Horna and Cuevas: Two Deep Reinforcement Learning Agents

A report by Carlos Gavidia-Calderon

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

Learning algorithm

Our agent relies on the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm proposed by Lowe et al.[1]. The MADDPG algorithm has the following characteristics:

1. It is a multi-agent extension of the Deep-Deterministic Policy Gradient (DDPG) algorithm, proposed by Lillicrap et al[2].
2. During training, each agent selects an action based on their corresponding actor component, making a decision based on the output of the critic component.
3. During testing, each agent does not need the critic's input to perform.

MADDPG qualifies as an actor-critic reinforcement learning algorithm. The role of the critic component is to enhance the decision-making process of the actor, by predicting the value of potential action-state pairs. In MADDPG, the critic of each agent has access to the state and actions of *all the agents* during training time. This differs from traditional actor-critic algorithms, where the critic has only local visibility.

During testing time, the agent does not need the critic input and can rely entirely on the policy learned by the actor component. MADDPG design enables its application to both cooperative and competitive environments.

Hyperparameters

The following table contains the hyperparameters used during training.

---

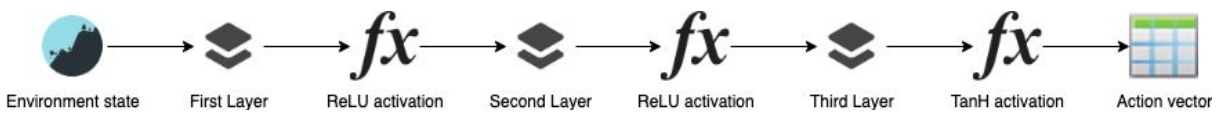
[1] Lowe, Ryan, et al. "Multi-agent actor-critic for mixed cooperative-competitive environments." *Advances in Neural Information Processing Systems*. 2017.
[2] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971* (2015).

Hyperparameter	Value	Description
Replay buffer size	1000000	The maximum number of experiences to store in the replay buffer. When full, newer experiences replace older ones
Learning frequency	4	This number controls how often we update the networks' weights. It is measured in time steps.
Batch size for training	1024	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 MADDPG adjusts the weights of the network with respect to the gradient of the loss.
Learning rate for Critic	0.0001	This is a parameter of the Adam optimizer attached to the critic's network. It is used to control how much MADDPG adjusts the weights of the network with respect to the gradient of the loss.
Discount factor (gamma)	0.95	This value represents how much we value future rewards when compared with present ones.
Tau	0.001	Periodically, MADDPG 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. This value contains its standard deviation.

Neural Network Architecture

In our MADDPG implementation, both the local and the target network have the same architecture. This applies for both the actor and critic components.

Each agent has their own actor and critic components. The following figure shows the actor's network:

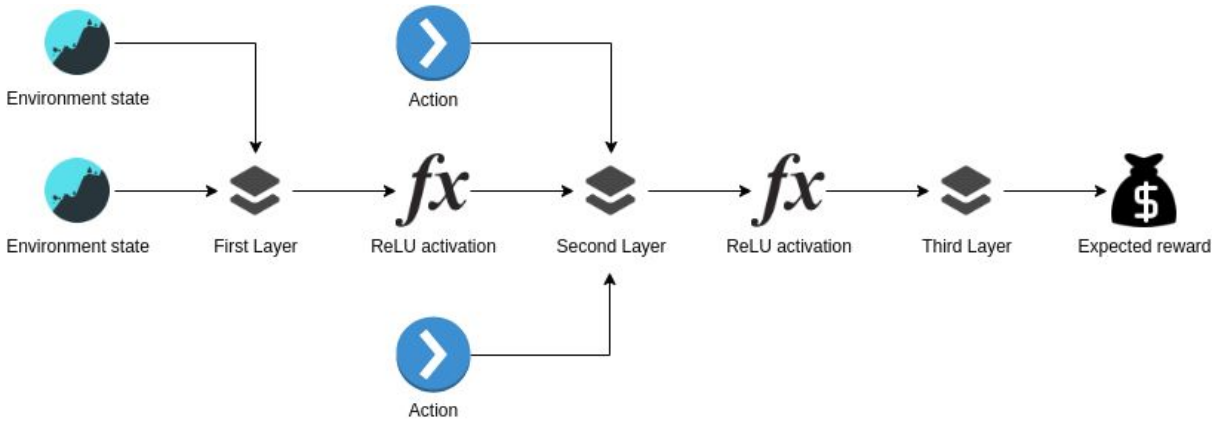


As seen, we rely on three fully connected layers:

Layer	Input Features	Output Features
First layer	24 (size of the state space)	400
Second layer	400	300
Third layer	300	2 (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:



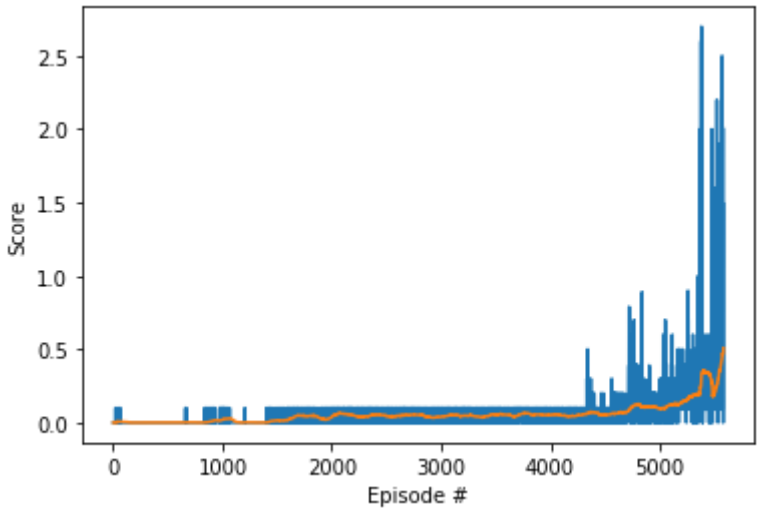
We rely as well in three fully-connected layers:

Layer	Input Features	Output Features
First layer	48 (state for all agents)	400
Second layer	404 (includes actions for all agents)	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 maximum score obtained by both agents at each training episode. The yellow line shows the average of this measure over the last 100 episodes. Training stopped when both agents produce an average maximum score of 0.5 over 100 episodes, which it accomplished at episode 5584 (average score = 0.508).



Ideas for Future Work

We have not explored the hyper-parameter space, relying mostly on the values used in our previous DDPG implementation[3].

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

Finally, we noted different behaviour on several training runs: the number of episodes required to solve the environment varied significantly, as well as its evolution over time. Even in some training runs, learning did not converge. Given more time and computational resources, we would like to perform several training runs, keeping parameter values constant, to explore the variability of learning convergence.

[3] https://github.com/cptanalatriste/luke-the-reacher