# Table of Contents



# Learning Algorithm

## Deep Deterministic Policy Gradient (DDPG)

**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**

## Code Walk-through

File: Continuous_Control.ipynb

1. Instantiate ddpg_agent
2. Initialize Variables
3. Outer Loop (1000 Episodes)
    a. Reset the Reacher Environment
    b. Retrieve the Initial State from the Environment
    c. Reset the Learning Agent
    d. Inner Loop (1 Full Episode)
        i. Get Action for Given State from Agent
        ii. Send the Action to the Environment
        iii. Get the New State
        iv. Get the Reward
        v. Check if Episode is Done
        vi. Transition to the next state
        vii. Accumulate rewards for the Episode
        viii. Exit Loop when Done
    e. Store each Episode’s Score for the last 100 Episodes
    f. Store each Episode’s Score for Plotting

## Model Architecture for Neural Network

**Actor Model:**

Three layer Neural Network

1$^{st}$ Layer takes as input the number of STATES, 128 units as output
2$^{nd}$ Layer – 128 units as input, 128 units as output
3$^{rd}$ Layer – 128 units as input, output is the number of ACTIONS

Rectified Linear Units connect the Layers together with a Batch Normalization for the first layer.

**Critic Model:**

Three layer Neural Network

1$^{st}$ Layer takes as input the number of STATES, 128 units as output
2$^{nd}$ Layer – 128 units + number of ACTIONS as input, 128 units as output
3$^{rd}$ Layer – 128 units as input, output is 1

Rectified Linear Units connect the Layers together with a Batch Normalization for the first layer.

## Hyper-parameters

| Hyper Parameter | Value | Description |
|---|---|---|
| BUFFER_SIZE | $1 \times 10^{6}$ | replay buffer size |
| BATCH_SIZE | 1024 | Mini batch size |
| GAMMA | 0.8 | discount factor |
| TAU | $1 \times 10^{-3}$ | for soft update of target parameters |
| LR_ACTOR | $2 \times 10^{-4}$ | learning rate of the actor |
| LR_CRITIC | $2 \times 10^{-4}$ | learning rate of the critic |
| WEIGHT_DECAY | 0 | L2 weight decay |
| THETA | 0.15 | |
| SIGMA | 0.1 | |

## Experimentation

Batch Size. Experimented with different batch sizes [64, 128, 256, 512, 1024]. Settled on 1024, as the algorithm learned much quicker (less episodes). Approximately 1/3 the number of Episodes at 1024.

Gamma. Experimented with different GAMMA settings. Slowly decreasing from .99, to .9, .8, then .1. Decreasing to .8 allowed the Agent to learn in less episodes, decreasing to .1 didn't have a positive impact.

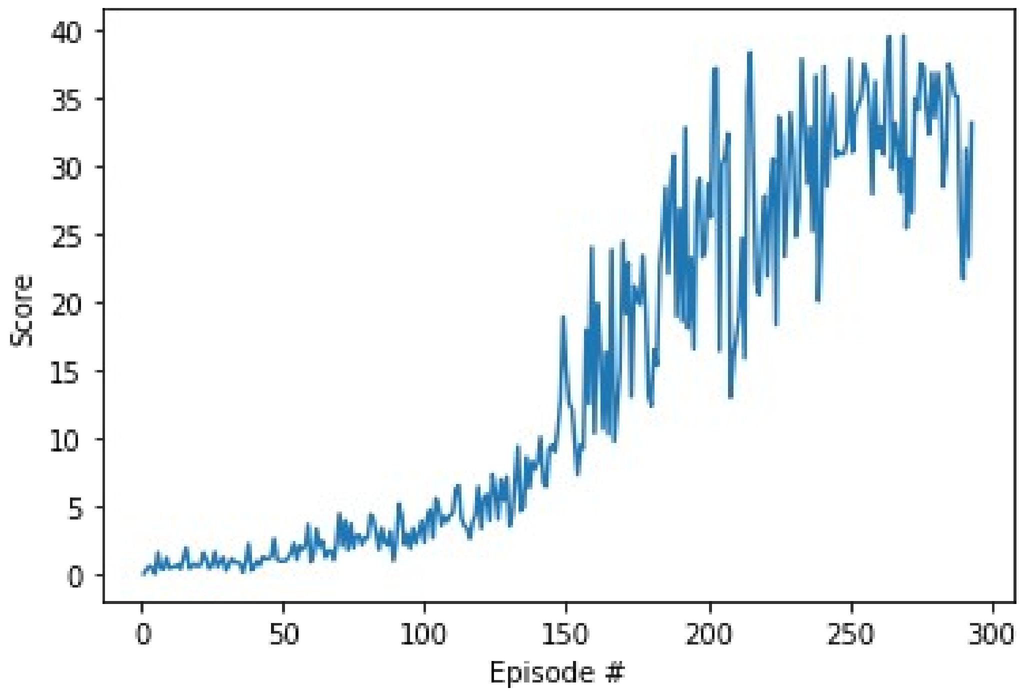
L2 Weight Decay. Started with an L2 weigh decay of $1 \times 10^{-2}$ and then settled on 0 for performance reasons.

TAU. Tested .01 and .001. When testing at .01, the agent appeared to have poor stability in learning.

Network Size. Started the network at 400, 300 first and second layers but decreased to 128, 128. Utilizing a smaller network while still maintaining good learning was the goal as a smaller network will reduce the chances of over-fitting to the training data.

## Results

(Goal: Average Score of 30 over 100 episodes)




Episode 100........................................................................................................Average Score: 1.65
Episode 200......................................................................................................Average Score: 12.66
Episode 293......................................................................................................Average Score: 30.05
Environment solved in 293 episodes!...................................................................Average Score: 30.05

## Ideas for Future Work

Attempt the multi-agent version of the Reacher problem.

## References


@misc{lillicrap2015continuous,

title={Continuous control with deep reinforcement learning},

author={Timothy P. Lillicrap and Jonathan J. Hunt and Alexander Pritzel and Nicolas Heess and Tom Erez and Yuval Tassa and David Silver and Daan Wierstra},

year={2015},

eprint={1509.02971},

archivePrefix={arXiv},

primaryClass={cs.LG}

}