Deep Deterministic Policy Gradients to solve Continuous Control Tasks

Mateo Neira

August 11, 2020

Introduction

In this project we use a deep deterministic policy gradient (DDPG) to solve a continuous control task problem. DDPG is an off-policy learning algorithm for continuous actions [4]. The algorithm combines both Q-learning [8] and Policy gradients[6], and consists of two model: Actor and Critic. The Actor directly maps the states to continuous action values, while the Critic is a Q-value network that takes in states and actions as inputs and outputs the Q-value.

The algorithm has three main components: experience replay [5], target networks for both the Actor and Critic that are time delayed copies of the original network and updated using soft-updates, and uses batch learning [1]. The experience replay uses a replay buffer to store experience tuples, the networks are then trained by sampling experience from the replay buffer which eliminates temporal dependencies between experiences and insures data is iid. The target networks are similar to the target networks used in [5], but are updated by: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$, in this way the target values are constrained to change slowly, which greatly improves the stability of learning. Lastly, *batch normalisation* normalises each dimension across the samples in a mini batch to have unit mean and variance, which helps the networks learn more efficiently.

Here we implement a modified version of the original DDPG algorithm proposed in [4] to solve a continuous control task. The task requires the agent to move a double-jointed arm by applying torque to the two joints in order to track a moving target.

0.1 Environment

The environment used for this project is the Reacher environment from Unity, using its Machine Learning Agents Toolkit [2]. The environment consists of 20 agents, and the goal of each agent is to track a moving target by controlling a double-jointed arm. A reward of +0.1 is received each step an agent's hand is in the targets location. The agent perceives a 33 dimensional state space corresponding to position, rotation, velocity, and angular velocities of the two arm rigid bodies. The agent can take actions by applying torque to the two joints (for a total of 4 dimensions), that can range from -1 to 1. The environment is considered solved when the average score of all agents is at least +30, averaged across 100 episodes.

1 Implementation

1.1 Deep Deterministic Policy Gradient

The implementation used follows the original algorithm proposed in [4] with minor adjustments to speed-up training for this specific environment. The algorithm is outlined in (DDPG, Algorithm 1).

1.1.1 Actor-Critic Network

We used Adam [3] for learning the neural network parameters with a learning rate of 10^{-3} and 10^{-3} for the actor and critic respectively. For Q we did not use a L_2 weight decay, as it decreased performance, and used a discount factor of $\gamma = 0.99$. For the soft target updates we used $\tau = 0.001$. The neural networks used the rectified non-linearity for all hidden layers. The final output layer of the actor was a $\tanh$ layer, to bound the actions. The networks had 2 hidden layers with 128 units each. Gradient clipping was used on the

Algorithm 1 DDPG algorithm (adapted from original paper [4])

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
Initialize epsilon for noise decay ϵ
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration for (independent process for each agent)
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t^i$ according to the current policy and exploration noise $\times\epsilon$
    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 minimising the loss on the clipped gradient of: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    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:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

    update epsilon: $\epsilon \leftarrow max(0.05, \epsilon - 1e^{-6})$
  end for
end for

Critic network in order to avoid exploding gradients, which improved learning performance. Actions were not included until the 2nd hidden layer of Q.The final layer weights and biases of both the actor and critic were initialized from a uniform distribution $[-3 \times 10^{-3}, 3 \times 10^{-3}]$ and $[3 \times 10^{-3}, 3 \times 10^{-3}]$. This was to ensure the initial outputs for the policy and value estimates were near zero, similar to the original paper. The other layers were initialised from uniform distributions $[-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}}]$ where f is the fan-in of the layer. We trained with minibatch sizes of 256, and replay buffer size of 10^6.

1.1.2 Ornstein–Uhlenbeck process

The original DDPG algorithm uses an Ornstein—Uhlenbeck process [7] to generate temporally correlated exploration. The Ornstein-–Uhlenbeck process models the velocity of a Brownian particle with friction, which results in temporally correlated values centred around 0. We tested tested the implementation provided by Udacity Bipedal environment with $\theta = 0.15$ and $\sigma = 0.05$, and noticed that the mean of the resulting noise was not centred around zero. We modified the implementation to get a correct noise $\mathcal{N}$ fig. 1, and improved learning performance significantly. Additionally a noise decay ϵ was added to slowly decrease the noise at each training step.

2 Results

With this implementation of DDPG we were able to solve the environment in under 130 episodes, as seen in fig. 2. The relatively small amounts of training episodes required to solve the environment can be mainly be attributed to efficient exploration of the environment by the added noise provided by the Ornstein—Uhlenbeck process with $\sigma = 0.05$, which provided a noise centred at zero and with a range between (-0.2, 0.2). Having each agent have a independent noise process also increase exploration efficiency as it allowed different agents to explore different actions when given the same state.

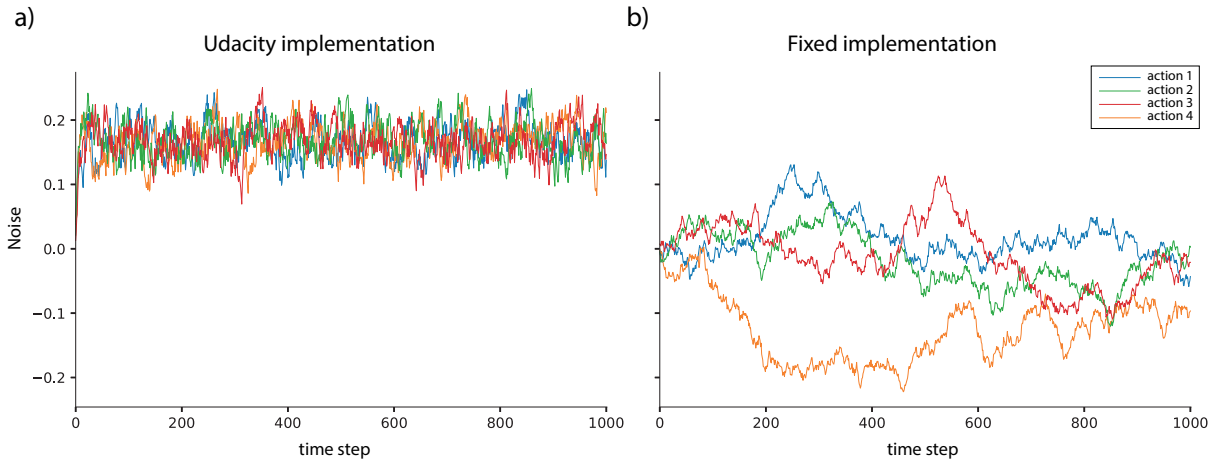

Figure 1: Ornstein—Uhlenbeck process. a) implementation provided by Udacity in the Bipedal environment, b) fixed implementation used to solve project.

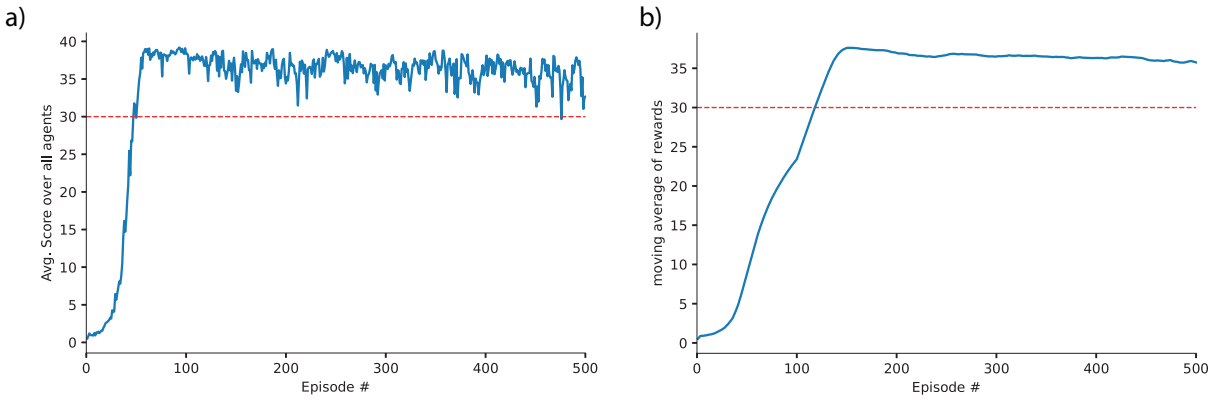

Figure 2: Training performance. a) average score over all agents for each episode. b) moving average of 100 episodes of average score over all agents.

3 Conclusions

We implemented and tested a DDPG to solve a continuous control problem. Several changes where made to the original algorithm in order to improve performance in our specific environment. The changes where primarily aimed at improving the exploration of the environment in order to learn more efficiently, as well as keeping the gradients from exploding to stabilise training. Specifically, a separate noise process was applied to each agent so that agents with the same state space take different actions and ensure a optimal exploration of the environment. Additionally, the L_2 weight decay of 10^{-2} as proposed in the original paper was removed as it slowed down training. Finally, gradient clipping by re-scaling the gradient by its vector norm. With this implementation we were able to solve the environment in 130 episodes. For future extensions, we can implement and compare other policy gradient methods, such as Proximal Policy Optimisation (PPO), Trust Region Policy Optimisation (TRPO), as well other Actor-Critic methods such as Advantage Actor Critic (A2C) and compare their performance.

References

[1] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[2] Arthur Juliani et al. “Unity: A general platform for intelligent agents”. In: *arXiv preprint arXiv:1809.02627* (2018).

[3] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[4] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[5] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[6] Richard S Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.

[7] George E Uhlenbeck and Leonard S Ornstein. “On the theory of the Brownian motion”. In: *Physical review* 36.5 (1930), p. 823.

[8] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.