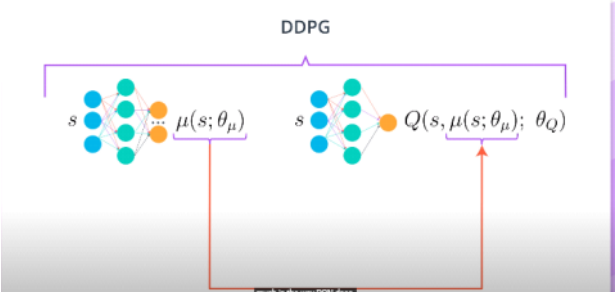

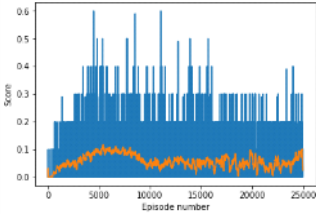
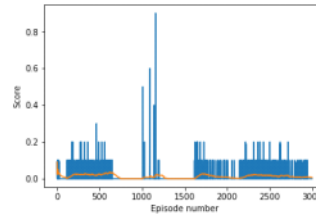
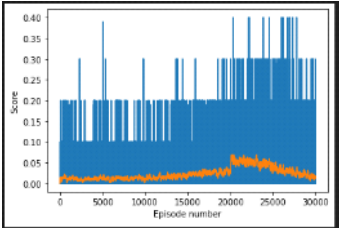
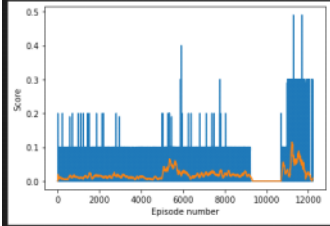


S.No	Topic	Details
1.	Project Goal	<p>This Tennis project is as part of Udacity Nanodegree - AI Deep Reinforcement Learning Expert and aims to develop a collaborative AI Agent - Tennis- to Self-Play In Continuous space using Policy-based 'Actor-critic' Methods using Deep Neural Networks.</p> <p>From <https://github.com/SENCIAIRResearcher/blob/master/README.md></p>
2.	Scope	<ul style="list-style-type: none">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 <p>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,</p> <ul style="list-style-type: none">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. <p>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.</p>
3.	Purpose	<ul style="list-style-type: none">One of the primary goal of AI is to solve complex tasks in high dimensional , sensory inputs . Though Deep Q Network (DQN) proved 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 can't find the action that maximizes the action-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 spaceDDPG Paper: https://arxiv.org/abs/1509.02971
4.	Solution Approach -Policy based Methods	<ul style="list-style-type: none">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 OnlineDevelop 'Actor-Critic' agent uses Function approximation to learn a policy (action) and value function<ul style="list-style-type: none">Have 2 Neural Networks<ul style="list-style-type: none">One for an Actor - Takes stats information as an input and actions distribution as an output<ul style="list-style-type: none">Take the action to move to next state and check the reward (Experience)and using TD estimate of the reward to predict the Critic's estimate for the next stateNext one for a Critic - Takes states as input and state value function of Policy as output.<ul style="list-style-type: none">Learn to evaluate the state value function V_{π} using TD estimateTo 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<ul style="list-style-type: none">Bootstrapping - generalization of a TD and Monte-Carlo estimatesTD is one step bootstrapping and MC is infinite bootstrappingMainly to reduce biasness and variances under controlled & fast convergenceLike 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<ul style="list-style-type: none">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 successfullyBuffer size can be large so allowing the algorithm to benefit from learning across a set of uncorrelated transitionsLittle 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 <p>#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</p> <p>Approximate the Maximizer - to calculate the new target value for training the action value function</p> $Q(s, \mu(s; \theta_{\mu}); \theta_Q)$  <p>Image source: Udacity DRLND</p> <p>Regular/local network - UpToDate network since training is done here but target network used to predict the stabilize strain</p> <p>#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</p> <p>Since multi-agent nature,</p> <ol style="list-style-type: none">Having 'Replay Memory' common for both agents so all experiences by each agents getting stored in common Memory for replayActor Network made it to common for both 2 agents since need an collaborative env<ol style="list-style-type: none">Idea to learn from both agents stats,rewards to make it more syncMain (Tennis.ipynb) --> MultiAgent --> Agent<ol style="list-style-type: none">Multiagent has common Replay memory and Actor NetworkCreate a child instance and set their actor network same but created own critic network
5.	Algorithm	<p>Deep deterministic Policy Gradient</p> <p>Published as a conference paper at ICLR 2016</p> <hr/> <p>Algorithm 1 DDPG algorithm</p> <p>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</p> <p>for episode = 1, M do Initialize a random process N_t for action exploration Receive initial observation state s_1 for $t = 1, T$ do Select action $a_t = \mu(s_t; \theta^{\mu}) + N_t$ 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: $\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s_i, a; \theta^Q) _{a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s; \theta^{\mu}) _{s_i}$ 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</p> <hr/> <p>#Crux of DDPG in 9 simple steps for both AI and Human Values</p> 

6.	Hyper parameters - Value configurations	<div> <div> Udacity Workspace # 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 </div> <div> Local Windows Env 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 </div> </div>
7.	Result & Rewards plot	<div> <div> Udacity Workspace  <pre>#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(isodes_score,avg_score)</pre>  </div> <div> Local Windows Setup  <pre>D: ML #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(isodes_score,avg_score)</pre>  </div> </div> <p>Yet to achieve 0.5 and work in progress and will re-submit again</p>
8.	Source code Details	<ol style="list-style-type: none"> Nn_model.py - <ul style="list-style-type: none"> 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 DDPG : Agent.py : Agent to have properties and functions covering <ul style="list-style-type: none"> 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 DDPGMultiAgentTennis.py <ul style="list-style-type: none"> 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 Tennis.ipynb - Python Notebook covers all the Code and detailed executed report <ol style="list-style-type: none"> Libraries , Environment and Agent initialization , DDPG , Rewards summary and plot Made some function to make ease the training process since have to try multiple iterations <ol style="list-style-type: none"> TrainMultipleAgents - <ol style="list-style-type: none"> Pass the MultiAgent Instance and Env and Score counters and set the number of iteration to be executed ShowGraph and Zoomplot for little customized way of see graph to decide on Hyperparams tuning Save model weights Checkpoint_actor30.pth - saved model weights for Actor Checkpoint_critic30.pth - saved model weights for Critic <p>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 .</p> <p>Tried every step learning, only end of episodes learning- retrian only with high rewards- every 1000 episodes , tuned the params</p>
9.	Ideas for future work	<ol style="list-style-type: none"> 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!!