

---

## Simple implementation of Reinforcement Learning (A3C) using Pytorch

This is a toy example of using multiprocessing in Python to asynchronously train a neural network to play discrete action CartPole and continuous action Pendulum games. The asynchronous algorithm I used is called Asynchronous Advantage Actor-Critic or A3C.

I believe it would be the simplest toy implementation you can find at the moment (2018-01).

### What are the main focuses in this implementation?

- Pytorch + multiprocessing (NOT threading) for parallel training
- Both discrete and continuous action environments
- To be simple and easy to dig into the code (less than 200 lines)

### Reason of using Pytorch instead of Tensorflow

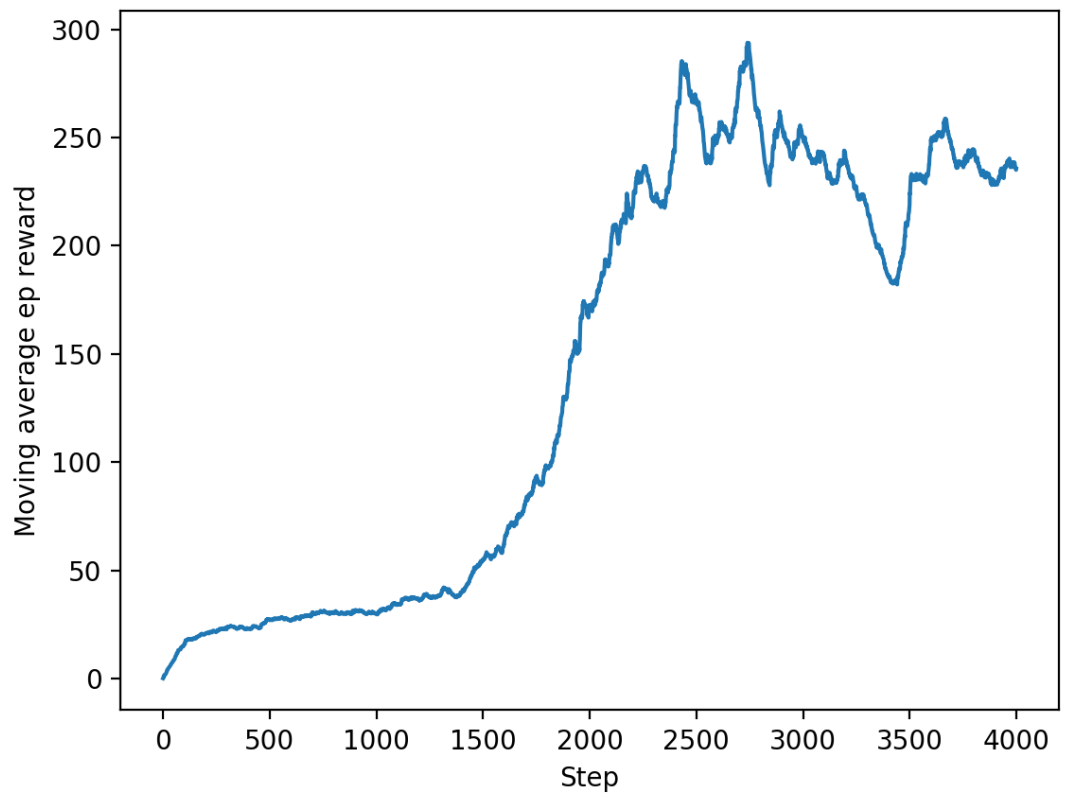
Both of them are great for building your customized neural network. But to work with multiprocessing, Tensorflow is not that great due to its low compatibility with multiprocessing. I have an implementation of Tensorflow A3C build on threading. I even tried to implement distributed Tensorflow. However, the distributed version is for cluster computing which I don't have. When using only one machine, it is slower than threading version I wrote.

Fortunately, Pytorch gets the multiprocessing compatibility. I went through many Pytorch A3C examples (there, there and there). They are great but too complicated to dig into the code. Therefore, this is my motivation to write my simple example codes.

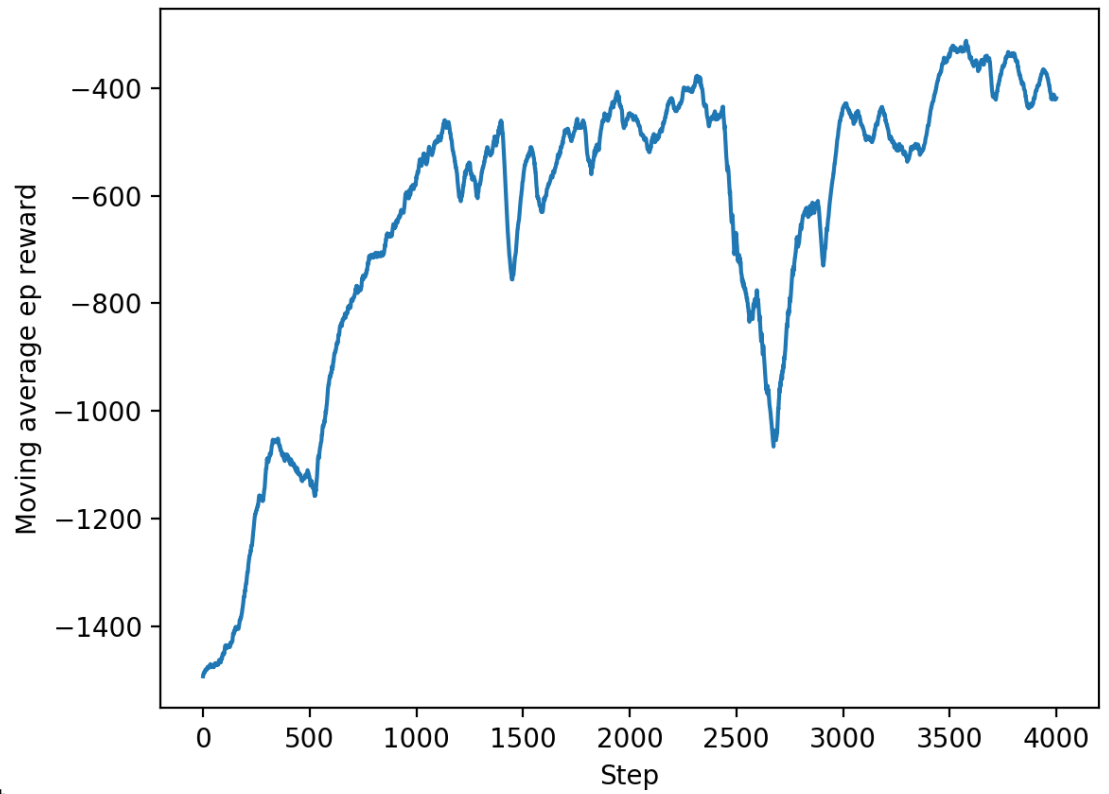
BTW, if you are interested to learn Pytorch, there is my simple tutorial code with many visualizations. I also made the tensorflow tutorial (same as pytorch) available in here.

### Codes & Results

- `shared_adam.py`: optimizer that shares its parameters in parallel
- `utils.py`: useful function that can be used more than once
- `discrete_A3C.py`: CartPole, neural net and training for discrete action space
- `continuous_A3C.py`: Pendulum, neural net and training for continuous action space



CartPole result



Pendulum result

### Dependencies

- pytorch >= 0.4.0
- numpy
- gym
- matplotlib