Can RL Agents Learn to Cooperate? - Using Actor Critic to Beat 2v2 Soccer

Joshua Peterson

Abstract

When a reinforcement learning agent interacts with the real world, the desired behavior is much more complex than in a one versus one game. There is a need to develop reinforcement learning strategies that work within cooperative and competitive scenarios. This work attempts to solve this by emulating the strategy used in training for team sports. By having one agent be in charge of the value function for all other agents being trained, cooperation can more easily develop. This study shows that RL agents can learn to cooperate, and that RL agents learn more quickly with a common coach.

Intro

At the beginning of this project I wanted to explore reinforcement learning agents cooperating and competing within the same environment. I found a soccer environment which had cooperation and competition built in. The environment was developed by Deep Mind using the Mujoco physics engine. As I worked on the environment however, I ran into several different problems and eventually built a solution which did show the agent learning.

The environment is represented by 111 different numbers, representing information about the ball, other players, goal, and field. The input into the environment is a list of all the agent's actions representing how much the agent runs, turns, and jumps.

The environment only rewards the agents if they score a goal. An environment having sparse reward structures is a difficult problem for reinforcement learners to overcome. This is due to the difficulty in evaluating how good a given state is, particularly if the experience did not result in a reward.

I attempted to solve the sparse reward environment by adding onto the existing reward structure with other rewards. The idea was to give the agent a small reward for getting close to the ball, a larger reward for getting the ball close to the goal, and the largest reward for scoring a goal.

The following reward structures offered no tangible results:

- negative reward for how far away from the ball the agent was at each timestep
- positive reward for how close the agent was to the ball at each timestep
- positive reward for the distance between the ball and goal for each timestep

For each of these reward structures the agent had a difficult time seeing a cause-effect relationship between the agent's actions and the given reward.

In addition to adding subgoal reward structures, I also added an exploration rate in an attempt to generate scores from

random input. This strategy produced no noticeable change in learning.

Another aspect that made the environment difficult was having multiple agents. Reinforcement learning agents have a difficult time learning in multiagent environments; because an agent is unaware of other agents acting in the environment, so it seems that the environment is changing of its own accord. Figure 3 shows an example of this happening. Near the end the blue agent scores and the red agent gets confused because of the negative reward associated with being scored on. I attempted two different strategies to overcome this obstacle.

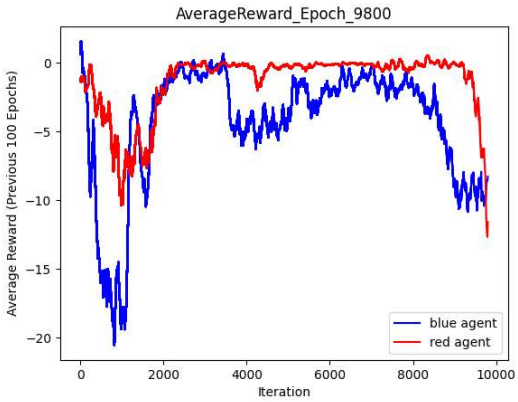

Figure 2 - Multiple agents causing memory loss

The first thing I tried was a method called MADDPG (Multi Agent Deep Deterministic Policy Gradient) . This method modifies the existing actor critic method, by giving the critic access to the observation space and actions of all agents. In this method the policy is generated only from an actor's state space, but the value of any given space takes into account the states and actions of other agents in the environment.

Another thing I attempted to alleviate this problem was reducing the number of agents in the environment from four to two. I decreased the number of agents to reduce computation time as well as to reduce the likelihood of other agents adding complexity as well as to reduce computation time.

Another aspect of the environment that made reinforcement learning harder was the representation of the action space. In some environments, such as the mountain car, the action input is one of a few options. In this environment the action is represented by three sliders representing how much to run, turn and jump. This difference in action representation required a different kind of neural network architecture and rendered many of my experiments obsolete, because I was using the incorrect action representation.

The last difficulty represented by this environment was the computation required to successfully do reinforcement learning. Without access to a high performance graphics card. This put a large hold on my effectiveness due to longer runtimes to see if changes introduced learning into the system.

To help with this problem I reduce the state representation space to only include the position of the ball, goal, and field corners, as well as the velocity of the ball. This change reduced the state array going into the neural network from 111 parameters to only 29 parameters.

Results

I tried two different algorithms throughout the course of this project. The first, MADDPG, I have already covered. MADDPG did not produce any kind of learning curve, though there were some problems in my implementation regarding the representation of the action space.

The strategy that did seem to work was a simple actor critic where one neural

network was tasked with evaluating how good a state was and another was tasked with deciding what action to take. Figure 1 shows the neural network architecture which I used to get eventual results in the project. Both the critic and the actor take in the same observation space, unlike MADDPG. The critic has a single layer outputting the value of the state. The actor has two layers leading into three unique layers outputting the value used to calculate the action the actor should take.

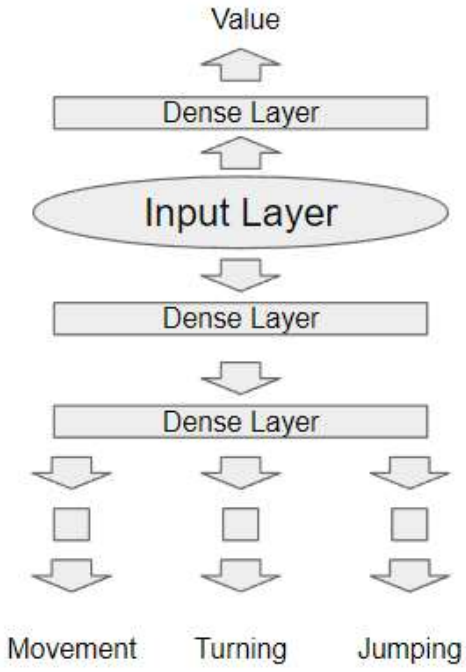

Figure2 - Neural Network Architecture

The loss function that eventually worked was the Mean Squared Logarithmic Error loss function. The critic's loss function is calculated as the difference between the expected value and the actual cumulative reward. This value is then used to calculate the loss of the actor by subtracting it from the mean of the -log of the actions taken.

Figure 2 shows the learning curve obtained from using the loss function and neural network.

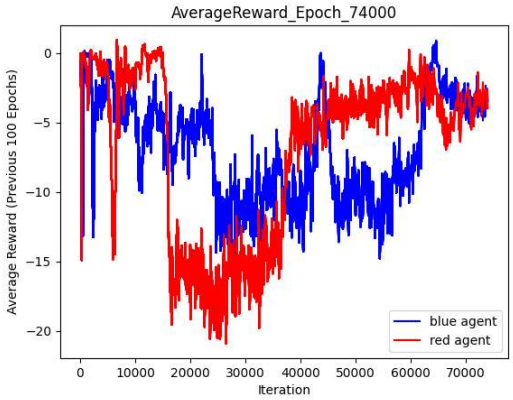

Figure 3 - Average reward using NN architecture

Conclusions

Reinforcement learning is a valuable tool that can be used to solve many different problems. It is however difficult to get working for particular problems such as the 2v2 soccer environment I was using. Many different aspects go into whether or not an agent learns, and my experience with this project has both increased my understanding and appreciation of state of the art reinforcement learning work.

Sources

https://www.youtube.com/watch?v=tZTQ6S9PfkE

https://arxiv.org/pdf/1706.02275.pdf

https://github.com/deepmind/dm_control.git

My github (currently private)
https://github.com/joshualp2000/CS5640-Project.git