next section we take a look at the result of both training phase of the model and the evaluation steps done through training.

4 Results

Every model trained is saved to the folder called *result/* and then a sub-folder is created called test and a number generated, where a **model_test.md** is placed containing all the hyper-parameters and the result of that run. A tensorboard file is also generated in to the logging folder, which contains a scalar plot over average_100_score and the average score for each episode Lastly the folder contains the trained model weights, this is the file you would load to use the trained model. In this section we are going through the result of the folder *solved* that will be located in the results folder.

Training

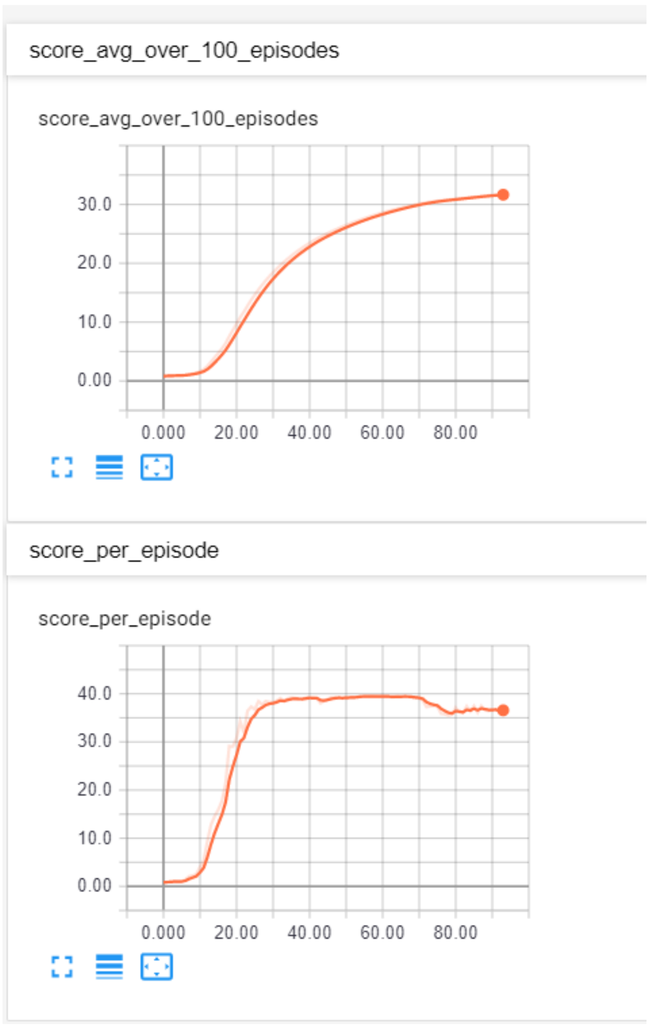

Fig. 2: The first plot is an average score over the last 100 episodes (or all episodes if total of episodes is lower than 100). The second plot is showing the average score of all 20 agents for each episode

In figure 2 a plot over the training period representing the average scores on the y-axis and the episodes on the x-asis, we can make the observation that during the first 20 episodes the agent is exploring the state space and try to couple the state of being within the sphere with good rewards and being outside the sphere gives bad rewards. After the first 20 episodes it is clear to see that this link is created and the agent keeps and average score for each episode over 30. So by the definition of solved given by udacity, the environment was solved at episode 0, since the average of the average scores from episode 0 to 100 was greater than +30. The full training values are shown in Apendix A.

5 Future improvement

Some future improvement that I didn't get to implement (yet), is the use of N-step learning. A behaviour we might see from using N-step is the arm should be staying more inside the centre of the sphere. Since a missed state will propagate to its states before it. Another improvement is using a Priority experienced replay buffer. The paper: Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards have already tried to come up with a priority function for DDPG, we might be able to transfer the priority function to work with TD3. Lastly, since the actor is basically a DQN, we could try to change it with a more advanced implementation, like the rainbow model. I'm not sure the last improvement is an improvement.

Appendix: A

Hyper-parameters used:

```
batch_size=512
buffer_size=1048576
discount=0.99
episodes=1000
eval=False
exploration_noise=0.1
load_model_path=’’
lr_actor=0.0004
lr_critic=0.0004
max_timesteps=1000
noise_clip=0.5
policy_noise=0.2
result_folder=’C:\\Fevre\\Code\\Python\\TD3_continues_control\\results’
seed=0
slow_and_pretty=False
steps_before_train=1
tau=0.001
train_delay=2
train_iterations=1
weight_decay=1e-06
```

All episodes:

```
Episode 1 Average Score: 0.83   current mean: 0.83  Min:0.22 Max:2.22 Duration:27.52
Episode 2 Average Score: 0.87   current mean: 0.92  Min:0.25 Max:1.91 Duration:27.72
Episode 3 Average Score: 0.90   current mean: 0.94  Min:0.25 Max:2.40 Duration:27.74
Episode 4 Average Score: 0.95   current mean: 1.10  Min:0.23 Max:2.79 Duration:27.80
Episode 5 Average Score: 0.95   current mean: 0.95  Min:0.15 Max:3.50 Duration:28.29
Episode 6 Average Score: 0.97   current mean: 1.06  Min:0.08 Max:2.50 Duration:29.22
Episode 7 Average Score: 1.03   current mean: 1.38  Min:0.07 Max:3.36 Duration:28.92
Episode 8 Average Score: 1.16   current mean: 2.09  Min:0.70 Max:3.90 Duration:29.57
Episode 9 Average Score: 1.28   current mean: 2.24  Min:0.81 Max:3.74 Duration:29.66
Episode 10 Average Score: 1.41   current mean: 2.54  Min:0.87 Max:4.14 Duration:30.24
Episode 11 Average Score: 1.65   current mean: 4.04  Min:2.13 Max:8.70 Duration:30.27
Episode 12 Average Score: 1.95   current mean: 5.30  Min:2.29 Max:8.33 Duration:30.73
Episode 13 Average Score: 2.52   current mean: 9.30  Min:4.19 Max:15.44 Duration:31.30
Episode 14 Average Score: 3.25   current mean: 12.77  Min:7.26 Max:20.40 Duration:32.12
Episode 15 Average Score: 4.00   current mean: 14.57  Min:6.47 Max:21.97 Duration:32.50
Episode 16 Average Score: 4.75   current mean: 15.91  Min:5.84 Max:28.20 Duration:33.60
Episode 17 Average Score: 5.51   current mean: 17.72  Min:6.81 Max:31.72 Duration:33.43
Episode 18 Average Score: 6.39   current mean: 21.36  Min:4.09 Max:34.35 Duration:33.82
Episode 19 Average Score: 7.59   current mean: 29.11  Min:9.35 Max:38.79 Duration:34.54
Episode 20 Average Score: 8.66   current mean: 29.04  Min:15.82 Max:39.31 Duration:35.25
Episode 21 Average Score: 9.71   current mean: 30.79  Min:16.55 Max:39.65 Duration:35.14
Episode 22 Average Score: 10.83   current mean: 34.25  Min:22.26 Max:39.27 Duration:35.75
Episode 23 Average Score: 11.75   current mean: 31.92  Min:15.82 Max:39.62 Duration:36.09
Episode 24 Average Score: 12.77   current mean: 36.42  Min:24.84 Max:39.40 Duration:37.03
Episode 25 Average Score: 13.75   current mean: 37.30  Min:30.17 Max:39.57 Duration:37.68
Episode 26 Average Score: 14.64   current mean: 36.74  Min:26.73 Max:39.65 Duration:38.03
Episode 27 Average Score: 15.52   current mean: 38.43  Min:34.23 Max:39.66 Duration:37.84
Episode 28 Average Score: 16.31   current mean: 37.75  Min:30.77 Max:39.65 Duration:37.76
Episode 29 Average Score: 17.08   current mean: 38.47  Min:34.06 Max:39.68 Duration:39.07
Episode 30 Average Score: 17.78   current mean: 38.28  Min:33.74 Max:39.65 Duration:39.91
Episode 31 Average Score: 18.44   current mean: 38.19  Min:33.08 Max:39.69 Duration:39.64
Episode 32 Average Score: 19.07   current mean: 38.58  Min:33.56 Max:39.66 Duration:40.97
Episode 33 Average Score: 19.68   current mean: 39.07  Min:36.76 Max:39.67 Duration:40.83
Episode 34 Average Score: 20.23   current mean: 38.37  Min:31.67 Max:39.63 Duration:40.48
Episode 35 Average Score: 20.76   current mean: 39.03  Min:36.14 Max:39.63 Duration:40.67
Episode 36 Average Score: 21.28   current mean: 39.19  Min:37.51 Max:39.62 Duration:41.38
Episode 37 Average Score: 21.76   current mean: 39.09  Min:33.87 Max:39.66 Duration:42.03
Episode 38 Average Score: 22.21   current mean: 38.83  Min:34.25 Max:39.62 Duration:42.64
Episode 39 Average Score: 22.64   current mean: 38.88  Min:36.33 Max:39.63 Duration:42.96
Episode 40 Average Score: 23.05   current mean: 39.18  Min:37.45 Max:39.63 Duration:43.27
Episode 41 Average Score: 23.44   current mean: 39.29  Min:37.21 Max:39.65 Duration:44.14
Episode 42 Average Score: 23.82   current mean: 39.08  Min:37.19 Max:39.62 Duration:44.57
Episode 43 Average Score: 24.17   current mean: 39.04  Min:35.85 Max:39.64 Duration:45.00
Episode 44 Average Score: 24.49   current mean: 38.03  Min:32.96 Max:39.65 Duration:45.27
Episode 45 Average Score: 24.80   current mean: 38.49  Min:36.63 Max:39.63 Duration:45.86
Episode 46 Average Score: 25.10   current mean: 38.95  Min:36.11 Max:39.63 Duration:46.20
Episode 47 Average Score: 25.41   current mean: 39.25  Min:36.81 Max:39.64 Duration:46.43
Episode 48 Average Score: 25.69   current mean: 39.27  Min:38.55 Max:39.60 Duration:48.31
Episode 49 Average Score: 25.97   current mean: 39.27  Min:38.48 Max:39.64 Duration:47.23
Episode 50 Average Score: 26.23   current mean: 39.04  Min:37.86 Max:39.59 Duration:47.76
Episode 51 Average Score: 26.49   current mean: 39.24  Min:38.47 Max:39.66 Duration:48.26
Episode 52 Average Score: 26.73   current mean: 39.27  Min:38.06 Max:39.61 Duration:48.43
Episode 53 Average Score: 26.97   current mean: 39.28  Min:37.96 Max:39.62 Duration:50.12
Episode 54 Average Score: 27.20   current mean: 39.36  Min:38.57 Max:39.63 Duration:49.88
Episode 55 Average Score: 27.42   current mean: 39.55  Min:39.22 Max:39.64 Duration:50.26
Episode 56 Average Score: 27.64   current mean: 39.48  Min:38.99 Max:39.65 Duration:50.53
Episode 57 Average Score: 27.85   current mean: 39.40  Min:38.43 Max:39.70 Duration:50.97
Episode 58 Average Score: 28.05   current mean: 39.54  Min:38.73 Max:39.67 Duration:49.90
Episode 59 Average Score: 28.24   current mean: 39.44  Min:38.92 Max:39.68 Duration:49.05
Episode 60 Average Score: 28.43   current mean: 39.40  Min:37.76 Max:39.66 Duration:51.87
Episode 61 Average Score: 28.61   current mean: 39.49  Min:38.66 Max:39.68 Duration:49.74
Episode 62 Average Score: 28.78   current mean: 39.44  Min:38.89 Max:39.65 Duration:53.06
Episode 63 Average Score: 28.95   current mean: 39.34  Min:38.71 Max:39.58 Duration:51.24
Episode 64 Average Score: 29.11   current mean: 39.27  Min:38.27 Max:39.59 Duration:51.34
Episode 65 Average Score: 29.27   current mean: 39.45  Min:39.05 Max:39.65 Duration:51.70
Episode 66 Average Score: 29.42   current mean: 39.38  Min:38.20 Max:39.65 Duration:51.58
Episode 67 Average Score: 29.57   current mean: 39.47  Min:38.28 Max:39.64 Duration:49.80
Episode 68 Average Score: 29.72   current mean: 39.39  Min:38.29 Max:39.63 Duration:51.30
Episode 69 Average Score: 29.86   current mean: 39.25  Min:38.61 Max:39.61 Duration:52.13
Episode 70 Average Score: 29.99   current mean: 39.10  Min:37.77 Max:39.61 Duration:55.58
Episode 71 Average Score: 30.12   current mean: 39.06  Min:36.81 Max:39.65 Duration:55.70
Episode 72 Average Score: 30.23   current mean: 38.61  Min:36.56 Max:39.61 Duration:50.84
Episode 73 Average Score: 30.33   current mean: 37.26  Min:30.58 Max:39.61 Duration:51.13
Episode 74 Average Score: 30.43   current mean: 37.39  Min:32.78 Max:39.18 Duration:50.44
Episode 75 Average Score: 30.52   current mean: 37.40  Min:32.69 Max:39.22 Duration:51.31
Episode 76 Average Score: 30.61   current mean: 37.34  Min:34.28 Max:39.54 Duration:50.40
Episode 77 Average Score: 30.68   current mean: 35.88  Min:31.26 Max:39.46 Duration:50.64
Episode 78 Average Score: 30.74   current mean: 35.76  Min:30.35 Max:39.60 Duration:53.57
Episode 79 Average Score: 30.80   current mean: 35.50  Min:32.29 Max:38.61 Duration:50.59
Episode 80 Average Score: 30.86   current mean: 35.72  Min:30.59 Max:39.28 Duration:49.65
Episode 81 Average Score: 30.94   current mean: 37.10  Min:34.69 Max:39.65 Duration:48.21
Episode 82 Average Score: 31.00   current mean: 36.16  Min:34.11 Max:38.72 Duration:48.67
Episode 83 Average Score: 31.06   current mean: 35.97  Min:30.78 Max:39.62 Duration:51.06
Episode 84 Average Score: 31.14   current mean: 37.39  Min:34.35 Max:39.37 Duration:48.91
Episode 85 Average Score: 31.20   current mean: 36.34  Min:22.17 Max:39.45 Duration:49.88
Episode 86 Average Score: 31.27   current mean: 37.46  Min:34.06 Max:39.24 Duration:50.01
Episode 87 Average Score: 31.33   current mean: 36.10  Min:31.76 Max:38.89 Duration:50.21
Episode 88 Average Score: 31.40   current mean: 37.48  Min:34.78 Max:39.33 Duration:49.51
Episode 89 Average Score: 31.46   current mean: 36.59  Min:33.11 Max:39.17 Duration:48.42
Episode 90 Average Score: 31.51   current mean: 36.44  Min:26.13 Max:39.32 Duration:48.02
Episode 91 Average Score: 31.57   current mean: 36.65  Min:31.09 Max:39.00 Duration:48.06
Episode 92 Average Score: 31.63   current mean: 36.74  Min:33.27 Max:39.51 Duration:47.89
Episode 93 Average Score: 31.67   current mean: 36.07  Min:29.03 Max:39.51 Duration:47.75
Episode 94 Average Score: 31.73   current mean: 36.74  Min:33.38 Max:39.42 Duration:47.97
Episode 95 Average Score: 31.78   current mean: 36.51  Min:33.99 Max:39.57 Duration:49.42
Episode 96 Average Score: 31.82   current mean: 35.73  Min:31.09 Max:38.86 Duration:50.83
Episode 97 Average Score: 31.87   current mean: 36.96  Min:35.51 Max:38.90 Duration:50.26
Episode 98 Average Score: 31.91   current mean: 35.82  Min:30.26 Max:38.09 Duration:49.77
Episode 99 Average Score: 31.96   current mean: 36.47  Min:32.88 Max:38.40 Duration:49.32
Episode 100 Average Score: 32.01   current mean: 37.52  Min:33.16 Max:39.61 Duration:49.42
Environment solved in 100 episodes!
```