Deep Reinforcement Learning Agent for Reacher environment

Daniel Szemerey

Udacity

Author Note

This project is an obligatory delivery for Project II in

Udacity's Deep Reinforcement Learning Nanodegree

Abstract

This report summarizes the work done on Project II of the Deep Reinforcement Learning Nanodegree.

An agent utilizing Deep Deterministic Policy Gradient (DDPG) was developed and tuned to solve the single agent Reacher environment, which runs on unity engine.

The DDPG agent was able to solve the environment under 250 episodes.

Keywords: Artificial Intelligence, Reinforcement Learning, DDPG, Actor-Critic

Table of Contents

Environment

Goal and Reward: Keep the end of the effector in the given moving goal sphere. Interaction is episodic (but could be continuous), with a hard step limit of 1000.

state space:	**33** (position, rotation, velocity and angular velocity)
action space:	**4** (continuous, all between -1 and 1)
agents (brains):	**1**
considered solved:	**> +30.0** avg. over 100 episodes
termination criteria:	**1000** time steps
reward:	**+0.1** at each time step when arm is in the goal sphere

source of the environment: Udacity - Deep Reinforcement Learning engine: unityagents

Agent with Deep Deterministic Policy Gradient

To solve the environment a popular Agent leveraging Deep Deterministic Policy Gradients has been chosen. This network is considered by many to be an Actor-Critic method. In the learning step the agent selects a next step to calculate the Temporal Difference (which is biased in regards of the actual value) with a Policy network (which has large variance in regards of the actual value). This way both bias and variance is decreased.

Below you can find the final network parameters that showed reliable convergence:

Actor Network Architecture

Input Layer size	33
Hidden Layer size 1	160
Hidden Layer size 2	160
Output Layer size	4 (action-size)
Activation function	ReLU
Output function	tanh

Critic Network Architecture

Input Layer size	33
Hidden Layer size 1	160
Action size (appended to hidden layer 1)	4
Hidden Layer size 2	160
Output Layer size	1
Activation function	ReLU
Output function	-

Training Hyperparameters

Input Layer size	33
TAU *(soft update of target network parameter)*	1e-4
Learning Rate	1e-4
Target network is updated every {} episode:	1
MU *(Noise)*	0.0
Theta *(Noise)*	0.15
Sigma *(Noise)*	0.05
Random seed	1

Results

After initial test runs establishing that the models architecture has the biggest influence on result, experiments were run on a fixed set of training hyperparameters – only network size was modified. An Initial run with both Actor and Critic hidden layer size of [192, 192] was run.

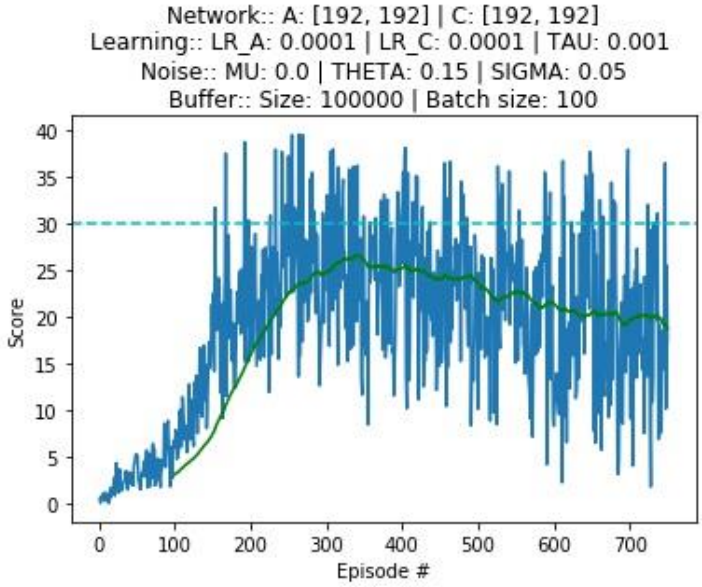


Figure 1 Run for 700 Episodes

The agent nearly achieved 30 per 100 episodes, but crashed after 400 episodes. To further test if increase or decreasing the node number in a layer would help, a test run with *Actor* [64, 64] *Critic* [64, 64] and *Actor* [240, 240] *Critic* [240, 240].

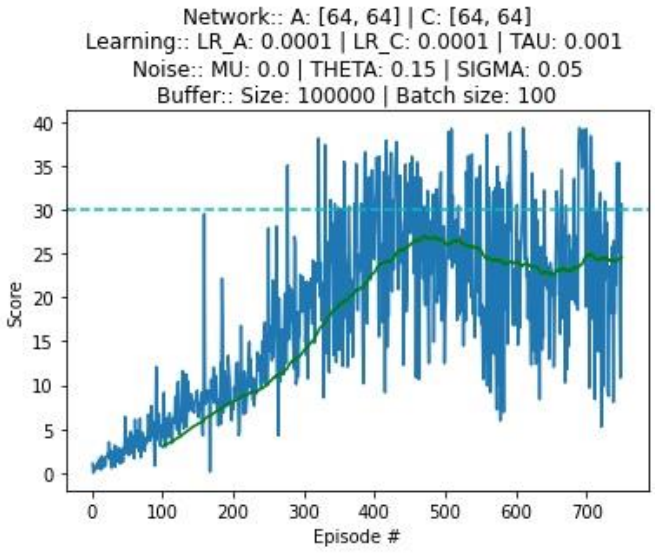


Figure 2

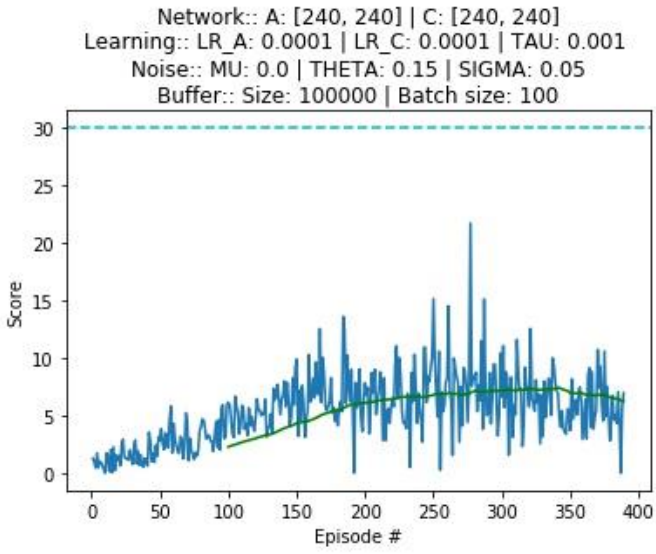


Figure 3

As Figure 2 and 3 shows, decreasing the network size both improves performance and training time – smaller network is less computationally expensive to train. The larger network's training was terminated early (after 400 runs) as it showed no convergence and calculating each episode was expensive. Given that the smaller network resulted in better performance, I experimented with turning it even smaller, to *Actor* [48, 48] *Critic* [48, 48].

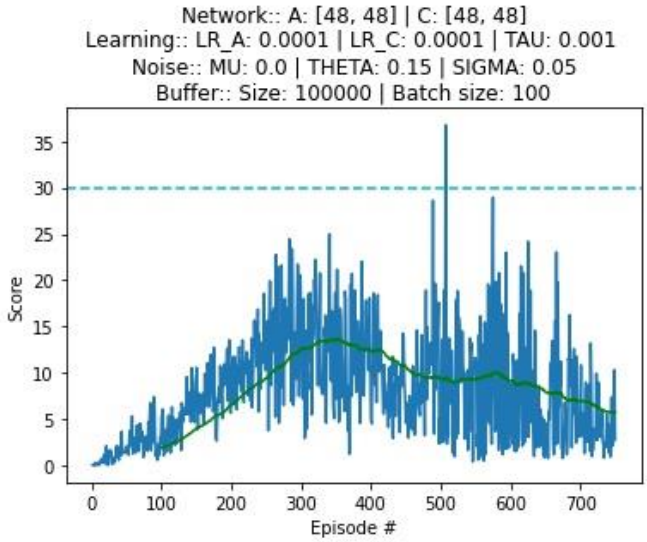


Figure 4

The Agent performs poorly, as per Figure 4. This could be due to the policy function being estimated is higher dimensional, than what the network can output. To counteract an expirement with a larger network was performed - *Actor* [128, 128] *Critic* [128, 128].

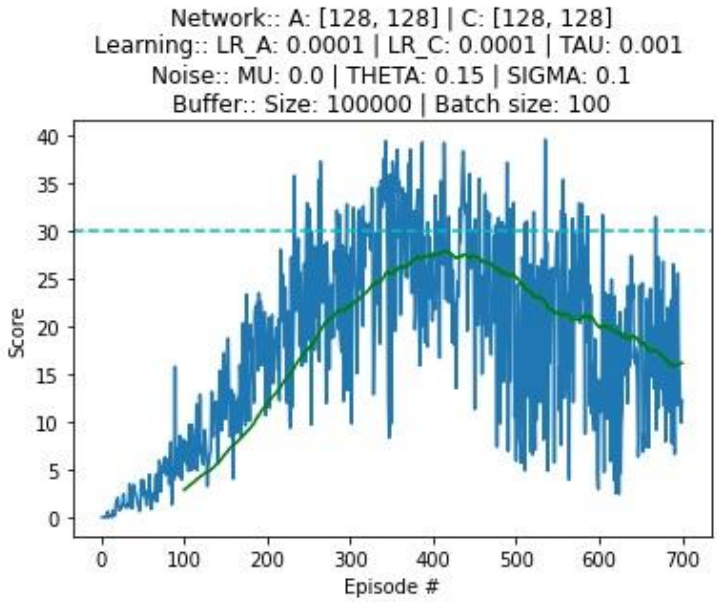


Figure 5

As demonstrated in Figure 5, this larger network nearly achieves the desired 30.0 points across 100 episodes. As a next step an even bigger network between the initial *Actor* [196, 196] *Critic* [196, 196] and the *Actor* [128, 128] *Critic* [128, 128] was tested.

The new agent with *Actor* [160, 160] *Critic* [160, 160] performed substantially better than any of the previous networks. It achieved the 30.0 point average in 237 steps, and achieved on an individual run over 30 points after 149 episodes.

The network also showed stable results, with performance dropping for a short period of time to 29.0, but regaining the average score shortly after.

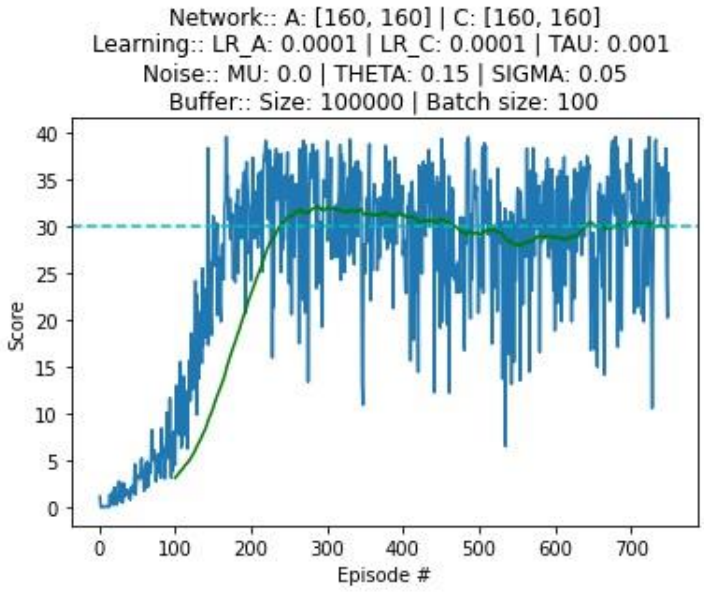
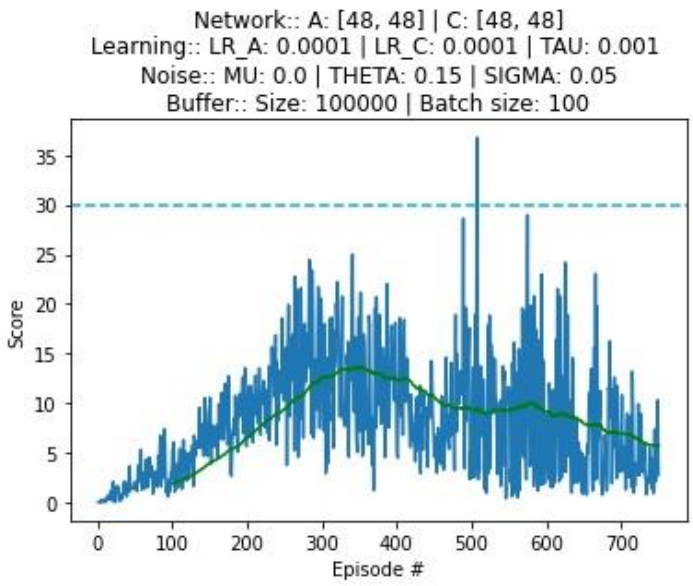
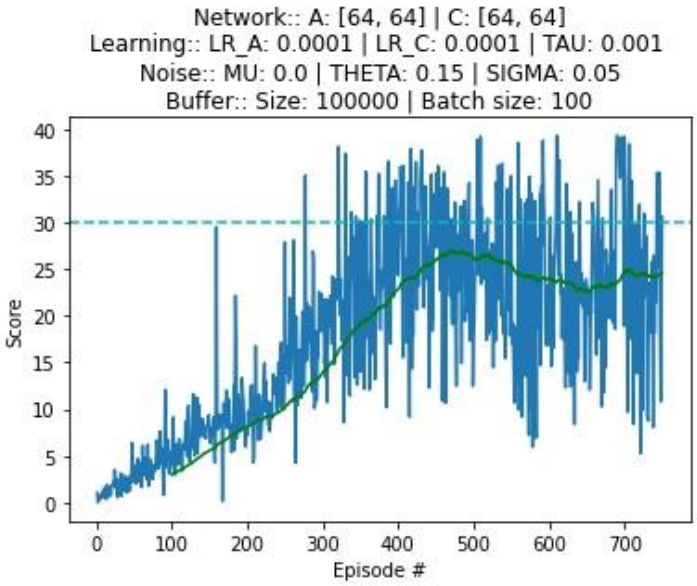
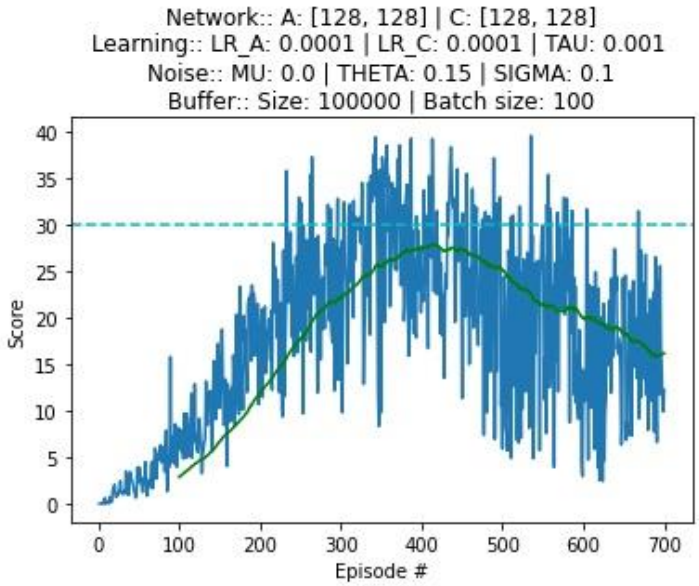


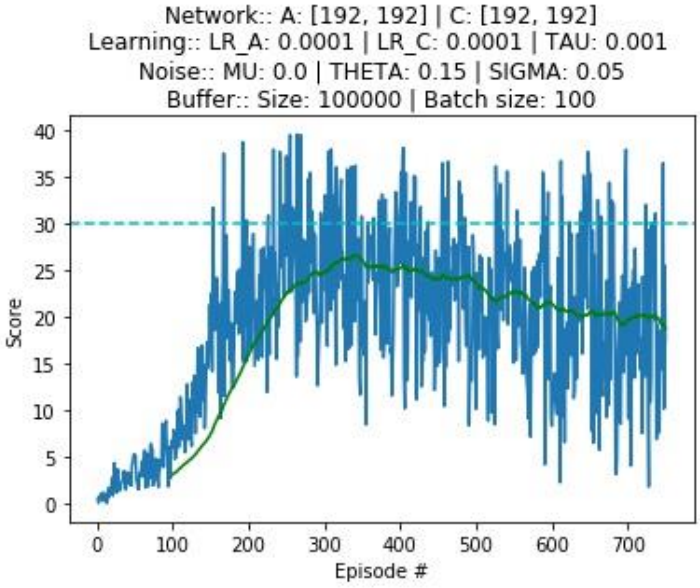
Figure 6 Final Agent's Performance

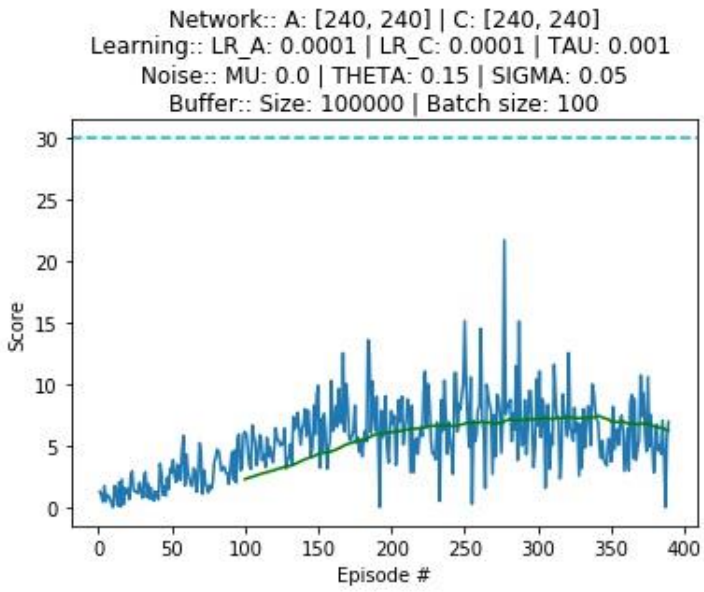
All Results

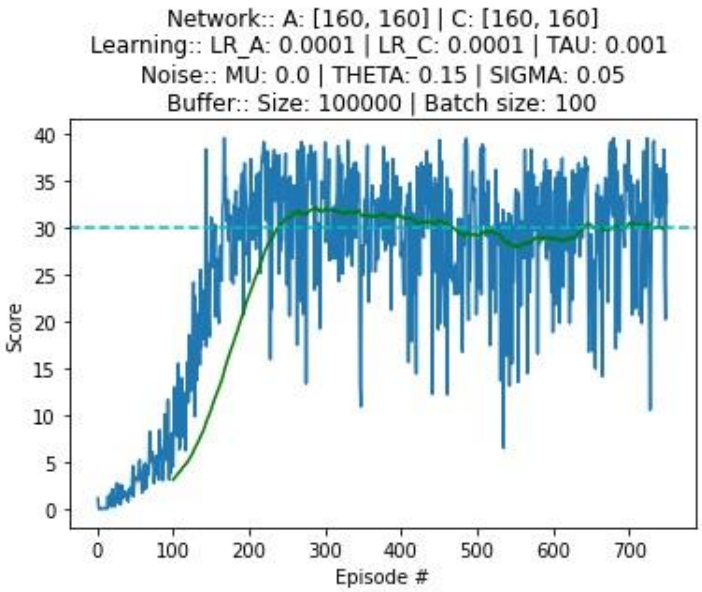

Network:: A: [48, 48] | C: [48, 48]
Learning:: LR_A: 0.0001 | LR_C: 0.0001 | TAU: 0.001
Noise:: MU: 0.0 | THETA: 0.15 | SIGMA: 0.05
Buffer:: Size: 100000 | Batch size: 100
Score
Episode #


Network:: A: [64, 64] | C: [64, 64]
Learning:: LR_A: 0.0001 | LR_C: 0.0001 | TAU: 0.001
Noise:: MU: 0.0 | THETA: 0.15 | SIGMA: 0.05
Buffer:: Size: 100000 | Batch size: 100
Score
Episode #


Network:: A: [128, 128] | C: [128, 128]
Learning:: LR_A: 0.0001 | LR_C: 0.0001 | TAU: 0.001
Noise:: MU: 0.0 | THETA: 0.15 | SIGMA: 0.1
Buffer:: Size: 100000 | Batch size: 100
Score
Episode #


Network:: A: [192, 192] | C: [192, 192]
Learning:: LR_A: 0.0001 | LR_C: 0.0001 | TAU: 0.001
Noise:: MU: 0.0 | THETA: 0.15 | SIGMA: 0.05
Buffer:: Size: 100000 | Batch size: 100
Score
Episode #


Network:: A: [240, 240] | C: [240, 240]
Learning:: LR_A: 0.0001 | LR_C: 0.0001 | TAU: 0.001
Noise:: MU: 0.0 | THETA: 0.15 | SIGMA: 0.05
Buffer:: Size: 100000 | Batch size: 100
Score
Episode #


Network:: A: [160, 160] | C: [160, 160]
Learning:: LR_A: 0.0001 | LR_C: 0.0001 | TAU: 0.001
Noise:: MU: 0.0 | THETA: 0.15 | SIGMA: 0.05
Buffer:: Size: 100000 | Batch size: 100
Score
Episode #

Future Ideas

There are several ways to further improve the algorithm:

- Modify the buffer to sample according to *Prioritized Replay*. This would add further complexity to the hyperparameters, which would need to be
- Add additional layers to the model. Two layers that the project will be expanded with are *batch normalization layers*, so that sample experiences are normalized to a fixed range between layers. Although the literature is not clear on this, but a dropout layer could be introduced in the beginning of the training, helping the agent generalize across the solution space when in a local optimum. The *dropout layer* can later be switched off, as overfitting is in general desirable in Deep Reinforcement Learning problems.
- Create more elaborate model architectures. Currently the *Critic* network receives the actions, by appending it to the second hidden layer. It could be interesting to feed the actions in different locations.

References

Grokking, Deep Reinforcement Learning, Miguel Morales, 2020

Dataset

Entire Project Folder with code can be found here (folder): deep-reinforcement-learning/p2_continuous-control at master · szemyd/deep-reinforcement-learning (github.com)