README.md

Project 2

Continuous Control

Deep Reinforcement Learning Udacity Nanodegree

Task

Navigate robot arm to move into target sphere on the rewards awarded using deep reinforcement learning.

Input

Vector of size 33.

Rewards

Based on the time spent in the target sphere.

Goal

Reach average reward of at least 30 points per episode.

Solution

We used code from previous code example as Deep Deterministic Policy Gradient learning repo link.

1 paddle problem

Parameters

- `mx` - number of maximal training iterations `1000`

Modification of nosie generator

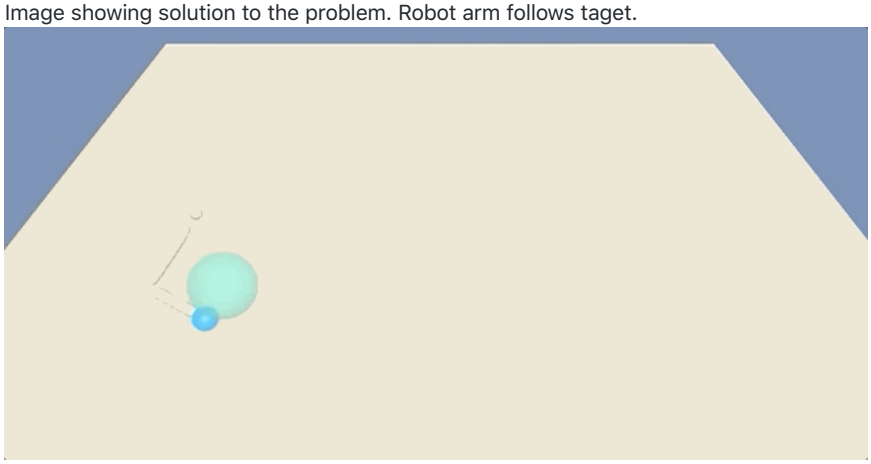
Code below is highlighting change of noise generator from the original `ddpg-bipedal` example.

```
class OUNoise:
    """Ornstein-Uhlenbeck process."""
    ...
        """Sigma reduction"""
        self.sigma = max(self.sigma_min, self.sigma*self.sigma_decay)
```

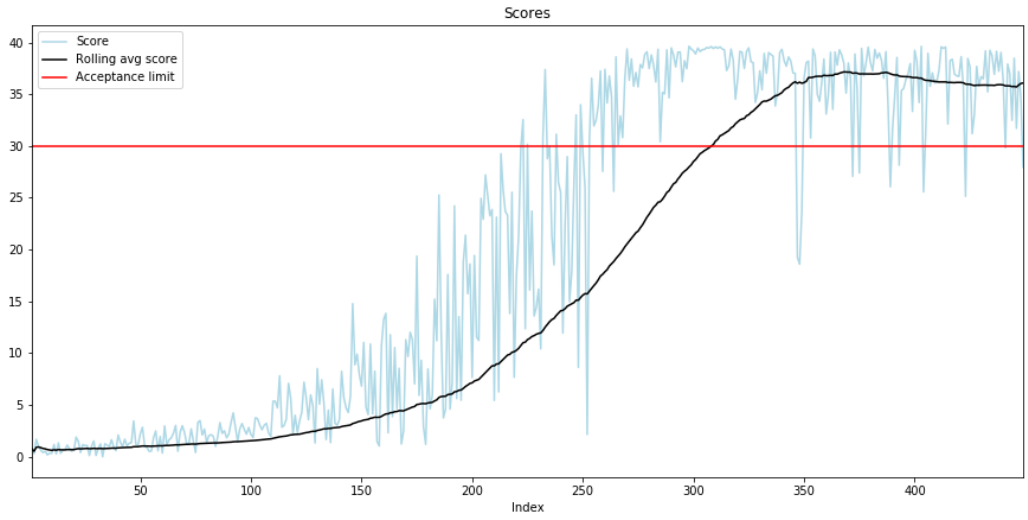
Training

Without the modification of Ornstein-Uhlenbeck process I was not able to reach the training goal. Neural network architecures are the same as in `ddpg-bipedal` for actor 1st layer contains 256 neurons and 2nd layer has 128 neurons 3rd layer has 4 neurons. Ativation function is hyperbolic tangens because action is vector of size 4 and values are continuous from -1 to 1. For critic base configuration is the same as for actor with different 3rd layer which contains only 1 neuron and linear activation function. We have experimented with hyperbolic tangens as activation function but the results were not good enough. We suspect it has to do with magnitude of the error. Model was saved every time new rolling average maximum was reached. If no new rolling average maximum was reached in 80 episodes traning was terminated.

To reproduce achieved results parameters you can run `python3 train.py` (assuming all of the `requirements.txt` are fullfilled).
Image showing solution to the problem. Robot arm follows taget.

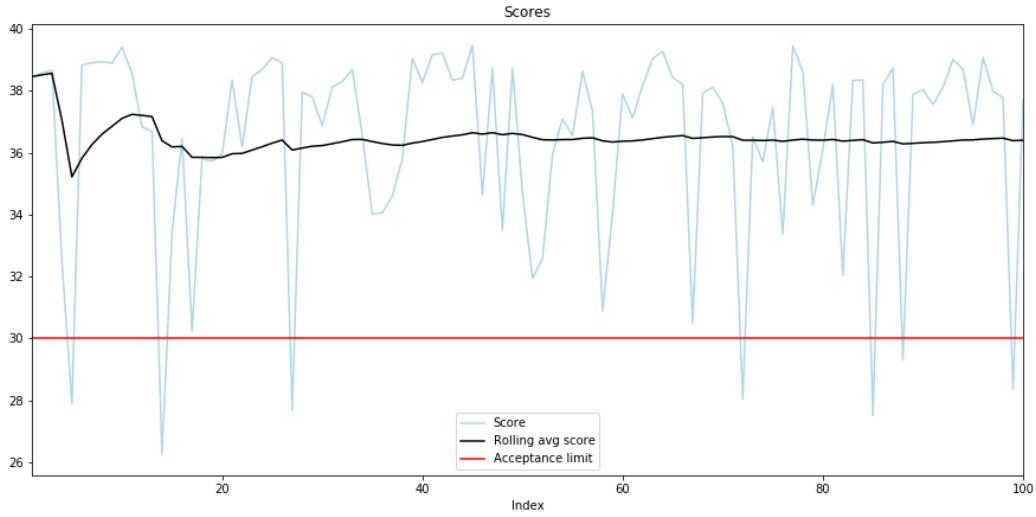


- Objective was reached in 309 episodes.
- Maximum rolling average was reached in 368 episodes.
- Max rolling average: 37.159
- Max score: 39.63



Test

- Test run with model weights loaded from `model/weights_local.torch` and `model/weights_target.torch` weights for cirtic are stored and loaded as well.
- To test included model you can run `python3 test.py` it will generate file data-test.csv with performace data recorded.
- Max rolling average: 38.56
- Max score: 39.47

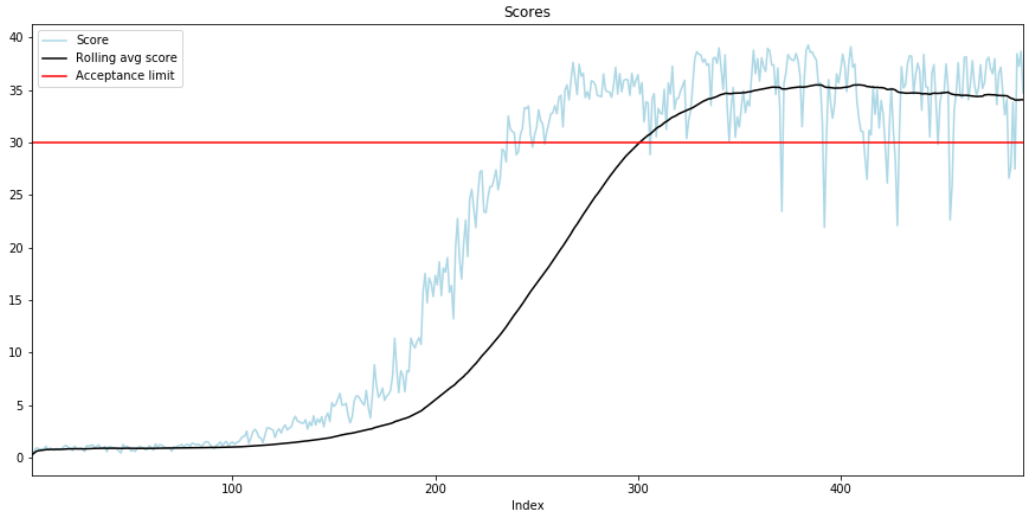


20 paddles problem

We modified code for 1 paddle to run for multiple paddles. Episode training increased from 2.5 seconds to 7.5 seconds training was done on Nvidia GPU.

Training

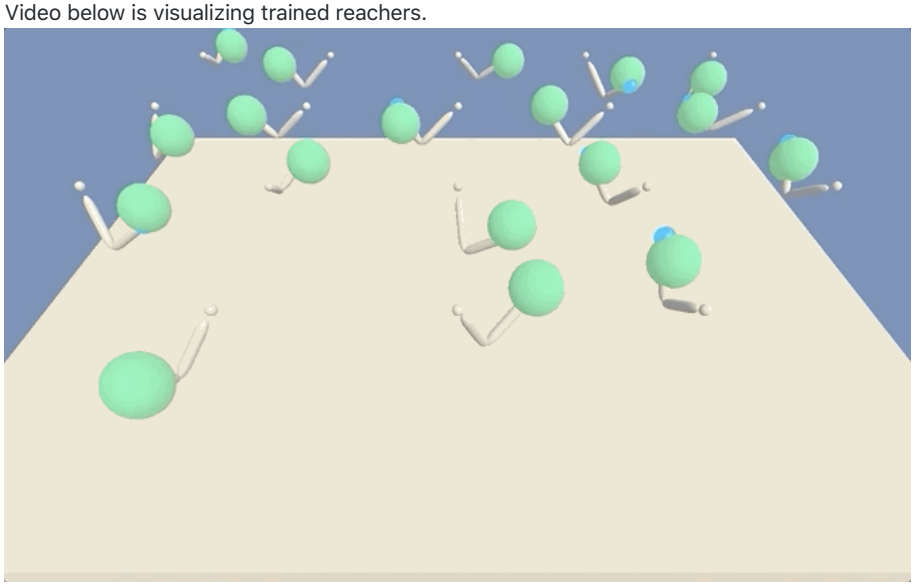
- Objective was reached in 301
- Maximum rolling average was reached in 409 episodes.
- Max rolling average: 35.504
- Max score: 39.286



Rolling average curve was smoother for 20 paddles problem than for 1 paddle. Objective score was reached at about the same time (slightly sooner for 20 paddles than for 1 paddle problem). Maximal rolling average of training score was reached later for 20 paddles problem.

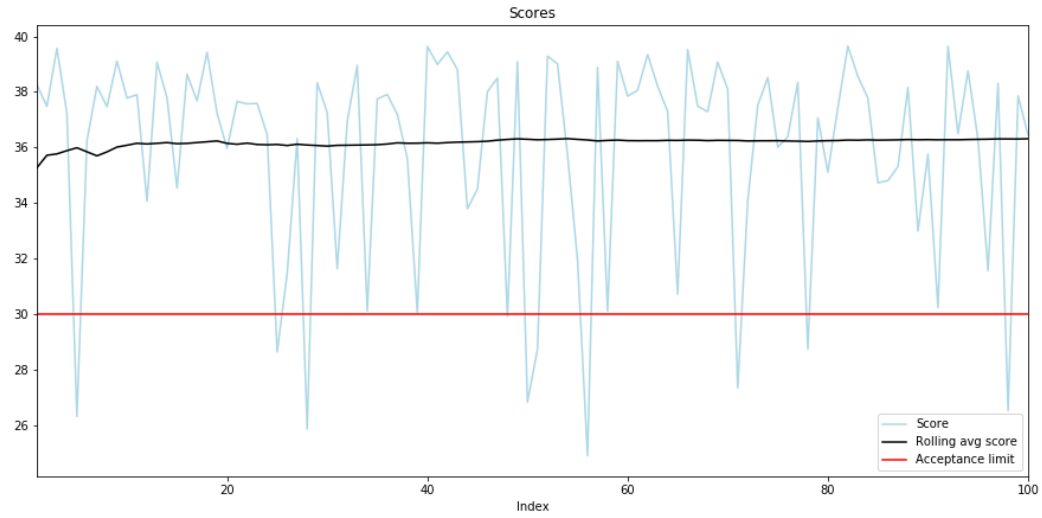
Test

Video below is visualizing trained reachers.



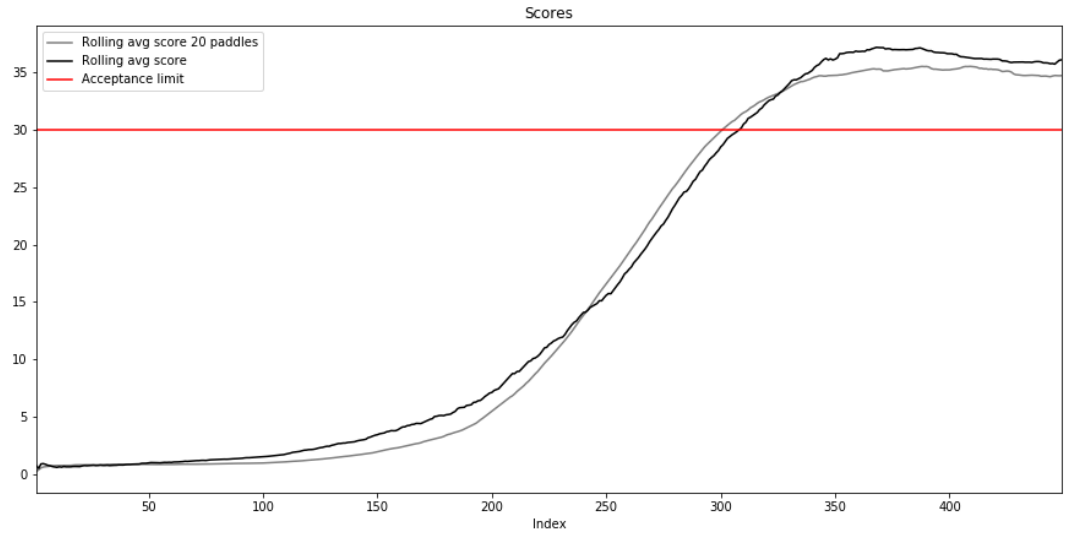
After loading training weights we recorded following test results.

- Average of test scores after 100 episodes was 36.18.



1 vs 20 paddles training

Graph below visualize difference between training 1 and 20 paddles. We can see that for 20 paddles rolling average is much smoother reaches the goal little bit faster but does not reach as high values as for single paddle.



]

Conclusions

Model training was very sensitive to changes in noise or qnetwork architecture modifications. However network could be trained for the target score with wide range of number of neurons. Given the changes in convergence due to adding sigma decay. I think there is space for improvement in modifying noise generation. Another possible modification is neural network architecture.