Parallelization and reinforcement learning: case studies

Adam Jakubowski and Henryk Michalewski
Overview

- case study of distributed TensorFlow: *distributed training of Atari games*
- case study of Horovod: *Unreal Engine 4 and Carla*
- case study of barebone MPI: *Learning to Run using hundreds of cores*
Computational budget

1. Approximate scale of a single experiment
   - $2.5\times10^6 - 5\times10^6$ CPU core hours or
   - $1\times10^5 - 2\times10^5$ GPU hours
2. **Our constraints**: on average we have at our disposal
   - 220 Xeon servers and
   - 120 K40 GPU cards
Research Budget

1. Human resources sponsored by deepsense.ai, Intel and an automotive company.
2. We use the Prometheus supercomputer, located in the Academic Computer Center Cyfronet in the AGH University of Science and Technology in Kraków.
3. We use a GCP grant from Google.
4. We use a number of cards given by NVIDIA to the Department of Mathematics, Informatics and Mechanics of the University of Warsaw.
5. We have received 600 000 EUR on reinforcement learning from the National Science Center Poland.
Distributed Tensorflow and Atari
DDRL: Distributed Deep Reinforcement Learning

Goal: Train agents for playing Atari games on CPU, as quickly as possible (smallest time to reference score)

Initial resources:
- Optimized implementation of BA3C for CPU
- Access to a 50k cores CPU cluster

Final result:
- 21 minutes to 300 score in Atari Breakout
- Almost linear scaling for up to 768 cores

Experiments carried out by: Igor Adamski, Robert Adamski, Tomasz Grel, Adam Jędrych, Kamil Kaczmarek and Henryk Michalewski
DDRL: Asynchronous vs Synchronous Training

- Async is faster and easier to implement, but causes “stale gradients”
DDRL: Asynchronous vs Synchronous Training

Fig. 3: Typical asynchronous training attempt, 64 workers.
DDRL: Asynchronous vs Synchronous Training

- Sync works much better when using more than ~10 nodes (no stale gradients)
- Side effect: very large batch sizes (2048 samples)
DDRL: Large batch sizes

- Adam optimizer the only one that works for BA3C
- “Linear scaling rule” did not work for Adam (unstable training)
- Our approach: don’t increase the learning rate, allow more adaptability (smaller epsilon)

\[
m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \\
v_t = \beta_2 v_{t-1} + (1 - \beta_1) g_t^2 \\
\eta_t = \eta \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t}, \\
\theta_t = \theta_{t-1} - \eta_t \frac{m_t}{\sqrt{v_t + \hat{e}}}
\]
(c) First 20 minutes, $\hat{\epsilon} = 10^{-3}$

(d) First 20 minutes, $\hat{\epsilon} = 10^{-8}$
DDRL: Results
DDRL: Results

Table 3: Algorithm performance in 3 games tested in both papers. Best stable score and time (in hours) to achieve it are given. The data are based on the best reported results found in the training plots in [6,2,20].

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BeamRider</td>
<td>14900 (2.7h)</td>
<td>3000 (24h)</td>
<td>–</td>
<td>15000 (15h)</td>
</tr>
<tr>
<td>Breakout</td>
<td>350 (0.5h)</td>
<td>350 (21h)</td>
<td>400 (15h)</td>
<td>500 (11h)</td>
</tr>
<tr>
<td>Boxing</td>
<td>98 (0.5h)</td>
<td>92 (2h)</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Pong</td>
<td>20 (4h)</td>
<td>18 (1h)</td>
<td>17 (24h)</td>
<td>20 (8h)</td>
</tr>
<tr>
<td>Seaquest</td>
<td>1832 (0.5h)</td>
<td>1706 (24h)</td>
<td>1840 (24h)</td>
<td>2300 (24h)</td>
</tr>
<tr>
<td>SpaceInvaders</td>
<td>650 (0.5h)</td>
<td>600 (24h)</td>
<td>700 (24h)</td>
<td>1400 (15h)</td>
</tr>
</tbody>
</table>
# DDRL: Results

**Table 3:** Algorithm performance in 3 games tested in both papers. Best stable score and time (in hours) to achieve it are given. The data are based on the best reported results found in the training plots in [6,2,20].

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BeamRider</td>
<td>14900 (2.7h)</td>
<td></td>
<td>3000 (24h)</td>
<td></td>
</tr>
<tr>
<td>Breakout</td>
<td>350 (0.5h)</td>
<td></td>
<td>350 (21h)</td>
<td>400 (15h)</td>
</tr>
<tr>
<td>Boxing</td>
<td>98 (0.5h)</td>
<td></td>
<td>92 (2h)</td>
<td></td>
</tr>
<tr>
<td>Pong</td>
<td>20 (4h)</td>
<td></td>
<td>18 (1h)</td>
<td>17 (24h)</td>
</tr>
<tr>
<td>Seaquest</td>
<td>1832 (0.5h)</td>
<td></td>
<td>1706 (24h)</td>
<td>1840 (24h)</td>
</tr>
<tr>
<td>SpaceInvaders</td>
<td>650 (0.5h)</td>
<td></td>
<td>600 (24h)</td>
<td>700 (24h)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>15000 (15h)</td>
<td>500 (11h)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DDRL: Results

A work from Berkeley reduced our training times by a factor of 3 (using single NVIDIA machine DGX-1). With IMPALA & v-trace, a distributed framework introduced by DeepMind in the middle of 2018, the results can be reduced to seconds, however this may be much harder to re-create.

<table>
<thead>
<tr>
<th>Game</th>
<th>DDRL [h]</th>
<th>A3C [h]</th>
<th>GA3C [h]</th>
<th>BA3C [h]</th>
<th>A3C [h]</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeamRider</td>
<td>14900</td>
<td>3000</td>
<td></td>
<td></td>
<td>15000</td>
</tr>
<tr>
<td>Breakout</td>
<td>350</td>
<td>350</td>
<td></td>
<td>400</td>
<td>500</td>
</tr>
<tr>
<td>Boxing</td>
<td>98</td>
<td>92</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pong</td>
<td>20</td>
<td>18</td>
<td></td>
<td>17</td>
<td>20</td>
</tr>
<tr>
<td>Seaquest</td>
<td>1832</td>
<td>1706</td>
<td></td>
<td>1840</td>
<td>2300</td>
</tr>
<tr>
<td>SpaceInvaders</td>
<td>650</td>
<td>600</td>
<td></td>
<td>700</td>
<td>1400</td>
</tr>
</tbody>
</table>
Initial, 15 minutes and 30 minutes performance for Breakout and Boxing
An “enhanced version” of Boxing imagined by a neural network (from our recent experiment with Google Brain)
Unreal Engine 4

Adam Jakubowski
Commercial project for one of the world leaders of the automotive industry

We are researching viability of training agents in simulated environments that are deployed in real life

Our task - train an agent in simulation to drive some route and transfer this knowledge to real-life

Need sophisticated graphical environment with realistic visuals and physics

CARLA is our simulator of choice

Team members: Krzysztof Galias, Adam Jakubowski, Henryk Michalewski, Przemysław Podczasi, Paweł Zięcina
Are you familiar with those?
CARLA - reinforcement learning setup

- Custom OpenAI gym wrapper for CARLA
  https://github.com/openai/gym
- Project started from
  https://github.com/openai/baselines
- RL algos:
  - Proximal Policy Optimization (PPO)
  - \(v\)-trace

Trajectory visualization with model outputs
Parallelisation - why do we need it?

- **Standard reason #1 - speed**
  - CARLA is very expensive to run
    - around 60 frames in training per node == ~5M per day
- **RL usually takes millions of frames to solve environment**
  - Atari games like pong/breakout take around ~20M frames to solve
  - in some papers we can even see numbers up to billion!
- **RL is still very sensitive to hyperparameters**
  - need parallelism to perform fast hyperparameter searches
- **More environments in parallel == more diverse data in a batch**
  - consecutive observations in a trajectory are highly correlated
  - on-policy -> trajectories can be used only once for network update
  - if we want more diverse data in the batch we need more environments in parallel
  - we have hard-limit (due to GPU memory) on how many envs we can run on one node
    - we run 4-8 CARLA instances per node
- **Up to 200 CARLA simulators in one parallel experiment**
- **For atari up to 2000 environments in parallel**
How did we do it?
Horovod overview

- Library for distributed deep learning
- Developed by Uber - [https://github.com/uber/horovod](https://github.com/uber/horovod)
- Integrated with TensorFlow, Keras and PyTorch
- Uses MPI
- Data parallelism only
- Very simple to use
  - less than 1 MD to go from single-node code to multi-node code
import horovod.tensorflow as hvd

hvd.init()

(...) # setup code

optimizer = (...)
opt = hvd.DistributedOptimizer(opt)

(...)

# Ran before training. Copies weights from root node to other nodes so initializations are consistent across nodes
session.run(hvd.broadcast_global_variables(0))

python /home/user/project/train.py

mpirun -H vm1,vm2,vm3,vm4 python /home/user/project/train.py
Learning to Run using hundreds of cores
Case study - Learning to Run using hundreds of cores

- Aim: use RL to learn running
- biologically exact musculoskeletal model (Stanford Neuromuscular Biomechanics Laboratory)
- competition track on NIPS 2017
- team H. Michalewski, P. Miłoś and B. Osiński ended up at the 6th place (University of Warsaw, deepsense.ai)
- rewards for the distance covered by skeleton
- no domain knowledge used
Round 1 - jumper

- buggy environment

- combining two strategies
  - careful jumper
  - sprinter
Bipedal walking

- increased exploration using frameskip = 2

- this is an example of a widespread problem of RL, algorithms often find suboptimal solutions due to poor exploration
Reward shaping

- knee bending enforced with reward shaping
  - points for keeping knee angle below 180°
- side effect: faster learning