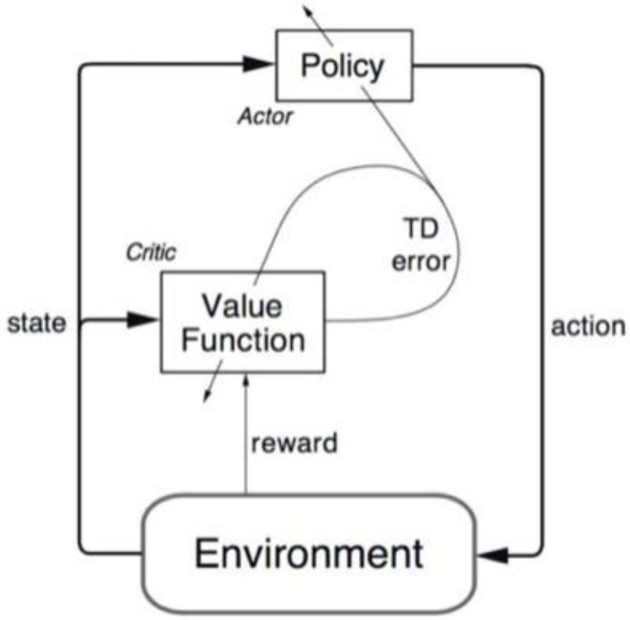
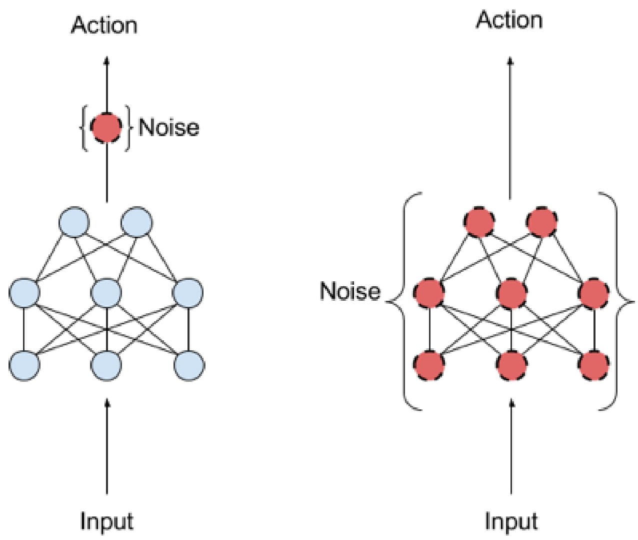
Learning Algorithm with few differences than Project 2

Deep Deterministic Policy Gradient (DDPG)

DDPG algorithm continuously improves the policy while exploring the environment and converges on large action space by using the actor-critic architecture. The actor specifies action
in a current state while critic criticizes the actions made by the actor by using Temporal Difference Error.

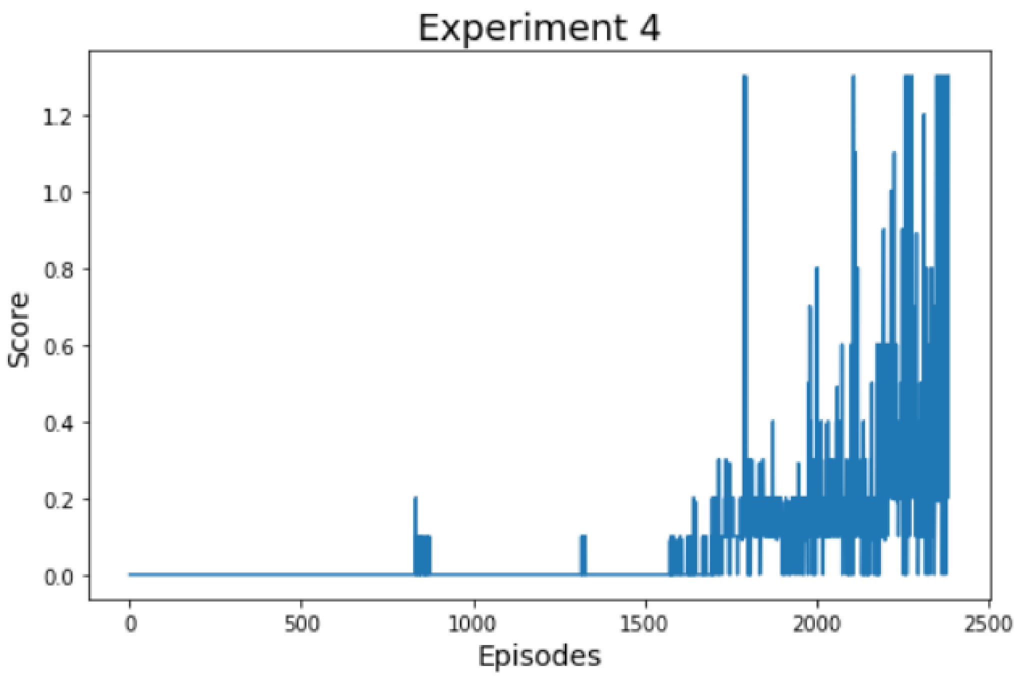


It maximizes the action-value function to compute the direction to change the current action to
increase overall discounted reward. However, it does not take into consideration how exploration is done. In the implemented DDPG code an agent adds its experience to the replay
buffer and local actor and critic are updated 10 times in a row using different samples from the
replay buffer. The OUNoise parameters are also experimented to add noise to the action space
of the policy.



The model is very simple to follow the Occam's Razor principle with parameters as follows:

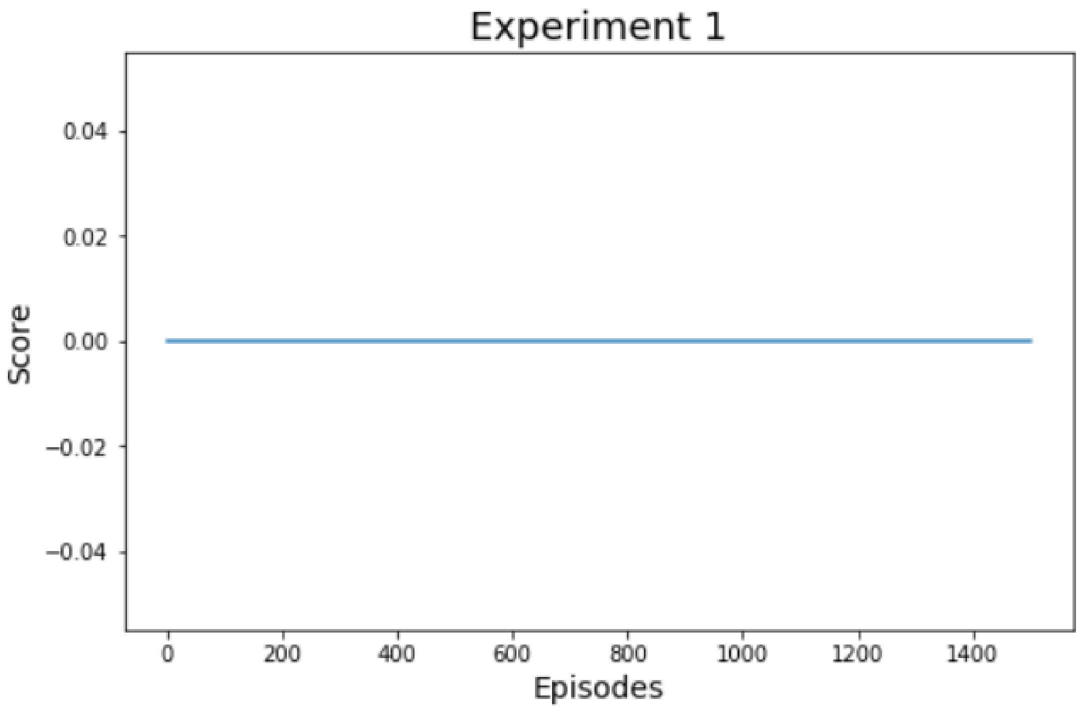
- 1 fully connected layer of 256 for actor
- 3 fully connected layer of 256,256 and 128
- BUFFER_SIZE = int(1e6) # replay buffer size
- BATCH_SIZE = 1024   # minibatch size
- GAMMA = 0.99   # discount factor
- TAU = 1e-3   # for soft update of target parameters
- LR_ACTOR = 1e-4   # learning rate of the actor
- LR_CRITIC = 1e-3   # learning rate of the critic
- WEIGHT_DECAY = 0   # L2 weight decay



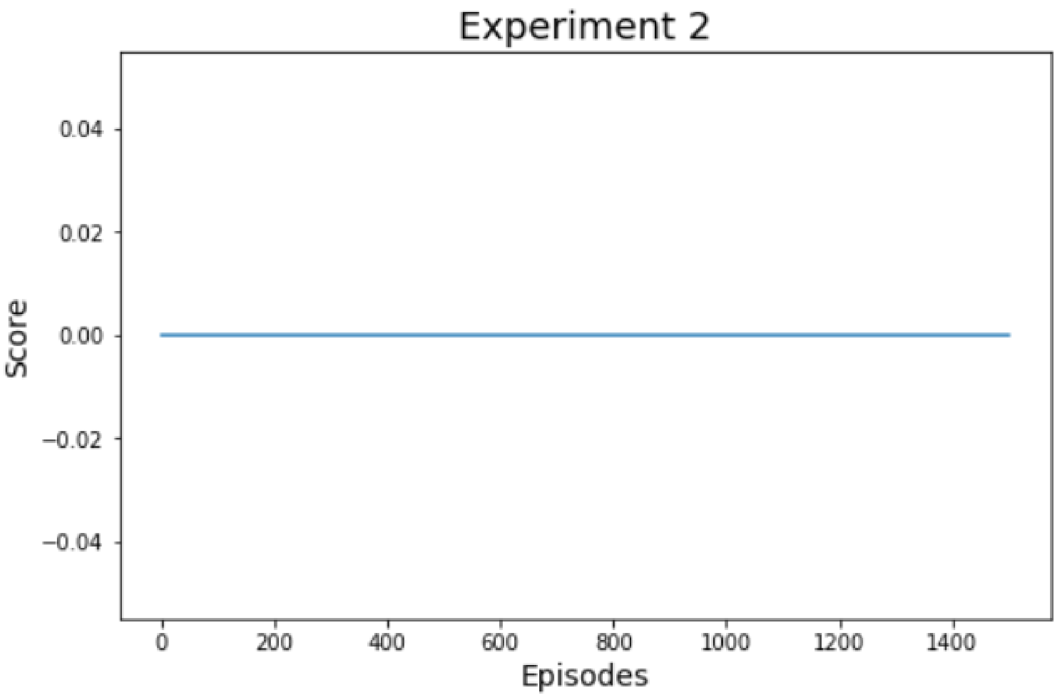
Environment solved in 2285 episodes! Average Score: 0.51

Parameters did not converge

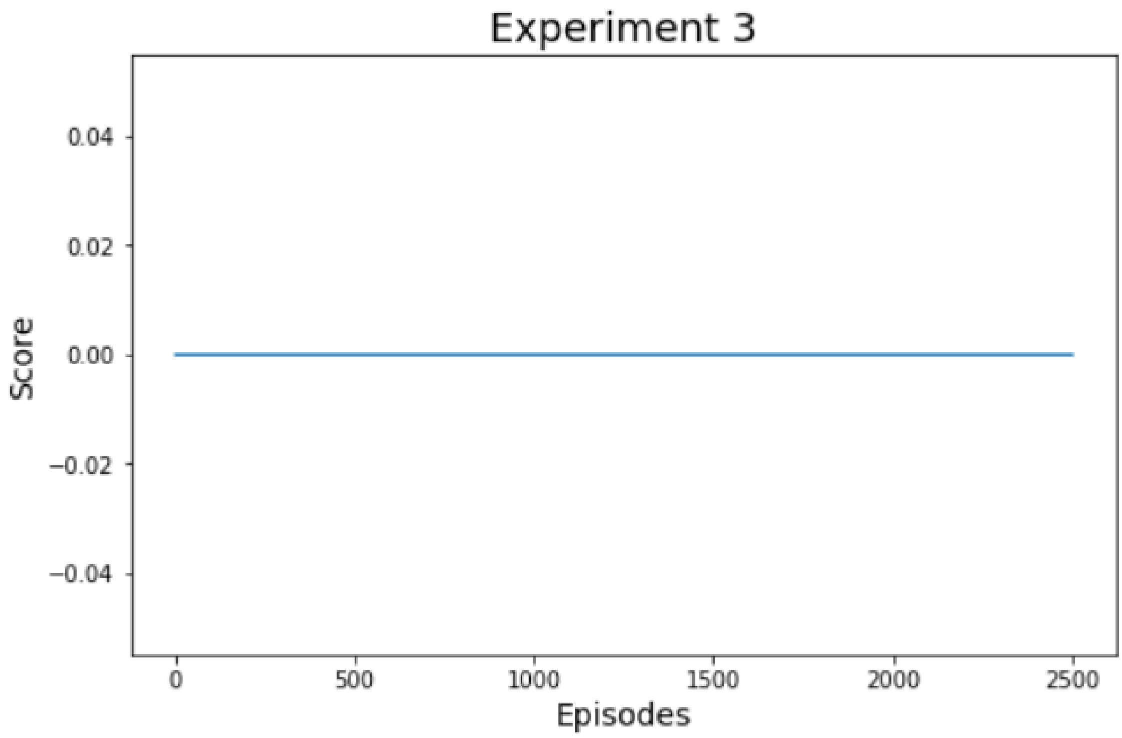
```
BUFFER_SIZE = int(1e6)  # replay buffer size
BATCH_SIZE = 256        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 1e-4         # learning rate of the actor
LR_CRITIC = 1e-3        # learning rate of the critic
WEIGHT_DECAY = 0.001    # L2 weight decay
```




```
BUFFER_SIZE = int(1e6)  # replay buffer size
BATCH_SIZE = 512        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 1e-4         # learning rate of the actor
LR_CRITIC = 1e-3        # learning rate of the critic
WEIGHT_DECAY = 0.1      # L2 weight decay
```




```
BUFFER_SIZE = int(1e6)  # replay buffer size
BATCH_SIZE = 512      # minibatch size
GAMMA = 0.99           # discount factor
TAU = 1e-3             # for soft update of target parameters
LR_ACTOR = 1e-4        # learning rate of the actor
LR_CRITIC = 1e-3       # learning rate of the critic
WEIGHT_DECAY = 0       # L2 weight decay
```



BUFFER_SIZE = int(1e6) # replay buffer size

BATCH_SIZE = 1024   # minibatch size

GAMMA = 0.99   # discount factor

TAU = 1e-3   # for soft update of target parameters

LR_ACTOR = 1e-4   # learning rate of the actor

LR_CRITIC = 1e-3   # learning rate of the critic

WEIGHT_DECAY = 0   # L2 weight decay

Ideas for Future Work

- Optimize parameters
- Increase the number of layers in the model
- Experiment with adding Batch Normalization and drop out in the model
- Implement Soccer

References

http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring18/ujjawal/DDPGAlgorithm.pdf
https://www.cs.ubc.ca/~gberseth/blog/demystifying-the-many-deep-reinforcement-learningalgorithms.html https://arxiv.org/pdf/1509.02971.pdf https://arxiv.org/abs/1604.06778
https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-bipedal
https://github.com/udacity/deep-reinforcement-learning/blob/master/ddpgpendulum/DDPG.ipynb
https://github.com/ShangtongZhang/DeepRL https://github.com/vy007vikas/PyTorch-ActorCriticRL
https://github.com/ikostrikov/pytorch-ddpg-naf