Report - Multi-Agent-DDPG-Tennis

Thursday, 20 August 2020 11:16 AM

<table>
<tr><th>S.No</th><th>Topic</th><th>Details</th></tr>
<tr><td>1.</td><td>Project Goal</td><td>This Tennis project is as part of Udacity Nanodegree - AI Deep Reinforcement Learning Expert and aims to develop a collaborative AI Agent - Tennis- to Self-Play in Continuous space using Policy-based 'Actor-critic' Methods using Deep Neural Networks.

From <https://github.com/SENC/AIReacher/blob/master/README.md></td></tr>
<tr><td>2.</td><td>Scope</td><td>• Develop an collaborative DDPG Agents using 'actor-critic' methods - which should learn the best policy to maximize its rewards by taking best actions in the given continuous environment

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.

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.</td></tr>
<tr><td>3.</td><td>Purpose</td><td>• One of the primary goal of AI is to solve complex tasks in high dimensional , sensory inputs . Though Deep Q Network (DQN) pr oved to be high performance on many Atari video games but handles well in discrete and low -dimensional action spaces .DQN can't applied directly to continuous domain since the core part to find the action that maximizes the acti on-value function.
• This project aims to build a model-free, off-policy actor-critic [Deterministic Policy - action-value] algorithm using deep function approximators that can learn policies in continuous space
• DDPG Paper: https://arxiv.org/abs/1509.02971</td></tr>
<tr><td>4.</td><td>Solution Approach -Policy based Methods</td><td>• Policy Gradients - An alternative to the familiar DQN (Value based method) and aims to make it perform well in continuous state space. Off-policy algorithm - Essential to learn in mini-batches rather than Online
• Develop 'Actor-Critic' agent uses Function approximation to learn a policy (action) and value function
  • Have 2 Neural Networks
    ○ One for an Actor - Takes stats information as an input and actions distribution as an output
      ▪ Take the action to move to next state and check the reward (Experience)and using TD estimate of the reward to predict the Cri tic's estimate for the next state

    ○ Next one for a Critic - Takes states as input and state value function of Policy as output.
      ▪ Learn to evaluate the state value function Vπ using TD estimate
    To calculate the advantage function and train the actor using this value.
    So ideally train the actor using the calculated advantages as a baseline.
  • Instead of having baseline using TD estimate , can use Bootstrapping to reduce the variance
    ○ Bootstrapping - generalization of a TD and Monte-Carlo estimates
      ▪ TD is one step bootstrapping and MC is infinite bootstrapping
      ▪ Mainly to reduce biasness and variances under controlled & fast convergence

• Like DQN , have 'Replay Memory' - a digital memory to store past experiences and correlates set of actions -REINFORCE- to choose actions which mostly yields positive rewards
  • Randomly collect experiences from the Replay Memory in to Mini-batches so the experiences may not be in same correlation as Replay Memory to train the Network successfully
  • Buffer size can be large so allowing the algorithm to benefit from learning across a set of uncorrelated transitions
• Little change in 'Actor-critic ' when using DDPG - to approximate the maximizer over the Q value of next state instead of baseline to train the value

#1 In Actor NN : used to approximate the maximizer - an optimal best policy (action) deterministically - so the Critic learns to evaluate the optimal policy - Action-Value function for the best action
  Approximate the Maximizer - to calculate the new target value for training the action value function
  Q(s, μ (s ; θμ);θ Q)

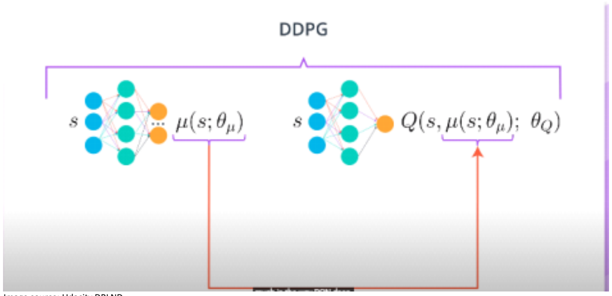
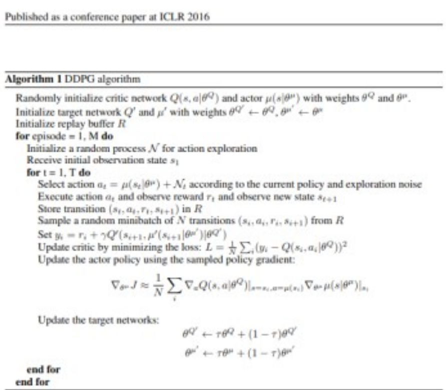


Image source: Udacity DRLND

Regular/local network - UpToDate network since training is done here but target network used to predict the stabilize strain

#2 Soft Target updates:
Weight of the target network are updated by having them slowly track the learned networks to improve the stability of learning

Since multi-agent nature,
1. Having 'Replay Memory' common for both agents so all experiences by each agents getting stored in common Memory for replay
2. Actor Network made it to common for both 2 agents since need an collaborative env
  a. Idea to learn from both agents stats,rewards to make it more sync
3. Main (Tennis.ipynb) --> MultiAgent --> Agent
  a. Multiagent has common Replay memory and Actor Network
    i. Create a child instance and set their actor network same but created own critic network</td></tr>
<tr><td>5.</td><td>Algorithm</td><td>Deep deterministic Policy Gradient



Published as a conference paper at ICLR 2016

Algorithm 1 DDPG algorithm

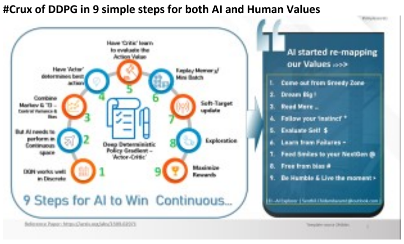
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

#Crux of DDPG in 9 simple steps for both AI and Human Values

</td></tr>
</table>

6. Hyper parameters - Value configurations

Udacity Workspace	Local Windows Env

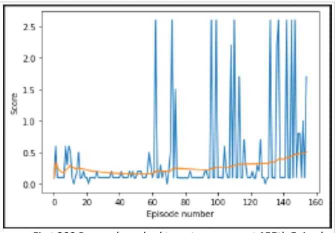
# Hyperparameters 	
SEEDC= 5
SEED = 1
GAMMA = 0.997999 # discount factor
TAU =0.0013 # for soft update of target parameters
LR_ACTOR =0.0002] # learning rate of the actor
LR_CRITIC =0.0003 # learning rate of the critic
BUFFER_SIZE =100000 #config["BUFFER_SIZE"]
BATCH_SIZE =512 # config["BATCH_SIZE"]
THETA = 0.77
SIGMA =0.9
EXPLORE = 0.79997
	HP :
	SEEDC=5
	Seed 1,
	Gamma :0.997 ,
	TAU:0.0013,
	LR_Act :1.3e-07 ,
	LR_Critic 1e-07 ,
	Mu 0.0,
	Theta 0.17,
	Sigma 0.79,
	ExploreFactor 0.9,
	IsTargetHardcopy:True

	Final one after multiple iteration with tuning Gamma : 0.998789 exploreFactor:0.5024, LrAct :1e-07, LrCritic:2.5e-06, Tau:0.0013, Sigma:0.29, Theta:0.193

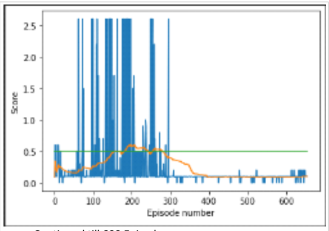
7. Result & Rewards plot

ENVIRONMENT SOLVED at 155th Episodes with Max Score of 2.6

Max Score achieved in episode 155 : **0.5020000075176358**



First 300 Run and reached target avg score at **155**th Episode



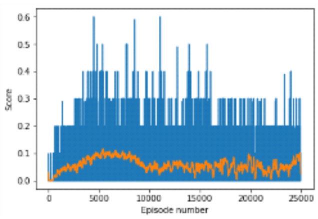
Continued till 600 Episodes

HyperParams:
Gamma:0.998789 , exploreFactor:0.09024,LrAct :0.0001,LrCritic:0.00025,Tau:0.0013,Sigma:0.29,Theta:0.193

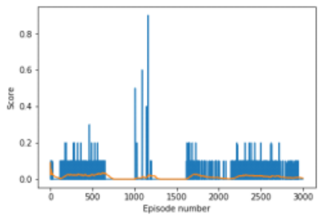
```
[15]
TestMultiAgents(env,colabAI2,episodes_score,scores_window,avg_score,n_episodes=500)
save_model_weights(colabAI2)

Episode: 60 Score: 0.30000000447034836  AvgScore (100+): 0.16600000249842803 CurrentWindowMax: 0.6000000089406967
Episode: 70 Score: 0.0  AvgScore (100+): 0.1980000029717173 CurrentWindowMax: 2.600000038743019
Episode: 80 Score: 0.09000000171363354  AvgScore (100+): 0.24050000361166896 CurrentWindowMax: 2.600000038743019
Episode: 90 Score: 0.10000000149011612  AvgScore (100+): 0.2258888922838701 CurrentWindowMax: 2.600000038743019
Episode: 100 Score: 2.600000038743019  AvgScore (100+): 0.26520000398159027 CurrentWindowMax: 2.600000038743019
Episode :100 score : 2.600000038743019   Avg Score(100+): 0.26520000398159027  WinMax 2.600000038743019
Episode: 110 Score: 0.10000000149011612  AvgScore (100+): 0.27520000413060186 CurrentWindowMax: 2.600000038743019
Episode: 120 Score: 0.10000000149011612  AvgScore (100+): 0.31530000472441316 CurrentWindowMax: 2.600000038743019
Episode: 130 Score: 0.0  AvgScore (100+): 0.3242000048607588 CurrentWindowMax: 2.600000038743019
Episode: 140 Score: 0.10000000149011612  AvgScore (100+): 0.3940000059083104 CurrentWindowMax: 2.600000038743019
Episode: 150 Score: 0.800000011920929  AvgScore (100+): 0.47300000708550216 CurrentWindowMax: 2.600000038743019
Max Score achieved in episode 155 : 0.5020000075176358
```

HISTORY

Udacity Workspace



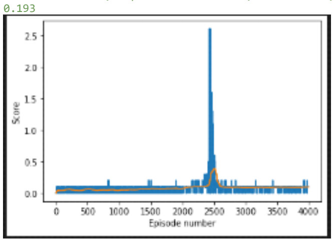

```
#HP HP :Seed 1,Gamma :0.9987999 , TAU:0.0011, LR_Act :0.00019 , LR_Critic 0.00019 ,
#Mu 0.0, Theta 0.26, Sigma 0.36, ExploreFactor 0.0003, IsTargetHardcopyFalse
showGraph(episodes_score,avg_score)
```

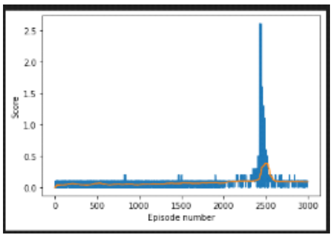


Local Windows Setup

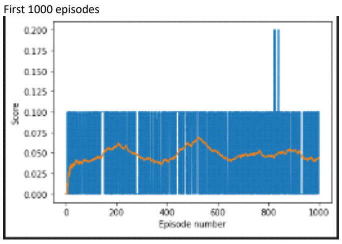
After updated the NN_Model 0 input neurons to 500/300 from 256/128 and batch norm for actor too

```
#Gamma:0.998789 , exploreFactor:0.5024,LrAct :1e-07,LrCritic:2.5e-06,Tau:0.0013,Sigma:0.29,Theta:0.193
```

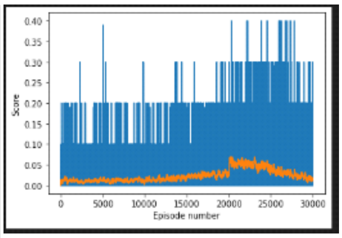




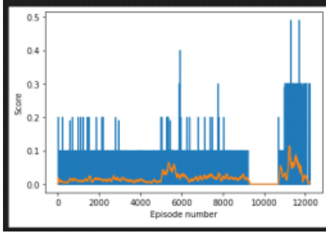
First 1000 episodes



Prev Iteration




```
#Training Tennis Agents
#HP
#Tune #3
#Gamma: 0.997899 Explore Factor: 0.17
#Gamma:0.9978 Explore factor : 0.0003 Lr for actor and Critic 0.00001 sigma 0.6 theta 0.17
showGraph(episodes_score,avg_score)
```

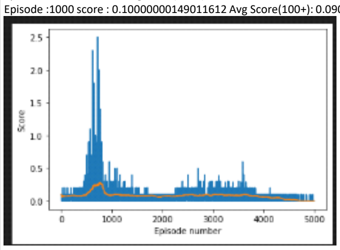


Learnings :

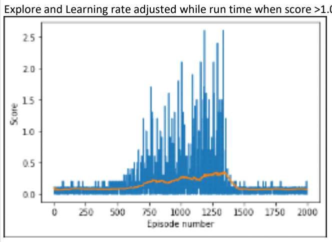
1. Its difficult to get this solved in first iteration itself
2. Since every step learning took long time and finally ends up very low score not even avg 0.1
3. So called the learning (Critic) only end of episodes but so started with random play but stored high rewards in separate memory
4. Once I get max rewards greater than 0.19 , saved those NN Weights and re-used with change in hyper params
5. Gamma mostly 0.9 so nothing much change as we know it should be .9 standard
6. Explore factor - Sampling/Noise is key …tried with 0.9 to 0.0032 to get some high score 1.7 just one episode
7. Tried tuning Learning rate for Actor and Critic and once I get consistent score (minimum learning) then kind of fixed
8. Tried both Udacity workspace plus local windows Unity and when I saw how the agent plays…could change the sigma,theta and explore factor to start with and one point made Sigma and Theta constant
9. Played with Explore factor - set 1 to 0..0032 and cud see the impact and memory
10. Refresh the memory since I created 2 separate Replay Buffer - one to record current rewards and another one to record only high scores > 0.19 to start and then changed to 0.39
11. Could see some improvement and got hit by max 0.4 to 0.6
12. Then Changed the Neural Network input neurons from 250 to 500 and hidden from 126 to 300 and added Batch Normalization in actor network too and increased the Dropout probablity from 0.01 to 0.03
13. Though got hit max 2.6 one time but average score was not even crossing 0.04 . This is where It took some days for me to tune Hyper params …started with quick play (Call learn at end of episodes for first 5K episodes and then called Learn at every step from 5001 to 10K episodes. Could see some improvement in learning
14. Using the weighted model , re-tried and started again from fresh but learning called at every step some iteration called learn 2 times (it took extra time to complete - sometime I used to start the training for whole night for 10k episodes
15. Finally got avg score increase upto 0.09 over 100+ episodes - but max score was 2.00
16. From that weight files, started one more quick run (calling Learning only end of episode) to sense the performance
17. Finally made it and solved the environment

WINDOWS:

Episode: 10 Score: 0.0 AvgScore (100+): 0.059000000916421415 CurrentWindowMax: 0.10000000149011612
Episode: 100 Score: 0.0 AvgScore (100+): 0.06960000105202198 CurrentWindowMax: 0.20000000298023224
Episode: 200 Score: 0.10000000149011612 AvgScore (100+): 0.08480000127106906 CurrentWindowMax: 0.20000000298023224
Episode: 300 Score: 0.09000000171363354 AvgScore (100+): 0.0766000011563301 CurrentWindowMax: 0.20000000298023224
Episode: 400 Score: 0.09000000171363354 AvgScore (100+): 0.07320000112056732 CurrentWindowMax: 0.20000000298023224
Episode: 500 Score: 0.0 AvgScore (100+): 0.098400001488626 CurrentWindowMax: 0.4000000059604645
Episode: 600 Score: 0.09000000171363354 AvgScore (100+): 0.18230000274255873 CurrentWindowMax: 1.1000000163912773
Episode: 620 Score: 0.10000000149011612 AvgScore (100+): 0.19510000294074417 CurrentWindowMax: 2.3000000342726707
Episode: 700 Score: 0.30000000447034836 AvgScore (100+): 0.23760000359266997 CurrentWindowMax: 2.3000000342726707
Episode: 770 Score: 1.4000000208616257 AvgScore (100+): 0.28230000426992774 CurrentWindowMax: 2.500000037252903
Episode: 800 Score: 0.10000000149011612 AvgScore (100+): 0.24890000380575658 CurrentWindowMax: 2.500000037252903
Episode :900 score : 0.0 Avg Score(100+): 0.1079000016860664 WinMax 0.9000000134110451
Episode :1000 score : 0.10000000149011612 Avg Score(100+): 0.090700001437217 WinMax 0.6000000089406967



Explore and Learning rate adjusted while run time when score >1.0 and <.6 while training

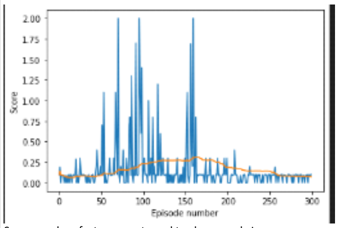


Used this saved model as a base and tried more iteration with different params

Episode: 10 Score: 0.09000000171363354 AvgScore (100+): 0.07400000132620335 CurrentWindowMax: 0.19000000320374966
Episode: 20 Score: 0.10000000149011612 AvgScore (100+): 0.07100000120699405 CurrentWindowMax: 0.19000000320374966
Episode: 30 Score: 0.10000000149011612 AvgScore (100+): 0.08600000143051148 CurrentWindowMax: 0.30000000447034836
Episode: 40 Score: 0.09000000171363354 AvgScore (100+): 0.0782500013243407 CurrentWindowMax: 0.30000000447034836
Episode: 50 Score: 0.20000000298023224 AvgScore (100+): 0.08820000145584345 CurrentWindowMax: 0.4000000059604645
Episode: 60 Score: 0.10000000149011612 AvgScore (100+): 0.11166666845480601 CurrentWindowMax: 1.1000000163912773
Episode: 70 Score: 0.10000000149011612 AvgScore (100+): 0.13542857356369495 CurrentWindowMax: 1.1000000163912773
Episode: 80 Score: 0.0 AvgScore (100+): 0.1583750024670735 CurrentWindowMax: 1.5000000223517418
Episode: 90 Score: 0.10000000149011612 AvgScore (100+): 0.15377778017686472 CurrentWindowMax: 1.5000000223517418
Episode: 100 Score: 0.10000000149011612 AvgScore (100+): 0.17910000277683139 CurrentWindowMax: 2.3000000342726707
Episode :100 score : 0.10000000149011612 Avg Score(100+): 0.17910000277683139 WinMax 2.3000000342726707
Episode: 110 Score: 0.09000000171363354 AvgScore (100+): 0.20630000317469238 CurrentWindowMax: 2.3000000342726707
Episode: 120 Score: 0.10000000149011612 AvgScore (100+): 0.21530000330880283 CurrentWindowMax: 2.3000000342726707
Episode: 130 Score: 0.10000000149011612 AvgScore (100+): 0.22650000346824528 CurrentWindowMax: 2.3000000342726707
Episode: 140 Score: 0.30000000447034836 AvgScore (100+): 0.24990000380203128 CurrentWindowMax: 2.3000000342726707
Episode: 150 Score: 0.4000000059604645 AvgScore (100+): 0.2569000039063394 CurrentWindowMax: 2.3000000342726707
Episode: 160 Score: 0.09000000171363354 AvgScore (100+): 0.25860000394284727 CurrentWindowMax: 2.3000000342726707
Episode: 170 Score: 0.09000000171363354 AvgScore (100+): 0.2465000037662685 CurrentWindowMax: 2.3000000342726707
Episode: 180 Score: 0.30000000447034836 AvgScore (100+): 0.2293000035174191 CurrentWindowMax: 2.3000000342726707
Episode: 190 Score: 0.0 AvgScore (100+): 0.22550000345334412 CurrentWindowMax: 2.3000000342726707

Updated Lr parameter in runtime- once get avg >0.25

Episode: 10 Score: 0.09000000171363354 AvgScore (100+): 0.07400000132620335 CurrentWindowMax: 0.19000000320374966 Episode: 20 Score: 0.10000000149011612 AvgScore (100+): 0.07100000120699405 CurrentWindowMax: 0.19000000320374966 Episode: 30 Score: 0.10000000149011612 AvgScore (100+):0.08600000143051148 CurrentWindowMax: 0.30000000447034836 Episode: 40 Score: 0.09000000171363354 AvgScore (100+): 0.0782500013243407 CurrentWindowMax: 0.30000000447034836 Episode: 50 Score: 0.20000000298023224 AvgScore (100+): 0.08820000145584345 CurrentWindowMax: 0.4000000059604645 Episode: 60 Score: 0.10000000149011612 AvgScore (100+): 0.11166666845480601 CurrentWindowMax: 1.1000000163912773 Episode: 70Score: 0.10000000149011612 AvgScore (100+): 0.13542857356369495 CurrentWindowMax: 1.1000000163912773 Episode: 80 Score: 0.0 AvgScore(100+): 0.15462500241119415 CurrentWindowMax: 2.0000000298023224 Episode: 90 Score: 0.20000000298023224 AvgScore (100+): 0.17733333607514698 CurrentWindowMax: 2.0000000298023224 Episode: 100 Score: 0.0 AvgScore (100+): 0.23140000354498624 CurrentWindowMax: 2.0000000298023224 Episode :100 score : 0.0 Avg Score(100+): 0.23140000354498624 WinMax 2.0000000298023224 Episode: 110 Score: 0.30000000447034836 AvgScore (100+): 0.24590000374242663 CurrentWindowMax: 2.0000000298023224 Episode: 112 Avg Score 0.2529000038467348-so resetting learning rate :1e-10,and explore 0.032 Episode: 120 Score: 0.20000000298023224 AvgScore (100+): 0.266900004055351 CurrentWindowMax: 2.0000000298023224 Episode: 130 Score: 0.30000000447034836 AvgScore (100+): 0.2731000041402876 CurrentWindowMax: 2.0000000298023224 Episode: 140 Score: 0.10000000149011612 AvgScore (100+): 0.2736000041291118 CurrentWindowMax: 2.0000000298023224 Episode: 150Score: 0.30000000447034836 AvgScore (100+): 0.2707000040821731 CurrentWindowMax: 2.0000000298023224 Episode: 160 Score: 2.0000000298023224 AvgScore (100+): 0.3095000466778876 CurrentWindowMax: 2.0000000298023224 Episode: 170 Score: 0.09000000171363354 AvgScore (100+): 0.2955000445917246 CurrentWindowMax: 2.0000000298023224 Episode: 180 Score: 0.30000000447034836 AvgScore (100+): 0.2783000042103231 CurrentWindowMax: 2.0000000298023224 Episode: 190 Score: 0.09000000171363354 AvgScore (100+): 0.2511000038124621 CurrentWindowMax: 2.0000000298023224 Episode: 199 Avg Score getting drop 0.1882000287145378-so reset learning rate 1e-6 and explore to 0.7 Episode: 200 Score: 0.30000000447034836 AvgScore (100+): 0.19120000291615724 CurrentWindowMax: 2.0000000298023224 Episode :200 score : 0.30000000447034836 Avg Score(100+): 0.19120000291615724 WinMax 2.0000000298023224 Episode: 210 Score: 0.10000000149011612 AvgScore (100+): 0.17820000272244216 CurrentWindowMax: 2.0000000298023224 Episode: 220 Score: 0.10000000149011612 AvgScore (100+): 0.15510002338195062 CurrentWindowMax: 2.0000000298023224 Episode: 230 Score: 0.10000000149011612 AvgScore (100+): 0.14520000223070384 CurrentWindowMax: 2.0000000298023224 Episode: 240 Score: 0.0 AvgScore (100+): 0.14610000224784017 CurrentWindowMax: 2.0000000298023224 Episode:250 Score: 0.10000000149011612 AvgScore (100+): 0.14300000220537185 CurrentWindowMax: 2.0000000298023224 Episode: 260 Score: 0.10000000149011612

		AvgScore (100+): 0.09010000141337514 CurrentWindowMax: 0.800000011920929 Episode: 270 Score: 0.09000000171363354 AvgScore (100+): 0.0838000013306737 CurrentWindowMax: 0.4000000059604645 Episode: 280 Score: 0.09000000171363354 AvgScore (100+): 0.0788000012616789 CurrentWindowMax: 0.4000000059604645 Episode: 290 Score: 0.09000000171363354 AvgScore (100+): 0.07780000124126672 CurrentWindowMax: 0.4000000059604645  Seems explore factr may not need to change only Lr
8.	Source code Details	1. **Nn_model.py** - • Convolutional Neural Network model with 3 layer architecture • Having constructor to initialize seed and Input , Output and Hidden layers • Feedforward function to Neuron activation using Relu function to make output 0 or >0 [y = max(0, x)] and method to reset the weights 2. **DDPG : Agent.py** : Agent to have properties and functions covering • local and Target networks which are common for both 2 agents in Tennis Env • Act - Returns the actions (policy) each agent to STEP next into Env • Learn - Deep Q learning - Read from Replay Memory and update Q Target based on current vs expected rewards • Replay Memory for Experience Replay • Noise - exploration functionality **3. DDPGMultiAgentTennis.py** • Basically a Class for multiple agents • Having common Replay Memory for both Agents for collaboration • Having common Actor Network for both Agents to get trained and learn • Act function will trigger Agent.step functions • ReplayExp : Important function for DDPG Implementation 4. **Tennis.ipynb** - Python Notebook covers all the Code and detailed executed report 1. Libraries , Environment and Agent initialization , DDPG , Rewards summary and plot 2. Made some function to make ease the training process since have to try multiple iterations a. TrainMultipleAgents - i. Pass the MultiAgent Instance and Env and Score counters and set the number of iteration to be executed ii. ShowGraph and Zoomplot for little customized way of see graph to decide on Hyperparams tuning iii. Save model weights 5. Checkpoint_actor30.pth - saved model weights for Actor 6. Checkpoint_critic30.pth - saved model weights for Critic **#Learnings**: Its very unstable and tried dynamic tuning of Hyperparams and changed Memory with High rewards too . Mostly Gamma , Theta and exploration Factor make the impact .. Tried every step leanring, only end of episoudes learning- retrian only with high rewards- every 1000 episodes , tuned the params
9.	Ideas for future work	1. Solve a more difficult continuous control environment where the goal is to teach a creature with four legs to walk forward without falling. Ref https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Examples.md#crawler
10.	In Simple	**A BIG THANKS TO UDACITY TEAM and my Reviewers!!**