Gradient Descent Algorithms on Modern Hardware

Yujing Ma, Florin Rusu, Martin Torres
University of California Merced

Motivation

- 47x higher than CPU on deep learning inference
  [https://www.nvidia.com/en-us/data-center/tesla-v100/]
- In open-source TensorFlow, simple placer: GPU are preferred over CPU.
- GPU is not always superior and has lower speedup than the theoretical value. (Lee, Victor W., et al. “Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU.”)

Problem & Contribution

- In-depth comparison on which GD algorithm performs better on which architecture and with what type of data
  1. GD algorithm (model update): sync, async
  2. computing architecture: CPU, GPU
  3. data sparsity: dense, sparse

Gradient Descent

- Batch Gradient Descent (Parallel BGD)
  1. Compute gradient: \( \text{in parallel} \) for \( i = 1 \) to \( N \) do \( \frac{\partial}{\partial \theta} J(\theta) \) \( (\theta; x_i, y_i) \)
  2. Update model: \( \theta \leftarrow \theta - \alpha \cdot \nabla J \)

- Mini-Batch Gradient Descent (Hogbatch)
  for each batch: \( \text{in parallel} \) for \( i = 1 \) to \( N \) do \( \frac{\partial}{\partial \theta} J(\theta) \) \( (\theta; x_i, y_i) \)
  2. Update model: \( \theta \leftarrow \theta - \alpha \cdot \nabla J \)

- Stochastic Gradient Descent (Hogwild)
  for \( i = 1 \) to \( N \) do \( \frac{\partial}{\partial \theta} J(\theta) \) \( (\theta; x_i, y_i) \)
  2. Compute gradient estimate: \( \frac{1}{m} \sum_{i=1}^{m} \nabla J(\theta; x_i, y_i) \)
  3. Update model: \( \theta \leftarrow \theta - \alpha \cdot \nabla J \)

Datasets

<table>
<thead>
<tr>
<th>dataset</th>
<th>#examples</th>
<th>#features</th>
<th>#NNZ/example (avg)</th>
<th>sparse size</th>
<th>dense size</th>
</tr>
</thead>
<tbody>
<tr>
<td>covtype</td>
<td>581,012</td>
<td>54</td>
<td>0 to 114 (12)</td>
<td>4.4 MB</td>
<td>155 MB</td>
</tr>
<tr>
<td>w8a</td>
<td>64,700</td>
<td>300</td>
<td>0 to 114 (12)</td>
<td>4.4 MB</td>
<td>155 MB</td>
</tr>
<tr>
<td>rcv1</td>
<td>677,399</td>
<td>47,236</td>
<td>4 to 1,224 (73)</td>
<td>1.2 GB</td>
<td>256 GB</td>
</tr>
<tr>
<td>news</td>
<td>19,996</td>
<td>1,355,191</td>
<td>1 to 16,423 (455)</td>
<td>134 MB</td>
<td>217 GB</td>
</tr>
</tbody>
</table>

* compressed sparse row (CSR) used for sparse representation

Synchronous GD

- news, logistic regression

Asynchronous GD

- asynchronous GD performance to 1\% error

Sync GD vs. Async GD

CPU vs. GPU

On sync GD, optimized GPU kernels provide better hardware efficiency than TensorFlow. On async GD, the performance of GPU is worse than CPU about 10 times.

CPU is faster than GPU in time to convergence. On dense and low-dimensional data, cpu-seq is faster. On sparse data, cpu-par dominates.

System Specification

- CPU: 2x Intel Xeon E5-2660 v4
  2*14 cores, 2*28 threads, max turbo frequency: 3.20 GHz
- GPU: NVIDIA Tesla K80
  2*2,496 cores (13 MP * 192 cores/MP), boost clock: 876 MHz

Asynchronous GD (cont.)

- covtype, logistic regression

Hybrid GD

- covtype, multilayer perceptron

cpu-hb

gpu-mgd

hybrid-average

hybrid-merge

GPU has better hardware efficiency and faster time to convergence. When the dataset is more sparse, the speedup of gpu over cpu-par is higher.