T. HOEFLER, T. BEN-NUN

Optimizing and Benchmarking Large-Scale Deep Learning
Machine Learning Day at ISC’19, Frankfurt, Germany

WITH CONTRIBUTIONS FROM DAN ALISTARH, YOSUKE OYAMA, CEDRIC RENGLI, AND OTHERS AT SPCL, IST AUSTRIA, AND TOKYO TECH
A brief theory of supervised deep learning (minibatch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x): X \rightarrow Y$

network structure (fixed)
weights $w$ (learned)

$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$

$f(x) = f_n \left( f_{n-1} \left( f_{n-2} \left( \ldots f_1(x) \ldots \right) \right) \right)$

$\ell_{sq}(w, x) = (f(x) - l(x))^2$

$\ell_{0-1}(w, x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$

$\ell_{ce}(w, x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$
A brief theory of supervised deep learning (minibatch SGD)

\[ f(x) = f_n \left( f_{n-1} \left( \ldots f_1(x) \ldots \right) \right) \]

\[ \ell_{ce}(w, x) = - \sum_{i} l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}} \]

\[ w^* = \arg\min_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim D} [\ell(w, x)] \]

Deep Learning is Supercomputing!

Cat 0.54  
Dog 0.28  
Airplane 0.07  
Horse 0.33  
Banana 0.02  
Truck 0.02  

\[ \geq \text{TBs of random access} \]

100MiB-26GiB and beyond

\[ \geq 22k \text{-} \text{millions} \]

\[ \geq 100 \text{MiB} - 26 \text{GiB} \text{ and beyond} \]

\[ \geq 22k \text{ - } \text{millions} \]
Computational Principles

Operators
Minibatch Stochastic Gradient Descent (SGD)

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019
In cuDNN there are ~16 convolution implementations.
Performance depends on temporary memory (workspace) size.
Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms.

How to choose microbatch sizes and algorithms?

Microbatching Strategy

- None (undivided)
- Powers-of-two only
- Any (unrestricted)

Fast (up to 4.54x faster on DeepBench)

Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis

https://www.arxiv.org/abs/1802.09941

Deep Neural Networks (DNNs) are becoming an important tool in modern computing applications. Accelerating their training in a major challenge and techniques range from distributed algorithms to low-level circuit design. In this survey, we describe the problem from a theoretical perspective, followed by approaches for its parallelization. Specifically, we present trends in DNN architectures and the resulting implications on parallelization strategies. We discuss the different types of concurrency in DNNs: synchronous and asynchronous stochastic gradient descent; distributed system architecture; communication schemes; and performance modeling. Based on these approaches, we extrapolate potential directions for parallelism in deep learning.

CCS Concepts: • General and reference → Surveys and overviews; • Computing methodologies → Neural networks; Distributed computing methodologies; Parallel computing methodologies; Machine learning;

ACM Reference Format:

1 INTRODUCTION

Machine Learning, and in particular Deep Learning [LeCun et al. 2015], is a field that is rapidly taking over a variety of aspects in our daily lives. In the core of deep learning lies the Deep Neural Network (DNN), a construct inspired by the interconnected nature of the human brain. Trained properly, the expressiveness of DNNs provides accurate solutions for problems previously thought to be unsolvable, simply by observing large amounts of data. Deep learning has been successfully implemented for a plethora of subjects, ranging from image classification [He et al. 2015], through speech recognition [Amodei et al. 2016] and medical diagnosis [Urejan et al. 2013], to autonomous driving [Bojarski et al. 2016] and defeating human players in complex games [Silver et al. 2018].
Model parallelism – limited by network size

- Parameters can be distributed across processors
- Mini-batch has to be copied to all processors
- Backpropagation requires complex communication every layer

Pipeline parallelism – limited by network size

- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time “Bubble”

G. Blelloch and C.R. Rosenberg: Network Learning on the Connection Machine, IJCAI’87
Data parallelism – limited by batch-size

- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

Hybrid parallelism

- Layers/parameters can be distributed across processors
- Can distribute minibatch
- Often specific to layer-types (e.g., distribute fc layers but handle conv layers data-parallel)
  - Enables arbitrary combinations of data, model, and pipeline parallelism – very powerful!

J. Dean et al.: Large scale distributed deep networks, NIPS’12.
T. Ben-Nun, T. Hoeffer: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019
Updating parameters in distributed data parallelism

Parameter server (sharded) $w' = u(w, \nabla w)$

$T = 2L + 2P \gamma m / s G$

Collective operations
- Topologies
- Neighborhood collectives
- RMA?

$T = 2L \log_2 P + 2\gamma m G (P - 1) / P$

Hierarchical Parameter Server
S. Gupta et al.: Model Accuracy and Runtime Tradeoff in Distributed Deep Learning: A Systematic Study. ICDM’16

Adaptive Minibatch Size
S. L. Smith et al.: Don’t Decay the Learning Rate, Increase the Batch Size, arXiv 2017
Communication optimizations

- Different options how to optimize updates
  - Send $\nabla w$, receive $w$
  - Send FC factors $(o_{l-1}, o_l)$, compute $\nabla w$ on parameter server
    - *Broadcast factors to not receive full $w*$
  - Use lossy compression when sending, accumulate error locally!

- Quantization
  - Quantize weight updates and potentially weights
  - Main trick is stochastic rounding [1] – expectation is more accurate
    - *Enables low precision (half, quarter) to become standard*
  - TernGrad - ternary weights [2], 1-bit SGD [3], ...

- Sparsification
  - Do not send small weight updates or only send top-k [4]
    - *Accumulate omitted gradients locally*

---

[3] F. Seide et al. 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs, In Interspeech 2014

source: ai.intel.com
SparCML – Quantized sparse allreduce for decentral updates

\[ \nabla w_1 + \nabla w_2 + \nabla w_3 + \nabla w_4 \]

Microsoft Speech Production Workload Results – 2 weeks → 2 days!

<table>
<thead>
<tr>
<th>System</th>
<th>Dataset</th>
<th>Model</th>
<th># of nodes</th>
<th>Algorithm</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piz Daint</td>
<td>ImageNet</td>
<td>VGG19</td>
<td>8</td>
<td>Q4</td>
<td>1.55 (3.31)</td>
</tr>
<tr>
<td>Piz Daint</td>
<td>ImageNet</td>
<td>AlexNet</td>
<td>16</td>
<td>Q4</td>
<td>1.30 (1.36)</td>
</tr>
<tr>
<td>Piz Daint</td>
<td>MNIST</td>
<td>MLP</td>
<td>8</td>
<td>Top16_Q4</td>
<td>3.65 (4.53)</td>
</tr>
<tr>
<td>EC2</td>
<td>MNIST</td>
<td>MLP</td>
<td>8</td>
<td>Top16_Q4</td>
<td>19.12 (22.97)</td>
</tr>
</tbody>
</table>

C. Renggli et al. SparCML: High-Performance Sparse Communication for Machine Learning, arXiv 2018
Reproducing and Benchmarking Deep Learning

- **End result – generalization**

---

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Focus</th>
<th>Metrics</th>
<th>Criteria</th>
<th>Customizability</th>
<th>DL Workloads</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Perf Con Acc Tim Cos Ene Util Mem Tput Brk Sca Com TTA FTA Lat Clo Ope Inf Ops Img Obj Spe Txt RL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kaggle [21]</td>
<td>![thumbs_up] ![thumbs_down] ![slashes]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ImageNet [13]</td>
<td>![thumbs_up] ![slashes]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLLP [30]</td>
<td>![thumbs_up] ![slashes]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep500</td>
<td>![thumbs_up] ![slashes]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE II**: An overview of available DL benchmarks, focusing on the offered functionalities. **Perf**: Performance, **Con**: Convergence, **Acc**: Accuracy, **Tim**: Time, **Cos**: Cost, **Ene**: Energy, **Util**: Utilization, **Mem**: Memory Footprint, **Tput**: Throughput (Samples per Second), **Brk**: Timing Breakdown, **Sca**: Strong Scaling, **Com**: Communication and Load Balancing, **TTA**: Time to Accuracy, **FTA**: Final Test Accuracy, **Lat**: Latency (Inference), **Clo**: Closed (Fixed) Model Contests, **Ope**: Open Model Contests, **Inf**: Fixed Infrastructure for Benchmarking, **Ops**: Operator Benchmarks, **Img**: Image Processing, **Obj**: Object Detection and Localization, **Spe**: Speech Recognition, **Txt**: Text Processing and Machine Translation, **RL**: Reinforcement Learning Problems, ![thumbs_up]: A given benchmark does offer the feature, ![thumbs_down]: Planned benchmark feature.

---

- **Sample throughput**

---

### Existing Deep Learning Frameworks

<table>
<thead>
<tr>
<th>System</th>
<th>Operators</th>
<th>Networks</th>
<th>Training</th>
<th>Dist. Training</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sta</td>
<td>Cus</td>
<td>Def</td>
<td>Cnt</td>
</tr>
<tr>
<td>(L) cuDNN</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Red.png" alt="Red" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Red.png" alt="Red" /></td>
</tr>
<tr>
<td>(L) MKL-DNN</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Red.png" alt="Red" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(F) TensorFlow [1]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(F) Caffe, Caffe2 [21]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(F) [Py]Torch [10, 35]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(F) MXNet [6]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(F) CNTK [48]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(F) Theano [4]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(F) Chainer[MN] [44]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(F) Darknet [38]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(F) DL4j [43]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(F) DSSTNE</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(F) PaddlePaddle</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(F) TVM [7]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(E) Keras [8]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(E) Horovod [42]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(E) TensorLayer [14]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(E) Lasagne</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
<tr>
<td>(E) TFlearn [11]</td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
<td><img src="Green.png" alt="Green" /></td>
</tr>
</tbody>
</table>

- Customizing operators relies on framework
- Network representation
- Dataset representation
- Training algorithm
- Distributed training (e.g., asynchronous SGD)

Deep500

- Deep learning **meta-framework**: a framework for frameworks to reside in

Deep500

- Deep learning **meta-framework**: a framework for frameworks to reside in

```python
inference()
inference_and_backprop()
add_node()
add_edge()
remove_...
```

Deep500

- Deep learning meta-framework: a framework for frameworks to reside in

```
train()

Sampler

Dataset

HDD

Optimizer

minimize() step()

next()

get(i)

Training Runner (Trainer)

Executor

Network

Operators

CustomOp

ONNX
```

Deep500

- Deep learning meta-framework: a framework for frameworks to reside in

**Metrics**

For Benchmarking: Recipes

Fixed definitions + mutable definitions + acceptable metric set = Recipe
For Benchmarking: Recipes

Fixed definitions + mutable definitions + acceptable metric set = Recipe

```python

"""
A recipe for running the CIFAR-10 dataset with ResNet-44 and a momentum optimizer, with metrics for final test accuracy. """

import deep500 as d5
from recipes.recipe import run_recipe

# Using PyTorch as the framework
import deep500.Frameworks.pytorch as d5fw

# Fixed Components
FIXED = {
    'model': 'resnet',
    'model_kwargs': dict(depth=44),
    'dataset': 'cifar10',
    'train_sampler': d5.ShuffleSampler,
    'epochs': 1
}

# Mutable Components
MUTABLE = {
    'batch_size': 64,
    'executor': d5fw.from_model,
    'executor_kwargs': dict(device=d5.GPUDevice()),
    'optimizer': d5fw.MomentumOptimizer,
    'optimizer_args': (0.1, 0.9),
}

# Acceptable Metrics
METRICS = [
    (d5.TestAccuracy(), 0.9)
]

if __name__ == '__main__':
    run_recipe(FIXED, MUTABLE, METRICS) or exit(1)
```

https://github.com/deep500/deep500/blob/master/recipes/cifar10_resnet44.py

For Customizing: New Operator

```python
class IPowOp(CustomPythonOp):
    def __init__(self, power):
        super(IPowOp, self).__init__()
        self.power = power
        assert int(power) == power  # integral

    def forward(self, inputs):
        return inputs[0] ** self.power

    def backward(self, grads, fwd_inputs, fwd_outputs):
        return (grads[0] * self.power *
                (fwd_inputs[0] ** (self.power - 1)))
```

```cpp
template<typename T>
class ipowop : public deep500::CustomOperator {
    protected:
        int m_len;
    public:
        ipowop(int len) : m_len(len) {}  
        virtual ~ipowop() {}  

        void forward(const T *input, T *output) {
            #pragma omp parallel for
            for (int i = 0; i < m_len; ++i)
                output[i] = std::pow(input[i], DPOWER);
        }

        void backward(const T *nextop_grad,
                      const T *fwd_input_tensor,
                      const T *fwd_output_tensor,
                      T *input_tensor_grad) {
            #pragma omp parallel for
            for (int i = 0; i < m_len; ++i) {
                input_tensor_grad[i] = nextop_grad[i] * DPOWER *
                                       std::pow(fwd_input_tensor[i], DPOWER - 1);
            }
        }
};
```

For Customizing: Distributed Optimization

class ConsistentNeighbors(DistributedOptimizer):
    # Follows communication scheme from https://arxiv.org/pdf/1705.09056.pdf

def step(self, inputs):
    self.base_optimizer.new_input()
    for param in self.network.get_params():
        self.base_optimizer.prepare_param(param)
    output = self.executor.inference_and_backprop(inputs, self.base_optimizer.loss)
    gradients = self.network.gradient(self.base_optimizer.loss)
    for param_name, grad_name in gradients:
        param, grad = self.network.fetch_tensors([[param_name, grad_name]])
        grad = self.communication.reduce_from_neighbors(grad) / 3
        param = self.base_optimizer.update_rule(grad, param, param_name)
        self.network.feed_tensor(param_name, param)
    return output
HPC for Deep Learning – Summary

- A supercomputing problem - amenable to established tools and tricks from HPC
- Concurrency is easy to attain, hard to program beyond data-parallelism
- Main bottleneck in distributed is communication – reduction by using the robustness of SGD
- Co-design is prevalent
- Very different environment from traditional HPC
  - Trade-off accuracy for performance!
- Main objective is generalization
  - Performance-centric view in HPC can be harmful for accuracy

https://www.arxiv.org/abs/1802.09941

Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis

TAL BEN-NUN* and FOREST HOFLEER, ETH Zürich

---

Deep Neural Networks (DNNs) are becoming an important tool in modern computing applications. Accelerating their training is a major challenge and techniques range from distributed algorithms to low-level circuit design. In this survey, we describe the problem from a theoretical perspective, followed by approaches for its parallelization. Specifically, we present trends in DNN architectures and the resulting implications on parallelization strategies. We discuss the different types of concurrency in DNNs: synchronous and asynchronous stochastic gradient descent distributed system architectures, communication schemes, and performance modeling. Based on these approaches, we enumerate potential directions for parallelism in deep learning.

CCS Concepts: → General and reference → Theory and systems; → Computing methodologies → Neural networks; → Distributed computing methodologies → Parallel computing methodologies; → Machine learning.

Additional Key Words and Phrases: Deep Learning, Distributed Computing, Parallel Algorithms.

ACM Reference Format:
Next steps – Community!

- More recipes
- More datasets (scientific computing)
- Use concepts from HPC to improve ML
  - Better formats
  - Communication schemes
- Implement reproducible methods
- Metrics and aggregate scores

https://www.deep500.org/
https://www.github.com/deep500/deep500