Luke The Reacher: A Deep Reinforcement Learning Agent

A report by Carlos Gavidia-Calderon

In this report, we describe in detail the algorithm we used to train our agent. We also elaborate on how our agent performed after training and some ideas on how to improve this performance.

Learning algorithm

Our agent relies on the Deep Deterministic Policy Gradient (DDPG) algorithm proposed by Lillicrap et al. from Google Deepmind[1]. A DDPG agent has the following characteristics:

1. Both an actor and critic are updated during the learning process.
2. In addition to the local network, both actor and critic have an additional target network.
3. During training, DDPG samples experiences uniformly from a finite replay buffer. Also, the actions produced by the agent are perturbed with stochastic noise.

The actor goal is to, given a state, produce an action that maximizes expected reward. In contrast, the critic produces the expected reward given a state-action pair. When training is complete, the actor should contain an optimal policy for the environment.

By keeping a target network separate for both actor and critic, DDPG avoids destabilizing training with a constantly shifting target. By sampling experiences randomly from the replay buffer during training, DDPG avoids correlations between subsequent states that can affect the training process. The noise added to actions favours the exploration of the state space.

Hyperparameters

The following table contains the hyperparameters used during training.

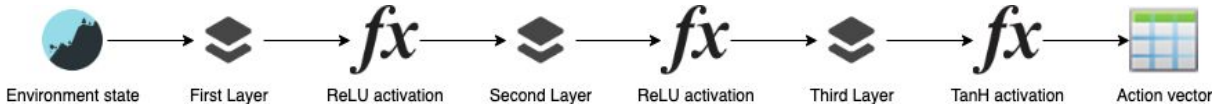
Hyperparameter	Value	Description
Replay buffer size	100000	The maximum number of experiences to store in the replay buffer. When full, newer experiences replace older ones

[1] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

Batch size for training	128	The number of experiences to sample from the replay buffer at each training step.
Learning rate for Actor	0.0001	This is a parameter of the Adam optimizer attached to the actor's network. It is used to control how much DDPG adjusts the weights of the network with respect to the gradient of the loss.
Learning rate for Critic	0.001	This is a parameter of the Adam optimizer attached to the critic's network. It is used to control how much DDPG adjusts the weights of the network with respect to the gradient of the loss.
Discount factor (gamma)	0.9	This value represents how much we value future rewards when compared with present ones.
Tau	0.001	Periodically, DDPG transfers the weights from the local network to the target network, for both actor and critic. This parameter controls the update process.
Standard deviation for Gaussian noise	0.1	Instead of using a Ornstein–Uhlenbeck process for noise generation, we rely on a mean-zero Gaussian noise.

Neural Network Architecture

According to DDPG, both the local and the target network have the same architecture. This applies for both actor and critic. The following figure shows the actor's network:

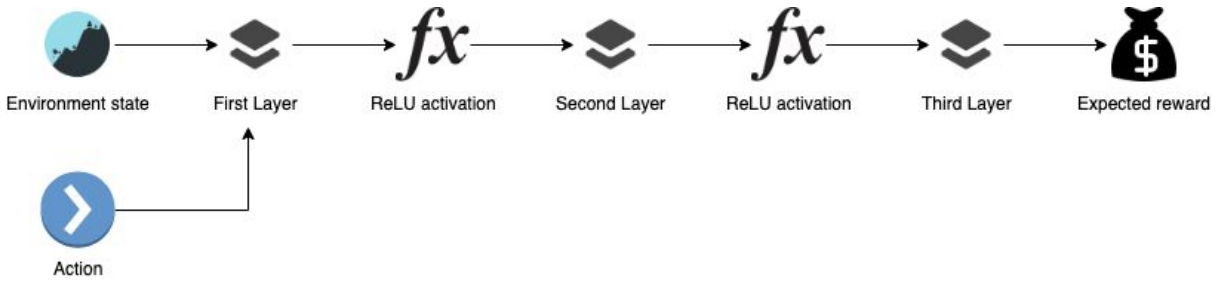


As seen, we rely on three fully connected layers:

Layer	Input Features	Output Features
First layer	33 (size of the state space)	400
Second layer	400	300
Third layer	300	4 (size of action vector)

The first and second layers are followed by a ReLU activation function. In the third layer, we apply the hyperbolic tangent activation to ensure that the elements of the action vector are in range.

The next figure shows the critic's network:



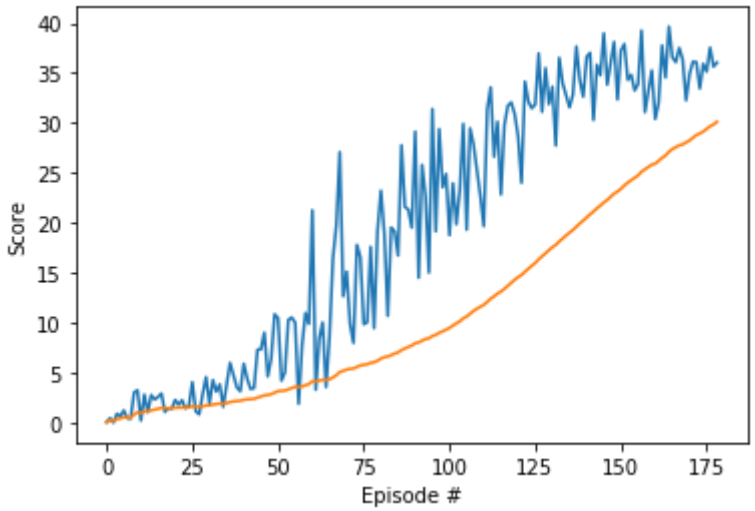
We rely as well in three fully-connected layers:

Layer	Input Features	Output Features
First layer	37 (size of the state + action space)	400
Second layer	400	300
Third layer	300	1

The first and second layers are followed by a ReLU activation function.

Plot of rewards

The blue line in the following plot contains the scores obtained by our agent at each training episode. The yellow line shows the average of the las 100 episodes. Training stopped when our agent obtained an average score of 30.0 over 100 episodes, which it accomplished at episode 179 (average score = 30.12).



Ideas for Future Work

We have not explored the hyper-parameter space, relying mostly on the values used in the DDPG implementation provided during the module[2].

The same applies to the network's architecture. The three-layer structure and the number of nodes per layer were also taken from this reference implementation. Additional layers with a different number of nodes can potentially accelerate convergence.

Finally, we addressed the single-agent version of the environment, training over a CPU. The multi-agent version of Reacher would allow training several agents in parallel, taken advantage of the performance gain of training with a GPU. PPO[3] appears to be an algorithm well suited for the multi-agent environment.

[2] https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum
[3] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).