Collaboration And Completion

Shreyas Chandra Sekhar

Abstract—Unity Reactor Environment was considered to study collaboration to complete a given task by two agents. Two agents control rackets to bounce a ball over a net. MADDPG was used to train the agents to collaborate and keep playing the rally as long as possible. A mean score of 0.5+ was attained for the last 100 consecutive episodes.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Localization.

1 Introduction

Multi agent systems available all around us. It can be any environment which involves more than one individual. This can be collaborative or competitive environment. The agents work towards a common goal in a Collaborative environments to complete a task together. Competitive environments were one agent tries to win over another. In either case the every agent in the system learns of its own as well as from the other agents as we human do.

There are many applications of Multi-agent systems. It can be from group of drones to deliver a package or constructing a building. An interacting robots with humans are multi-agents system.

2 Background

As mentioned multi-agent systems are all around us. Though it is difficult to train every agent in the system there are few advantage in training multiple agents together. The agents can share their experience all learn from each other. This leads to finding a source of communication between them which needs additional hardware and software.

Multi agent systems are robust. An agent can be replaced if they fail. Other agents in the system can take over the task of the failed agent. The complexity of the system increasing with the number of agents in the system. So, the Hardware and Software capabilities of the agents will have be analyzed for a multi agent system.

2.0.1 MARL

The process of training multi agents in deep reinforcement learning is termed as MARL. In a singe agent system action spaces would just be for that agent however in a multi-agent system would have action spaces relating to actions by every agent. Each of them has their own individual policy which would return an action for their observations. Each of them would have their own set of actions. Apart from this there is a joint set of actions in this system. This is represented by Markov games which is tuple as written below.

[1]

- Ai: Joint Action space of agent i
- Oi: Observation space of agent i
- Ri: Reward functions of agent i which returns a real value for acting an action from a particular state

$$\langle n, S, A_1, \ldots, A_n, O_1, \ldots, O_n, R_1, \ldots, R_n, \pi_1, \ldots, \pi_n, T \rangle$$

n : number of agents
S : set of environment states
$A : A_1 \times A_2 \cdots \times A_n$
$R_i : S \times A_i \to R$
$\pi_i : O_i \to A_i$
$T : S \times A \to S$

Fig. 1. Markov Games

- PIi: Policy of each agent i. Given the observations returns the probability distribution Ai
- T: State transition function. Given the current state and joint action function, it provides the probability distribution of set of next states.

The State functions are still Markovian as in MDP for a single agent. As for Markovian, next state depends only upon the present state and the actions taken in this state. However the transition function depends on the joint action.

2.1 Approaches to MARL

- Non-Stationary: Train all the agents independently without considering the existence of other agents. Every agent considers other agents to be part of environment and leans its own policy. As all are learning simultaneously, the environment is seen as the change from single agent dynamically. This condition is called Non-Stationary of the environment. In most single agent algorithm, it is assumed that the environment is stationary which leads to convergence guarantees. Using this in Non-Stationary would have no guaranty to converge.
- Meta-Agent: Meta-agent considers the existence of multiple agents. A single policy is learnt for all the agents. It takes present state of the environment as input and provides the Action of each agent in the form of singe Joint Action vector. A single reward function given the environment state and action vector return global reward. The joint action space increase exponentially with the number of agents.

If the environment is partially observable, or the agents can only see locally, each agent will have a different observation of the environment state. This leads to further complexity to speculate the environment with local observation. So, this approach works well only if each agent knows well about the entire environment.

$$\text{Policy} : S \rightarrow A_1 \times A_2 \cdots \times A_n$$

$$R : S \times A \rightarrow \text{Real Number}$$

Fig. 2. Markov Games

[1]

2.1.1 Multi-Agent scenarios)

Multi-Agent systems can be of three types. Namely, Cooperative, Competitive and Mixed Environments. Agents in a Cooperative environment looks to maximize the global rewarded of the system. Agents in Competitive environment are just concerned to maixmize their own rewards and the agents in Mixed Environment will have the characteristics of both Cooperative and Competitive agents.

2.1.2 MADDPG)

MADDPG is a multi-agent version of DDPG. DDPG is a off-policy Actor-Critic algorithm using target networks. The input of the actor network is the current state and the output would be a real value vector of the action chosen from a continuous action space. This algorithm is called Multi-Agent Deep Deterministic Policy Gradient (MADDPG). It uses framework of centralized training with de-centralized execution.

During training the Critic for each agent uses extra information like states observed and actions taken by all the other agents. The actor, their just one observation space for each agent. Each actor has access only to its observations and actions. During execution time, only these actors are present and hence own observations and actions are used. Learning critic for each agent allows us to use a different reward structure for each. This makes it enabled to use the same algorithm for all types of mixed agent scenarios, Cooperative, Competitive and Mixed scenarios.

[1]

2.1.3 Environment)

Unity Tennis environment was used for this experiment.

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

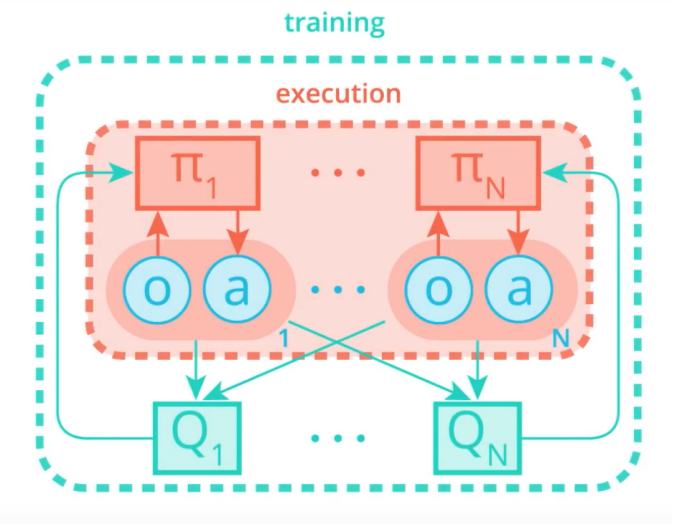

Fig. 3. Markov Games

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.

3 Results

There are many tuning parameters in the algorithms. There were many attempts made for tuning and the agents were trained to score 1.6 during last 100 episodes. This took 3000 episodes. The minimum target score of 0.5 was achieved in 2067 episodes. The desired results were achieved and the project was completed successfully.

[1]
[1]

4 Discussion

There are many hyper parameters to tune.

- BUFFER_SIZE = int(1e5) replay buffer size
- BATCH_SIZE = 128 minibatch size
- GAMMA = 0.99 discount factor
- TAU = 1e-3 for soft update of target parameters
- LR_ACTOR = 1e-4 learning rate of the actor
- LR_CRITIC = variable learning rate of the critic
- WEIGHT_DECAY = 0.0001 L2 weight decay

Most of these values here were taken from the reference paper. There were many trials with different tuning values.

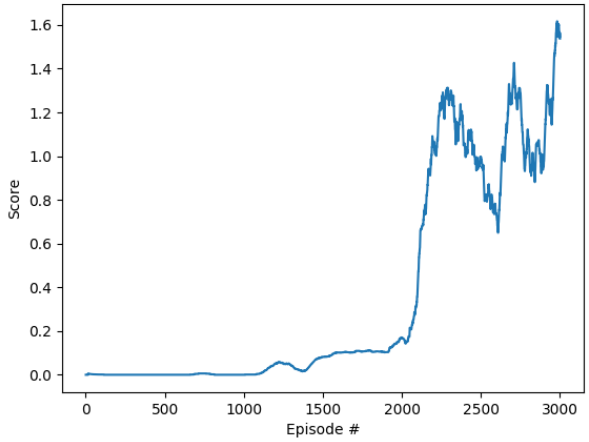

Fig. 4. Training

Buffer size was always kept at the mentioned value. Many trials of Batch size were tried. This did not show major impact on the result. Gamma was always kept at 0.99. TAU was taken from the reference paper and so Weight Decay. The learning rate of Actor and Critic showed huge impact on the results. Having them below 0.0001 or above 0.0002 showed very slow learning. The sweet shot was in between 0.0001 to 0.0002. There were many trials between these numbers finally the required score of 30+ could be achieved with learning rate of 0.0001 for both actor and critic after 190 trails.

5 Conclusion / Future work

There are many tuning parameter to be considered for improving the training time. Further analysis would be consider changing he DQN network for faster tuning.

The performance of the algorithm will be explored with Play Soccer Unity environment.

References

[1] A. T. J. H. P. A. I. M. Ryan Lowe, Yi Wu, *LATEX: Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments*. NIPS, 2017.