COLLABORATION AND COMPETITION

Sudhir Kumar Suman
16D070027
Dept. of Electrical Engineering
IIT Bombay

Saarthak Kapse
16D070041
Dept. of Electrical Engineering
IIT Bombay

July 16, 2020

1 Introduction

For artificial intelligence(AI) to reach its full potential, AI system need to interact effectively with humans and with other agents. There already exists such system in which this type of agent-human and agent-agent interaction takes place on massive scale such as stock market. Close interaction,collaboration and compete with others can yield immense advance in human life quality and successive changes in evolution of human population. Beside our intelligence which is predefined by our genes, our interaction and gathered experience with others Agents can reinforce our general mental capability to perceive differences, solve unconventional problems, comprehend complex ideas,etc. Therefore, the important step to build artificially intelligent system based on deep reinforcement learning approach is to involve interaction between multiple agents. One step along this path is to train AI agents to interact with other agents in both cooperative and competitive setting. This project deals with one similar setup and environment here we train two agents to play tennis in which the agent must bounce the ball between one another while not drooping the ball out of bounds.In this environment if the agent hits the ball over the net then it will receive a positive reward.If an agent lets a ball hit the ground or hits the ball out of bounds then it will receive a negative reward.Thus the goal of each agent is to keep the ball in play as long as possible

2 Related Work

There has been several approaches developed to solve Tennis Unity ML-Agents such as Q-learning or policy gradient. But these method are poorly suited to multi-agent environments problems.One issue behind using this method as discussed in [2] is that, as training progresses each agent's policy get changed and environment becomes non-stationary from perspective of any individual agent in a way that is not explainable by changes in 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. On the other hand Policy gradient methods usually exhibit very high variance when coordination of multiple agents is required. Applying these methods to competitive environments is also challenging from optimization perspective. One way of solving this environment is by using DDPG(Deep Deterministic Policy Gradient),in which each agent is trained using its own action as solved by Markus Buchholz[4]. Other method is by using MADDPG algorithm as done by PHRABAL[7]. And several other ways with some tweak in DDPG and MADDPG, like using Prioritized experience replay and adding exploration noise like Ornstein-Uhlenbeck Process as used in DDPG paper.

3 Problem Setup

Here we propose to solve Tennis environment which is similar but not identical to Tennis environment of Unity ML-Agents.In this environment, two agents control rackets to bounce a ball over the net. If an agent hits the ball over the net, it receives a reward of +0.01.If an agent lets the ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01.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 from the environment. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.The task is episodic and to solve the

environment agents must get an average score of +0.5 (over 100 consecutive episodes after taking the maximum over both agents). So as reward is +0.01 to hit ball over net and -0.01 if agent hit ball out of bound or hit the ground, to reach average of +0.5 over 100 consecutive episodes agents need to collaborate to keep playing so that rewards of both agents keep adding. Whereas they will also compete to gain more points than opponent agent. Specifically, After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields two potentially different scores. We then took the maximum of these two scores in order to get a single score for each episode.

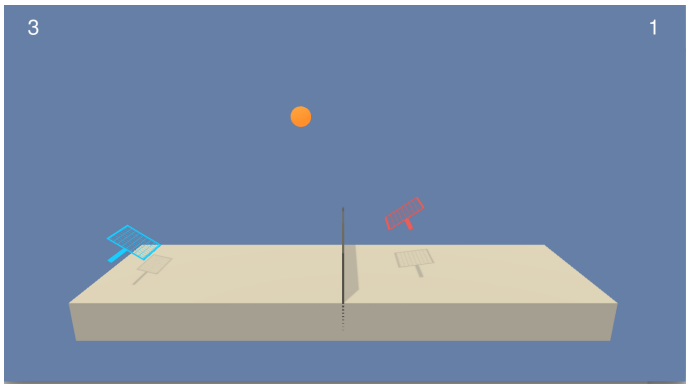


Figure 1: Tennis Environment

4 Method

Since the environment requires the training of two separate agents and under some situation, the agents need to collaborate to not let the ball hit the ground and in some situations will also compete with each other in order to gather more points. Since the action space is continuous which allow the agent to execute more complex and precise movements. There are unlimited range of possible action values that control the movement of agents even though each tennis agent can only move forward, backward and jump. Due to this complexity of environment, we need an algorithm that allows the tennis agent to utilize its full range of movement. For the purpose of solving this environment we can use the policy based method as policy based method are well suited for continuous action spaces.

So we solve this environment by Multi-Agent Deep deterministic policy gradient algorithm. We used Actor-Critic networks same as Deep deterministic policy gradients (DDPG)[3] algorithm but instead of training each agent to learn from its own action, we incorporate actions taken by both agents. The role of the actor is to determine best action and the critic evaluate the quality of action as determined by actor. It was beneficial as environment state depends on the actions taken by both the agents. Both the agents have separate actor and critic neural network. For each agents, actor take state of both the agents and gives the action to take and critic network take both state and action of both the agent and return Q-Value which is then used to make actor network learn.

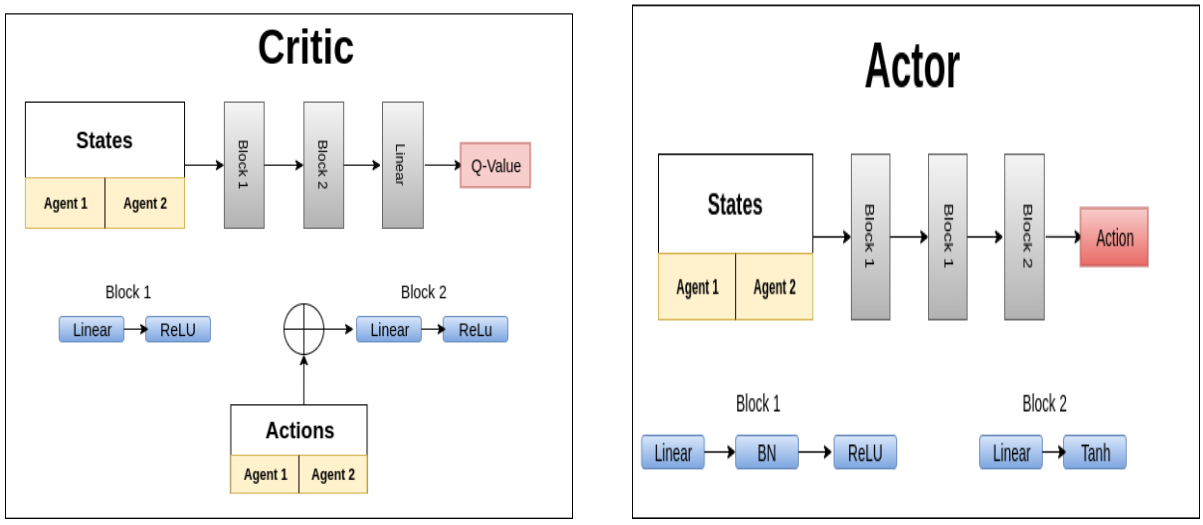
4.1 Network Architecture

4.1.1 Critic Network

Critic Network is a feed-forward neural network. Input to this network is states of both the agents and actions taken by both the agents and output of this network is Q-Value of the given state and action. State of each agent is of size 24 and actions taken by each agent is of size 2. For Critic we used a two layer neural network with input size of 48 unit, first hidden layer of size 256, second hidden layer of size 128 and the network output layer is of size 1. One point to be noted is we first give both agents state of total size 48 to model as input. Which is then passed through model and gives output of first hidden layer of size 256. Which is then concatenated with actions of both agents which is of size 4. So input is passed as 48 -> 256 which is then concatenated with actions and thus become of size 260 which is then passed to second hidden layer as (256+4) -> 128. And then is further passed through model and output of size 1 is generated. We tried giving states and action both together as input but model was taking to much episodes to learn and it was very unstable. In this network we used ReLU as activation function.We have avoided to use BatchNorm for Critic. We tried out with Dropout layer but it was making our model unstable.

4.1.2 Actor Network

Actor is also a feed-forward network whose input is states of both the agents and output of this model is action to be taken by the agent. So output of this model is of size 2, as there are two continuous action in this Tennis environment. Two continuous actions which the agent can take is to moves toward (or away from) the net, and jumping. For Actor we used a two layer neural network with input size of 48 unit, first hidden layer of size 256, second hidden layer of size 128 and the network output layer is of size 2. For actor we used ReLU as activation function for block1 as shown in Figure 2(b). And Tanh for output layer, as actions are in range [-1,1]. In this network using BatchNorm was beneficial. Plots with and without BatchNorm is provided in Results section below. Here too dropout was making model very unstable.



(a) Critic Network (b) Actor Network

Figure 2: Network Architecture of critic and actor

4.2 Algorithm

We ran our model till mean reward of last 100 episodes reaches 0.5. In each episode, game is continued till one agent drops the ball or hit it out of bound. Till the end of episode, every time ball is kept into play (that is to and fro motion) reward of +0.01 for each agent is added and the agent who drop the ball its reward is subtracted by 0.01 and for each episode max of reward of both agent is taken as reward of that episode.

We address the issue of exploration vs exploitation by using Ornstein-Uhlenbeck process as suggested in the paper by Google DeepMind. This process adds some noise to the action values at every time. This noise is correlated to previous noise and therefore tends to stay in the same direction for longer duration without canceling itself out. This allows the agent to maintain velocity and explore the action space with more continuity in each episode. We are also giving exploration boost to our model at the start of episodes by multiplying it by Ornstein-Uhlenbeck noise with a noise coefficient which we are decaying every episodes to eventually reach zero. After it reaches zero than it is assumed that the model has explored enough and now it only have to exploits thus we can see plots in results section that average reward of last 100 episodes keeps increasing suddenly after some 200-300 episodes as compared to episodes where it exploring more.We are clipping the actions in -1 to 1 after action is added with (noise*noise coefficient).

In each episodes while ball is kept into play, agents keep acting and we store it in Replay Buffer and actions are given to environment which gives the reward of that particular actions and also next state which is stored in Reply Buffer. Pseudo code of the algorithm is written below. Actor model has a actor_local and actor_target. Whereas Critic model has a critic_local and critic_target. Where target models give target values and local models give expected values, MSE_Loss(Mean Squared Error) is used between target values and expected values to train both models.

Algorithm 1 Multi-Agent Deep Deterministic Policy Gradient for N agents

for episode = 1 to M **do**
  Initialize random process N for action exploration
  Receive initial state x
  for $t = 1$ to max-episode-length **do**
    For each agent i, select action $a_i = \mu_{\theta_i}(o_i) + N$ w.r.t the current policy and exploration
    Execeute action a = $(a_1, \ldots, a_N)$ and observe reward r and new state x'
    Store (x,a,r,x') in replay buffer D
    $x \leftarrow x'$
    for agent i = 1 to N **do**
      Sample a random minibatch of S samples (x^j, a^j, r^j, x'^j) from D
      Set $y^j = r_i^j + \gamma Q_i^{\mu'}(x'^j, a'_1, \ldots, a'_N)|_{a'_k = \mu'(\sigma_k^j)}$
      Update critic by minimizing the loss $L(\theta_i) = \frac{1}{S}\sum_j (y_j - Q_i^{\mu'}(x'^j, a'_1, \ldots, a'_N))^2$
      Update actor using the sampled policy gradient:
      $\Delta_{\theta_i} \simeq \frac{1}{S}\sum_j \Delta_{\theta_i}\mu_i(o_i^j)\Delta_{a_i} Q_i^{\mu'}(x'^j, a_1^j, \ldots, a_N^j)|_{a'_i = \mu'(\sigma_i^j)}$
    end for
    Update target network parameters parameter for each agent i:
    $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
  end for
end for

Algorithm 2 Learning of Actor-Critic and Agent network (Simplified)

REQUIRE: states, actions, rewards, next_states, dones = Replay Buffer
Update_critic
  actions_next = actor_target(next_states)
  Q_targets_next = critic_target(next_states, actions_next)
  Q_targets = rewards + (discount_factor * Q_targets_next * (1 - dones))
  Q_expected = critic_local(states, actions)
  critic_loss = MSE_Loss(Q_expected, Q_targets)
Update_actor
  actions_pred = actor_local(states)
  actor_loss = -critic_local(states, actions_pred).mean()
Update_target_networks
  critic_target= tau*critic_local + (1-tau)*critic_target
  actor_target= tau*actor_local + (1-tau)*actor_target

5 Result

In this section we will present you the dependencies of model on some of the hyper parameter. Here we show ablation study of our model on 3 hyper parameter. First we have shown that, how model depends on value of TAU. TAU is the soft update parameter, and it decides how much the target model for both actor_target and critic_target should change after critic and actor local models are updated. See Algorithm 2.
Then we have shown how using BatchNorm helps model to become more stable and learn fast as compared to not using it in neural network. One point to be noted is we are only using in BatchNorm in Actor model, in both actor_target and actor_local. See figure 2. We haven't used BatchNorm in Critic Network, but have used it in Actor Network in Block 1. And finally we have shown dependency on Discount factor.

5.1 Hyper parameters

Here we have shown that the main hyper parameters of the model need to be tuned. We mainly focused on TAU, discount factor, noise_coefficient_start and using of BatchNorm. We tried tuning hyper parameters and the best which work on our model is in table below.

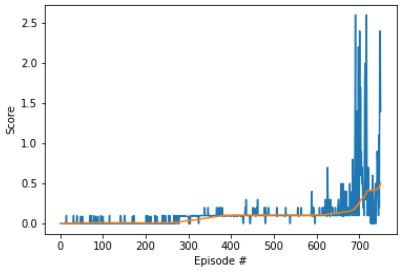
Parameter	Value	Description
Buffer Size	1e6	Reply Buffer Size
Batch_Size	128	Minibatch size
Discount_factor	0.99	Discount factor
TAU	8e-2	Soft update of target parameters
LR_ACTOR	1e-3	Actor learning rate
LR_Critic	1e-3	Critic learning rate
WEIGHT_DECAY	1e-6	L2 weight decay
Noise_coefficient_start	20	Exploratory boost coefficient
Noise_coefficient_decay	250	Exploratory boost coefficient decay
Seed	0	To make results deterministic
Sigma	0.2	OU Noise standard deviation

5.2 Analysis

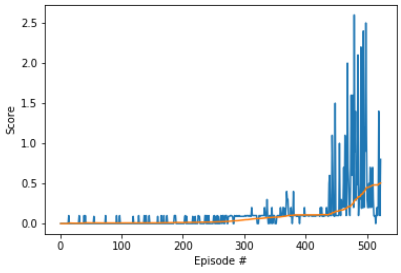
In all the plots orange line is moving average and when it reaches 0.5 our task ends. And here blue line is score value at each episodes and its fluctuating too much but as model starts to learn its value increases rapidly. We have done ablation study on TAU, discount_factor, Batchnorm and noise_coefficient_start and learning rate of both actor and critic model. But to keep the paper short we only present model dependency on TAU, BatchNorm, and Discount factor as model is most sensitive to these parameters.

5.2.1 Dependency on TAU

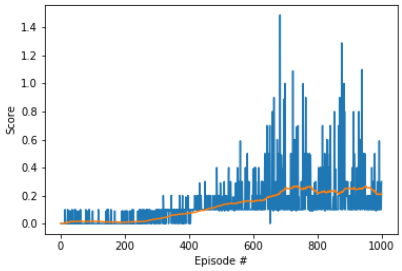
We can observe that at high TAU values like at TAU=0.4 the model is not able to learn well and is saturated at mean reward of around 0.25. Its obvious that high TAU value will make the model unstable as stated in literature and at TAU=0.01 and TAU=0.08 we can see that after some episodes average score(orange line) increases with high slope. When TAU=0.08 the average reward of last 100 episodes reached 0.5 around 400 episodes. Whereas in TAU=0.01 it takes around 650 episodes to reach the goal.



(a) TAU = 0.01



(b) TAU = 0.08

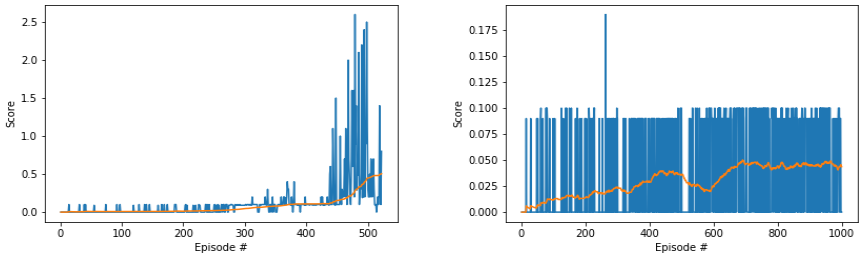


(c) TAU = 0.4

Figure 3: Score Plot variations on TAU value

5.2.2 Dependency on BatchNorm

We can observe with BatchNorm our target, average reward of 0.5 of the last 100 episodes is reached in 400 episodes. Whereas when BatchNorm is not used in model, the model is not able to learn and it is saturating at much small average reward than 0.5. The main reason of this is without BatchNorm gradients may explode. One way to make model learn without using BatchNorm is by gradient clipping but by using BatchNorm we can avoid the problem of exploding gradients.

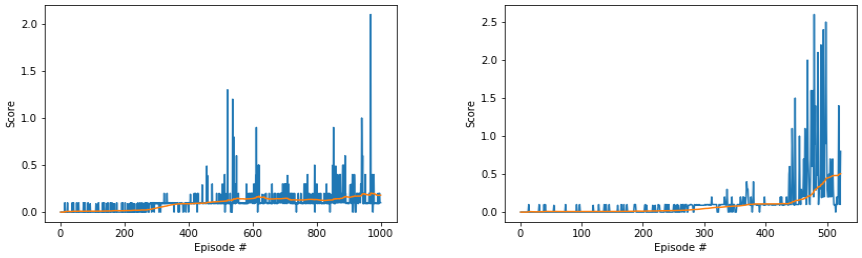


(a) Batchnorm is Used (b) Batchnorm is Not Used

Figure 4: Score Plot variations on using Batchnorm or not using it.

5.2.3 Dependency on Discount Factor

We observe that we need to set discount factor high. When discount factor is 0.90, our model is learning but it is learning slowly and also it is saturating and is not able to reach target. As we keep discount factor 0.99 our model was able to learn and there was not any issue of saturation hence we finally kept discount factor fixed at 0.99 as our optimal value .



(a) Discount Factor = 0.90 (b) Discount Factor = 0.99

Figure 5: Score Plot variations on Discount Factor.

6 Conclusion & Future Work

Here we solved the Tennis environment using multi-agent learning algorithm and we observed that our model is very sensitive to value of TAU. Batch Normalization and Discount factor and tuning of TAU is very important to train model fast and make it stable. Keeping high value of discount factor is really important to solve our task. We finally concluded that BatchNorm is really important for model to learn to avoid model from exploding gradient.
Here we used MADDPG instead of DDPG because in DDPG we give state and action of that agent to its critic and actor network, whereas in MADDPG we give state and action of both the agents to critic and actor network of both the agents. This will be beneficial as environment is not stationary and depends on other agent too. One more reason is as aim of this task is to reach mean reward of last 100 episodes greater than 0.5, so its sort of collaboration task and hence it would be beneficial if both agents get states and action of itself and opponent agent. Collaboration as agents will try to keep the ball in play and hence increasing episode reward, and Competition as agent will try get more reward than the other agent in each episode.

The following things can be explored to improve training and make model more stable:-

1. Add prioritized experience replay. That is instead of giving experiences randomly to model, it will be better if we give more priority to experiences which has higher magnitude of error. This way rare experiences would have better probability of selecting in experiences list. This is what exactly prioritized experience replay does and we think it may make training fast and more smooth.
2. Instead of using a separate critic for both the agents we can have a one central critic which may help collaboration and thus playing game for long.
3. Experiment with architecture of network and make it more deeper by adding more hidden layers. And tuning learning rate and also figuring out why dropout is failing in this task.
4. Experimenting more with noise_coefficient_start and its decay rate will tune how much of exploration is actually required. And thus we can exploit more and thus can train model faster.

References

[1] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, Igor Mordatch Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments *arXiv preprint arXiv:1706.02275*, 2017.

[2] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra Continuous control with deep reinforcement learning *arXiv preprint arXiv:1706.02971*, 2015.

[3] Udacity Tennis Environment *Github repository Github:deep-reinforcement-learning*, 2018.

[4] Markus Buchholz Deep Reinforcement Learning. Multi Agent Environment *Medium Article Medium:deep-reinforcement-learning-multi-agent-environment-collaboration-and-competition*, 2016.

[5] Henry Learning to play Tennis using DDPG *Medium Article Medium:learning-to-play-tennis-using-ddpg*, 2018.

[6] Thomas Tracey Training bots to Play Tennis *Medium Article Medium:/training-two-agents-to-play-tennis*, 2018.

[7] Phrabal Tennis Collaboration *Github repository Github:PHRABAL-DRL-Tennis*, 2018.

[8] Tom Schaul, John Quan, Ioannis Antonoglou, David Silver Prioritized Experience Replay *arXiv preprint arXiv:1511.05952*, 2015.