Unity Tennis

Austin Silveria

We examine the MADDPG algorithm applied to the Unity Tennis environment. We then analyze the performance of our agent and propose ideas for future improvements.

1 Learning Algorithm

MADDPG is an off policy actor critic method that stores agent interactions in a replay buffer to be sampled from and used for training of actor and critic networks. This algorithm extends the DDPG algorithm to multi-agent environments by separating the observations of each agent to train the actor networks, but using all agent observations to train each agent's critic network. This allows the agents to act independently from one another in the environment, but also benefit from full state observations during training. Without using the observations of all agents to train the critic network, the environment would look as though it is changing dynamically from the perspective of each agent, breaking down the convergence properties of single agent reinforcement learning algorithms.

Gamma	.95
Tau	.0396
Actor LR	1e-4
Critic LR	1e-3
Batch Size	128
Gradient Clip	1

TABLE 1. The chosen hyperparameters for our implementation.

Table 1 shows the eight chosen hyperparameters for our implementation. A standard value of .95 was chosen for gamma to discount future rewards when computing the expected state-value. A tau value of .0396 corresponds to the "softness" of an update to the target model with respect to the local model. A smaller value pushes the weights of the target model's network in the direction of the local model a smaller amount than a larger value. The batch size corresponds to the number of experiences sampled at a time and can be adjusted based on the GPU memory of the running system to maximize utilization if running in parallel. A gradient clip value of 1 was also used to control the size of the gradients when updating the neural networks to ensure that

our updates were not too big and helped stabilize learning. This was important due to the Tennis environment being very unstable because the agents had to keep the ball in the air for an extended period of time to receive a large reward. With this reward structure, the agents only got to see high rewards when they had a streak of hits, which was difficult to come by when their policy networks had not been trained for a long period.

2 Performance Analysis

With a state dimension of 28 and an action dimension of 2, networks of 4 fully connected hidden layers were used to approximate the optimal policy for choosing actions and critiquing action choices. The layers consisted of 256, 128, 128, and 64 neurons, respectively, and used the ReLU activation function. A batch normalization layer was also used on the first layer of the networks to allow generalization to different inputs due to the very unstable environment. Figure 1 illustrates the best learning we recorded of an agent using the MADDPG algorithm. The agent's learning was very unstable at the beginning, but once it "found" long streaks of rewards it was able to train very quickly and achieve an average reward of 0.5. One interesting property of the MADDPG algorithm that was discovered in our implementation was that the agents benefitted from having their target networks updated before and after loss computations and backpropagation. This was implemented by switching the order in which the agents were updated each iteration to allow both agents to benefit from having their target networks updated before their loss calculation an equal amount.

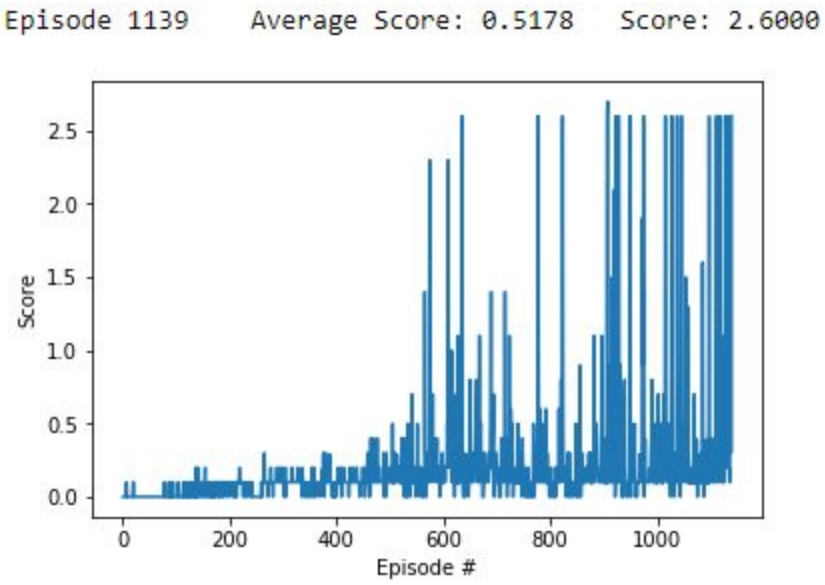

Figure 1: Best MADDPG performance on the Unity Tennis environment using the hyperparameters shown in Table 1.

3 Future Work

Hyperparameter Search with Parallel Seeds: Our implementation was very reliant on proper hyperparameters to achieve stable learning and avoid crashes. Using seeds for neural network implementations allows reproducible results of usually non-deterministic behavior of neural network training. By training multiple agents in parallel with the same seed it becomes possible to observe differences in training very early without having to wait for long sessions to complete before determining hyperparameter choices that lead to more stable training. With the use of various search methods such as genetic algorithms or other optimization techniques such as machine learning, a hands-off approach to hyperparameter tuning becomes available, as non-determinism will not play a role in skewing the results of different choices. It would be important to then scale this search across many seeds in parallel in order to find a set of hyperparameters that generalizes across the non-deterministic nature of neural networks in order to find a robust solution.