Multi-GPU Accelerated Methods for Deep Reinforcement Learning

Adam Stooke, Pieter Abbeel
(UC Berkeley)

28 Mar 2018
Recent Scaling Results in Supervised Learning

ResNet-50, ImageNet training: 1 GPU $\rightarrow$ 256 GPUS $\rightarrow$ 2 weeks $\rightarrow$ 1 hour

Reinforcement Learning

Big projects (e.g. AlphaZero, Poker)
- large-scale hardware utilization
- still exceptional

More common domain: Atari
- Variety of tasks (games)
- Visual perception
- Fast emulator
- Heavily benchmarked
- Other domains share RL implementation

Common Implementations in Deep RL (Atari):
- 0 to 1 GPUs
- 1 to 10 days training

Our target implementation:
- NVIDIA DGX-1: 8 GPUs, 40 CPU cores
- Training time: less, much less!

Key Challenges:
- Smaller neural networks → lower GPU utilization
- Requires NN inference engine (unlike supervised)

Outline

Background
• Reinforcement Learning (RL)
• Deep RL Algorithms

Neural Network (NN) Inference Engine
• CPU: Simulators, GPU: NN

Multi-GPU Framework for RL
• Synchronous & Asynchronous Optimization
• Example Results

Scaling Effects on Learning
• Batch Size
• Update Rule

Recap & Outlook
Outline

Background
• Reinforcement Learning (RL)
• Deep RL Algorithms

Neural Network (NN) Inference Engine
• CPU: Simulators, GPU: NN

Multi-GPU Framework for RL
• Synchronous & Asynchronous Optimization
• Example Results

Scaling Effects on Learning
• Batch Size
• Update Rule

Recap & Outlook
Background: Reinforcement Learning

**Learning Agent Maximizes:**

$$\mathbb{E}_{s, a \sim \pi}[R_t]$$

\[ R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \]

\( R \): discounted return

\( \gamma \): discount (e.g. 0.99)
Background: Deep RL Algorithms

Policy Gradient Methods (e.g. A3C)
Stochastic Policy: \( \pi(a|s; \theta) \)
Requires on-policy training data

\[
\text{while learning do:} \\
\quad \text{for } t = 1 \text{ to sample_horizon do:} \\
\quad \quad a_t \sim \pi(s_t; \theta) \\
\quad \quad s_{t+1}, r_t = \text{env. step}(a_t) \\
\quad \quad \text{data.append}(s_t, a_t, r_t, s_{t+1}) \\
\quad \quad \text{update } \theta \text{ using data} \\
\quad \quad \text{clear data} \\
\]

\[
\Delta \theta \sim \frac{1}{B} \sum_{i=1}^{B} \nabla_{\theta} \text{Loss(data}_i) \\
\theta \leftarrow \theta + \Delta \theta \\
\]

Common configurations:
- A few parallel sampling threads (CPU inference)
- CPU or 1-GPU training
- 1 day to fully train

Q-Learning Methods (e.g. DQN)
Q-function: \( Q(s, a; \theta) \)
greedy policy: \( a^* = \arg \max_a Q(s, a; \theta) \)

\[
\text{while learning do:} \\
\quad \text{for } t = 1 \text{ to sample_horizon do:} \\
\quad \quad a_t \sim \pi_Q(s_t; \theta) \\
\quad \quad s_{t+1}, r_t = \text{env. step}(a_t) \\
\quad \quad \text{replay.append}(s_t, a_t, r_t, s_{t+1}) \\
\quad \quad \text{replay.remove_oldest()} \\
\quad \quad \text{update } \theta \text{ using random data from replay} \\
\]

Common configurations:
- 1 sampling thread (CPU inference)
- 1-GPU training
- 10 days to fully train

Outline

Background
• Reinforcement Learning (RL)
• Deep RL Algorithms

Neural Network (NN) Inference Engine
• CPU: Simulators, GPU: NN

Multi-GPU Framework for RL
• Synchronous & Asynchronous Optimization
• Example Results

Scaling Effects on Learning
• Batch Size
• Update Rule

Recap & Outlook
Parallel Simulation, GPU Inference

GPU prefers batched processing ... Approach: *many simulators per GPU, synchronized*
Alternating Sampling

**Result:** Higher hardware utilization $\rightarrow$ faster sampling
Sampling Speed

Single-GPU, Multi-CPU Sampling Speed Measurement:
(small conv-net playing game “Breakout”)

Results:
• NN inference time completely hidden for 1 CPU core running sim
• Increased simulator count mitigates synchronization & latency losses
• Total DGX-1 throughput: 32,000 samples per second (2,000 x real-time)

Question:
Can RL algorithms use many sims?
Outline

Background
• Reinforcement Learning (RL)
• Deep RL Algorithms

Neural Network (NN) Inference Engine
• CPU: Simulators, GPU: NN

Multi-GPU Framework for RL
• Synchronous & Asynchronous Optimization
• Example Results

Scaling Effects on Learning
• Batch Size
• Update Rule

Recap & Outlook
Multi-GPU Framework

Organizing Principle: GPUs used homogeneously – everyone samples and optimizes

**Synchronous Optimization**
- GPUs hold identical NN parameter values

\[
g ← \nabla_\theta (local\_data) \quad \text{« store on GPU} \\
g ← \text{all_reduce}(g) \times 1/n \quad \text{« NCCL} \\
\theta ← \text{update\_rule}(g, \theta)
\]

**Asynchronous Optimization**
- GPUs hold different NN parameter values, \( \theta \)
- Central NN parameters on CPU, \( \tilde{\theta} \)
- Each GPU applies update to \( \tilde{\theta} \) when ready

Write-protect & mitigate lock contention:

\[
g ← \nabla_\theta (local\_data) \quad \text{« store on GPU} \\
\text{for } c = 1 \text{ to } \text{num\_chunks} \text{ do:} \\
\quad \text{acquire } \text{Lock}_c \\
\quad \theta_c ← \tilde{\theta}_c \quad \text{« pull to GPU} \\
\quad \theta_c ← \text{update\_rule}(g_c, \theta_c) \\
\quad \tilde{\theta}_c ← \theta_c \quad \text{« write to CPU} \\
\quad \text{release } \text{Lock}_c
\]

Question:
Can Deep RL Algorithms scale well to DGX-1?
- Preserve final game performance
- Preserve sample complexity

Batch Size ↑  →  GPU Utilization ↑, Communication ↓
Scaling: Policy Gradient Methods

Sample Complexity with increasing parallelization scale:

(pictured: asynchronous optimization, 1:1 learner:GPU)


Result: sample complexity preserved as compute resources increased (up to at least 8 GPUs)
Scaling: Policy Gradient Methods

Wall-Clock Time (minutes) using full DGX-1 (8-GPU, 40-CPU):

(pictured: two RL algorithms, synchronous and asynchronous versions each)

Result: 35x speedup vs typical, 16-CPU implementation (15 hours → 20 minutes)
### Scaling: Policy Gradient Methods

#### Training Time Scaling (200M frames)

<table>
<thead>
<tr>
<th>Method</th>
<th>1 GPU</th>
<th>2 GPUs</th>
<th>4 GPUs</th>
<th>8 GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2C</td>
<td>3.8</td>
<td>2.2</td>
<td>1.2</td>
<td>0.59</td>
</tr>
<tr>
<td>A3C</td>
<td>3.8</td>
<td>2.4</td>
<td>1.3</td>
<td>0.65</td>
</tr>
<tr>
<td>PPO</td>
<td>4.4</td>
<td>2.6</td>
<td>1.5</td>
<td>1.1</td>
</tr>
<tr>
<td>APPO</td>
<td>4.4</td>
<td>2.8</td>
<td>1.5</td>
<td>0.71</td>
</tr>
</tbody>
</table>

**Result**

8 GPUs vs 1: >6x speedup
Sample Complexity with increasing training batch size (standard=32):
(total use of training data held constant)

Categorical-DQN

Result: sample complexity preserved up to batch size 2,048
Scaling: DQN + Variants

Results:

- **1 GPU**: 20x speedup
- **8 GPU, Cat-DQN**: 120x speedup (10 days → 2 hours)
Gameplay while Learning

- Fully Random Initial
- “Beginner” 6M actions played
- “Advanced” 60M actions played
Outline

Background
- Reinforcement Learning (RL)
- Deep RL Algorithms

Neural Network (NN) Inference Engine
- CPU: Simulators, GPU: NN

Multi-GPU Framework for RL
- Synchronous & Asynchronous Optimization
- Example Results

Scaling Effects on Learning
- Batch Size
- Update Rule

Recap & Outlook
Batch Size Limits

One culprit: *Optimization difficulties* (regarding neural network weights)

Evidence: Secondary-Learner Experiment (DQN)

- 2\textsuperscript{nd} learner trains simultaneously, using only replay buffer of 1\textsuperscript{st} learner (two NNs)
- 1\textsuperscript{st} Learner : 2\textsuperscript{nd} Learner  Large Batch : Small Batch  --  OK
- 1\textsuperscript{st} Learner : 2\textsuperscript{nd} Learner  Small Batch : Large Batch  --  learning fails!
- *optimization speed* had greater effect than *exploration*

![Graph 1](image1)

![Graph 2](image2)
Update Rule: RMSProp

Result: RMSProp -- scale learning rate (with batch size) to maintain $\theta$ norm; Adam – constant l.r.
Outline

Background
• Reinforcement Learning (RL)
• Deep RL Algorithms

Neural Network (NN) Inference Engine
• CPU: Simulators, GPU: NN

Multi-GPU Framework for RL
• Synchronous & Asynchronous Optimization
• Example Results

Scaling Effects on Learning
• Batch Size
• Update Rule

Recap & Outlook
Contributions Summary

- Unified parallelization & acceleration framework for deep RL
- Efficient NN inference on GPU
- Larger training batches for efficiency
- Asynchronous optimization with GPUs

- Unprecedented wall-clock times without loss in sample complexity
- Promising scaling to:
  - more sophisticated NN agents
  - more challenging tasks
Open Questions / Future Directions

• Limits to batch size?
  • Exploration (state space; parameter space)
  • Optimization
    • Agility/speed in learning
    • Gradient information saturation / noise

• Speedup NN via reduced-precision arithmetic
  • Previously inhibited (in RL) by CPU-based inference
  • Volta GPUs: >2x speedup from FP32 to FP16

• Apply this framework to support other research!
  adam.stooke@berkeley.edu  (code release pending)
More Work Ahead

Game: Riverraid
Agent: well-trained

Can you spot the flaw in the strategy?