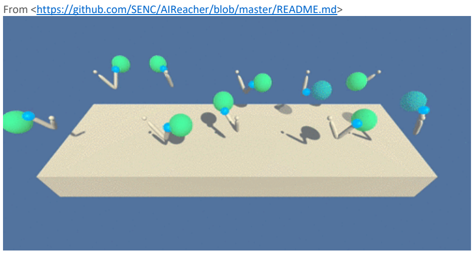
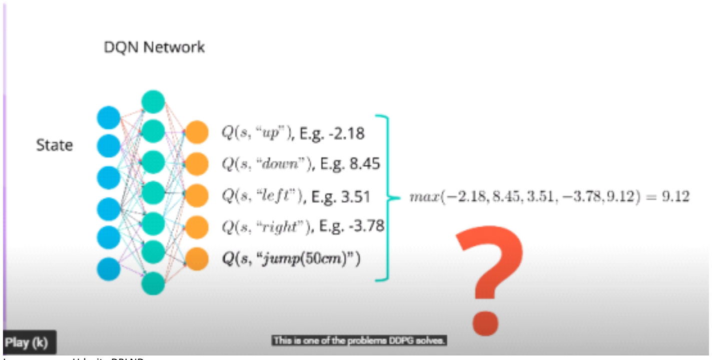
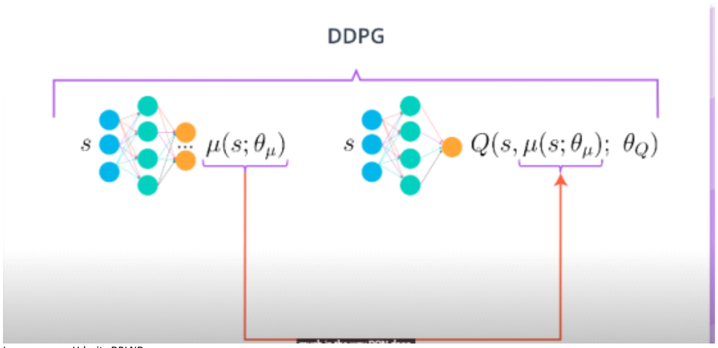
# Project 2 - Continuous Control -DDPG

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.<br><br>From <https://github.com/SENC/AIReacher/blob/master/README.md><br> |
| 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<br>• Goal The environment is considered solved, when the average (over 100 episodes) of those average scores is at least +30.<br>• Decided to solve the First Version<br>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.<br>• 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<br>• 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<br>• Develop 'Actor-Critic' agent uses Function approximation to learn a policy (action) and value function<br>  • Have 2 Neural Networks<br>    ○ One for an Actor - Takes stats information as an input and actions distribution as an output<br>      ▪ 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<br>    ○ Next one for a Critic - Takes states as input and state value function of Policy as output.<br>      ▪ Learn to evaluate the state value function Vπ using TD estimate<br>    To calculate **the advantage function** and train the actor using this value.<br>    So ideally train the actor using the calculated advantages as a baseline.<br>  • Instead of having baseline using TD estimate , can use Bootstrapping to reduce the variance<br>    ○ Bootstrapping - generalization of a TD and Monte-Carlo estimates<br>      ▪ TD is one step bootstrapping and MC is infinite bootstrapping<br>      ▪ Mainly to reduce biasness and variances under controlled & fast convergence<br><br>• 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<br>  • 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<br>  • Buffer size can be large so allowing the algorithm to benefit from learning across a set of uncorrelated transitions<br>• 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<br>• <br>Image source: Udacity DRLND<br><br>**#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<br>  Approximate the Maximizer - to calculate the new target value for training the action value function<br>  Q(s, μ (s ; θμ);θ Q)<br><br>Image source: Udacity DRLND<br><br>Regular/local network - UpToDate network since training is done here but target network used to predict the stabilize strain<br><br>#2 Soft Target updates:<br>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<br><br>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:

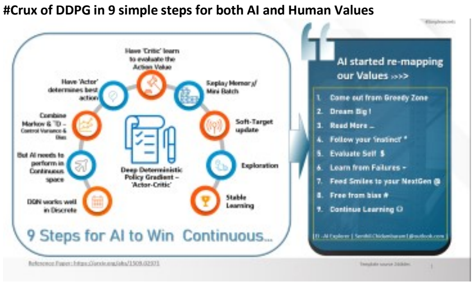
$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    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**




6. Hyper parameters

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)

7. Rewards

Workspace

jupyter Continuous_Control Last Checkpoint: 25 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
Episode 62      Average Score: 0.2200

In [13]: #Ver 7.3 19Jul
         #Change params to default
         #BUFFER_SIZE = 500000     #int(1e4)  # replay buffer size
         #BATCH_SIZE = 300         #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 = 1e-3                     # learning rate of the critic
         #WEIGHT_DECAY = 0

         scores = Ddpg()

         fig = plt.figure()
         ax = fig.add_subplot(111)
         plt.plot(np.arange(1, len(scores)+1), scores)
         plt.ylabel('Score')
         plt.xlabel('Episode #')
         plt.show()

Episode 100     Average Window Score: 1.7100
Episode 200     Average Window Score: 7.0200
Episode 300     Average Window Score: 11.8900
Episode 400     Average Window Score: 13.7100
Episode 500     Average Window Score: 16.9300
Episode 600     Average Window Score: 20.4300
Episode 700     Average Window Score: 22.0900
Episode 800     Average Window Score: 27.2800
Episode 868     Average Score: 30.1100
Environment solved in 768 episodes!     Average Score: 30.1100
```

**Kindly Note:**

1. There are 2 bugs in the DDPG function which I got to know after executed at 868 th Episode,s so not able to plot the reward graph
   Trying again to re-run as well ; will update but considering the timelines, and keeping the workspace active need more time and effort so submitting the report with weighted model file and Result achieved screenshot as well as the original ipython Notebook syncd in github
   https://github.com/SENC/AINavigator/blob/master/TrainingCode/AI_DeepQ_Navigator_Solution.ipynb

2. Environment solved - Print statement erroneously made episodes - 100 hence actual Episode solved at 868 but printed as 768

Error 1: Source to be fixed : Line 55 at DDPG Function
print ('AiReacher took {} hrs & {} mins to solve the env in {} episodes '.format(int(time_taken/3600),int(time_taken/60,i_episode)))

Error 2: Typo error -100
print('\nEnvironment solved in {:d} episodes!\tAverage Score: {:.4f}'.format(i_episode-100, np.mean(scores)))

```
Episode 100     Average Window Score: 1.7100
Episode 200     Average Window Score: 7.0200
Episode 300     Average Window Score: 11.8900
Episode 400     Average Window Score: 13.7100
Episode 500     Average Window Score: 16.9300
Episode 600     Average Window Score: 20.4300
Episode 700     Average Window Score: 22.0900
Episode 800     Average Window Score: 27.2800
Episode 868     Average Score: 30.1100
Environment solved in 768 episodes!     Average Score: 30.1100

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-d04fcc10640a> in <module>()
      9 #WEIGHT_DECAY = 0
     10 
---> 11 scores = Ddpg()
     12 
     13 fig = plt.figure()

<ipython-input-12-598689bedeb0> in Ddpg(n_episodes, max_t, print_every)
     53             torch.save(reacherAI.actor_local.state_dict(), 'checkpoint_actor30.pth')
     54             torch.save(reacherAI.critic_local.state_dict(), 'checkpoint_critic30.pth')
---> 55             print ('AiReacher took {} hrs & {} mins to solve the env in {} episodes '.format(int(time_taken/3600),int(t
ime_taken/60,i_episode)))
     56             break
     57 

ValueError: int() base must be >= 2 and <= 36
```

Error 2: Loading the weighted file - yet to fix it

**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
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!!**