MADDPG Algorithm - Tennis

Overview

In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play.

The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

The task is episodic, and in order to solve the environment, your agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically,

- After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. We then take the maximum of these 2 scores.
- This yields a single score for each episode.

The environment is considered solved, when the average (over 100 episodes) of those scores is at least +0.5.

At first both agents randomly take actions and records the feedback, but eventually they begin to take those experiences and learn from them using a Deep Deterministic Policy Gradient (DDPG) algorithm, used in a multi-agent setting.

The attached code written in Python, using PyTorch and presented in a Jupyter Notebook, demonstrates how the agents learn and eventually achieves an average max score of 0.5 over 100 consecutive episodes.

Model Architecture

The Udacity provided DDPG code in PyTorch was used and adapted for this environment by adding the multi-agent specifics - joining experiences from both agents while training separately.
I used two deep neural networks (actor-critic), each with two hidden layers of 400-300 nodes, with Relu activation functions on the hidden layers and tanh on the output layers.

DDPG algorithm description

Policy-Gradient (PG) algorithms optimize a policy end-to-end by computing noisy estimates of the gradient of the expected reward of the policy and then updating the policy in the gradient direction.

DDPG is a policy gradient algorithm that uses a stochastic behavior policy for good exploration but estimates a deterministic target policy, which is much easier to learn. Policy gradient algorithms utilize a form of policy iteration: they evaluate the policy, and then follow the policy gradient to maximize performance.

The Actor-Critic learning algorithm is used to represent the policy function independently of the value function. The policy function structure is known as the actor, and the value function structure is referred to as the critic. The actor produces an action given the current state of the environment, and the critic produces a TD (Temporal-Difference) error signal given the state and resultant reward. If the critic is estimating the action-value function, it will also need the output of the actor. The output of the critic drives learning in both the actor and the critic. In Deep Reinforcement Learning, neural networks can be used to represent the actor and critic structures.
In practice, we let the agent explore the environment according to initial network output and we store the results in a replay buffer (experiences). To deal with the exploration/exploitation dilemma, we add some random noise in the training, and also use the epsilon greedy approach. After enough experiences have been accumulated in the replay buffer, we sample a minibatch-sized number of experiences, and instruct the agent to learn from them. Learning process is governed by the following equations:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2)$$

Here, we sample a minibatch of size N from the replay buffer, with the i index referring to the i’th sample. The target for the temporal difference error computation, yi, is computed from the sum of the immediate reward and the outputs of the target actor and critic networks, having weights θμ′ and θQ′ respectively. Then, the critic loss can be computed with respect to the output Q(si,ai|θQ) of the critic network for the i’th sample. See [1] and [2] for more details.

Extension to MA-DDPG

Traditional reinforcement learning approaches such as Q-Learning or policy gradient are poorly suited to multi-agent environments. One issue is that each agent's policy is changing as training progresses, and the environment becomes non-stationary from the perspective of any individual agent (in a way that is not explainable by changes in the agent's own policy). This presents learning stability challenges and prevents the straightforward use of past experience replay, which is crucial for stabilizing deep Q-learning. Policy gradient methods, on the other hand, usually exhibit very high variance when coordination of multiple agents is required. Alternatively, one can use model-based policy optimization which can learn optimal policies via back-propagation, but this requires a (differentiable) model of the world dynamics and assumptions about the interactions between agents. Applying these methods to competitive environments is also challenging from an optimization perspective, as evidenced by the notorious instability of adversarial training methods.

The method used is centralized training with decentralized execution, allowing the policies to use extra information to ease training, so long as this information is not used at test time. It is unnatural to do this with Q-learning without making additional assumptions about the structure of the environment, as the Q function generally cannot contain different information at training and test time.

The MADDPG can be seen as a simple extension of actor-critic policy gradient methods where the critic is augmented with extra information about the policies of other agents, while the actor only has access to local information. For more detailed information see [4]

Hyperparameters

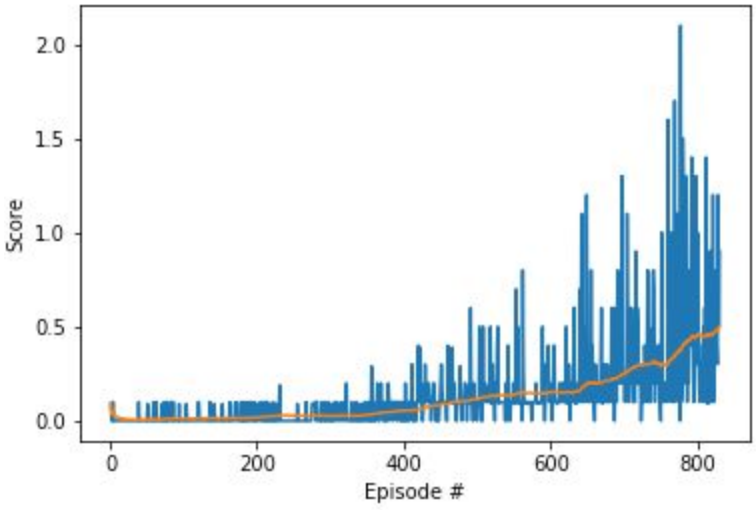
The learning rate that worked was 1e-4 for the actor and 1e-3 for the critic, a batch size of 128 were used along with replay buffer size of 500000, gamma .99 and Tau of 1e-3. There was no change to the default Ornstein-Uhlenbeck noise parameters (0.15 theta and 0.2 sigma.). I also removed the epsilon greedy part of DDPG.

Training

The training is mostly the same as for a normal DDPG-based task.

Results and Future Work

The model was able to achieve the 0.5 average reward goal after 829 episodes.



In the future, I plan experimenting with different network architectures and other combinations of hyperparameters. I am curious to see what are the most influential factors on network convergence and stable training. That is because I noticed so far

that there is still a lot of instability in training, depending on initial weight initialization and exploration vs exploitation set up. I think it also depends on the first episodes played, for the same architecture and same hyperparameters, network still converges, but after different training episodes, with a lot of variance (by example 2000 vs 8000). I plan to investigate more on this, as it takes quite a while for the training.

Bibliography

1. Silver, et al. Deterministic Policy Gradients
2. Mnih, et al. Human-level control through deep reinforcement learning
3. Patrick Emami Deep deterministic policy gradients in Tensorflow
4. Rove, et all Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments