DRLND - Collaboration & Competition

Learning Algorithm

The learning algorithm used in this project is the Proximal Policy Optimization (PPO) algorithm. PPO is a policy gradient algorithm which uses the following improvements for better learning:

- For efficiency, reuses trajectories sampled from an older policy, and re-weights them based on how likely they are with respect to the current policy.
- Clips the surrogate objective function obtained as a result of the above re-weighting, to ensure that the surrogate does not continue to have a gradient once a point near the true optimum is reached.
- Uses multiple trajectories sampled from a single policy, to get a better Monte-Carlo estimate of the expected return (which is the objective we are trying to maximize).
- Attempts to assign credit to actions by using a return value which includes only the immediate reward obtained by executing the action as well as all future rewards henceforth (i.e. no rewards from the past).

For this project, I used the PPO implementation available in the Shangton Zhang DeepRL repository (https://github.com/ShangtongZhang/DeepRL). The main enhancements I made for the sake of this project are the following:

- A custom extension of BaseTask (TennisTask) which could suitably wrap the current environment. Since this is a collaborative environment, the two agents cannot be seen in isolation. Accordingly, the TennisTask implementation creates a single compound agent whose state and action spaces are the concatenation of those of the two individual agents. Thus, the input to the PPO actor/critic networks becomes a single vector of length 48. The output of the actor network is also a vector of length 4, which is reshaped back to (2,2) before applying to the actual environment. Lastly, since the episodic reward of the compound agent is effectively the joint reward of the two agents, the compound reward at each step is computed as the sum of the individual step rewards. Owing to the symmetry of the problem, this compound episodic reward serves as a pretty good proxy for the actual metric we are trying to maximize, which is the individual episodic reward of the better of the two agents.
- Book-keeping for keeping track of the actual metric we are trying to maximize (the individual episodic reward of the better of the two agents). Two separate counters are maintained for accumulating the step rewards of each agent within a specific episode. At the end of the episode, the higher of the two is chosen as the reward for the current episode, and added to a sliding window of length 100 and a full history. The counters are also reset to zero after the end of each episode.

The following were the hyper-parameters chosen for this algorithm:

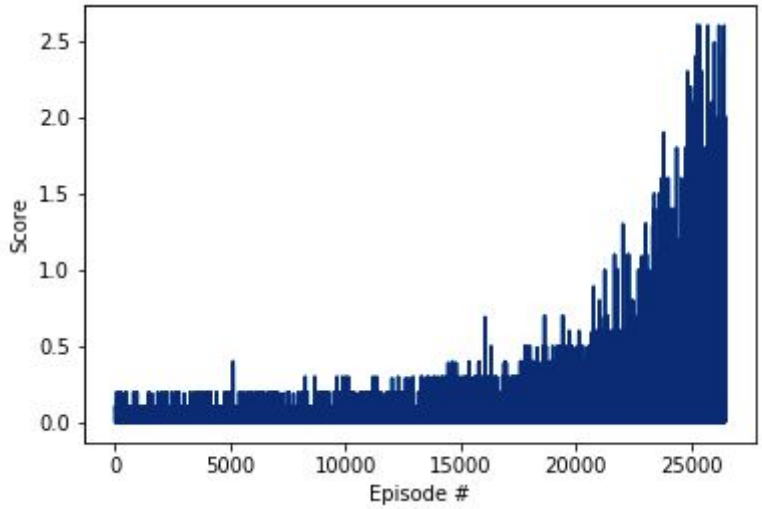
- Adam optimizer with a learning rate of 3e-4.
- Discount rate (gamma) of 0.99
- A tau value of 0.95 for soft-update of target networks.

- Gradient clipping using an absolute threshold of 5.
- An epsilon of 0.2 for clipping the surrogate objective.
- A maximum number of steps equal to 2e7.

The network architecture used was an actor-critic setup. The actor was a simple fully connected network with two hidden layers (each with 64 units) and a RELU activation. The output layer of the actor had as many units as the dimension of the compound action-space (4), followed by a tanh activation. Some Gaussian noise was further added to the predictions of the actor network. The critic was also a fully connected network with two hidden layers (each with 64 units) and a RELU activation. The output layer of the critic had a single unit (since it was emitting a scalar prediction for the value function), with a linear activation.

Results

The following is a plot of averaged episodic rewards over time. Note that each point here is the score of the better of the two agents at the end of an episode. Also note that though the curve fluctuates a lot, it has a generally upwards trend. Towards the end, the curve manages to reach high enough values that the mean over 100 consecutive points is +0.5, resulting in the environment being solved as per our definition.



A total of **26,474** episodes were needed to solve the environment.

Future Work

The compounding approach I used worked, but it isn't the only approach available for the current problem. Also, training was rather slow, and a huge number of episodes were needed to solve the environment. Accordingly, I can think of a couple of areas for future improvement:

- Tinker with the hyper-parameters which can speed up the training (e.g. the network learning rates).
- Use an alternative approach for training multiple agents (e.g. the MADDPG reference code provided in the classroom).