Collaboration and Competition Deep Q Learning Task

Approach taken

As this was a task featuring multiple agents in a continuous action space, a sensible approach required an algorithm suitable for these constraints.
Typically, single agent reinforcement learning algorithms won't translate naively to the multiple agent setting. A reason for this is that the environment is non stationary from the perspective of any one agent - as the environment can change due to the changes in the other agents. With this constraint in mind, I still opted to train both agents separately with DDPG, with no joined knowledge during training. The motivation here was that as the two agents were physically separated in this tennis like game, perhaps the non stationarity of the environment wouldn't be a major issue - as the agents are not directly interacting with each other.

DDPG is an actor critic method, which trains two neural networks - an actor and a critic. The actor is responsible for providing the actions taken by the agent - it maps from states to actions. The critic is responsible for evaluating the Q function - giving a score to state, action pairs proposed by the actor. By using the Q score from the critic as a reward function, the actor can be trained.

Both neural networks used two fully connected hidden layers (256 nodes in the first layer, and 128 in the second), with ReLU activation between the hidden layers. The actor used tanh activation in the output layer.

Footnote: there is a small bug in the implementation. When reporting scores for good runs (those with an average score of the past 100 runs of 14 of more), the notebook prints off an index for the iteration 100 less than the correct value. This is visually jarring, and incorrect, but has no impact beyond that.

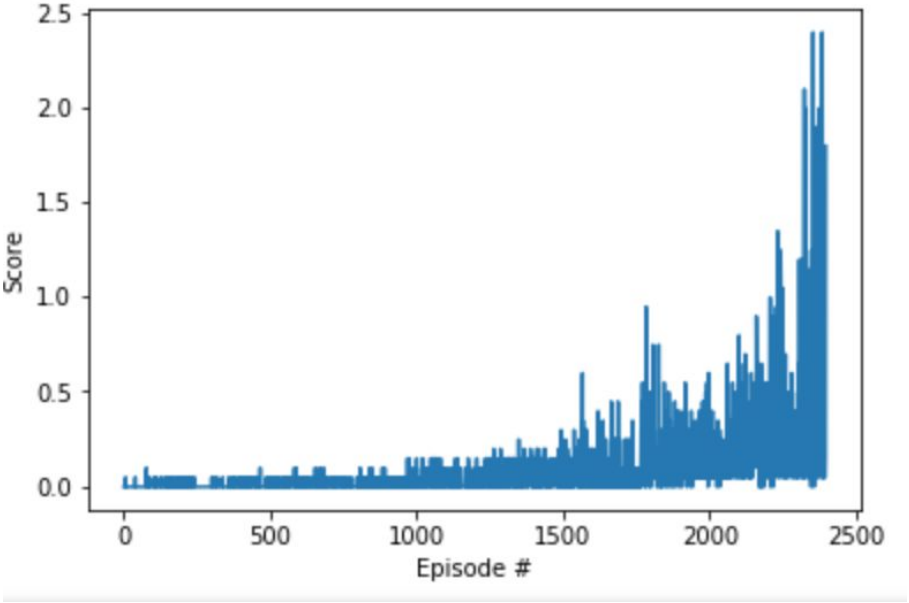
In this multi agent case, two independent DDPG agents with identical hyperparameters were trained in the environment, receiving separate rewards and state inputs, and producing separate actions.

Hyperparameters

- Buffer size (number of items held in replay buffer) = 100000
- Batch size = 128
- Gamma = 0.99
- Tau = 0.001
- Learning rate = 0.0001 for both neural networks

Results

Training was initially very slow, with no noticeable improvement made during the first 1000 iterations. It then picked up fairly rapidly, albeit with a few dropoffs in performance - possible because once each agent is comfortable returning the ball once, there's nothing fundamentally different with doing it twice, three times etc. It took around 2300 iterations for the agent to reach an average score (over the past 100 episodes) of 0.5



Footnote: there is a small bug in the implementation. When reporting scores for good runs (those with an average score of the past 100 runs of 14 of more), the notebook prints off an index for the iteration 100 less than the correct value. This is visually jarring, and incorrect, but has no impact beyond that.

Next steps

- Hyperparameter estimation with this setup was quite difficult, with more computational power available, something like grid search with multiple parallel runs would make this efficient, and would likely lead to better hyperparameters being chosen
- Due to computation limits, a small fully connected neural network was used. A larger network may be able to perform better on the task.
- The model operated over a small state space as it knew the inner mechanics of the world. It would be interesting to train a model from pixel information using a convolutional network.
- It's likely that multi agent DDPG (MADDPG) would outperform this solution. MADDPG allows the DDPG critics to be aware of the other agents (there is shared training), which may have led to faster convergence.

Footnote: there is a small bug in the implementation. When reporting scores for good runs (those with an average score of the past 100 runs of 14 of more), the notebook prints off an index for the iteration 100 less than the correct value. This is visually jarring, and incorrect, but has no impact beyond that.