Project 2: Continuous Control

Rory McGrath

February 28, 2019

1 Overview

In this environment, a double-jointed arm acts as the agent and 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. The objective 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 is between -1 and 1. This task is episodic, in order to solve the environment the agent must achieve an average score of +30 over 100 consecutive episodes.

2 Methods

An Actor-Critic approach was used to solve this problem. A value based method such as Deep Q Networks (DQN) [4] could have been adapted but it would require discretising the action space. Depending on the required granularity of the discretisation and the degrees of freedom in the arm this would lead to a rapidly growing action space [1]. With this limitation in mind the Actor-Critic algorithm Deep Deterministic Policy Gradients (DDPG)[2] was implemented.

Similar to DQN, DDPG uses deep artificial neural networks as large non-linear function approximators for the value function. In order to train these function approximators in a stable and robust way DDPG also uses a Replay Buffer[3] and updates the target networks with "soft target updates" [2]. The DDPG maintains four separate ANNs. Two are the local and target actor functions and two are the local and target critic functions. The actor function $\mu(s|\theta^{\mu})$ specifies the current policy by deterministically mapping states to a specific action. The critic function $Q(s, a)$ is learned using the Bellman equation. Given that the action space is in the range of -1 to 1 the final activation function in the actor network is a tanh function. The tanh function is given in equation [1]

$$f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \tag{1}$$

One issue with learning in continuous space is exploration. How does the agent explore the environment? Given that this is an off-policy algorithm we can construct an exploration policy μ' by adding noise sampled from a noise process N. Adding this noise to the actor policy yields an exploration policy [2].

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + N \tag{2}$$

When observing the behaviour of the trained agent this exploration can be turned off.

3 Results

Using the methodology described above [2] an agent was trained to solve the environment. The agent was written in pyTorch and was run using the openai gym toolkit.

The ANN used for the actor consisted of an input layer of 33 nodes, one dense hidden layer of 256 nodes and an output layer of 4 nodes. The activation functions for the input and hidden layer was the rectified linear units (ReLUs). The output layers activation function was the hyperbolic tangent function (tanh). This maps the output to a range between -1 and 1 which is the required input for the motors

The ANN used for the critic has a similar architecture and consisted of an input layer of 33 nodes, one dense hidden layer of 256 nodes and an output layer of 1 node. The action tensor was added to the last hidden layer similar to the paper [2].

The following hyper-parameters were used to train the agent:

Learning Updates	
Parameter	**Value**
γ	0.99
τ	$1e^{-3}$

Experience Replay	
Parameter	**Value**
Buffer Size	$1e^{5}$
Batch Size	1024

The model was trained for 3,000 episodes. After 1,993 episodes the model achieved it's target of an average of +30 over 100 episodes. A plot of the reward verses episode number is given in [Fig 1].

4 Future Work

While the initial results of this agent are promising, the agents behaviour is far from optimal. There are several improvements that could be implemented to decrease training time and increase the average reward.

Actor Network	
Parameter	**Value**
Update Rate	every 20 time-steps
Learning Rate	$1e^{-4}$

Critic Network	
Parameter	**Value**
Update Rate	every 20 time-steps
Learning Rate	$1e^{-3}$

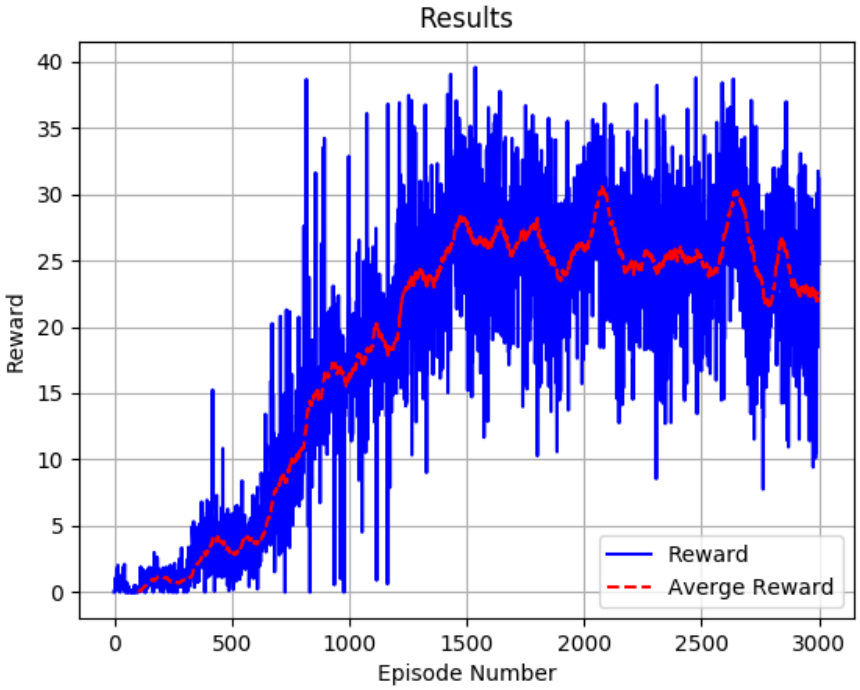

Figure 1: Training results

One of the most difficult aspects of this project was find hyper-parameter that lead to stable learning. Trying out variations of hyper-parameters along with different network architectures was extremely time consuming. A hyper-parameter optimisation technique such as grid search could be implemented to help converge on good choices for the parameters specified above. Here you could set the ranges and combinations you wanted to investigate and run the search on a gpu server.

The Replay Buffer could be updated to implement prioritised experience replay [5]. The key idea here is that 'important' transitions would be sampled more frequently than unimportant ones. The 'important' of transition is determined by the magnitude of it's temporal-difference error. This should allow the agent to learn more efficiently. Given that we are no longer uniformly sampling from the experiences importance sampling will have to be added to calculate the impact of each weight when updating.

References

[1] R. Bellman, Rand Corporation, and Karreman Mathematics Research Collection. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957. ISBN: 9780691079516. URL: `https://books.google.it/books?id=wdtoPwAACAAJ`.

[2] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. arXiv: `1509.02971 [cs.LG]`.

[3] Long-Ji Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine Learning* 8.3 (1992), pp. 293–321. ISSN: 1573-0565. DOI: 10.1007/BF00992699. URL: https://doi.org/10.1007/BF00992699.

[4] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (Feb. 2015), 529 EP –. URL: https://doi.org/10.1038/nature14236.

[5] Tom Schaul et al. "Prioritized Experience Replay". In: *CoRR* abs/1511.05952 (2015).