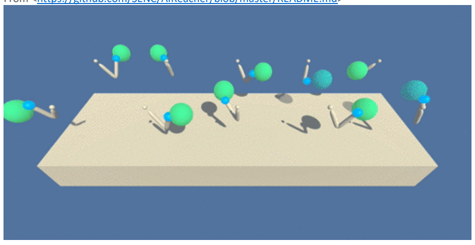
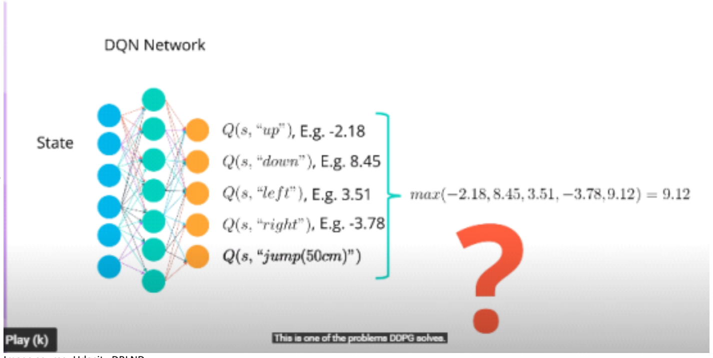
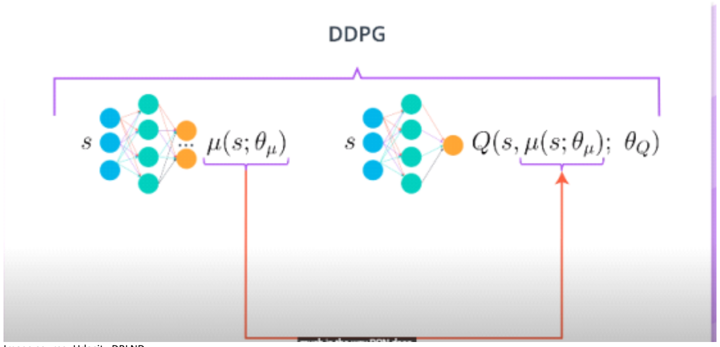
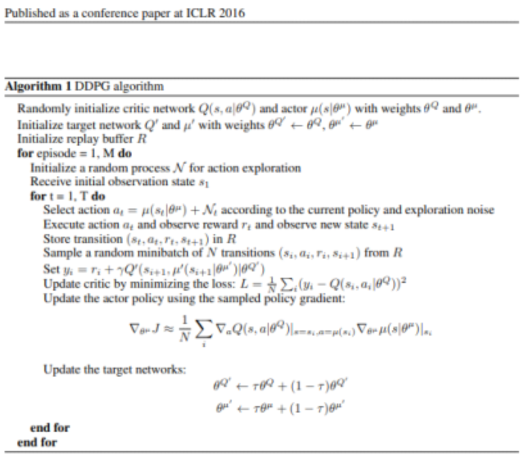
Project 2 - Continuous Control -DDPG

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 •  DQN Network State $Q(s, \text{"up"})$, E.g. -2.18 $Q(s, \text{"down"})$, E.g. 8.45 $Q(s, \text{"left"})$, E.g. 3.51 $Q(s, \text{"right"})$, E.g. -3.78 $Q(s, \text{"jump(50cm)"})$ $max(-2.18, 8.45, 3.51, -3.78, 9.12) = 9.12$? Play (k) This is one of the problems DDPG solves. 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)  DDPG s $\mu(s;\theta_\mu)$ s $Q(s, \mu(s;\theta_\mu);\ \theta_Q)$ 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

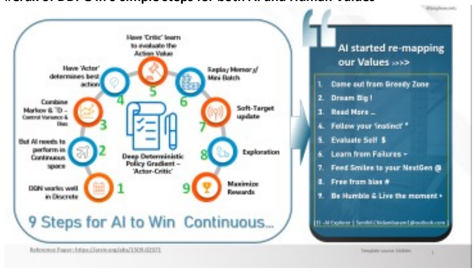


Published as a conference paper at ICLR 2016

Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ 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


Have 'Critic' learn to evaluate the Action Value
Have 'Actor' determines best action
Replay Memory/ Mini Batch
Combine Markov & TD – Control Variance & Bias
Soft-Target update
But AI needs to perform in Continuous space
Deep Deterministic Policy Gradient – 'Actor-Critic'
Exploration
DQN works well in Discrete
Maximize Rewards
1
2
3
4
5
6
7
8
9
9 Steps for AI to Win Continuous...
AI started re-mapping our Values >>>>
1. Come out from Greedy Zone
2. Dream Big !
3. Read More ..
4. Follow your 'instinct'
5. Evaluate Self $
6. Learn from Failures
7. Feed Smiles to your NextGen
8. Free from bias
9. Be Humble & Live the moment

6. Hyper parameters

```
BUFFER_SIZE = 500000 # replay buffer size
BATCH_SIZE = 256        # minibatch size
GAMMA = 0.997           # discount factor
TAU = 0.0013            # for soft update of target parameters
LR_ACTOR = 0.00013    # 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 = 3         #reacherAI = AiAgent(state_size,action_size,random_seed=3)
```

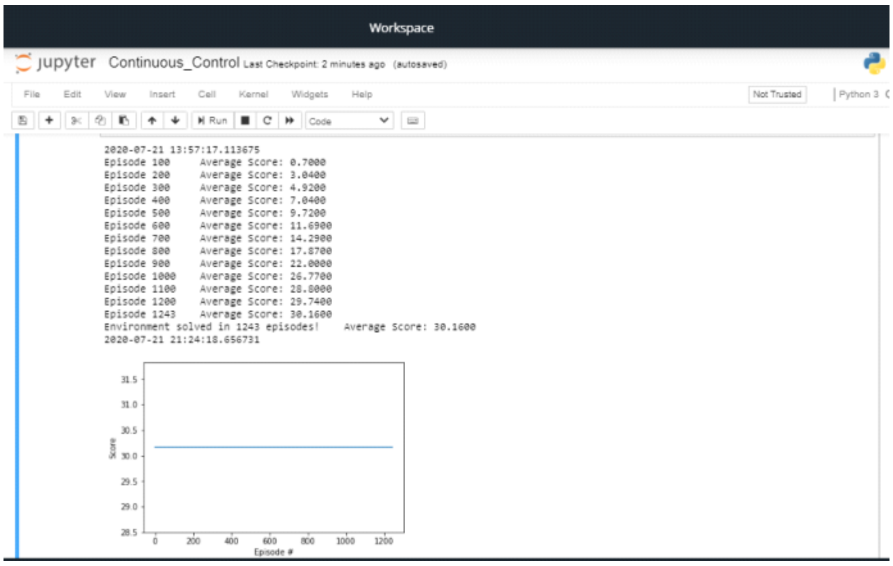
7. Rewards

After multiple iteration and parameter tuning , finally Environment getting solved and please find the Score and reward chart

```
2020-07-21 13:57:17.113675
Episode 100      Average Score: 0.7000
Episode 200      Average Score: 3.0400
Episode 300      Average Score: 4.9200
Episode 400      Average Score: 7.0400
Episode 500      Average Score: 9.7200
Episode 600      Average Score: 11.6900
Episode 700      Average Score: 14.2900
Episode 800      Average Score: 17.8700
Episode 900      Average Score: 22.0000
Episode 1000     Average Score: 26.7700
Episode 1100     Average Score: 28.8000
Episode 1200     Average Score: 29.7400
Episode 1243     Average Score: 30.1600
```

Environment solved in 1243 episodes!   Average Score: 30.1600

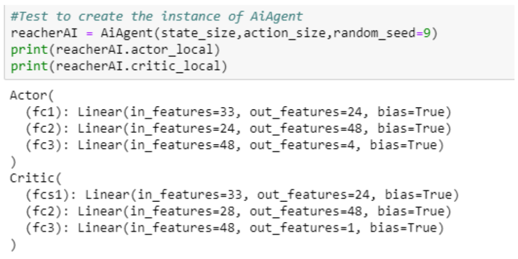
2020-07-21 21:24:18.656731



```
2020-07-21 13:57:17.113675
Episode 100     Average Score: 0.7000
Episode 200     Average Score: 3.0400
Episode 300     Average Score: 4.9200
Episode 400     Average Score: 7.0400
Episode 500     Average Score: 9.7200
Episode 600     Average Score: 11.6900
Episode 700     Average Score: 14.2900
Episode 800     Average Score: 17.8700
Episode 900     Average Score: 22.0000
Episode 1000    Average Score: 26.7700
Episode 1100    Average Score: 28.8000
Episode 1200    Average Score: 29.7400
Episode 1243    Average Score: 30.1600
Environment solved in 1243 episodes!    Average Score: 30.1600
2020-07-21 21:24:18.656731
```

8. Source code

1. Nn_model.py -Convolutional Neural Network model with 3 layer architecture



```
#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. Agent : Agent with properties for
   - local and Target networks ,
   - soft update ,
   - Noise for exploration
   - Replay Memory for Experience Replay
   - step, act, reset , learn functions

3. Continuous Control.ipynb - Python Notebook covers all the Code and Results
4. Checkpoint_actor30.pth - saved model weights for Actor
5. Checkpoint_critic30.pth - saved model weights for Critic

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!!**