DRL - Project 2 - Reacher

Christian Eberhardt

November 2019

1 Introduction

The goal of this project is to train an agent, in this case a two armed robot with a endpoint. The robot must keep this endpoint inside a sphere that circles the robot at some arbitrary constant speed above the robots fixed base. The environment is the 20-agent Reacher Unity-environment from the Unity ML-Agents Toolkit. The version used is version 0.4 of the interface. See the **README.md** of the github repository, chrillemanden/DRLND-DDPG-Reacher, for a description of the environment and an implementation of a reinforcement learning algorithm that solves the environment. The solution to the environment is inspired from the ddpg_pendulum and ddpg_bidepal exercises in the Udacity Deep Reinforcement Learning Nanodegree. The learning algorithm is described in detail in the paper Continuous Control with Deep Reinforcement Learning.

2 Learning Algorithm

The reinforcement learning algorithm used to solve the environment is DDPG (Deep Deterministic Policy Gradient). This algorithm was chosen to accommodate the continuous states space and the continuous action space. It is an Actor-Critic method and therefore accommodates both an Actor model as well as a Critic Model, both deep neural networks. The Actor is used to approximate the optimal policy deterministically while the Critic learns to evaluate the optimal action value function by using the Actor's best believed action. This way the Critic maps a state and action pair to a Q-value. The learning algorithm is described in pseudocode in figure 1.

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

Figure 1: DDPG algorithm.

Experience Replay is used in the training algorithm to stabilise training. Experience tuples of states, actions, rewards and next states are stored in a Replay Buffer and used for learning, so the agent can learn from past and current experiences.
Noise is also added in the learning algorithm to the action values sent to the environment. The noise added is described by the Ornstein-Uhlenback process. Noise is added to the action values to make training faster. The hyperparameters used are shown in table 1.

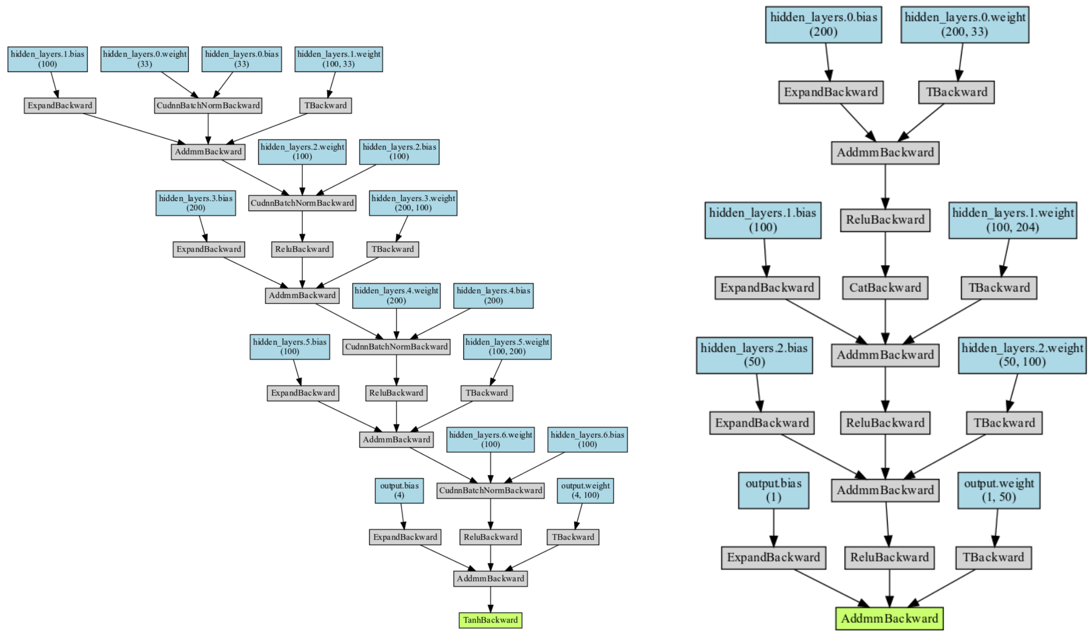
γ	0.99
τ	10^{-3}
Actor α	$5 \cdot 10^{-3}$
Critic α	$4 \cdot 10^{-4}$
Actor network size	[33, 100, 200, 100, 4]
Critic network size	[33, 200, 100, 50, 1]
Noise standard deviation	0.2
Episodes between updating networks	4
Replay buffer size	10,000
Replay buffer batch size	64

Table 1: The hyperparameters used for the agent with the best performance.

3 Neural Networks

The Actor and Critic deep neural networks each have a copy that is used as a target network. So in total four networks are used, two for the Actor and two for the Critic. The network for the

Actor model is visualised in figure 2a. The Actor network consist of an arbitrary number of fully connected layers with batch normalisation between the layers as an option and rectifying linear units (ReLU) in the nodes. The input is the state vector of 33 dimensions and the output is an action vector with four action values. The output layer uses the hyperbolic tangent function, *tanh()*, in the activation units. The network for the Critic model is visualised in figure 2b. The Critic network consists of an arbitrary number of fully connected layers with ReLU's in the nodes. The input is the state vector of 33 dimensions as well as the action values vector that is concatenated in the second layer of the network. It has just one output.
The Adam optimizer is used in all the networks.



(a) A visualisation of the actor network using torchviz.

(b) A visualisation of the critic network using torchviz.

Figure 2: Visualisations of the networks used. The images can be found in higher resolution in the github repository, chrillemanden/DRLND-DDPG-Reacher.

4 Plot of Rewards

In this project four different agents using DDPG as the learning algorithm was tested. Table 2 highlights the differences between the agent parameters.
Figure 3 show the score at every episode for the four agents. As can be seen from the figure, agent 2 and three do not manage to solve the environment within the provided 300 episodes. It is possible that that they would have achieved a solution if they had been trained more, but the purpose was to compare them to the other agents. This suggests that batch normalisation in the layers of the actor as well as the addition of noise to the action values to boost exploration is necessary to make the agent learn the optimal policy. Agent 4 compared to agent 1 has a bigger network with more nodes in each layer and an additional layer in both the Critic and Actor network and also learns much faster. Agent 4 reaches a score above 30 around episode 27 and achieves a mean score above

Agent #	With BatchNorm	With Noise	Model Layers Sizes	Reached Score of 30 at episode
1	True	True	Both: [50, 50]	136
2	False	True	Both: [50, 50]	Never
3	True	False	Both: [50, 50]	Never
4	True	True	Actor: [100, 200, 100] Critic: [200, 100, 50]	27

Table 2: The settings of the different agents tested

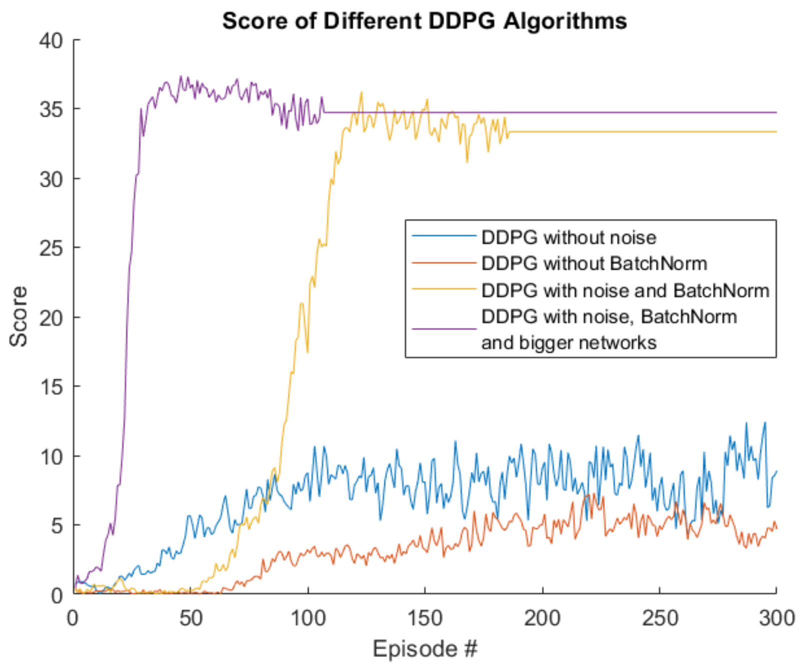


Figure 3: The scores for the four different agents

30 for the past 100 episodes at episode 106.

5 Ideas for Future Work

It was indicated that adding noise to the action values promotes exploration and makes the agent learn the optimal policy much faster. However the noise in this project is added to the action values. Studies might suggest that learning might be even faster if noise is injected in each of the nodes in the network as parameter noise as discussed in this paper, this paper and mentioned in this paper.
Additionally it would be interesting to try to solve the environment using Proximal Policy Optimisation (PPO).