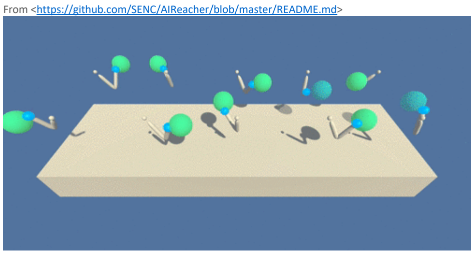
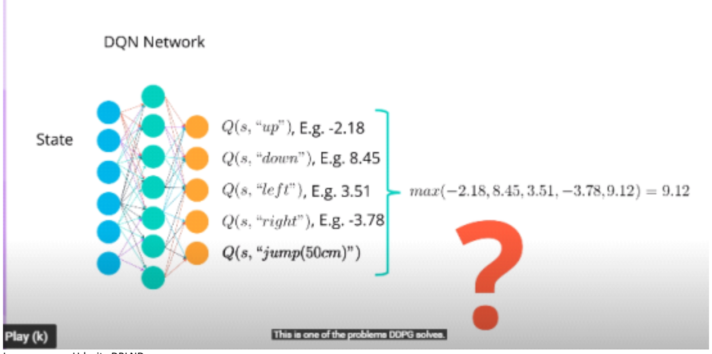
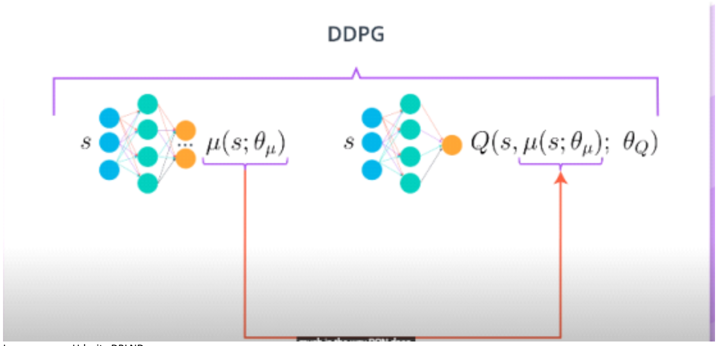
Report - Continuous Control -DDPG

Sunday, 19 July 2020   11:39 PM

S.No	Topic	Details
1.	Project Goal	This Reacher project is as part of Udacity Nanodegree - AI Deep Reinforcement Learning Expert and aims to develop an AI Agent - "a double-jointed arm" - move to target location in Continuous space using Policy-based 'Actor-critic' Methods using Deep Neural Networks. From <https://github.com/SENC/AIReacher/blob/master/README.md> 
2.	Scope	• Develop an AI Agent using 'actor-critic' methods - which should learn the best policy to maximize its rewards by taking best actions in the given continuous environment • Goal The environment is considered solved, when the average (over 100 episodes) of those average scores is at least +30. • Decided to solve the First Version Option 1: The task is episodic and the Agent must get an average score of +30 over 100 consecutive episodes
3.	Purpose	• 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 part to 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 space • DDPG Paper: https://arxiv.org/abs/1509.02971

4.	Solution Approach -Policy based Methods	• 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 Critic'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
		• 
		Image source: Udacity DRLND
		
		#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**

5.	Algorithm	Deep deterministic Policy Gradient Published as a conference paper at ICLR 2016

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:

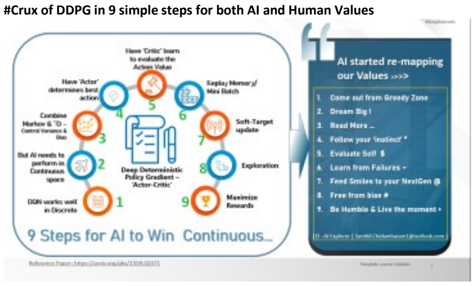
$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    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



6. Hyper parameters - Value configurations

Udacity Workspace	Local Windows Env

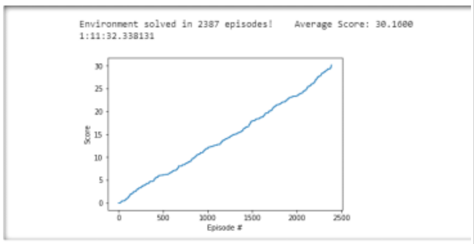
BUFFER_SIZE = 500000 # replay buffer size
BATCH_SIZE = 300 # minibatch size
GAMMA = 0.997 # discount factor
TAU = 0.0013 # for soft update of target parameters
LR_ACTOR = 0.0002 # learning rate of the actor
LR_CRITIC = 1e-3 # learning rate of the critic
WEIGHT_DECAY = 0 # L2 weight decay
theta=0.17 # Noise Sampling
sigma=0.24
Random seed = 9 #reacherAI = AiAgent(state_size,action_size,random_seed=9)
	# Iteration Ver 15 - Test run with 100 Episodes
	#BUFFER_SIZE = 500000 #int(1e4) # replay buffer size
	#BATCH_SIZE = 500 #128 # minibatch size
	#GAMMA = 0.997 # discount factor
	#TAU = 0.0013 # for soft update of target parameters
	#LR_ACTOR = 0.0002 # learning rate of the actor
	#LR_CRITIC = 0.0001 # learning rate of the critic
	#WEIGHT_DECAY = 0 # L2 weight decay
	
	#Noise mu=0., theta=0.17, sigma=0.24
	#Random seed = 17

7. Result & Rewards plot

Udacity Workspace

Environment solved in 2387 episodes!

Average Score: 30.1600



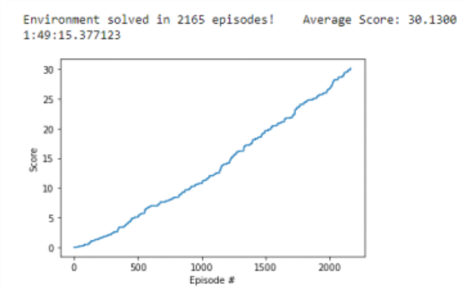
#Episode 100 Average Score: 0.0097 AcScore 0.9700
#Episode 200 Average Score: 0.0180 AcScore 2.7700
#Episode 300 Average Score: 0.0123 AcScore 4.0000
#Episode 400 Average Score: 0.0088 AcScore 4.8800
#Episode 500 Average Score: 0.0128 AcScore 6.1600
#Episode 600 Average Score: 0.0066 AcScore 6.8200
#Episode 700 Average Score: 0.0125 AcScore 8.0700
#Episode 800 Average Score: 0.0100 AcScore 9.0700
#Episode 900 Average Score: 0.0154 AcScore 10.6100
#Episode 1000 Average Score: 0.0130 AcScore 11.9100

#Episode 1100 Average Score: 0.0072 AcScore 12.6300
#Episode 1200 Average Score: 0.0133 AcScore 13.9600
#Episode 1300 Average Score: 0.0101 AcScore 14.9700
#Episode 1400 Average Score: 0.0112 AcScore 16.0900
#Episode 1500 Average Score: 0.0176 AcScore 17.8500
#Episode 1600 Average Score: 0.090 AcScore 18.7500
#Episode 1700 Average Score: 0.0137 AcScore 20.1200
#Episode 1800 Average Score: 0.0145 AcScore 20.5700
#Episode 1900 Average Score: 0.0113 AcScore 22.7000
#Episode 2000 Average Score: 0.0072 AcScore 23.4200
#Episode 2100 Average Score: 0.0120 AcScore 24.6200
#Episode 2200 Average Score: 0.0190 AcScore 26.5200
#Episode 2300 Average Score: 0.0206 AcScore 28.5800

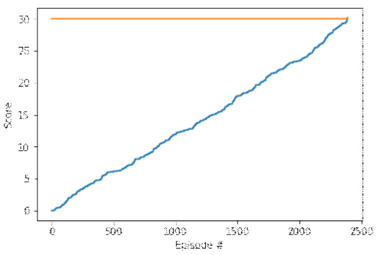
Local Windows Env

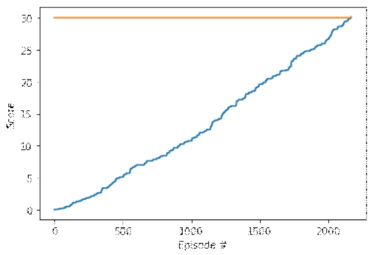
Environment solved in 2165 episodes!

Average Score: 30.1300

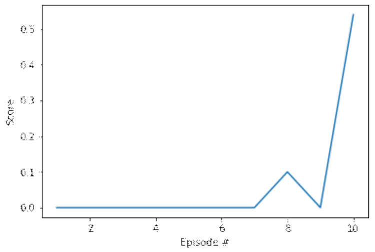


#Episode 100 Average Score: 0.0048 AcScore 0.4800
#Episode 200 Average Score: 0.0099 AcScore 1.4700
#Episode 300 Average Score: 0.0087 AcScore 2.3400
#Episode 400 Average Score: 0.0122 AcScore 3.5600
#Episode 500 Average Score: 0.0152 AcScore 5.0800
#Episode 600 Average Score: 0.0190 AcScore 6.9800
#Episode 700 Average Score: 0.0064 AcScore 7.6200
#Episode 800 Average Score: 0.0081 AcScore 8.4300
#Episode 900 Average Score: 0.0123 AcScore 9.6600
#Episode 1000 Average Score: 0.0108 AcScore 10.7400
#Episode 1300 Average Score: 0.0208 AcScore 16.2000
#Episode 1500 Average Score: 0.0153 AcScore 19.6000
#Episode 1700 Average Score: 0.0098 AcScore 21.8500
#Episode 1900 Average Score: 0.0087 AcScore 25.1500
#Episode 2000 Average Score: 0.0153 AcScore 26.6800
#Episode 2100 Average Score: 0.0205 AcScore 28.7300

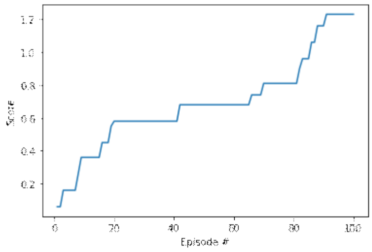




Test Run with 10 Episodes



Test Run with 100 Episodes



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

```
#Test to create the instance of AiAgent
reacherAI = AiAgent(state_size,action_size,random_seed=9)
print(reacherAI.actor_local)
print(reacherAI.critic_local)
```

```
Actor(
  (fc1): Linear(in_features=33, out_features=24, bias=True)
  (fc2): Linear(in_features=24, out_features=48, bias=True)
  (fc3): Linear(in_features=48, out_features=4, bias=True)
)
Critic(
  (fcs1): Linear(in_features=33, out_features=24, bias=True)
  (fc2): Linear(in_features=28, out_features=48, bias=True)
  (fc3): Linear(in_features=48, out_features=1, bias=True)
)
```

2. **DDPG_Agent.py** : Agent to have properties and functions covering
   - local and Target networks ,
   - soft update ,
   - Noise for exploration
   - Replay Memory for Experience Replay
   - step, act, reset , learn functions

S.No	Activity	Core
1.	Initialize local and target network for both Actor and Critic	Deep CNN - 3 Layers
2.	Initialize replay memory based on Buffer size, Mini Batch and seed	Recall Experience
3.	Call "Learn " Function when actor.step : Core Algo - Update Policy and Value params based on experiences	**Get Next_Action and Qvalue to Q_targets = r + γ * critic_target(next_state, actor_target(next_state))** where: actor_target(state) -> action critic_target(state, action) -> Q-value From the Memory , take random set of experiences and predict target based on current states and reward
4.	• Get the Next_action from action_target and caluclate Q_target Value for given (s,a) • Compute Q target from current reward • Compute the Critic Loss by Qexpected - Qtarget • Optimizer	Basically , update the weights of Actor and Critic Network targeting minimize the loss with Current vs Expected Result
5.	Smooth copy of local to target network	Stable Learning

3. **Continuous Control.ipynb** - Python Notebook covers all the Code and detailed executed report
   1. Libraries , Environment and Agent initialization , DDPG , Rewards summary and plot
4. Checkpoint_actor30.pth - saved model weights for Actor
5. Checkpoint_critic30.pth - saved model weights for Critic

#Learnings: Sometimes it make us focus in depth understanding and tuning of algos but may miss simple front end part especially People like me :) when learning new curious to try quick lunch.
But this project work taught me ,
- sometimes we needs to more patience (TRAINING PHASE -almost 6-12 hrs in initial times-),
- where need quick try and
- where need to focus on core and
- how to be productive(when training starts, I started preparing the Report and exploring Research papers though it tempted tosee agent's performance)

1. The silly mistake I did was reset the scores 'counter array' in wrong place so only recent scores scored -

2. Before running long hr training process (like here avg 8 to 10 hrs it took for me to achieve 30+ score') please dry run your logic with simple 1 to 10 episodes
3. After all my 2 extended nights, but got an error on last Printing the time taken so made me restart from point 0 :|
4. Cud see the OS also matters ...same code , i tried running in Udacity provided workspace (linux) and Windows in my local machines. Same logic but coud see difference in scores average in each episodes. So i tested initial 200 runs to see which seed gets me relatively high score chosen to tune the parameters
5. Buffer size can be large but keep the Mini batch smaller- random sampling for correlations-[it worked well 256-300 for me]
6. Agent storing history to Replay Memory - its critical .. I was oversighted initially by just calling the" aiReacher.Step" function but got right scores by having the condition once the Episode complete with 'done'. Instead of storing all time steps states , just store memories when the episodes gets 'done'

7. Couldn't find much difference using of GPU vs CPU . So completely ran in 'CPU' and saved 'GPU' hrs for next project
8. Biggest challenge is Udacity Workspace have to be active - 3 times got disconnected after 6 hrs and nearing closure all work vanished. Highly recommend try this in Local Machine

		and once you know what are all the parameters to be tuned after seeing the intial 100 to 300 Episodes then use the parameters in workspace to get it done 9.
9.	Ideas for future work	1. Work on Option 2 and try parallel learning PPO 2. 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	**THANKS TO UDACITY TEAM!!**