GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING

M. Babaeizadeh†,‡, I. Frosio†, S. Tyree†, J. Clemons†, J. Kautz†

†University of Illinois at Urbana-Champaign, USA ‡NVIDIA, USA

An ICLR 2017 paper

A github project
THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:

"MY CODE'S COMPILING."

HEH! GET BACK TO WORK!

COMPILING!

OH. CARRY ON.
THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:

“MY DNN IS TRAINING”

Hey! Get back to work!

Training!

Oh. Carry on.
GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING
GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING

Learning to accomplish a task

Image from www.33rdsquare.com
GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING

Definitions

$S_t = S_{t+1}$

✓ Environment
✓ Agent
✓ Observable status $S_t$
✓ Reward $R_t$
✓ Action $a_t$
✓ Policy $a_t = \pi(S_t)$
GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING

Definitions

\[ S_t, R_t \]

\[ a_t = \pi(S_t) \]

Deep RL agent
GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING

Definitions

$S_t, R_t$

$\Delta \pi(\cdot)$

$a_t = \pi(S_t)$
GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING
GPU-BASED **A3C** FOR DEEP REINFORCEMENT LEARNING

GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING
GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING

The GPU
GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING
LOW OCCUPANCY (33%)
GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING

HIGH OCCUPANCY (100%)
GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING

HIGH OCCUPANCY (100%), LOW UTILIZATION (40%)
GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING

HIGH OCCUPANCY (100%), HIGH UTILIZATION (100%)
GPU-BASED A3C FOR DEEP REINFORCEMENT LEARNING

BANDWIDTH LIMITED
MAPPING DEEP PROBLEMS TO A GPU

REGRESSION, CLASSIFICATION, ...

100% utilization / occupancy

REINFORCEMENT LEARNING

data

status, reward

labels

action
Agent 1
\( S_t, R_t \)
\( a_t = \pi(S_t) \)

Agent 2
\( S_t, R_t \)
\( a_t = \pi(S_t) \)

Master model

Agent 16
\( S_t, R_t \)
\( a_t = \pi(S_t) \)

\[ a_t = p(S_t) \]
A3C

Agent 1

\[ a_t = \pi(S_t) \]

Agent 2

\[ a_t = \pi(S_t) \]

\[ R_0, R_1, R_2, R_3, R_4 \]

Agent 16

\[ a_t = \pi(S_t) \]

\[ R_0, R_1, R_2, R_3, R_4 \]

Master model
A3C

Agent 1

\( S_t, R_t \)

\( a_t = \pi(S_t) \)

Agent 2

\( S_t, R_t \)

\( a_t = \pi(S_t) \)

\( \ldots \)

Agent 16

\( S_t, R_t \)

\( a_t = \pi(S_t) \)
A3C

Agent 1

$S_t, R_t$

$a_t = \pi(S_t)$

Agent 2

$S_t, R_t$

$a_t = \pi(S_t)$

Agent 16

$S_t, R_t$

$a_t = \pi(S_t)$

Master model

Small inference batch size (1), low occupancy
Agent 1

\[ S_t, R_t \]

\[ a_t = \pi(S_t) \]

\[ \Delta \pi(\cdot) \]

Small training batch size (5), low occupancy

Agent 2

\[ S_t, R_t \]

\[ a_t = \pi(S_t) \]

Agent 16

\[ S_t, R_t \]

\[ a_t = \pi(S_t) \]
A3C

Agent 1

\[ a_t = \pi(s_t) \]

Agent 2

\[ a_t = \pi(s_t) \]

... Agent 16

\[ a_t = \pi(s_t) \]

\[ S_t, R_t \]

\[ S_t, R_t \]

\[ S_t, R_t \]

\[ R_0, R_1, R_2, R_3, R_4 \]

\[ R_0, R_1, R_2, R_3, R_4 \]

\[ R_0, R_1, R_2, R_3, R_4 \]

\[ \Delta \pi(\cdot) \]

\[ \pi'(\cdot) \]

Master model

Intense traffic, low utilization
GPU-based A3C

El Capitan big wall, Yosemite Valley
GA3C (INFEERENCE)

$\{a_t\}$

$\{S_t\}$

predictors

prediction queue

Agent 1

Agent 2

Agent N

Large inference batch size
Large training batch size, avoid model broadcasting
GA3C

Agent 1

Agent 2

Agent N

prediction queue

predictors

Master model

training queue

trainers

\( a_t \)

\( S_t \)

\( \{a_t\} \)

\( \{S_t\} \)

\( \Delta \pi(\cdot) \)

\( \{S_t, R_t\} \)
Learn how to balance

El Capitan big wall, Yosemite Valley
GA3C: PREDICTIONS PER SECOND (PPS)

Agent 1

prediction queue

Agent 2

Agent N

prediction queue


t

training queue

predictors

trainers

\begin{align*}
&\{a_t\} \\
&\{S_t\} \\
&\{S_t, R_t\}
\end{align*}

\Delta \pi (\cdot)
AUTOMATIC SCHEDULING
Balancing the system at run time

ATARI Boxing

ATARI Pong

\[ N_p = \# \text{ predictors}, \quad N_T = \# \text{ trainers}, \quad N_A = \# \text{ agents}, \quad \text{TPS} = \text{training per second} \]
THE ADVANTAGE OF SPEED

More frames = faster convergence

- **Optimal** $N_T = 4, N_P = 2, N_A = 32$
- **Sub-optimal** $N_T = N_P = N_A = 1$
- **Dynamic** starting from $N_T = N_P = N_A = 1$

---

**Pong**

- **94M**
- **80M**
- **64M**

**Boxing**

- **94M**
- **80M**
- **63M**
- **10M**

---

**Time (hours)**

- **01**
- **02**
- **03**

**Score**

- **20**
- **10**
- **0**
- **-10**
- **-20**

---

**Legend**

- Blue: Optimal $N_T = 4, N_P = 2, N_A = 32$
- Orange: Sub-optimal $N_T = N_P = 1, N_A = 32$
- Red: Sub-optimal $N_T = N_P = N_A = 1$
- Green: Dynamic, starting from $N_T = N_P = N_A = 1$
LARGER DNNS
For real world applications (e.g. robotics, automotive)

A3C (ATARI)

Conv 16
8x8 filters, stride 4

Conv 32
4x4 filters, stride 2

FC 256

Conv 32
8x8 filters, stride 1, 2, 3, 4

Conv 32
4x4 filters, stride 2

Conv 64
4x4 filters, stride 2

FC 256

Others (robotics)

Timothy P. Lillicrap et al., Continuous control with deep reinforcement learning, International Conference on Learning Representations, 2016.

GA3C VS. A3C*: PREDICTIONS PER SECONDS

* Our Tensor Flow implementation on a CPU

<table>
<thead>
<tr>
<th>DNN Size</th>
<th>PPS</th>
<th>Stride</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small DNN</td>
<td>A3C</td>
<td>GA3C</td>
</tr>
<tr>
<td>Large DNN, stride 4</td>
<td>4x</td>
<td>11x</td>
</tr>
<tr>
<td>Large DNN, stride 3</td>
<td>11x</td>
<td>12x</td>
</tr>
<tr>
<td>Large DNN, stride 2</td>
<td>12x</td>
<td>20x</td>
</tr>
<tr>
<td>Large DNN, stride 1</td>
<td>20x</td>
<td>45x</td>
</tr>
</tbody>
</table>
CPU & GPU UTILIZATION IN GA3C
For larger DNNs

Utilization (%)

Small DNN  Large DNN, stride 4  Large DNN, stride 3  Large DNN, stride 2  Large DNN, stride 1

CPU %

GPU %
GA3C POLICY LAG
Asynchronous playing and training
STABILITY AND CONVERGENCE SPEED
Reducing policy lag through min training batch size
Balancing computational resources, speed, and stability.

GA3C (45x faster)

GPU-based A3C

El Capitan big wall, Yosemite Valley
**RESOURCES**

**THEORY**


**CODING**

GA3C, a GPU implementation of A3C (open source at https://github.com/NVlabs/GA3C).

A general architecture to generate and consume training data.
Policy lag
Multiple GPUs
Why TensorFlow
Replay memory
...

Github: https://github.com/NVlabs/GA3C

ICLR 2017: Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU.
BACKUP SLIDES
POLICY LAG IN GA3C

\[ f_\pi (\theta) = \log \pi (a_t | s_t; \theta) (R_t - V (s_t; \theta_t)) + \beta H (\pi (s_t; \theta)) \]  \hspace{1cm} (1)

\[ f_v (\theta) = (R_t - V (s_t; \theta))^2 \]  \hspace{1cm} (2)

\[ \nabla_\theta \left[ \log \pi (a_{t-k} | s_{t-k}; \theta) (R_{t-k} - V (s_{t-k}; \theta_t)) + \beta H (\pi (s_{t-k}; \theta)) \right] \]  \hspace{1cm} (3)

\[ \nabla_\theta \left[ \log \left( \pi (a_{t-k} | s_{t-k}; \theta) + \epsilon \right) (R_{t-k} - V (s_{t-k}; \theta_t)) + \beta H (\pi (s_{t-k}; \theta)) \right] \]  \hspace{1cm} (4)

Potentially large time lag between training data generation and network update
<table>
<thead>
<tr>
<th></th>
<th>AMIDAR</th>
<th>BOXING</th>
<th>CENTIPEDE</th>
<th>NAME THIS GAME</th>
<th>PACMAN</th>
<th>PONG</th>
<th>QBERT</th>
<th>SEAQUEST</th>
<th>UP-DOWN</th>
<th>Time</th>
<th>PPS*</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human [2]</td>
<td>1676</td>
<td>10</td>
<td>10322</td>
<td>6796</td>
<td>15375</td>
<td>16</td>
<td>12085</td>
<td>40426</td>
<td>9896</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>A3C-1 [9]</td>
<td>284</td>
<td>34</td>
<td>3773</td>
<td>5614</td>
<td>3307</td>
<td>11</td>
<td>13752</td>
<td>2300</td>
<td>54525</td>
<td>1 day</td>
<td>352</td>
<td>CPU</td>
</tr>
<tr>
<td>A3C-4 [9]</td>
<td>264</td>
<td>60</td>
<td>3756</td>
<td>10476</td>
<td>654</td>
<td>6</td>
<td>15149</td>
<td>2355</td>
<td>74706</td>
<td>4 days</td>
<td>352</td>
<td>CPU</td>
</tr>
<tr>
<td>GA3C</td>
<td>218</td>
<td>92</td>
<td>7386</td>
<td>5643</td>
<td>1978</td>
<td>18</td>
<td>14966</td>
<td>1706</td>
<td>8623</td>
<td>1 day</td>
<td>1080</td>
<td>GPU</td>
</tr>
</tbody>
</table>

Table 3: Average scores on a subset of Atari games achieved by: a human player [2]; A3C after one and four days of training on a CPU [9]; and GA3C after one day of training. (*Note: predictions per second (PPS) is measured on our TensorFlow CPU and GPU implementations of A3C, while scores are from the cited publications.*)
## Training a larger DNN

<table>
<thead>
<tr>
<th>System</th>
<th>DNN</th>
<th>PPS</th>
<th>Utilization (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>A3C</td>
<td>GA3C</td>
</tr>
<tr>
<td>System I</td>
<td>small</td>
<td>352</td>
<td>1361</td>
</tr>
<tr>
<td></td>
<td>large, stride 4</td>
<td>113</td>
<td>1271</td>
</tr>
<tr>
<td></td>
<td>large, stride 3</td>
<td>97</td>
<td>1206</td>
</tr>
<tr>
<td></td>
<td>large, stride 2</td>
<td>43</td>
<td>874</td>
</tr>
<tr>
<td></td>
<td>large, stride 1</td>
<td>11</td>
<td>490</td>
</tr>
<tr>
<td>System II</td>
<td>small</td>
<td>116</td>
<td>728</td>
</tr>
<tr>
<td></td>
<td>large, stride 1</td>
<td>12</td>
<td>336</td>
</tr>
<tr>
<td>System III</td>
<td>small</td>
<td>300</td>
<td>1248</td>
</tr>
<tr>
<td></td>
<td>large, stride 1</td>
<td>38</td>
<td>256</td>
</tr>
</tbody>
</table>
BALANCING COMPUTATIONAL RESOURCES

The actors: CPU, PCI-E & GPU

Trainings per second

Training queue size

Prediction queue size

Generating and consuming data with constant queue sizes maximizes speed