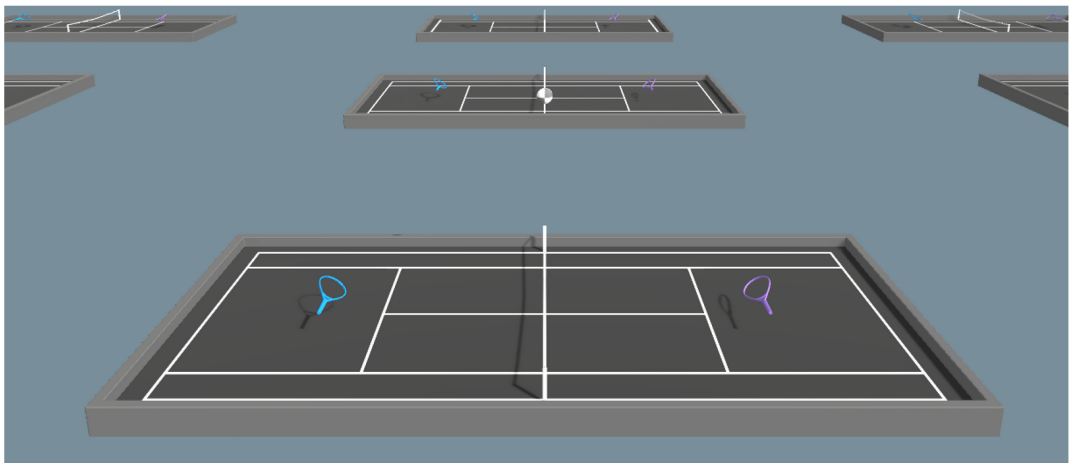
REPORT

PROJECT 3

AUTHOR NAME:VIGNESH.B.YAADAV

Tennis



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 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.

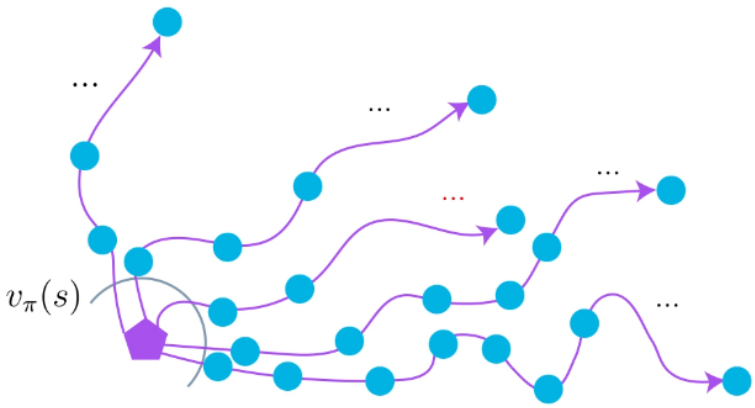
This yields a single score for each episode.

What is the diff b/w value-based method vs policy-based method?
If an agent is set to use Deep Neural Network to estimate value function then its value-based.
If an agent uses a Deep Neural Network to approximate the optimal Policy then it's a Policy-based method. DQN is a value-based method.
In the value-based method, we try to approximate the optimal value for a given state "s" Vπ(s) -> V*(s)
Qπ(s, a) -> Q*(s, a).
Policy-based methods contain 2 types
-> Stochastic policy
-> Deterministic policy

In the value-based method, the actions are calculated using expectation returns.

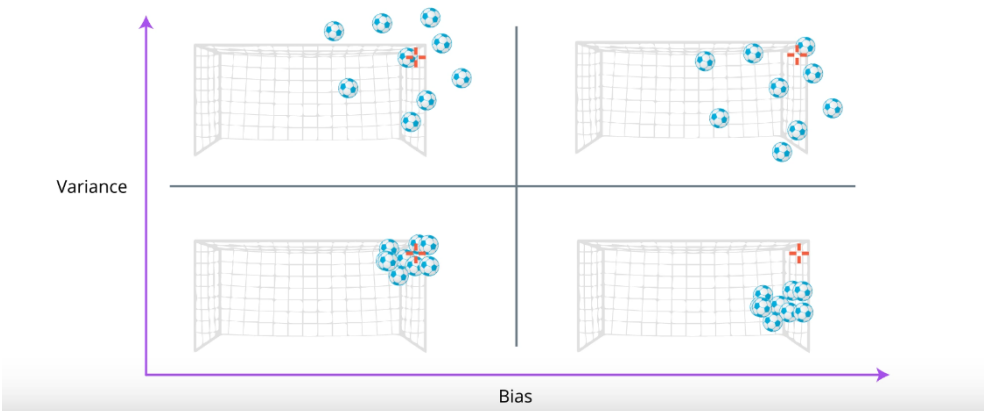
What's the best way to estimate value for Actor-Critic methods?

-> Montecarlo method:
In this method, all the rewards are added up with or without consideration of discounted return.
To calculate the value function we need to average the estimates. This method is unbiased but has a high variance.
-> TD Estimate:
In this method, we estimate the current state to the next state which is estimate over the estimate.
TD Estimate has a feature of bootstrapping were we leverage the estimate of the current state to calculate the next state. It has low variance but high bias.



$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$
$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$
$$a_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$$

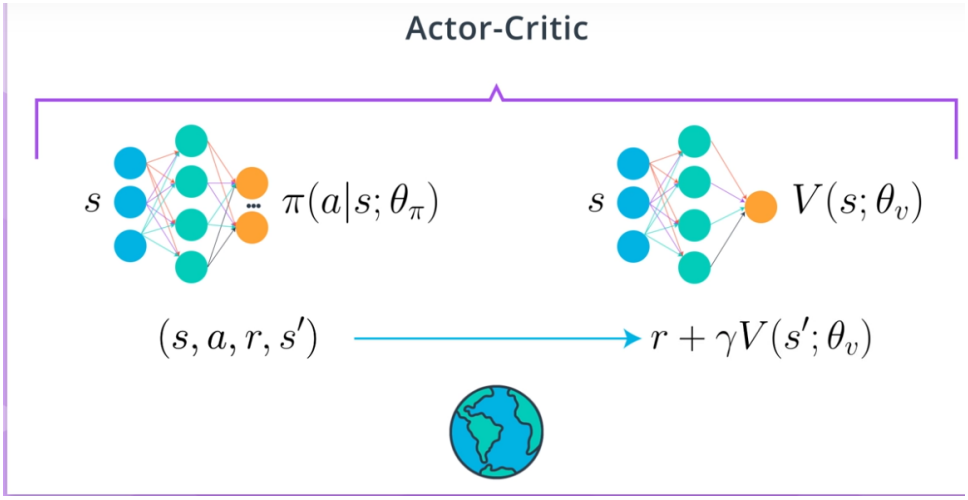
TD Estimate



How does a Basic Actor-Critic Agent work?

Actor-Critic Agent is an agent that uses function approximation to learn a policy and value Function to update the policy.
The actor-Critic method mainly consists of 2 neural networks 1. Actor-Network(Monte Carlo), 2. Critic network(TD Estimate)

A very basic Actor critic agent works as follows 1 takes in a state and outputs a probability distribution over actions, another network 2 takes in a state and outputs a state value function of policy π



Hears an intro on A3C (Asynchronous Advantage Actor-Critic):

As suggested by the name we are calculating Aπ and the critic will be learning to estimate Vπ.
A3C can use a single CNN with actor and critic sharing weights. Sharing whats is a more complex approach but using 2 networks speeds up the process.
As suggested by the name we are calculating Aπ and the critic will be learning to estimate Vπ.
A3C can use a single CNN with actor and critic sharing weights. Sharing whats is a more complex approach but using 2 networks speeds up the process.

What is On Policy VS Off Policy method?

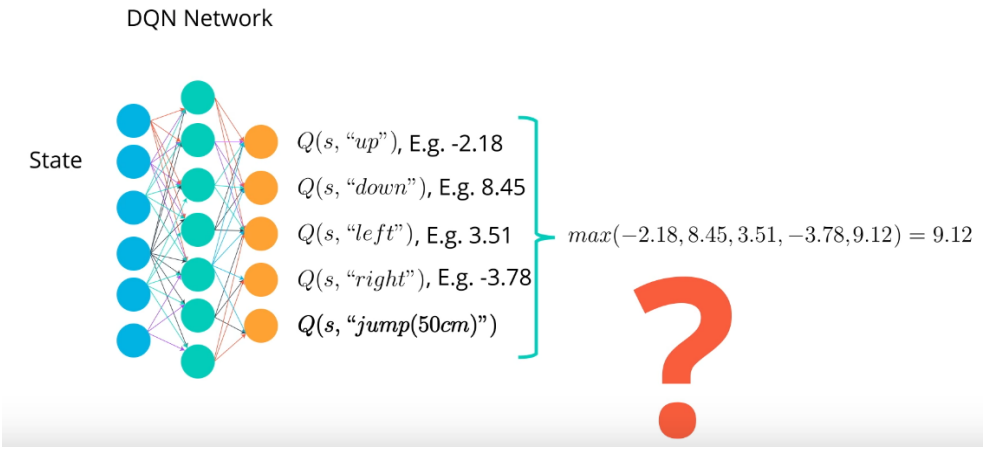
In On Policy, when policy used for interacting is the policy being learned. Eg: SARSA
In the Off Policy, when the policy is used for interacting with the environment is different from the policy being learned. Eg: Q-Learning

What is DDPG (Deep Deterministic Policy Gradient, Continuous control)?

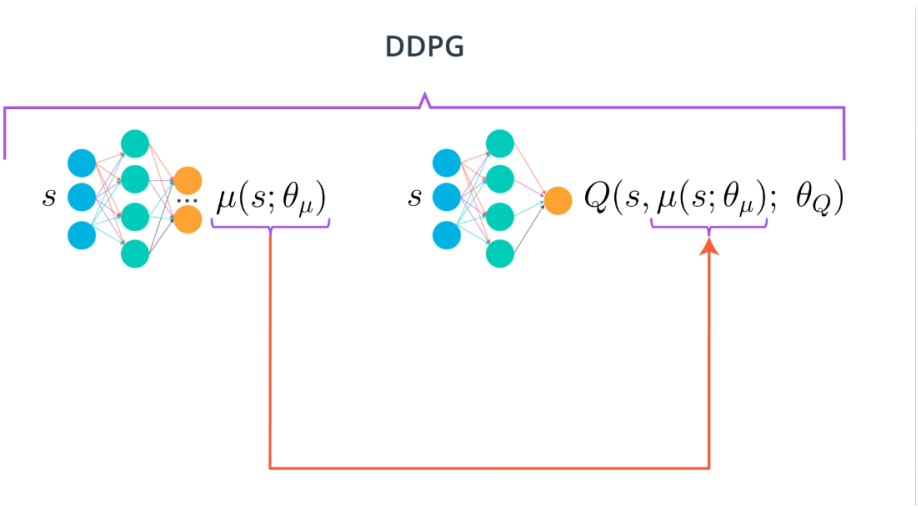
It's a different kind of Actor-Critic method, In DDPG the Critic is used to approximate the maximizer the vales of Q-vales of the next state.

Why not use DQN it approximates over continuous spaces?

DQN network takes in a state and gives an estimated return action-value function its quite good at giving out action-value function for n-actions space, but not capable of outputting a continuous range of action this is where DDPG can overcome it.



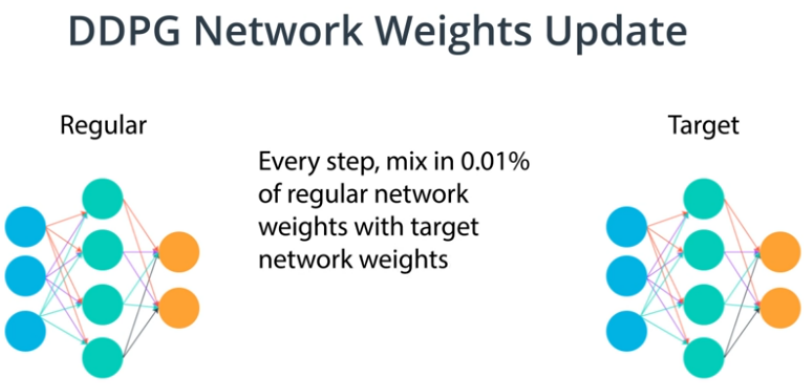
DQN network takes in a state and gives an estimated return action-value function its quite good at giving out action-value function for n-actions space, but not capable of outputting a continuous range of action this is where DDPG can overcome it.



DDPG has 2 Neural networks 1. Actor, 2.Critic. The actor approximates the optimal Policy deterministically. The critic gets to evaluate the optimal policy by using the actor's best-believed action, we use this actor again to calculate the new target value based on it.

Interesting aspects of DDPG are :
-> Replay Buffer
-> Softupdate

In DDPG we have two copies of network weights for each network local actor, target actor, local critic, target critic, but in DDPG targets are updated using soft update strategy its slowely blending of regual network weights with target



Model architecture

The Actor network consist of three fully connected layer with batchnormalization applied at the first layer. The network maps states to actions. It uses ReLU as activation function except the last layer where it use tanh.The critic network also consist of three fully connected layer with batchnormalization applied at the first layer. The network maps maps (state,action) pairs to Q-values. It uses ReLU as activation function in the first two layers and no activation function for the last layer.

Hyperparameters

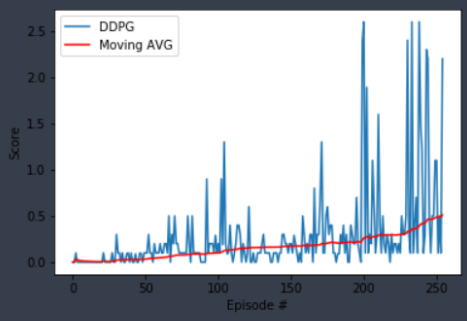
fc1_units=400 # Number of nodes in the first hidden layer
fc2_units=300 # Number of nodes in the second hidden layer
BUFFER_SIZE = int(1e6) # replay buffer size
BATCH_SIZE = 256 # minibatch size
GAMMA = 0.99 # discount factor
TAU = 2e-3 # for soft update of target parameters
LR_ACTOR = 1e-3 # learning rate of the actor
LR_CRITIC = 1e-3 # learning rate of the critic
WEIGHT_DECAY = 0 # L2 weight decay
LEARN_EVERY = 1 # learning timestep interval
LEARN_NUM = 10 # number of learning passes
GRAD_CLIPPING = 1.0 # gradient clipping
EPSILON = 1.0 # for epsilon in the noise process (act step)
EPSILON_DECAY = 1e-6 # epsilon decay rate

```
from workspace_utils import active_session

with active_session():
    scores, avgs = ddpg(actor_weights_name ='actor_tennis.pth', critic_weights_name='critic_tennis.pth')
```

```
Episode 237 (78s)       Max score: 0.700        Moving Avg: 0.368
Episode 238 (6s)        Max score: 0.100        Moving Avg: 0.369
Episode 239 (303s)      Max score: 2.600        Moving Avg: 0.394
Episode 240 (166s)      Max score: 1.500        Moving Avg: 0.408
Episode 241 (137s)      Max score: 1.200        Moving Avg: 0.419
Episode 242 (15s)       Max score: 0.100        Moving Avg: 0.420
Episode 243 (39s)       Max score: 0.300        Moving Avg: 0.422
Episode 244 (269s)      Max score: 2.300        Moving Avg: 0.444
Episode 245 (258s)      Max score: 2.200        Moving Avg: 0.463
Episode 246 (57s)       Max score: 0.500        Moving Avg: 0.465
Episode 247 (16s)       Max score: 0.100        Moving Avg: 0.464
Episode 248 (66s)       Max score: 0.500        Moving Avg: 0.467
Episode 249 (56s)       Max score: 0.500        Moving Avg: 0.471
Episode 250 (125s)      Max score: 1.100        Moving Avg: 0.480
Episode 251 (131s)      Max score: 1.100        Moving Avg: 0.489
Episode 252 (16s)       Max score: 0.100        Moving Avg: 0.489
Episode 253 (68s)       Max score: 0.500        Moving Avg: 0.491
Episode 254 (9s)        Max score: 0.100        Moving Avg: 0.490
Episode 255 (257s)      Max score: 2.200        Moving Avg: 0.511

Environment solved in 255 episodes.     Average score: 0.511    Total training time: 8696.894483804703s
```

The DDPG and the moving average has been plotted out

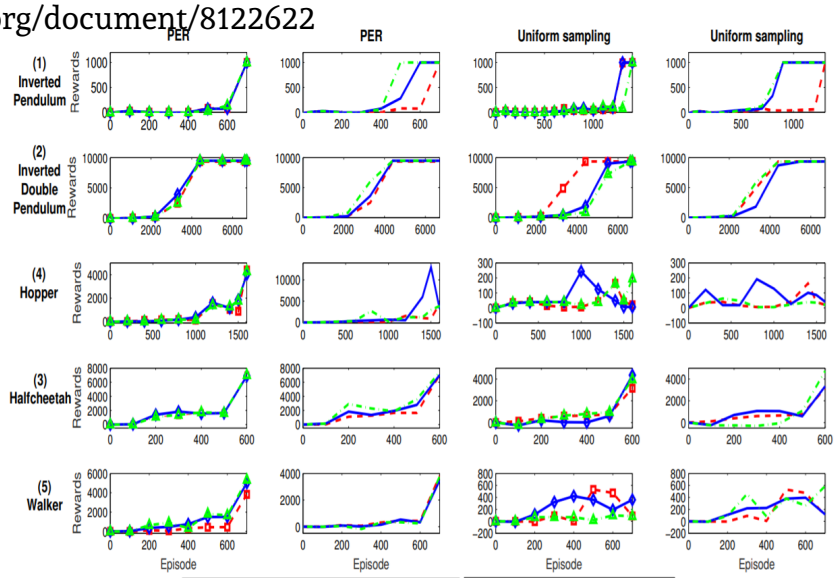
```
plt.plot(np.arange(len(scores)), scores, label='DDPG')
plt.plot(np.arange(len(scores)), avgs, c='r', label='Moving AVG')
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.legend(loc='upper left');
plt.show()
```



Future ideas to improve the agent's performance

-> Fine-tuning of hperparameters lead to better results and faster training time.
The process of setting the hyper-parameters requires expertise and extensive trial and error. There are no simple and easy ways to set hyper-parameters — specifically, learning rate, batch size, momentum, and weight decay.
" The hyper-parameter tuning process is a tightrope walk to achieve a balance between underfitting and overfitting." Source:https://towardsdatascience.com/hyper-parameter-tuning-techniques-in-deep-learning-4dad592c63c8

->DDPG can be improved by using prioritized experience replay
Using prioritized replay helps in taking better samples over Replay buffer.
More information of the code can be found in prioritised_replay_memory.py by using his as model you can increase he effiency of the model.
If intresed in learning more on this please refur to this paper :
https://ieeexplore.ieee.org/document/8122622



-> Using Multi Agent Actor Critic for Mixed Cooperative Competitive environments

Source : https://papers.nips.cc/paper/7217-multi-agent-actor-critic-for-mixed-cooperative-competitive-environments.pdf

Multi-agent policy gradient algorithm is where agents learn a centralized critic-based on the observations and actions of all agents. Empirically, this method outperforms traditional
RL algorithms on a variety of cooperative and competitive multi-agent environments. We can further improve the performance of our method by training agents with an ensemble of policies, an approach we believe to be generally applicable to any multi-agent algorithm.

One downside to this approach is that the input space of Q grows linearly (depending on what information is contained in x) with the number of agents N. This could be remedied in practice by, for example, having a modular Q function that only considers agents in a certain neighborhood of a given agent.