An Introduction to Apache SINGA V1

Wei Wang & Wenfeng Wu

singa.apache.org
On behalf of SINGA Team
Deep Learning

- **Is effective in**
  - Computer vision [1,2,3,4]
  - NLP [5]
  - Speech [6]
  - Games [7]

- **Is big in terms of**
  - Training dataset size, >1 million images[1], >10k hours audio[6]
  - Model parameter size, >100 Million [2]

- **Is Difficult to**
  - Understand (its effectiveness)
  - Tune the hyper-parameters)
  - Optimize (the training speed and memory)

---

Outline

Background
- DL models
- Training

Optimize DL Systems
- Efficiency
- Memory

SINGA V1
- Design
- Examples
Onset of Deep Learning

Multimedia Data

Social Media

Audio

E-commerce

Image/video

Health-care

Text

VocalIQ (acquired by Apple)

Madbits (acquired by Twitter)

Perceptio (acquired by Apple)

LookFlow (acquired by Yahoo! Flickr)

Deepomatic (e-commerce product search)

Descartes Labs (satellite images)

Clarifai (tagging)

Ldibon

ParallelDots

AlchemyAPI (acquired by IBM)

Semantria (NLP tasks >10 languages)
Deep Learning Models

- **Feed-forward neural nets, acyclic directed graphs**
  - multi-layer perceptron (MLP)
  - convolution neural network (CNN)

Left: MLP with 2 hidden layers; Right: feature transformation in one hidden layer. (source http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/)

Deep Learning Models

- **Feed-forward neural nets, acyclic directed graphs**
  - multi-layer perceptron (MLP)
  - convolution neural network (CNN)

- **Recurrent neural nets, e.g., LSTM, GRU**
  - can be converted into a feed-forward nets.
  - Energy models (shallow model)
  - Reinforcement learning + deep learning

unrolling a 2 layer RNN into 4 pieces (time-steps)
Model Training

- **Objective**\(^1\) \(\min_{\theta} L(f(x), y)\)
  - Loss function \(L()\) measures the discrepancy between the ground-truth and the prediction
  - \(\theta\) denotes all model parameters.
  - Given a dataset of a set of \(\{<x, y>\}\), the objective is to minimize the averaged loss (the following slides use a single pair for illustration).

\(^1\)Usually, there would be a regularization term in the objective for preventing overfitting. It is omitted here. More details could be found online at http://cs231n.github.io/neural-networks-2/
Model Training

- **Objective** \( \min_{\theta} L(f(x), y) \)
  - Loss function \( L() \) measures the discrepancy between the ground-truth and the prediction
  - \( \theta \) denotes all model parameters.

- **Algorithm: Stochastic Gradient Descent (SGD)**
  - Compute the gradients of \( L \) w.r.t model parameters via back-propagation (BP) [11]. Operations of each layer are computation intensive:
    - Neural net operations, e.g., convolution, pooling, normalization
    - Linear algebra operations, including matrix multiplication, e.g., \( x^*W+b \) (\( W \) could be larger than 4K * 4K), sigmoid.

## Model Training

- The training procedure is time consuming
  - Training dataset is large
  - ‘compute gradients’ phase is computation intensive
  - It takes many iterations to converge (may pass the dataset multiple times/epochs)
- The training procedure consumes a lot of memory
  - Model parameter set is large
  - Many layers $\rightarrow$ hidden features use a lot of memory
  - Device memory is small, e.g., GPU memory is 2G-12G

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of epochs</td>
<td>90</td>
<td>74</td>
<td>100</td>
<td>90</td>
</tr>
<tr>
<td>Parameter size (# of floats)</td>
<td>60M</td>
<td>144M</td>
<td>5M</td>
<td>0.27-19.4M</td>
</tr>
</tbody>
</table>
Efficiency Optimization

- Improve the speed of DL on a single device (GPU or CPU device)
  - All operations of one (BP) iteration create a **dataflow graph**.
  - Existing systems either do static (Theano[12] and TensorFlow[13]) or dynamic (MxNet[14]) dependency analysis to **parallelize** operations without data dependencies.

\[
y = \text{concatenate}(a_1, a_2)
\]
\[
a_1 = x \ast W_1 + b_1
\]
\[
a_2 = x \ast W_2 + b_2
\]
\[
x = \text{sigmoid}(x)
\]

Efficiency Optimization

- New hardware (not unique to DB)
  - CPU is not suitable for DL training, which requires high FLOPS.
  - Nvidia GPU + CUDA is the most widely used hardware and programming language.
  - FPGA [15] has the advantage in terms of power consumption (lower than GPU). Programming is difficult on FPGA. Memory is small on FPGA. But it is potential for the prediction phase (where the model is already trained), which uses less memory and can be pipelined.
  - 2nd gen Xeon (Knights Landing) has improved towards machine learning applications.
  - OpenCL works on all above hardware, hence could be useful for programming.
  - DL chip and instruction set [16] have been proposed, and the products would come out soon.
  - Database researchers have exploited new hardware for query processing [17,18] by optimizing the cache, memory, and communication, which could be useful for implementing DL operations

[18] Paul Johns*, Jiong He*, Bingsheng He. GPL: A GPU-based Pipelined Query Processing Engine. SIGMOD
Efficiency Optimization

- Distributed training for DL
  - Model parallelism (partition model)
  - Data parallelism (partition data and replicate model)
Efficiency Optimization

- Communication challenges for distributed training
  - Model parallelism only works for special models that has multiple parallel paths. Otherwise the communication cost of transferring intermediate features would be large.
  - Data parallelism is not scalable for models with large sizes of model parameters, e.g., VGG model has > 100 Million parameters. The communication cost of transferring parameter values (and gradients) would be the bottleneck.
Efficiency Optimization

- Communication challenges for distributed training
  - Model parallelism only works for special models that have multiple parallel paths. Otherwise, the communication cost of transferring intermediate features would be large.
  - Data parallelism is not scalable for models with large sizes of model parameters, e.g., VGG model has > 100 Million parameters. The communication cost of transferring parameter values (and gradients) would be the bottleneck.

- Optimization techniques:
  - Compression [22,23], including lossless and lossy compression (a hot research topic)
  - Infiniband (RDMA) [21]. More exploration, e.g., data compression on Infiniband.
  - Hybrid parallelism [19,20]. Automatic model/data partitioning for parallelism?

[22] Hao Zhang, Zhiting Hu, Jinliang Wei, Pengtao Xie, Gunhee Kim, Qirong Ho, Eric Xing Poseidon: A System Architecture for Efficient GPU-based Deep Learning on Multiple Machines
[23] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, Dong Yu. 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs. Interspeech 2014.
Efficiency Optimization

- Consistency for distributed training
  - Synchronous training is limited to small (GPU) clusters
    - More workers → larger synchronization and communication overhead
    - More workers → BP time per (GPU) worker is smaller
    - For GPU workers, BP time is very small (compared with sync and communication time) → synchronization overhead would easily become the bottleneck → small GPU clusters
  - Asynchronous training may slow down the convergence rate or even diverge
    - More workers → larger staleness (staleness = global param version - local version)
    - More workers → larger momentum (i.e., may update the parameters in the wrong direction with a large step size) [20]
  - Hybrid:
    - Staleness-bounded asynchronous training would be promising [24]
    - Synchronous training with backup workers

Memory Optimization

- Techniques used by database systems [25]

<table>
<thead>
<tr>
<th>Aspects</th>
<th>Concerns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>cache consciousness, time/space efficiency</td>
</tr>
<tr>
<td>Data Layout</td>
<td>cache consciousness, space efficiency</td>
</tr>
<tr>
<td>Parallelism</td>
<td>linear scaling, partitioning</td>
</tr>
<tr>
<td>Concurrency Control/Transaction Management</td>
<td>overhead, correctness</td>
</tr>
<tr>
<td>Query Processing</td>
<td>code locality, register temporal locality, time efficiency</td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>durability, correlated failures, availability</td>
</tr>
<tr>
<td>Data Overflow</td>
<td>locality, <strong>paging strategy</strong>, hot/cold classification</td>
</tr>
</tbody>
</table>

Memory Optimization

- Optimize memory footprint for DL
  - Memory is used by:
    - Parameter values
    - Parameter gradients
    - Hidden layer values
    - Hidden layer gradients
  - GPUs and other new hardware have less memory than CPUs
    - e.g., 4G-12G, not large enough to hold VGG model.
  - Memory pool and Garbage collection
    - Release memory (via garbage collection) if it is not used in the rest of one iteration
    - Frequent malloc/free is costly for GPUs. Memory pool could help.
  - Swap data between GPU and CPU memory
    - Statically done by users [22]
    - Dynamically done by system using the paging strategies?
  - Drop some hidden layer values and redo the computation to recover them
    - Statically analyze the dataflow to decide which one to drop [23].

[27] Training Deep Nets with Sublinear Memory Cost. Tianqi Chen, Bing Xu, Chiyuan Zhang, Carlos Guestrin
Optimization techniques in existing DL systems

<table>
<thead>
<tr>
<th></th>
<th>TensorFlow</th>
<th>Caffe</th>
<th>Torch</th>
<th>MxNet</th>
<th>CNTK</th>
<th>Theano</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallelize operations</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Execution plan tuning</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Memory swapping</td>
<td>x</td>
<td>manual</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Intermediate data dropping</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>static</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Consistency</td>
<td>***</td>
<td>**</td>
<td>*</td>
<td>***</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Communication</td>
<td>-</td>
<td>*</td>
<td>-</td>
<td>-</td>
<td>**</td>
<td>-</td>
</tr>
<tr>
<td>H/W</td>
<td>***</td>
<td>***</td>
<td>***</td>
<td>**</td>
<td>**</td>
<td>***</td>
</tr>
</tbody>
</table>

The comparison may not be up-to-date and comprehensive as all system are under active development
- : unknown     x: not available    ✓: available    *: score
# SINGA V1

<table>
<thead>
<tr>
<th>Model</th>
<th>Layer</th>
<th>Initializer</th>
<th>Loss</th>
<th>Metric</th>
<th>Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>CppMath</td>
<td>CudaMath</td>
<td>OpenclMath</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device</td>
<td>CppCPU</td>
<td>CudaGPU</td>
<td>OpenclGPU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td>Memory</td>
<td>CPU/GPU/FPGA/ARM</td>
<td>Disk</td>
<td>InfiniBand/Ethernet</td>
<td></td>
</tr>
<tr>
<td>Tensor</td>
<td>Reader</td>
<td>Encoder</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Writer</td>
<td>Decoder</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Endpoint</td>
<td>Message</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SINGA V1: features

- Support general machine learning application via the Tensor abstraction
- Heterogeneous hardware to run Tensor math functions
  - CPU, Nvidia GPU, AMD GPU, Xeon Phi, FPGA, etc.
  - CPP + CUDA + OpenCL
- Runtime optimizations
  - Speed: analyze data dependency and estimate cost of execution plans
  - Memory: log commands and do drop/swap according to cost models
- Distributed training
  - Extend V0.3 to support automatic distributed training configuration
- Rafiki (deep learning as a service)
  - Easily deploy a deep learning model for service
  - Visualization
SINGA Applications

• Image Classification
  • Foodlg
SINGA Applications

- Malware Detection with SecureAge
- Virus/application files classification

<table>
<thead>
<tr>
<th>True Label</th>
<th>Predicted Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>4161 52</td>
</tr>
<tr>
<td>Virus</td>
<td>313 1874</td>
</tr>
</tbody>
</table>
SINGA Applications

- Health-care with NUH
- Disease progression modelling
- Patient readmission prediction
SINGA Applications -- StarHub
SINGA Applications -- StarHub

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8</td>
<td>10</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>12</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

... 10 ...

... to ...

---

25
Conclusion

- Deep learning is effective but difficult to optimize
- DB optimization techniques could be adapted for DL systems
- DL models can be applied to DB applications
- SINGA is working on
  - Optimizing speed, memory for single node
  - Optimizing communication/consistency for distributed training
  - Rafiki for easy deployment and visualization
Thank you!

http://singa.apache.org/
References


References

[22] Hao Zhang, Zhiting Hu, Jinliang Wei, Mengtao Xie, Gunhee Kim, Qirong Ho, Eric Xing Poseidon: A System Architecture for Efficient GPU-based Deep Learning on Multiple Machines
[23] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, Dong Yu, 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs. Interspeech 2014
References


[27] Training Deep Nets with Sublinear Memory Cost. Tianqi Chen, Bing Xu, Chiyuan Zhang, Carlos Guestrin

[28] Kim, Y. Convolutional Neural Networks for Sentence Classification. EMNLP 2014


[33] Xin Luna Dong and Divesh Srivastava. Knowledge curation and knowledge fusion: challenges, models, and applications. Tutorial in Sigmod'15
Synchronous training

- **Train the AlexNet [1] model with the mini-batch size = 512 would take 2s on a single GPU for one iteration**
  - With n GPU nodes
    - time per node would be $\geq 2/n$ seconds,
    - Size of data being transferred per node would be $60M \times 4 \times 2$ Bytes (60M is the number of parameters/floats, one float has 4 bytes, 2 for sending and receiving)
    - Rate of commodity Ethernet is 1Gbit/s ~ 100MB/s
    - Communication cost is $480/100=4.8s$
    - $2/n < 4.8 \rightarrow$ communication bottleneck
  - Set batch-size to 512*m
    - Time per node is $2*m/n$ seconds for one iteration
    - total training time would increase if $m > n$;
    - Communication bottleneck if $m < n$
- **GoogleLeNet has a smaller size of parameters and larger complexity per iteration.**
  - It would scale better than AlexNet model, but is still limited by the communication + synchronization overhead
SINGA V0.3: Neural Net and Layer

Feedforward models (e.g., CNN)

RNN

RBM

TrainOneBatch

NeuralNet

stop

labels
SINGA V0.3: Model Parallelism

- **NeuralNet Partitioning:**
  1. Partition all layers into different subsets
  2. Partition each single layer on batch dimension.
  3. Partition each single layer on feature dimension.
  4. Hybrid partitioning strategy of 1, 2 and 3.
SINGA V0.3: Data Parallelism

- **Synchronous training** (Google Sandblaster, Baidu AllReduce)

- **Asynchronous training** (Google Downpour, Hogwild!)

SINGA V0.3: Distributed Training Frameworks

Cluster Topology

Legends:
- Worker
- Server

Node
Group
Inter-node Communication

(a) Sandblaster
(b) AllReduce
(c) Downpour
(d) Distributed Hogwild

sync
async
Experimental Study

Train DCNN over CIFAR10: [https://code.google.com/p/cuda-convnet](https://code.google.com/p/cuda-convnet)

**Single Node**
- 4 NUMA nodes (Intel Xeon 7540, 2.0GHz)
- Each node has 6 cores
- hyper-threading enabled
- 500 GB memory

**Cluster**
- Quad-core Intel Xeon 3.1 GHz CPU and 8GB memory, 1Gbps switch
- 32 nodes, 4 workers per node

Synchronous
Experimental Study

Train DCNN over CIFAR10: [https://code.google.com/p/cuda-convnet](https://code.google.com/p/cuda-convnet)

Single Node

- Caffe
- SINGA

Cluster

Asynchronous
Experimental Study

AlexNet model for ImageNet dataset

(a) GTX 970
(b) GTX TiTan X

(a) A single node with multi-GPUs.
(b) A GPU cluster with 4 nodes.