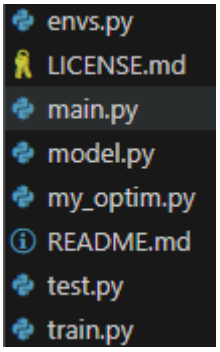
The best performing method, an asynchronous variant of actor-critic, surpasses the current state-of-the-art on the Atari domain while training for half the time on `a single multi-core CPU` instead of GPU.

```python
os.environ['OMP_NUM_THREADS'] = '1'
```

`OMP_NUM_THREADS`
OpenMP(Open Multi-Processing) 라이브러리를 사용할 때, 사용할 `스레드(thread)`의 수를 **'1'** 로 설정한다.



```python
from __future__ import print_function

import argparse
import os

import torch
import torch.multiprocessing as mp

import my_optim
from envs import create_atari_env
from model import ActorCritic

from test import test
from train import train

# Based on
# https://github.com/pytorch/examples/tree/master/mnist_hogwild
# Training settings

parser = argparse.ArgumentParser(description='A3C')

parser.add_argument('--lr', type=float, default=0.0001,
                    help='learning rate (default: 0.0001)')
parser.add_argument('--gamma', type=float, default=0.99,
                    help='discount factor for rewards (default: 0.99)')
parser.add_argument('--gae-lambda', type=float, default=1.00,
                    help='lambda parameter for GAE (default: 1.00)')
parser.add_argument('--entropy-coef', type=float, default=0.01,
                    help='entropy term coefficient (default: 0.01)')
parser.add_argument('--value-loss-coef', type=float, default=0.5,
                    help='value loss coefficient (default: 0.5)')
parser.add_argument('--max-grad-norm', type=float, default=50,
                    help='value loss coefficient (default: 50)')
parser.add_argument('--seed', type=int, default=1,
                    help='random seed (default: 1)')
parser.add_argument('--num-processes', type=int, default=4,
                    help='how many training processes to use (default: 4)')
parser.add_argument('--num-steps', type=int, default=20,
                    help='number of forward steps in A3C (default: 20)')
parser.add_argument('--max-episode-length', type=int, default=1000000,
                    help='maximum length of an episode (default: 1000000)')
parser.add_argument('--env-name', default='PongDeterministic-v4',
                    help='environment to train on (default: PongDeterministic-v4)')
parser.add_argument('--no-shared', default=False,
                    help='use an optimizer without shared momentum.')
```

```python
if __name__ == '__main__':
    os.environ['OMP_NUM_THREADS'] = '1'
    os.environ['CUDA_VISIBLE_DEVICES'] = ""

    args = parser.parse_args()

    torch.manual_seed(args.seed)
    env = create_atari_env(args.env_name)
    shared_model = ActorCritic(
        env.observation_space.shape[0], env.action_space)
    shared_model.share_memory()

    if args.no_shared:
        optimizer = None
    else:
        optimizer = my_optim.SharedAdam(shared_model.parameters(), lr=args.lr)
        optimizer.share_memory()

    processes = []

    counter = mp.Value('i', 0)
    lock = mp.Lock()

    p = mp.Process(target=test, args=(args.num_processes, args, shared_model, counter))
    p.start()
    processes.append(p)

    for rank in range(0, args.num_processes):
        p = mp.Process(target=train, args=(rank, args, shared_model, counter, lock, optimizer))
        p.start()
        processes.append(p)

    for p in processes:
        p.join()
```

PYTHON

```python
os.environ['OMP_NUM_THREADS'] = '1'
```

`OMP_NUM_THREADS`
OpenMP(Open Multi-Processing) 라이브러리를 사용할 때, 사용할 `스레드(thread)`의 수를 **'1'** 로 설정한다.

PYTHON

```python
os.environ['CUDA_VISIBLE_DEVICES'] = ""
```

`CUDA_VISIBLE_DEVICES`
CUDA를 사용할 때, 사용할 GPU 장치를 설정한다. 빈 문자열로 설정하면, CUDA가 사용할 수 있는 **GPU 장치를 비활성화**한다.

PYTHON

```
args = parser.parse_args()

torch.manual_seed(args.seed)
env = create_atari_env(args.env_name)
```

parser 객체가 명령줄 인수를 읽고, 정의된 인수들에 따라 파싱한다.
파싱된 인수들은 'args' 객체로 반환된다. 이 'args' 객체를 통해 프로그램 내에서 명령줄 인수를 접근할 수 있다.

PYTHON

```
shared_model = ActorCritic(env.observation_space.shape[0],
env.action_space)

shared_model.share_memory()
```

프로세스를 생성하고, 각각 다른 환경에서 훈련하되 `shared_memory`에 있는 model의 parameters은 공유한다.

PYTHON

```
if args.no_shared:
        optimizer = None

else:
        optimizer =
my_optim.SharedAdam(shared_model.parameters(), lr=args.lr)
        optimizer.share_memory()
```

PYTHON

```
processes = []
```

`test` 와 `train` processes이 포함되어 있다.

PYTHON

```
counter = mp.Value('i', 0)
```

Multi-process 간에 공유되는 변수이다.
모든 `test` , `train` 프로세스가 함께 공유하는 전역적인 카운터 역할을 한다.

```python
lock = mp.Lock()
```

Lock 객체를 생성한다.
여러 worker process가 동시에 shared_model을 업데이트할 때 데이터 경합을 방지한다.
한 worker가 shared_model을 업데이트하는 동안 다른 worker는 기다리게 한다.

- 4개의 train process와 1개의 test process를 생성한다.

```python
p = mp.Process(target=test, args=(args.num_processes, args,
shared_model, counter))
```

- `mp.Process` 새로운 프로세스 `p`를 생성한다.
- `target=test` test process를 실행하는 새로운 process `p` 를 지정한다.
- 새로 생성된 프로세스가 실행할 **목표 함수**를 지정한다. 여기서는 'test' 함수가 목표 함수로 지정되었다.
- `args=(args.num_processes, args, shared_model, counter)`

```python
p.start()
```

`mp.Process` 를 통해서 생성된 프로세스 `p` 를 시작한다.

```python
processes.append(p)
```

PYTHON

```
for rank in range(0, args.num_processes):
        p = mp.Process(target=train, args=(rank, args,
shared_model, counter, lock, optimizer))
        p.start()
        processes.append(p)
```

`target=train` 'train' 함수를 실행한다.

PYTHON

```
for p in processes:
    p.join()
```

각 프로세스가 종료될 때까지 기다린다.
`processes` 리스트에는 `test` 와 `train` 프로세스들이 포함되어 있다.

`p.join()` 는 프로세스 `p` 가 종료될 때까지 실행을 일시중지하고 기다리라는 의미이다.

PYTHON

```python
import multiprocessing as mp
import time
import random

def worker_func(worker_id, progress_counter):
    print(f"Worker {worker_id} is starting")

    sleep_time = random.uniform(0.5, 2.0)
    time.sleep(sleep_time)

    print(f"Worker {worker_id} has finished after
{sleep_time:.2f} seconds")

    with progress_counter.get_lock():
        progress_counter.value += 1

if __name__ == "__main__":
    processes = []
    num_processes = 5
    progress_counter = mp.Value('i', 0)

    for i in range(num_processes):
        p = mp.Process(target=worker_func, args=(i,
progress_counter))
        processes.append(p)
        p.start()

    for p in processes:
        p.join()

    print(f"All processes have finished. Total progress:
{progress_counter.value}")
```

PYTHON

```
Worker 0 is starting
Worker 2 is starting
Worker 1 is starting
Worker 4 is starting
Worker 3 is starting
Worker 1 has finished after 0.64 seconds
Worker 4 has finished after 1.22 seconds
Worker 3 has finished after 1.43 seconds
Worker 0 has finished after 1.61 seconds
Worker 2 has finished after 1.80 seconds
All processes have finished. Total progress: 5
```