LBANN
Livermore Big Artificial Neural Network

CSE 5194.01
Autumn ‘19

Quentin Anthony
The Ohio State University
E-mail: Anthony.301@osu.edu
Outline

• Introduction

• LBANN Design

• Experiments
Introduction: The Need for Large Networks

- Bias = how poorly the model “listens” to training data
- Variance = sensitivity to training data
- Larger models allow low bias
- Larger, complex datasets allow low variance

- Which is high variance?
- Which is high bias?
Introduction: The Need for HPC

- Larger datasets
- Larger networks
- Need for speed/accuracy

Examples:
- Logistic Regression – How long to train ResNet-50 on GTX 1080?
- How about 2,048 Tesla V100s?


Introduction: LBANN is Born!

LBANN: Livermore Big Artificial Neural Network

• Built for deep unsupervised feature learning on massive, unlabeled datasets

• Highly optimized for large neural networks on HPC systems

• Completely open-source: https://github.com/LLNL/lbann
Outline

• Introduction

• LBANN Design

• Experiments
LBANN Design: Model/Data Parallelism

• LBANN core data structure = distributed matrices
  – Enables splitting a model across HPC nodes
  – Built on Elemental, an open-source linear algebra framework on C++/MPI

• New HPC systems have node-local NVRAM
  – LBANN uses NVRAM to stage files from parallel file system

What type of parallelism is depicted here?
Figure 8: # parallel I/O streams fetching each mini-batch versus the total I/O time to load all images into memory. All tests were executed on 32 nodes, a hidden layer with 50K neurons, and 128 image mini-batch.
Aside: Autoencoders

• The choice of input features is as important (sometimes more important!) as the learning algorithm used
• Autoencoders learn great feature representations of data, especially for images (e.g. edges, curves, and faces)
• Autoencoders first define a reconstruction $r(x) = g(h(x))$
  – Input: $x$, Encoder: $h(x)$, Decoder: $g(x)$
  – $r(x)$ is the output of a one-hidden-layer network, $h(x)$ is the output of the hidden layer
  – Back-propagate to minimize loss: $L(r(x), x)) = ||r(x) - x||^2$
• Benefits: Remove labelling requirement for large datasets, human-readable features, efficient back-propagation
• Is this supervised or unsupervised learning?
Aside: Autoencoders

- Each square shows the image that maximally activates a hidden unit
- This autoencoder is learning edge features!

- Key Idea: The autoencoder is trying to learn the **bare minimum** information required to describe dataset. The bare minimum info are the features!
Aside: Autoencoders

Example: Stacked autoencoders for image classification

28 x 28
Aside: Autoencoders

Example: Stacked autoencoders for image classification

![Diagram of a stacked autoencoder](image)

28 x 28
Aside: Autoencoders

Example: Stacked autoencoders for image classification

28 x 28
Aside: Autoencoders

Example: Stacked autoencoders for image classification

28 x 28

Hidden layer 1

Hidden layer 2
Aside: Autoencoders

Example: Stacked autoencoders for image classification

28 x 28
Outline

• Introduction

• LBANN Design

• Experiments
Experiments: Setup

• To test LBANN scaling, train a single layer autoencoder using ImageNet 2012.
• Measure clock time required for a fixed amount of training
  – LLNL Catalyst HPC System
    • 324 Nodes
    • 24 Xeon EP X5660 Cores
    • 128 GB DRAM
    • 800 GB node-local NVRAM
• Network topology = ~197K \rightarrow X \rightarrow ~197K
  – X = \# neurons in fully connected hidden layer
• Tested on 50K, 100K, 400K neuron models (huge!)
Experiments: Autoencoders

- Autoencoder learning quality is measured by the reconstruction error
  - Compares the original image with the autoencoder layer’s output
- More training epochs leads to better performance (Figure 4)
- Larger hidden layers give lower errors! (Figures 3-4)
Experiments: Model Parallelism

• As a single model is distributed over multiple nodes, there are two ways to improve performance
  – Solve a fixed-sized model faster (strong scaling) (Figure 5)
  – Solve a larger problem in a “fixed” amount of time (weak scaling) (Figure 6)
Experiments: Data Parallelism

• Two techniques to provide data parallelism
  – Large mini-batches
    • Allows more data to be processed in parallel (less blocked matrix multiplications!)
    • Reduces number of synchronization steps (reduce communication!)
  – Parallel data movement
• In LBANN, each node handles a single mini-batch
  – Read from NVRAM or PFS, copy into local memory, distribute to nodes
• Most time-consuming part? Reading mini-batch from NVRAM/PFS
  – LBANN uses as many MPI ranks as possible to read in mini-batches concurrently and then distribute them round-robin
Experiments: Data Parallelism

**Figure 7:** Strong scaling mini-batch size to exploit data parallelism (50K neuron hidden layer).

**Figure 8:** # parallel I/O streams fetching each mini-batch versus the total I/O time to load all images into memory. All tests were executed on 32 nodes, a hidden layer with 50K neurons, and 128 image mini-batch.
Experiments: Tuning Elemental Library

- LBANN is built on a linear algebra software library: Elemental
  - Test 1: Vary the block size of the distributed matrix within Elemental (Figure 9)
  - Test 2: Vary the number of ranks per node (Figure 10)
- Both used a network with 400K hidden units, and a batch size of 256 and 128 images, respectively

Figure 9: Tuning Elemental’s distributed block size.

Figure 10: MPI ranks per node versus the number of threads available to Intel BLAS library.
Experiments: NOWLAB

Performance Comparison

Model: ResNet-50
4 GPUs per node
Metric: Images per sec

<table>
<thead>
<tr>
<th>Batch Size (per device)</th>
<th>GPU2</th>
<th>GPU4</th>
<th>GPU8</th>
<th>GPU16</th>
<th>GPU32</th>
<th>GPU64</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MV2-GDR</td>
<td>NCCL</td>
<td>MV2-GDR</td>
<td>NCCL</td>
<td>MV2-GDR</td>
<td>NCCL</td>
</tr>
<tr>
<td>1</td>
<td>18.61</td>
<td>27.91</td>
<td>55.98</td>
<td>48.67</td>
<td>105.46</td>
<td>80.36</td>
</tr>
<tr>
<td>2</td>
<td>38.69</td>
<td>55.15</td>
<td>111.39</td>
<td>88.89</td>
<td>200.99</td>
<td>148.59</td>
</tr>
<tr>
<td>4</td>
<td>84.92</td>
<td>88.52</td>
<td>209.22</td>
<td>193.47</td>
<td>393.74</td>
<td>317.15</td>
</tr>
<tr>
<td>8</td>
<td>103.80</td>
<td>103.75</td>
<td>231.81</td>
<td>235.45</td>
<td>462.39</td>
<td>444.67</td>
</tr>
<tr>
<td>16</td>
<td>116.46</td>
<td>111.82</td>
<td>240.75</td>
<td>238.69</td>
<td>488.10</td>
<td>463.57</td>
</tr>
<tr>
<td>32</td>
<td>120.97</td>
<td>114.36</td>
<td>241.30</td>
<td>237.91</td>
<td>496.70</td>
<td>470.33</td>
</tr>
</tbody>
</table>

- We performed our own evaluation of LBANN built on MVAPICH2-GPUDIRECT and NCCL communication runtimes
- Notice that increased batch size gives diminishing returns!
Conclusions

- LBANN is optimized for training huge DNNs on massive datasets
- LBANN targets HPC systems
- LBANN is built on the distributed linear algebra library Elemental
- Support for scalable model and data parallelism
Thank You!

Anthony.301@osu.edu

Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

The MVAPICH2 Project
http://mvapich.cse.ohio-state.edu/

The High-Performance Deep Learning Project
http://hidl.cse.ohio-state.edu/