Asynchronous advantage actor-critic (A3C) method on Atari games

Wei Chen, Dewen Zhong

December 7, 2016

Outline

Introduction

- FF neural networks
- CNN
- RNN (LSTM)
- Actor-critic method $\approx$ Q-learning $+$ policy gradient
- Asynchronous gradient descent
- CPU/GPU hybrid method

Actor-critic method

$$\text{Screenshots} \rightarrow \begin{cases} \text{Actor } \pi(a|s,\theta_p)\text{: probability of each action} \\ \text{Critic } V(s,\theta_v)\text{: expected discounted future reward} \end{cases}$$

Basic idea of parameter updates

The parameters should be updated every few steps such that

- $V(s,\theta_v)$ should reflect the expected discounted future reward (EDFR), i.e., $(R - V(s,\theta_v))^2$ should be minimized.
- An action that gives a higher/lower EDFR than current estimated one should be encouraged/discouraged, i.e., $-\log \pi(a|s,\theta_p)\,(R - V(s,\theta_v))$ should be minimized.

Asynchronous gradient descent

Why not experience replay?

- Requires large memory and computation.
- Uses data from old policies to train current policy.

Asynchronous gradient descent

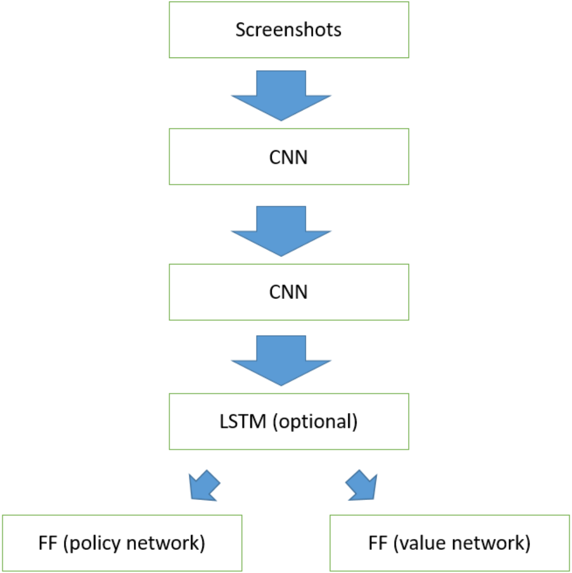
Why not experience replay?

- Requires large memory and computation.
- Uses data from old policies to train current policy.

Then how do we get uncorrelated training data?

- Run game agents concurrently, and train independent models.
- Update the parameters for the shared model asynchronously.

Model structure



Implementation issues

- For each process/thread: run an individual game simulator, and a training model
- Shared objects:
  - Global shared model (for asynchronous updates)
  - Shared RMSprop optimizer (to enhance stability)

Implementation issues

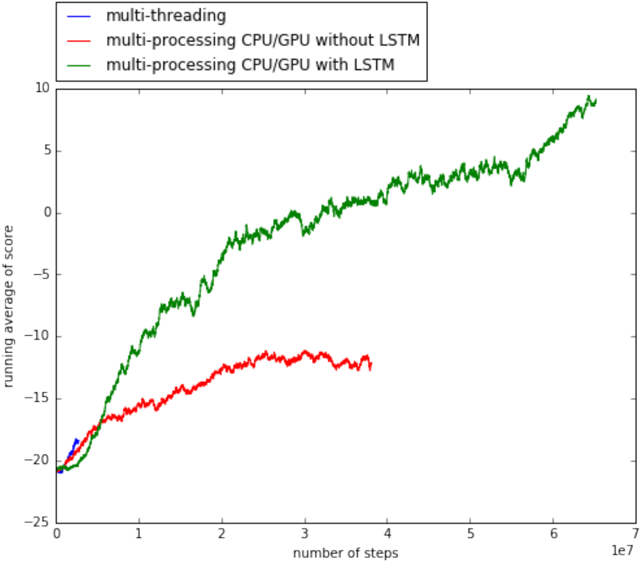
Q: How to parallelize training and do asynchronous updates?

- Multiprocessing in CPU: 6-7 h/M steps
  - Sometimes some processes get stuck and keep 'sched_yield()' (from tracing Linux kernel calls). This might be related to OpenBLAS issue (https://github.com/scikit-learn/scikit-learn/issues/2088)
- Multiprocessing in GPU: 0.6 h/M steps
  - Shared model does not work, multiprocessing module does not support shared arrays in GPU (https://github.com/muupan/async-rl/issues/10).

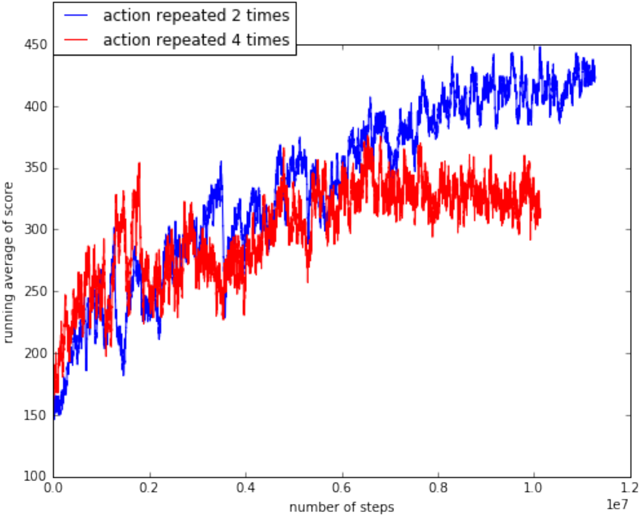
Implementation issues

- Multithreading in GPU: 5-6 h/M steps
  - Slow since only one thread can be executed each time due to GIL (global interpretation lock).
- Multiprocessing in GPU, while requesting shared weight and optimizer parameters from CPU memory in each iteration: 2 h/M steps
- Multiprocessing, with independent models in GPU, and shared model in CPU (parameters shared in CPU): 0.5 h/M steps
  - Reason: profiling shows that most time consuming part is backpropagation.

Results: Pong

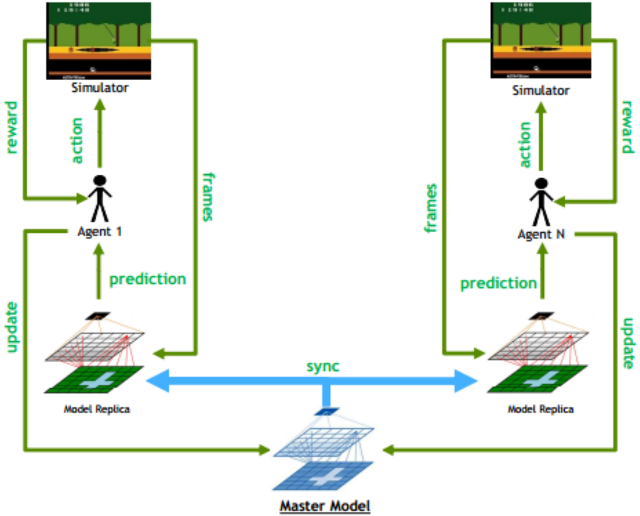


Results: Space-invaders

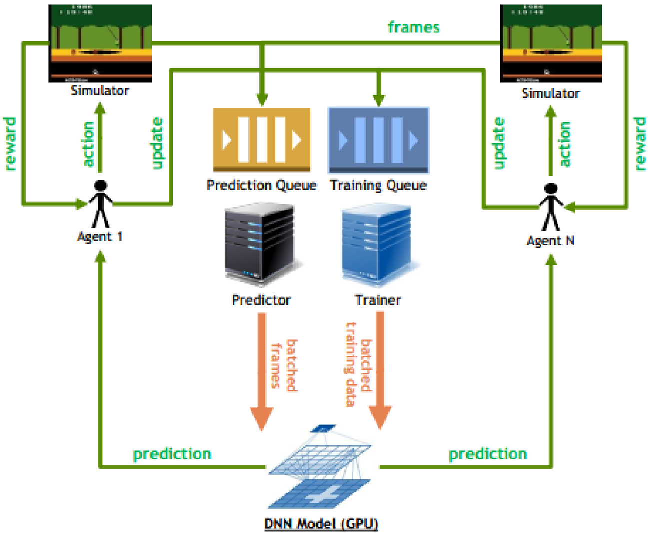


https://gym.openai.com/evaluations/eval_i57wuZpQhKHRhywgh4Ug

Latest progress: GA3C (ICLR 2017) by UIUC and Nvidia



Latest progress: GA3C (ICLR 2017) by UIUC and Nvidia



Conclusions

Conclusions

- A3C works for training Atari games, the implementation of training and parallelization part would significantly affect the efficiency.
- GA3C could possibly reduce training time by 1-2 order of magnitude.

Code

https://github.com/weiHelloWorld/IE598_project

Questions?