多 GPU 环境下使用 CNTK 进行并发深度模型训练

Taifeng Wang (王太峰)
微软亚洲研究院 主管研究员

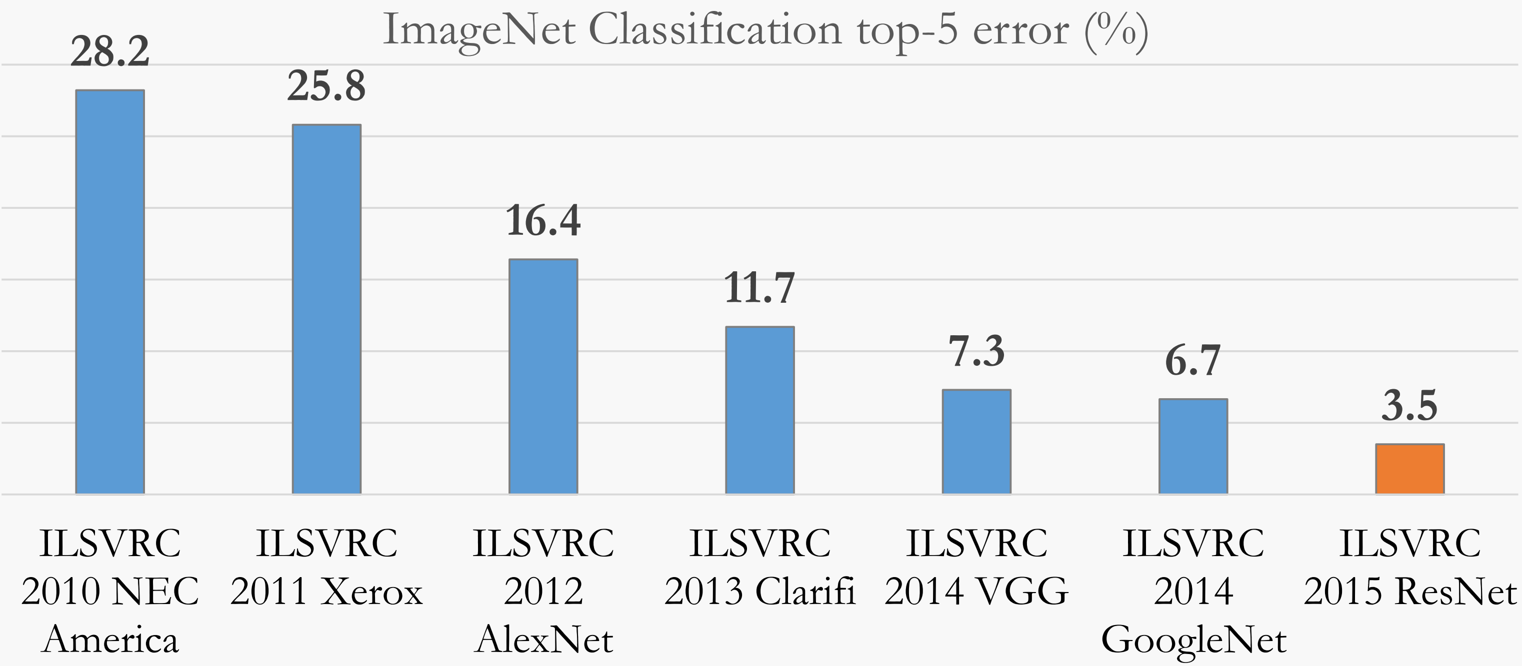
Microsoft

CHINA HADOOP SUMMIT

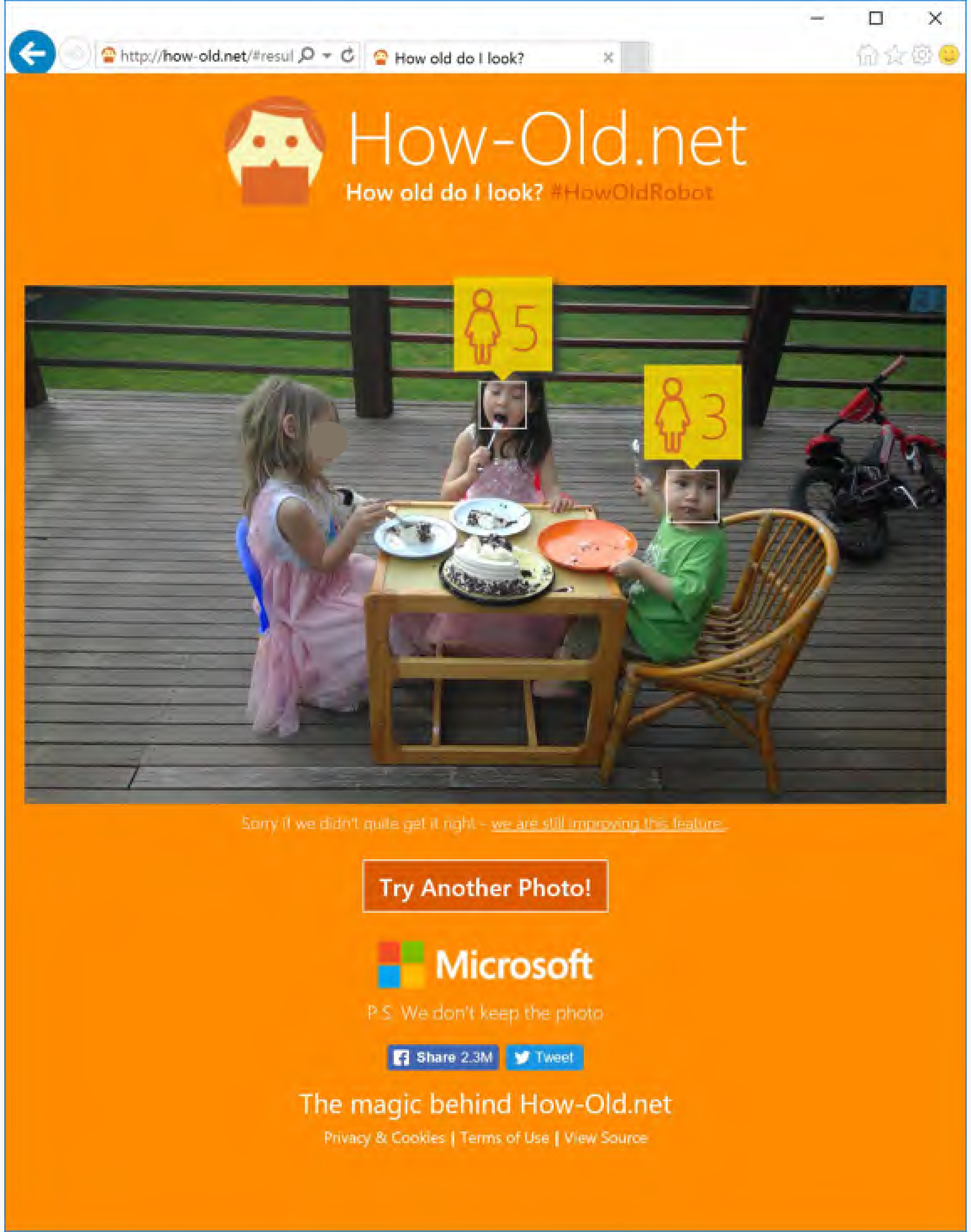
Machine Learning Group @ MSRA

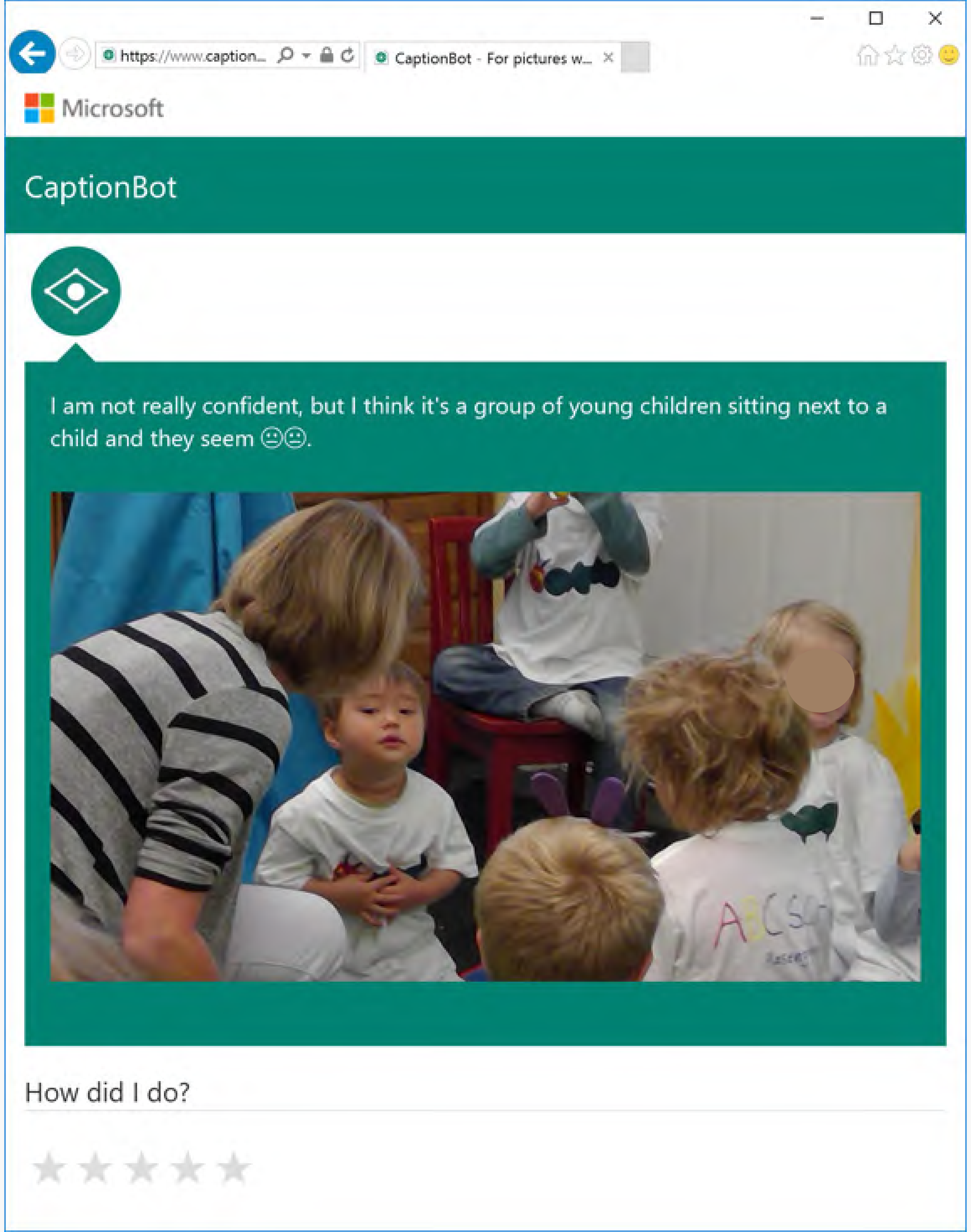
- 顶级学术期刊和会议上发表150余 篇论文，被引用万余次。
- 担任诸多顶级学术会议(SIGIR/ICML/NIPS/KDD/WWW/AAAI/ WINE/ICTIR等)组委会主席或领域 主席, 顶级学术期刊(TOIS/TWEB /Neurocomputing等)副主编。
- 发布或联合发布知名开源项目
  - 微软认知工具包(CNTK)
  - 微软图引擎 (Graph Engine)
  - 微软分布式机器学习工具包(DMTK)

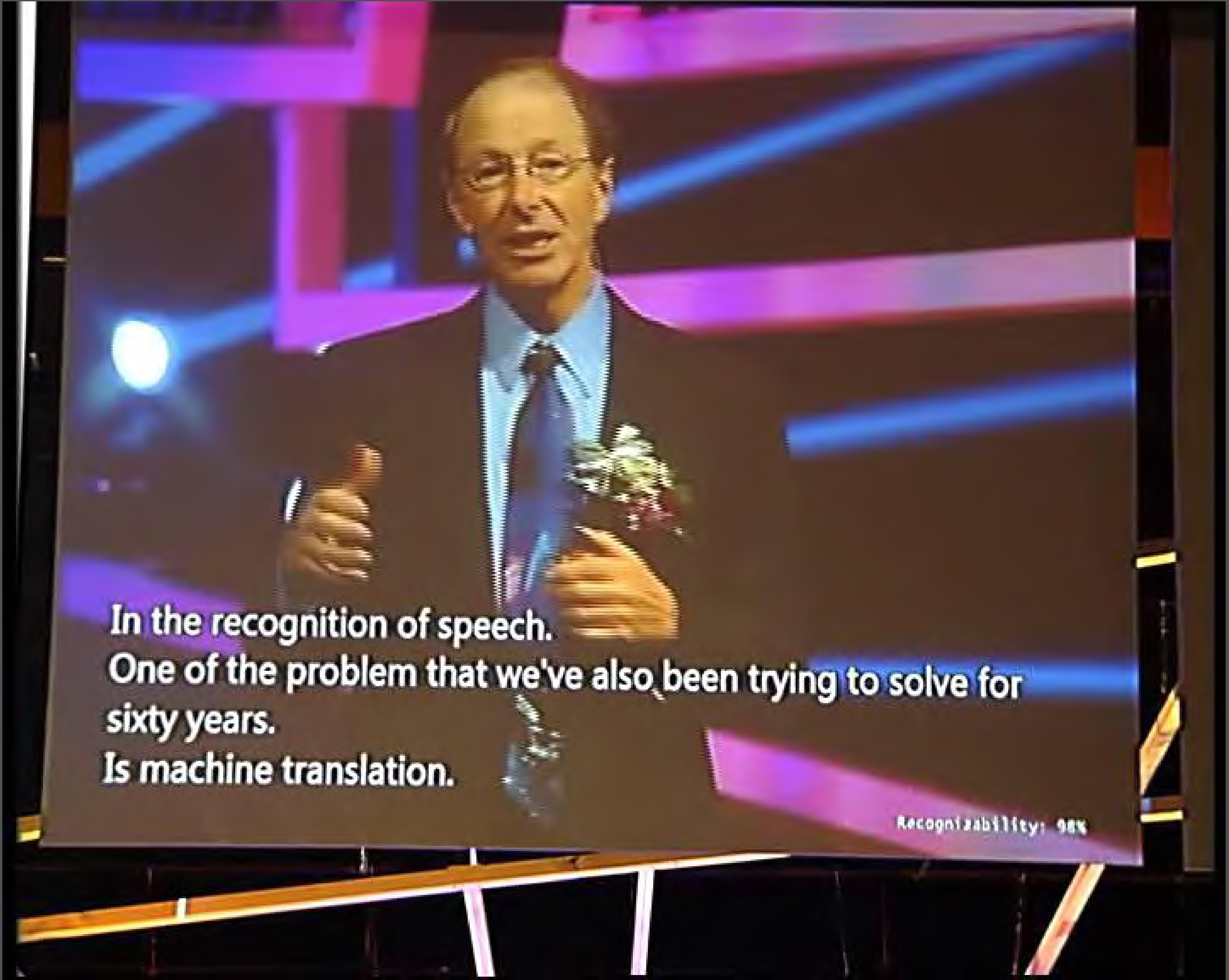
ImageNet: Microsoft 2015 ResNet



Microsoft had all **5 entries** being the 1-st places this year: ImageNet classification, ImageNet localization, ImageNet detection, COCO detection, and COCO segmentation


http://how-old.net/#resul
How old do I look?
How-Old.net
How old do I look? #HowOldRobot
5
3
Sorry if we didn't quite get it right - we are still improving this feature.
Try Another Photo!
Microsoft
P.S. We don't keep the photo
Share 2.3M
Tweet
The magic behind How-Old.net
Privacy & Cookies | Terms of Use | View Source


https://www.caption...
CaptionBot - For pictures w...
Microsoft
CaptionBot
I am not really confident, but I think it's a group of young children sitting next to a child and they seem 😐😐.
How did I do?


In the recognition of speech.
One of the problem that we've also been trying to solve for
sixty years.
Is machine translation.
Recognizability: 98%

Microsoft's historic speech breakthrough

- Microsoft 2016 research system for conversational speech recognition
- 5.9% word-error rate
- enabled by CNTK's multi-server scalability

[W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, G. Zweig: "Achieving Human Parity in Conversational Speech Recognition," https://arxiv.org/abs/1610.05256]

Historic Achievement: M

https://blogs.microsoft.com/next/2016/10/18/historic-achievement-micrc

Historic Achievement: Microsoft researchers reach human parity in conversational speech recognition

Microsoft researchers from the Speech & Dialog research group include, from back left, Wayne Xiong, Geoffrey Zweig, Xuedong Huang, Dong Yu, Frank Seide, Mike Seltzer, Jasha Droppo and Andreas Stolcke. (Photo by Dan DeLong)

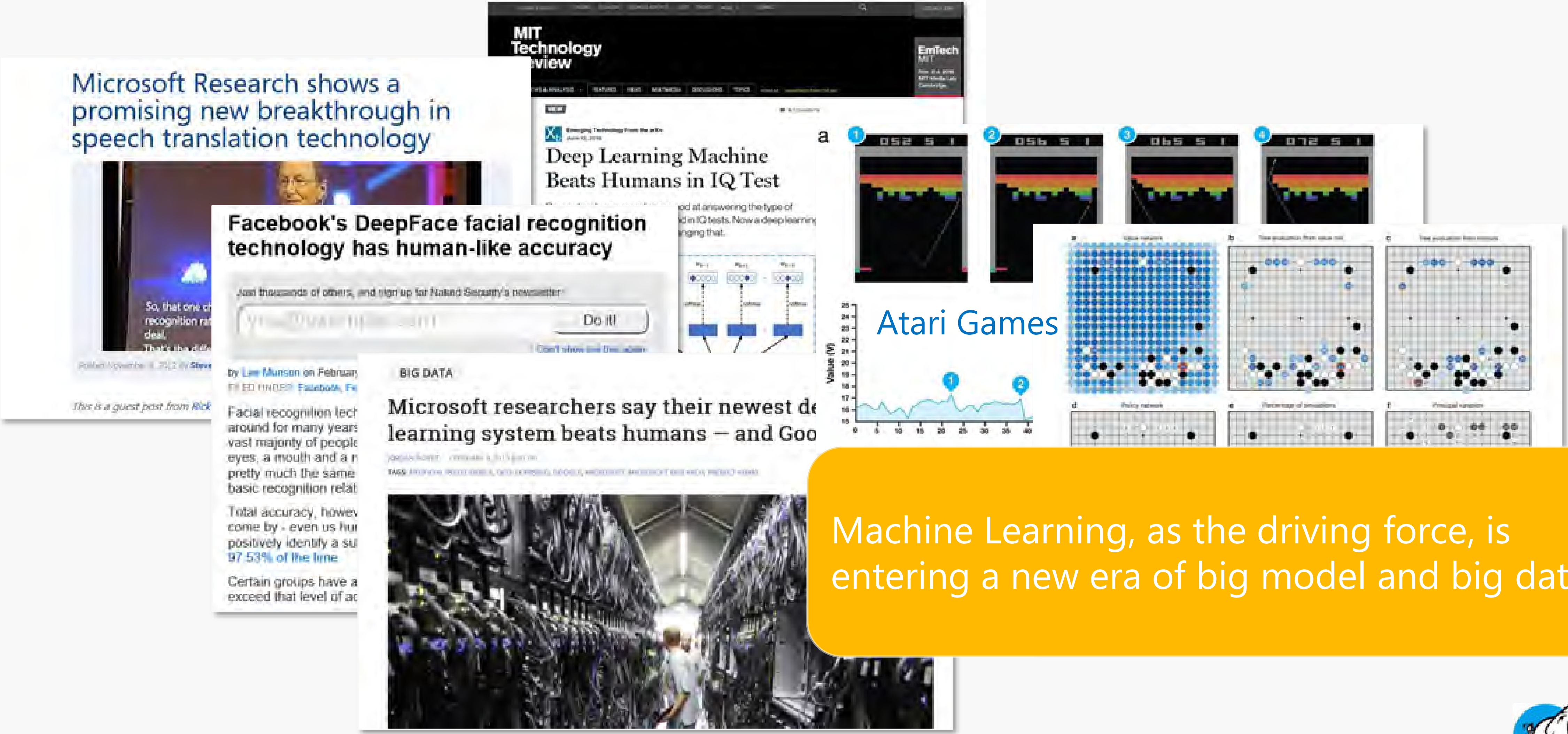
Posted October 18, 2016

By **Allison Linn**

Microsoft has made a major breakthrough in speech recognition, creating a technology that recognizes the words in a conversation as well as a person does.

Microsoft

AI Is Making Break-through!

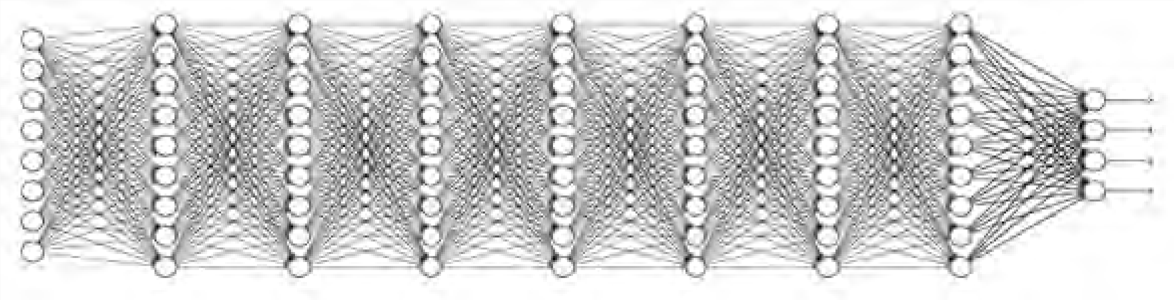


Machine Learning, as the driving force, is entering a new era of big model and big data.

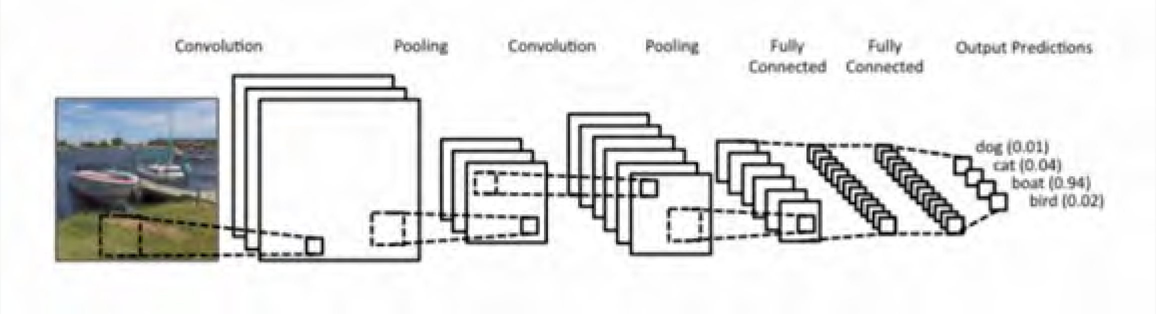
CHINA HADOOP SUMMIT

Deep Learning as Driving Force

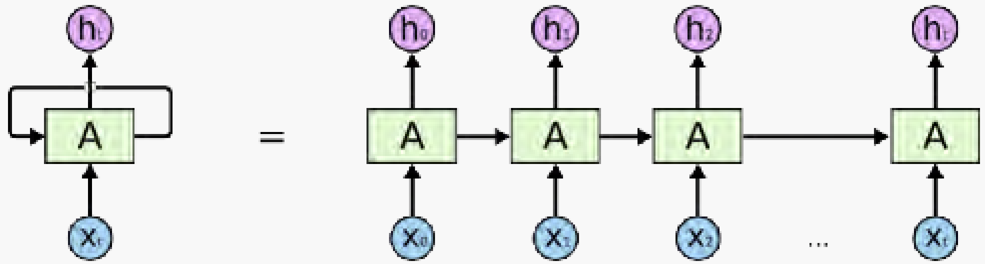
DNN: Deep Neural Networks



CNN: Convolutional Neural Networks



RNN: Recurrent Neural Networks



Some time cost analysis

- Time cost on train typical DNN models

Model	Hardware	Time cost
ResNet on ImageNet ~1M image samples for 1K classes	K40 * 8	~ 130 hours
GoogleNet on ImageNet	K40	~ 570 hours
2000h Speech LSTM model training	K40	~ 1100 hours
Neural Translation model	K40	~ 2000 hours

How to well utilize computation resources to speed up the training of big model over big data?

Distributed Deep Learning

Outline

- Quick Overview on CNTK
- Mini-tutorial on distributed machine learning
- Details on CNTK's parallel training

Outline

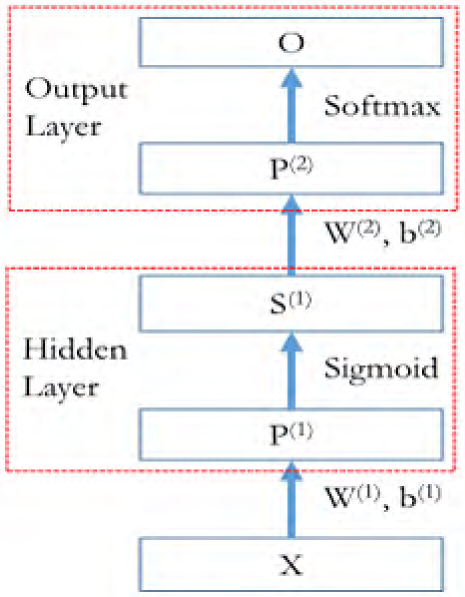
- Quick Overview on CNTK
- Mini-tutorial on distributed machine learning
- Details on CNTK's parallel training

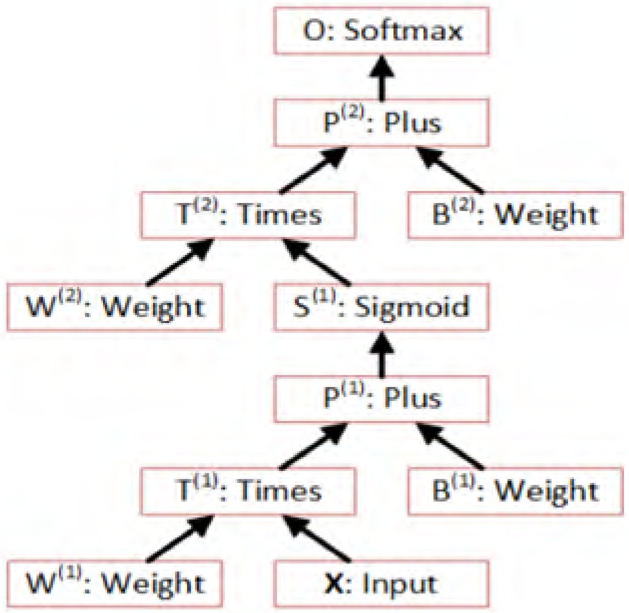
CNTK "Cognitive Toolkit"

- CNTK is Microsoft's **open-source**, **cross-platform** toolkit for learning and evaluating **deep neural networks**.
- CNTK expresses (nearly) **arbitrary neural networks** by composing simple building blocks into complex **computational networks**, supporting relevant network types and applications.
- CNTK is **production-ready**: State-of-the-art accuracy, efficient, and scales to multi-GPU/multi-server.
- Linux, Windows, docker, cudnn5, next: CUDA 8
- Python and C++ API (beta; C#/.Net on roadmap)

Computational Networks

- A generalization of machine learning models that can be described as a series of computational steps.





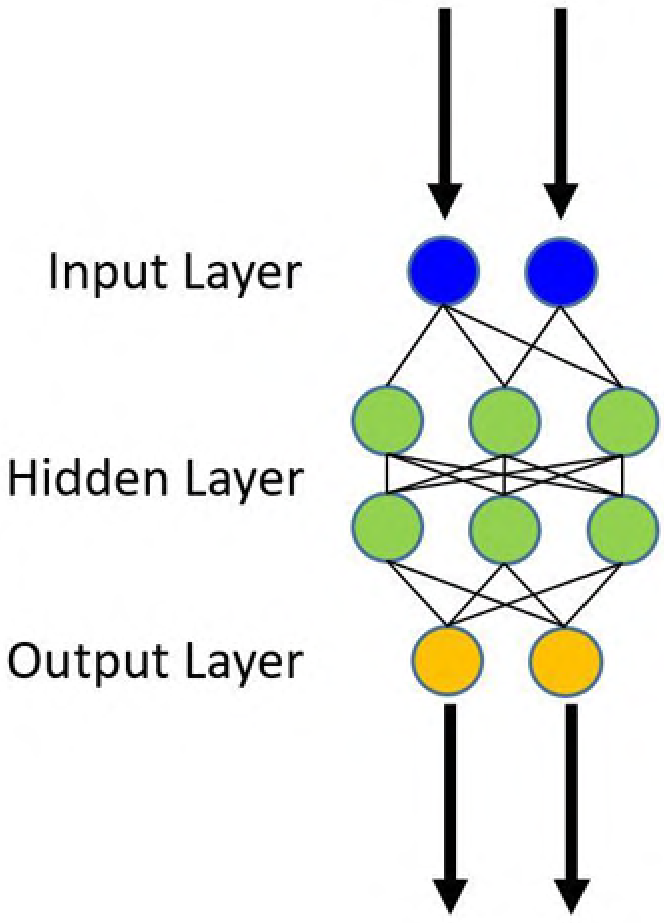
“CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications.”

"CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications."

example: 2-hidden layer feed-forward NN

$$
\begin{aligned}
h_1 &= \sigma(W_1\, x + b_1) \\
h_2 &= \sigma(W_2\, h_1 + b_2) \\
P &= \mathrm{softmax}(W_{\mathrm{out}}\, h_2 + b_{\mathrm{out}})
\end{aligned}
$$

with input $x \in \mathrm{R}^M$



“CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications.”

example: 2-hidden layer feed-forward NN

$$
\begin{aligned}
h_1 &= \sigma(W_1\, x + b_1) \\
h_2 &= \sigma(W_2\, h_1 + b_2) \\
P &= \mathrm{softmax}(W_{\mathrm{out}}\, h_2 + b_{\mathrm{out}})
\end{aligned}
$$

with input $x \in \mathrm{R}^M$ and one-hot label $y \in \mathrm{R}^J$
and cross-entropy training criterion

$$
\begin{aligned}
ce &= y^{\mathrm{T}} \log P \\
\textstyle\sum_{\mathrm{corpus}} ce &= \max
\end{aligned}
$$

Microsoft

"CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications."

example: 2-hidden layer feed-forward NN

$$
\begin{aligned}
h_1 &= \sigma(W_1\, x + b_1) \\
h_2 &= \sigma(W_2\, h_1 + b_2) \\
P &= \mathrm{softmax}(W_{\mathrm{out}}\, h_2 + b_{\mathrm{out}})
\end{aligned}
$$

```
h1 = sigmoid (x  @ W1   + b1)
h2 = sigmoid (h1 @ W2   + b2)
P  = softmax (h2 @ Wout + bout)
```

with input $x \in \mathrm{R}^M$ and one-hot label $y \in \mathrm{R}^J$
and cross-entropy training criterion

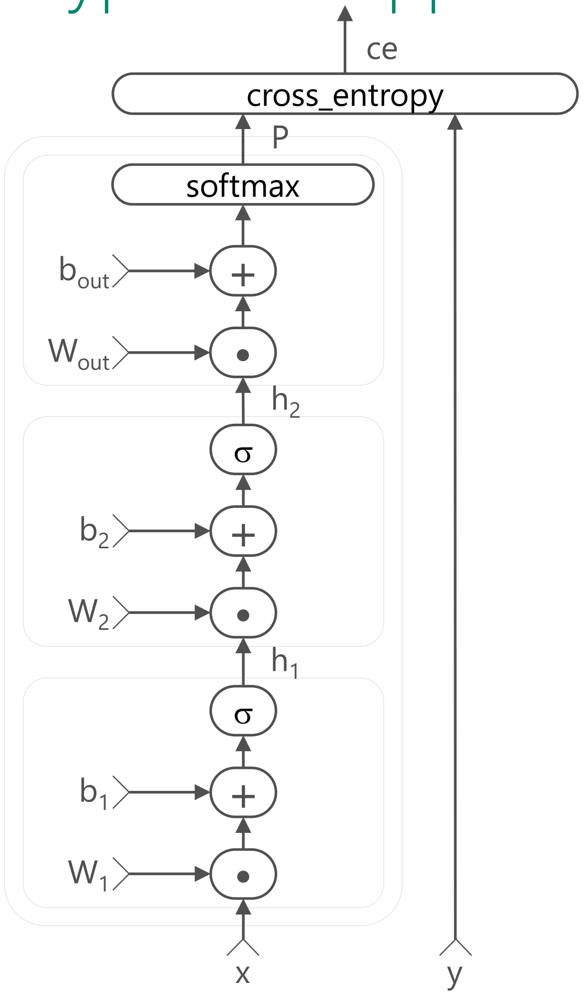
$$
\begin{aligned}
ce &= y^{\mathrm{T}} \log P \\
\textstyle\sum_{\mathrm{corpus}} ce &= \max
\end{aligned}
$$

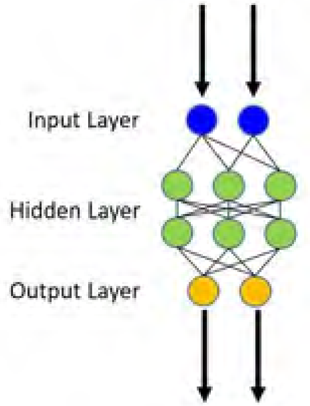
```
ce = cross_entropy (P, y)
```

"CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications."

```
h1 = sigmoid (x  @ W1   + b1)
h2 = sigmoid (h1 @ W2   + b2)
P  = softmax (h2 @ Wout + bout)
ce = cross_entropy (P, y)
```

“CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications.”

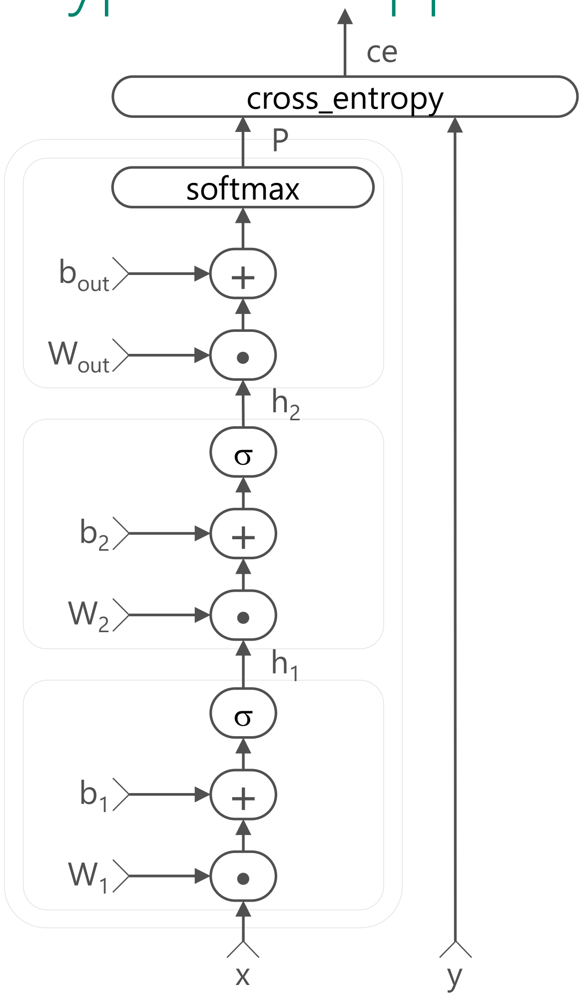





```
h1 = sigmoid (x  @ W1   + b1)
h2 = sigmoid (h1 @ W2   + b2)
P  = softmax (h2 @ Wout + bout)
ce = cross_entropy (P, y)
```

Microsoft

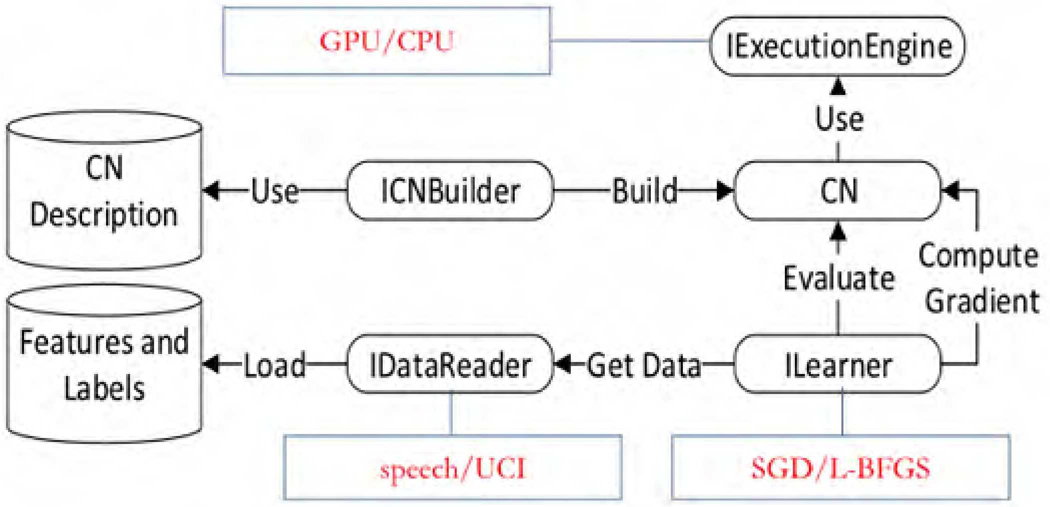
"CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications."



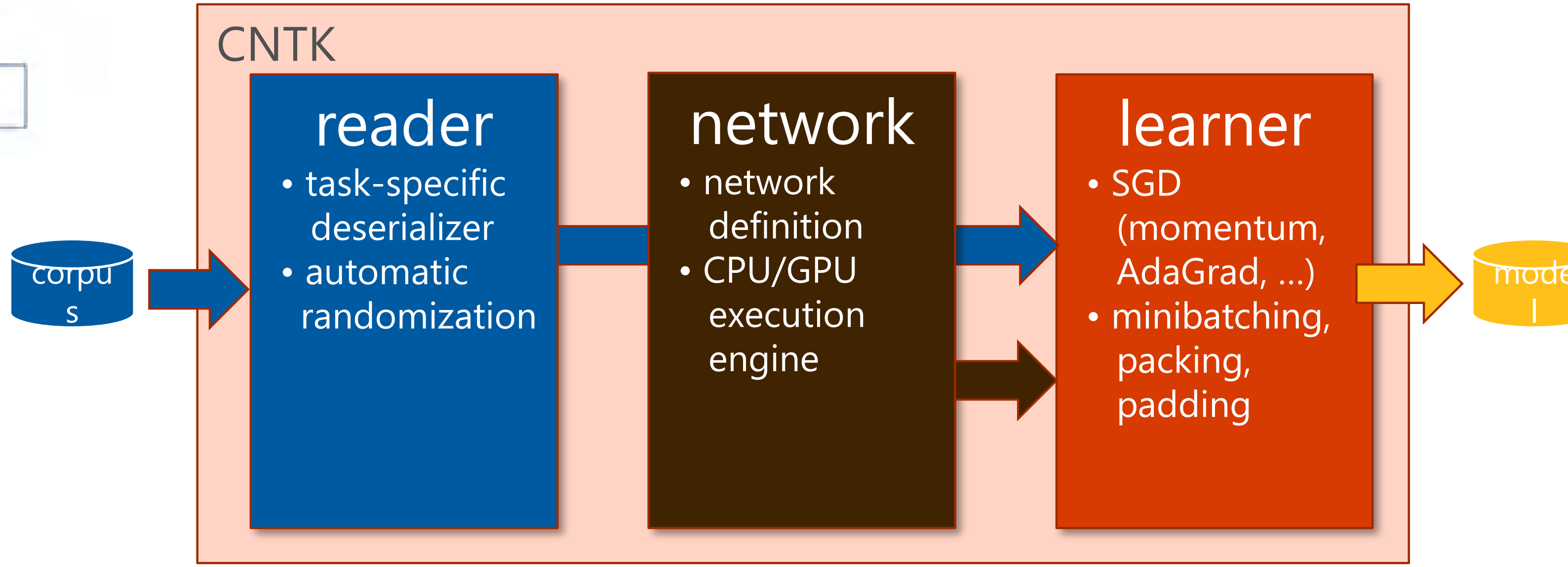
- nodes: functions (primitives)
  - can be composed into reusable composites
- edges: values
  - incl. tensors, sparse
- automatic differentiation
  - $\partial\mathcal{F} / \partial in = \partial\mathcal{F} / \partial out \cdot \partial out / \partial in$
- deferred computation → execution engine
- editable, clonable

LEGO-like composability allows CNTK to support wide range of networks & applications

CNTK Architecture



- Select right reader and learner to do an LEGO-like training
- Describe network as Computation Network
- Using Simple Network Builder, Brainscript or python to build Computation Network

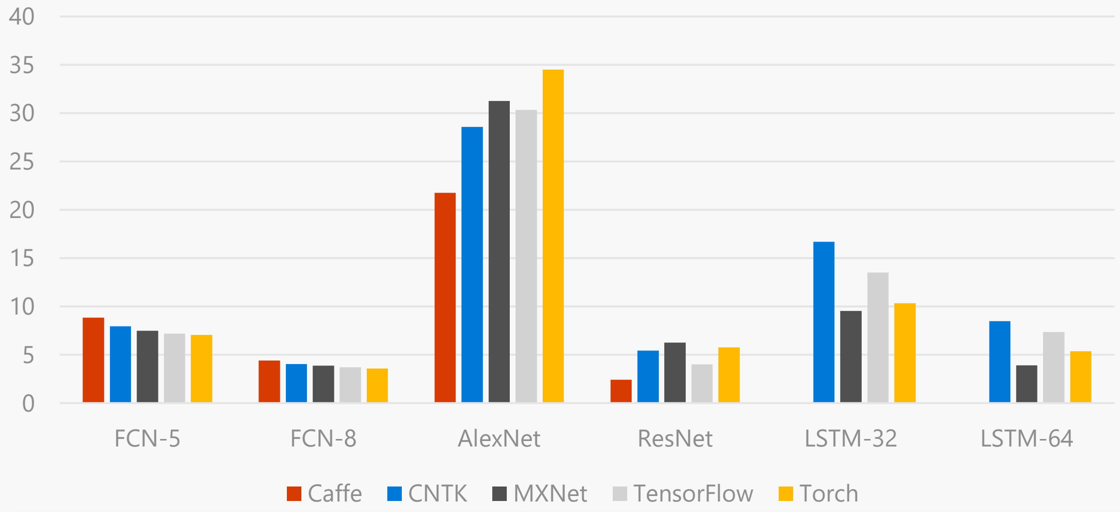


Microsoft

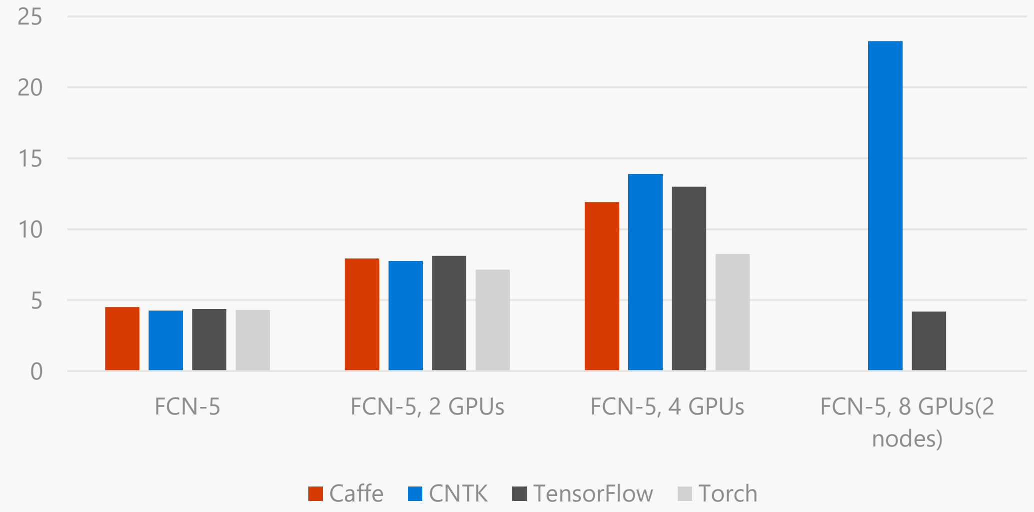
CNTK Benchmark

http://github.com/Microsoft/cntk

Speed Comparison for Single GPU Training
meausred by iterations*/second, higher=better



Speed Comparison for parallel Training
Measured by iterations / second, higher = better



- CNTK is fast for single-GPU training, and its speed is especially outstanding for RNN training. CNTK is also the best among all DNN tools in terms of scalability.

Outline

- Quick revisit on CNTK
- Mini-tutorial on distributed machine learning
- Details on CNTK's parallel training

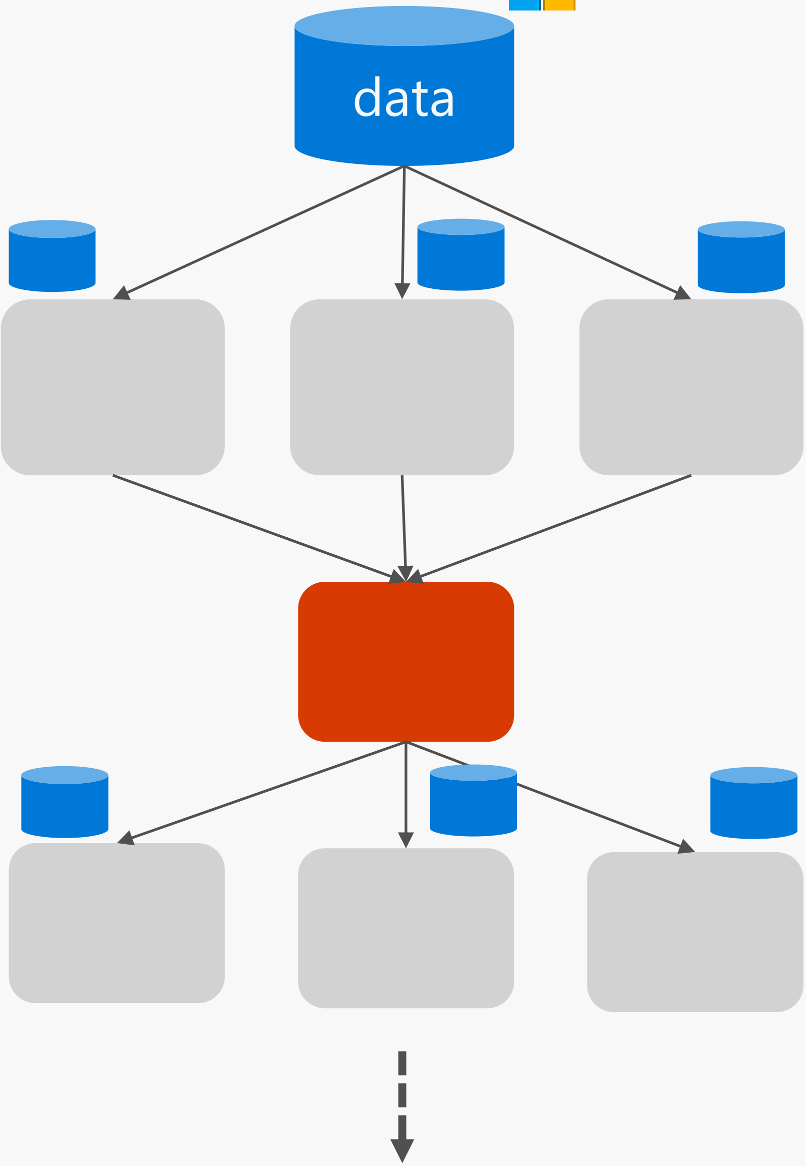
Data Parallelism

1. Partition the training data

2. Parallel training on different machines

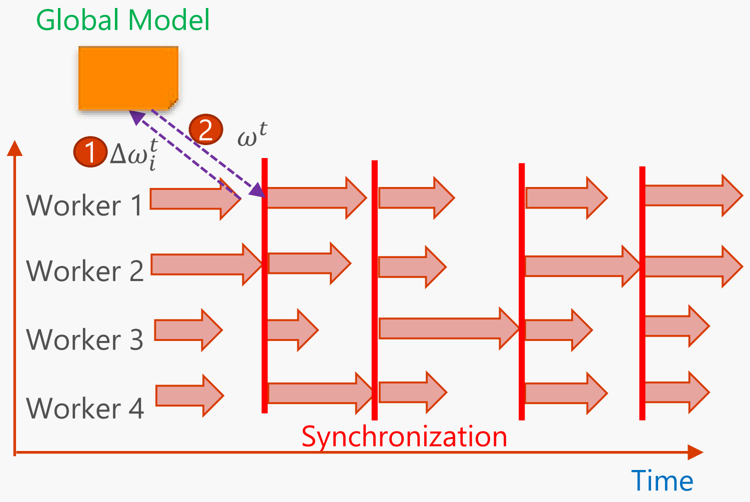
3. Synchronize the local updates

4. Refresh the local model with new parameters, go to 2.



BSP: Bulk Synchronous Parallel

Leslie G. Valiant

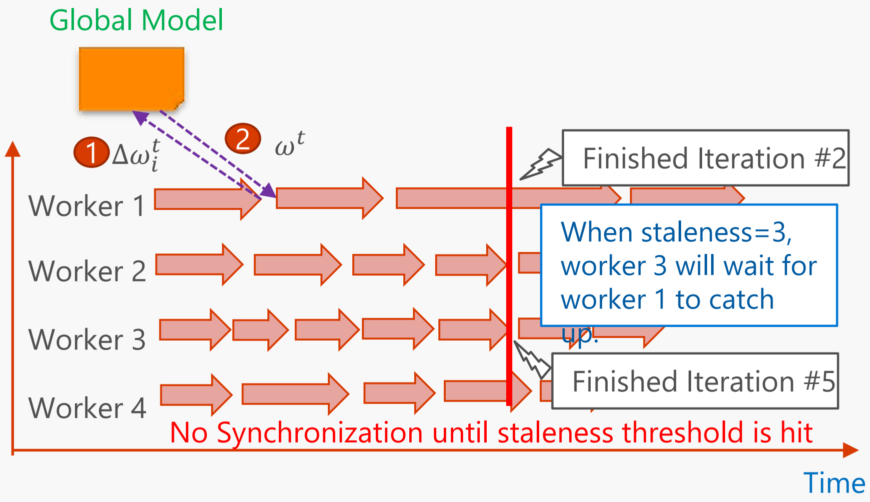


Individual workers synchronize with each other every (k) mini-batch:

1) Aggregate $\Delta\omega_i$'s from all workers to refine global model ω.
2) Broadcast global model ω back to each worker.
3) After receiving new global model, each worker starts next step of training.

- **BSP** is a well-defined mechanism, which can be equivalent to a single-machine SGD under certain conditions.
- **BSP** has convergence guarantee, but might be inefficient due to frequent synchronization.

SSP: Stale Synchronous Parallel

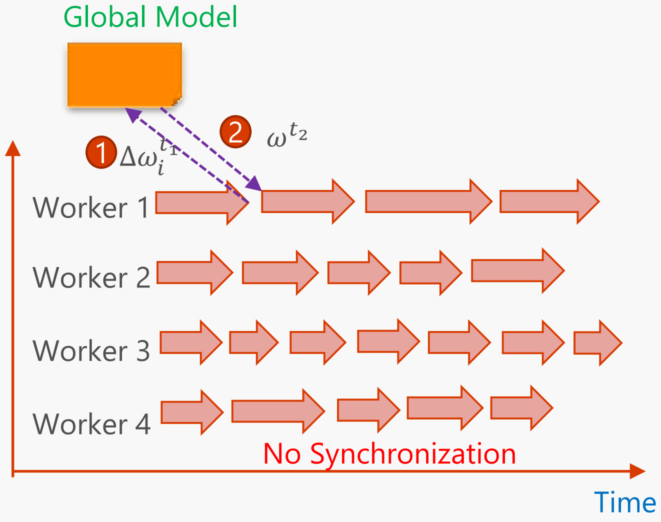


Individual worker pushes update $\Delta\omega_i$ to global model ω every (k) mini-batch, until notice that another worker is s steps behind. Thus SSP tradeoffs between BSP and ASP.

1) When $s = 0, SSP = BSP$.
2) When $s = \infty, SSP = ASP$.

- **SSP** tradeoffs efficiency and convergence: (1) It does not require strict synchronization (2) It does not allow workers' progresses to have large differences.
- **SSP** is proven to converge for convex loss and bounded staleness.

ASP: Asynchronous Parallel



Individual worker push its update $\Delta\omega_i$ to global model ω every (k) mini-batch, without waiting for others.

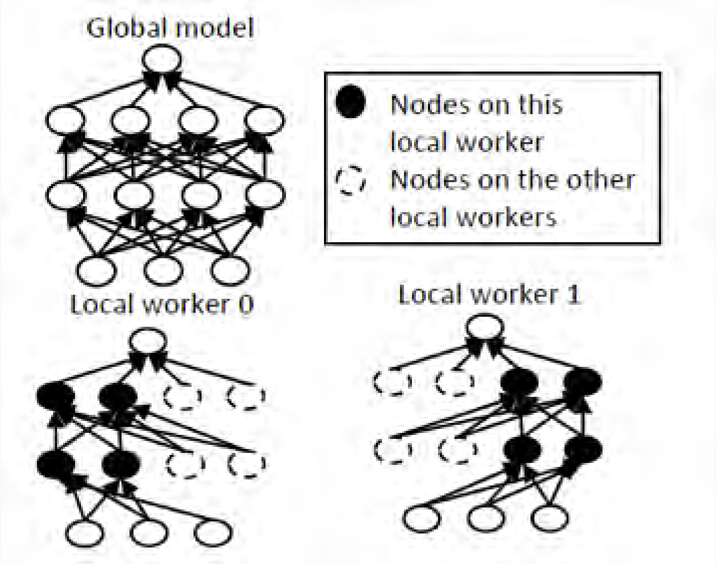
1) Push update $\Delta\omega_i$ to global model ω
2) Pull back whatever global model in the parameter server
3) Proceed training based on the latest ω in local machine.

- **ASP** always runs fast due to its asynchronous nature, no time wasted on waiting.
- **ASP**, in theory, might not converge when differences between workers' progresses are unbounded (straggler will destroy convergence by pushing stale $\Delta\omega$ onto global model).

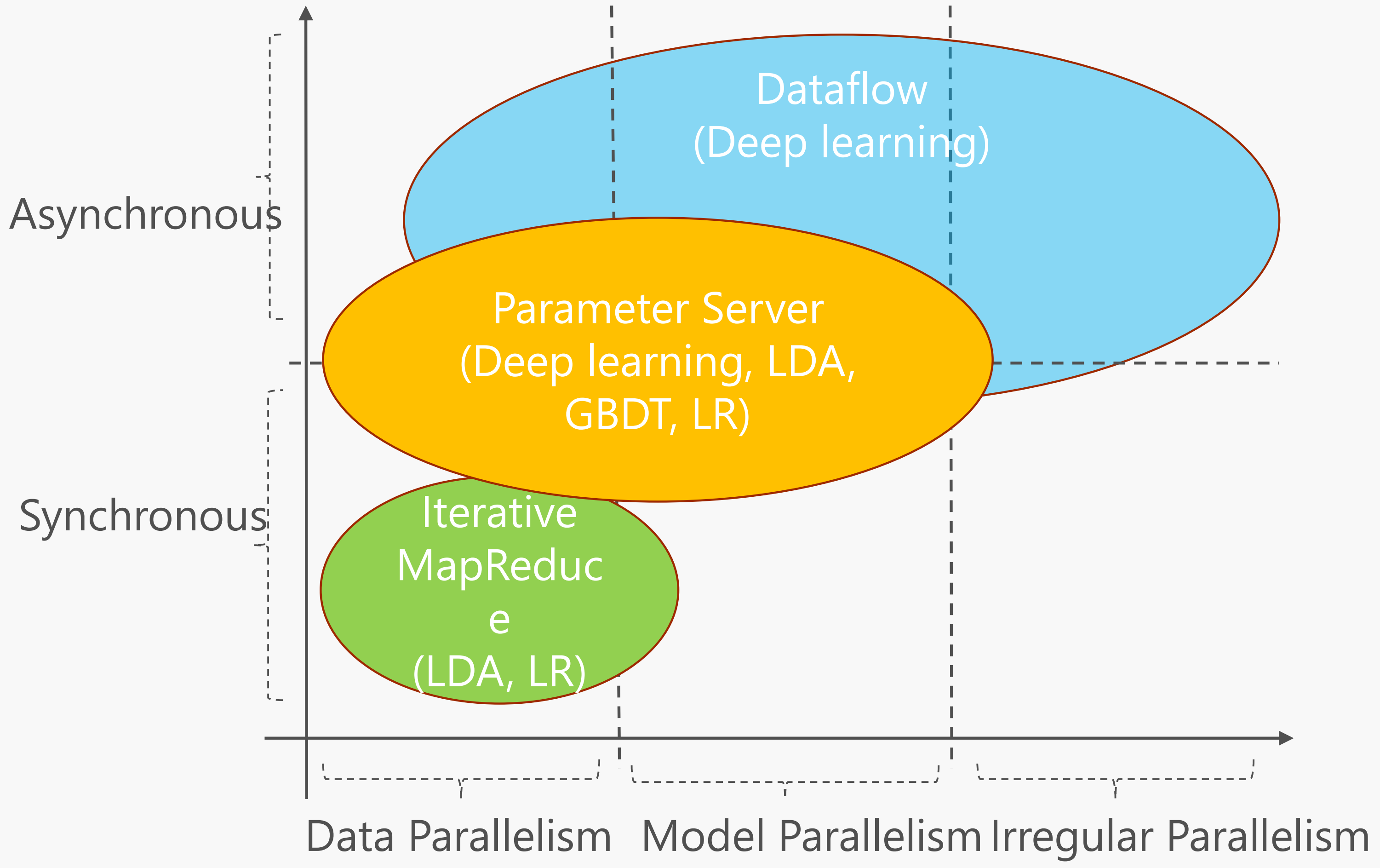
Model Parallelism

A system approach

- The global model is partitioned into K sub-models without overlap.
- The sub-models are distributed over K local workers and serve as their local models.
- In each mini-batch, the local workers compute the gradients of the local weights by back propagation.

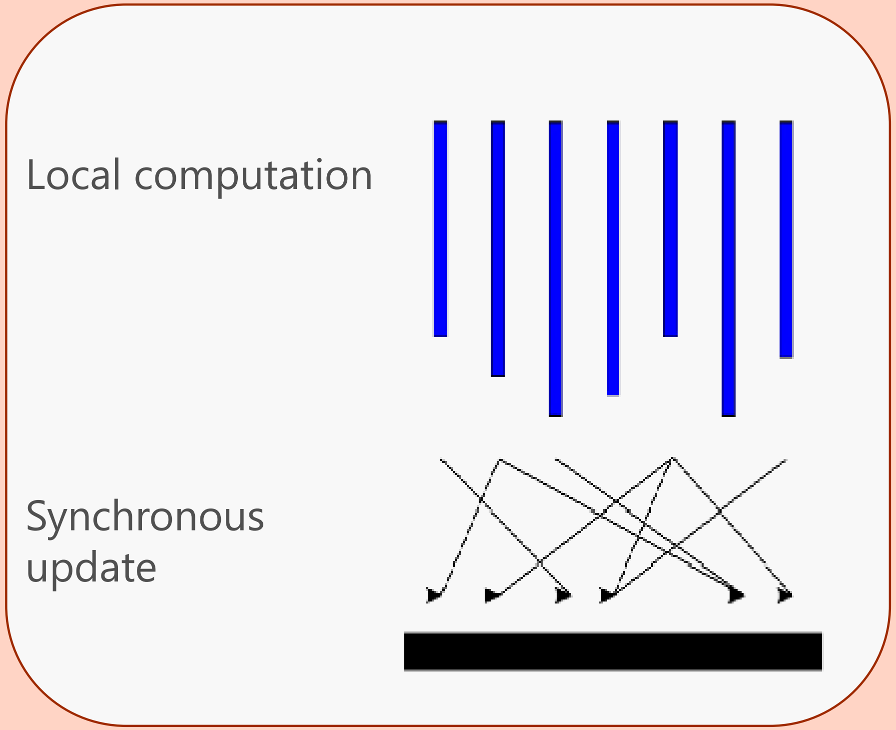


Evolution of Distributed ML Architectures



Evolution of Distributed ML Architectures

Iterative MapReduce

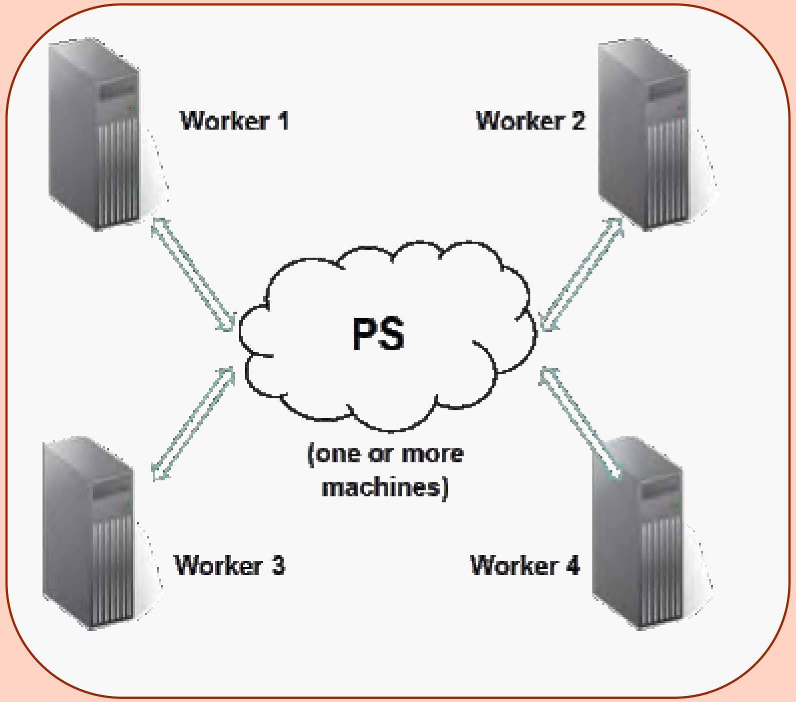


- Use MapReduce / AllReduce to sync parameters among workers
- Only synchronous update
- Example: Spark and other derived systems

Evolution of Distributed ML Architectures

Iterative MapReduce

Parameter Server



- Parameter server (PS) based solution is proposed to support:
  - Asynchronous update
  - Different mechanisms for model aggregation, especially in asynchronous manner
  - Model parallelism
- Example:
  - Google's DistBelief; Petuum
  - Multiverso PS

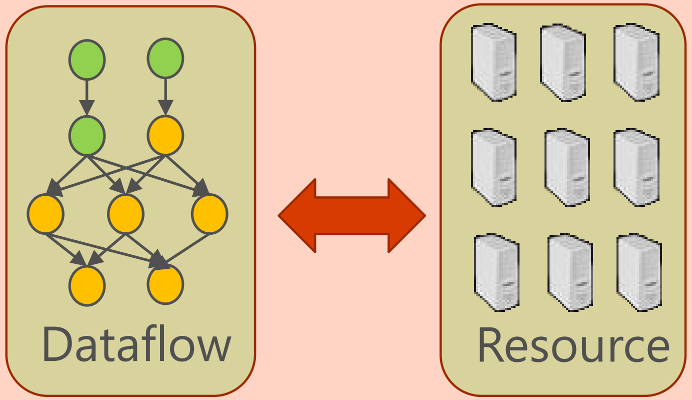
+ NIPS'12 DistBelief (Google), NIPS'13 Petuum (Eric Xing), OSDI'14 Parameter server (Mu Li), Multiverso PS… etc.

Evolution of Distributed ML Architectures

Iterative MapReduce

Parameter Server

Dataflow



Task scheduling & execution based on:

1. Data dependency
2. Resource availability

Dataflow based solution is proposed to support:

- Irregular parallelism (e.g., hybrid data- and model-parallelism), particularly in deep learning
- Both high-level abstraction and low-level flexibility in implementation

Example:

- Google's TensorFlow

+ Tensorflow, Eusys'07 Dryad (Microsoft), NSDI'12 Spark (AMP Lab), CNTK, MXNet… etc.

A Summary for Distributed ML System

Computation Infrastructure
parallel execution engine based on gRPC, packup message, communication, etc.

Computation allocation method
Manual assign to devices vs. automatic conduct optimized allocation

Parameter aggregation logic
Inherent all research output from parameter server side.

Industrial Practice

Microsoft

The focus is still in Data Parallelism

+ data is huge
+ model is in modest size
+ a cluster of machine are working together to speed up the data partition training

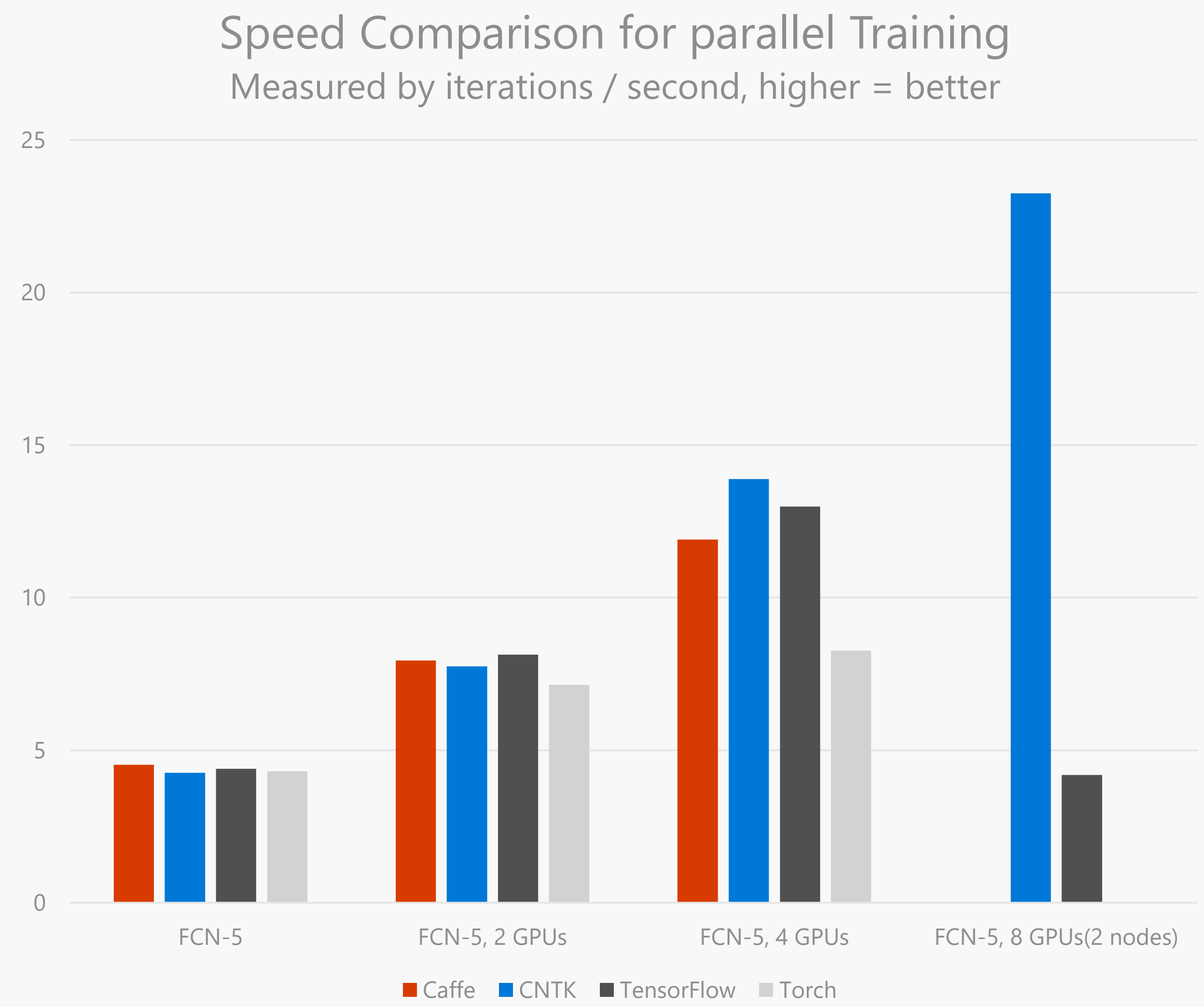
CHINA HADOOP SUMMIT

Outline

- Quick revisit on CNTK
- Mini-tutorial on distributed machine learning
- Details on CNTK's parallel training

CNTK's scale up experiment

Almost linear speed up from within node to cross node



https://github.com/guolinke/deep-learning-benchmarks

CNTK Deep dive: data-parallel training

- data-parallelism: distribute each minibatch over workers, then aggregate
- challenge: communication cost
- example: DNN, MB size 1024, 160M model parameters
  - compute per MB: → 1/7 second
  - communication per MB: → 1/9 second (640M over 6 GB/s)
  - can’t even parallelize to 2 GPUs: communication cost already dominates!

To tackle the parallelization problem

- Communicate less each time
- 1 bit SGD

- Communicate less often
- Auto mini batch sizing
- Block momentum

- Asynchronous and Pipelined processing
- \- ASGD with Multiverso

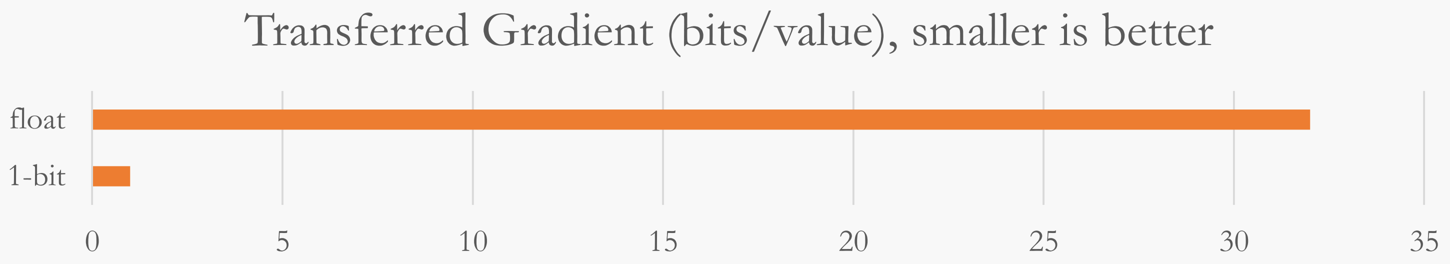
https://github.com/Microsoft/CNTK/wiki/Multiple-GPUs-and-machines

Microsoft

1-bit SGD

- quantize **gradients** to but **1 bit per value** with **error feedback**
  - All parameter was decided weather to plus/minus a same value, and carries over the rest value to next minibatch
  - Delay the updates procedure

$$
\begin{aligned}
G_{ij\ell}^{\text{quant}}(t) &= \mathcal{Q}(G_{ij\ell}(t) + \Delta_{ij\ell}(t-N)) \\
\Delta_{ij\ell}(t) &= G_{ij\ell}(t) - \mathcal{Q}^{-1}(G_{ij\ell}^{\text{quant}}(t))
\end{aligned}
$$



1-Bit Stochastic Gradient Descent and its Application to Data-Parallel Distributed Training of Speech DNNs, InterSpeech 2014, F. Seide, H. Fu, J. Droppo, G. Li, D. Yu

Using 1Bit technique of CNTK

Available on both Python and C++

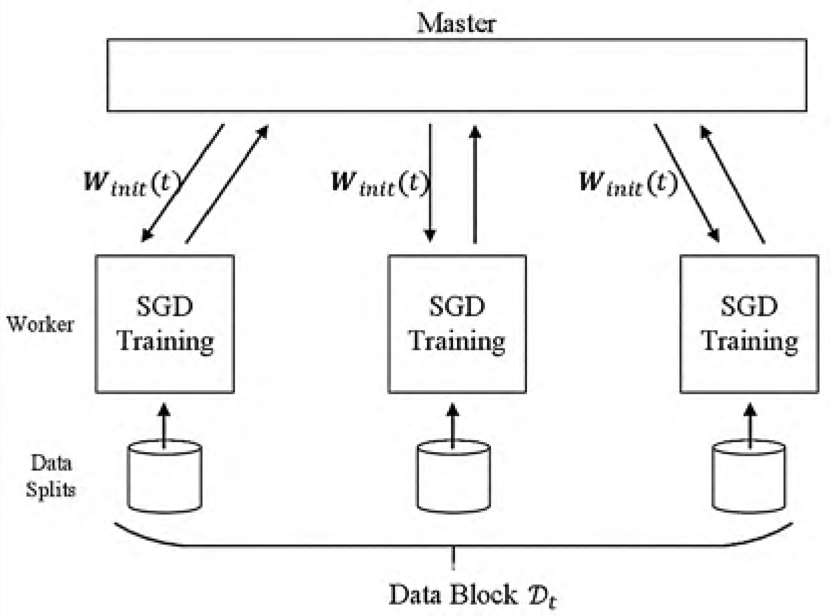
Brainscript(CNTK receipt)

```
SGD = [
    ...
    ParallelTrain = [
        DataParallelSGD = [
            gradientBits = 1
        ]
        parallelizationStartEpoch = 2  # warm start: don't use 1-bit SGD for first epoch
    ]
    AutoAdjust = [
        autoAdjustMinibatch = true       # enable automatic growing of minibatch size
        minibatchSizeTuningFrequency = 3 # try to enlarge after this many epochs
    ]
]
```

Python script

```
from cntk import distributed
...
distributed_after = epoch_size              # number of samples to warm start with
distributed_trainer = distributed.data_parallel_distributed_trainer(
    num_quantization_bits = 1,
    distributed_after = distributed_after)  # warm start: don't use 1-bit SGD for first epo
...
minibatch_source = MinibatchSource(
    ...,
    distributed_after = distributed_after)  # minibatch source becomes distributed after wa
...
trainer = Trainer(z, ce, pe, learner, distributed_trainer)
...
distributed.Communicator.finalize()      # must call this to finalize MPI, otherwise proce
```

Block momentum



- Select randomly an unprocessed data block denoted as $\mathcal{D}_t$
- Distribute N splits of $\mathcal{D}_t$ to N parallel workers
- Starting from an initial model denoted as $\boldsymbol{W}_{init}(t)$, each worker optimizes its local model independently by 1-sweep mini-batch SGD with momentum trick
- Average N optimized local models to get $\overline{\boldsymbol{W}}(t)$

K. Chen, Q. Huo: "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," ICASSP 2016

Block momentum

- Generate model-update resulting from data block $\mathcal{D}_t$:

$$\boldsymbol{G}(t) = \overline{\boldsymbol{W}}(t) - \boldsymbol{W}_{init}(t)$$

- Calculate global model-update:

$$\boldsymbol{\Delta}(t) = \eta_t \cdot \boldsymbol{\Delta}(t-1) + \varsigma_t \cdot \boldsymbol{G}(t)$$

  - ς_t: Block Learning Rate (BLR)
  - η_t: Block Momentum (BM)
  - When $\varsigma_t = 1$ and $\eta_t = 0$ → MA
- Update global model

$$\boldsymbol{W}(t) = \boldsymbol{W}(t-1) + \boldsymbol{\Delta}(t)$$

- Generate initial model for next data block
  - Classical Block Momentum (CBM)

$$\boldsymbol{W}_{init}(t+1) = \boldsymbol{W}(t)$$

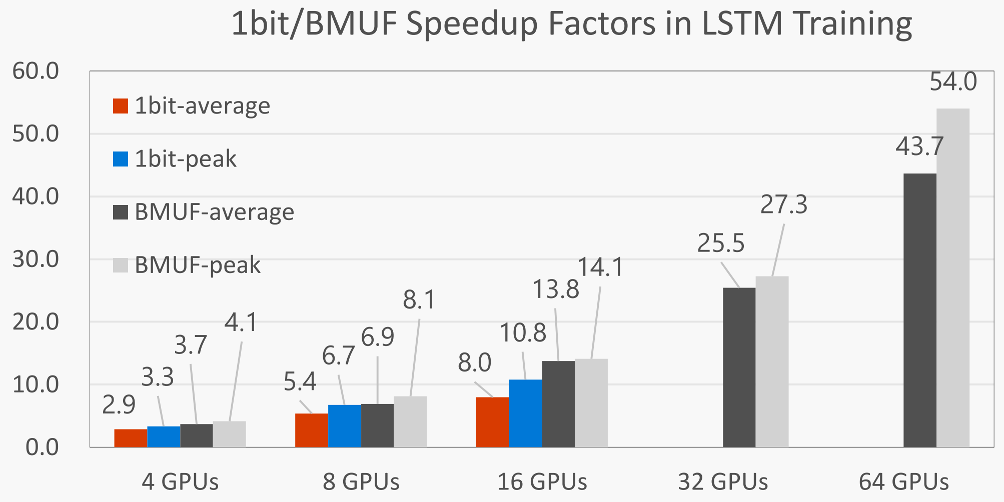
  - Nesterov Block Momentum (NBM)

$$\boldsymbol{W}_{init}(t+1) = \boldsymbol{W}(t) + \eta_{t+1} \cdot \boldsymbol{\Delta}(t)$$

K. Chen, Q. Huo: "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," ICASSP 2016

Block momentum

- Training data: 2,670-hour speech from real traffics of VS, SMD, and Cortana
  - **About 16 and 20 days to train DNN and LSTM on 1-GPU, respectively**
- Philly is too busy in I/O traffics:
  - Peak speedup factor
  - Average speedup factor



K. Chen, Q. Huo: "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," ICASSP 2016

Using BMUF in CNTK

Available on C++

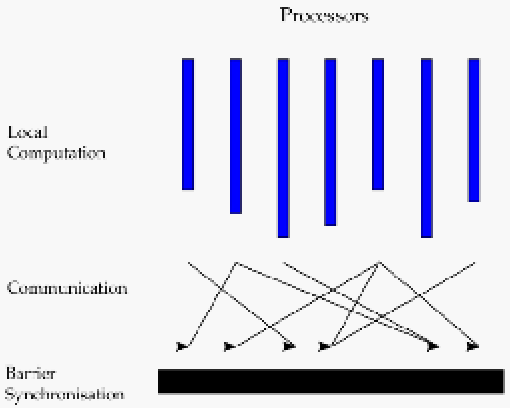
Brainscript(CNTK receipt)

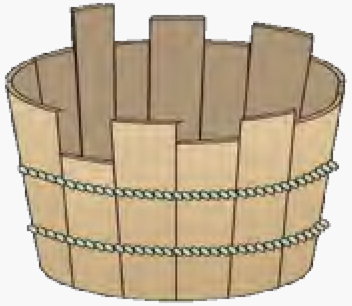
```
learningRatesPerSample=0.0005
# 0.0005 is the optimal learning rate for single-GPU training.
# Use it for BlockMomentumSGD as well
ParallelTrain = [
    parallelizationMethod = BlockMomentumSGD
    distributedMBReading = true
    syncPerfStats = 5
    BlockMomentumSGD=[
        syncPeriod = 120000
        resetSGDMomentum = true
        useNesterovMomentum = true
    ]
]
```

Asynchronous and Pipelined processing

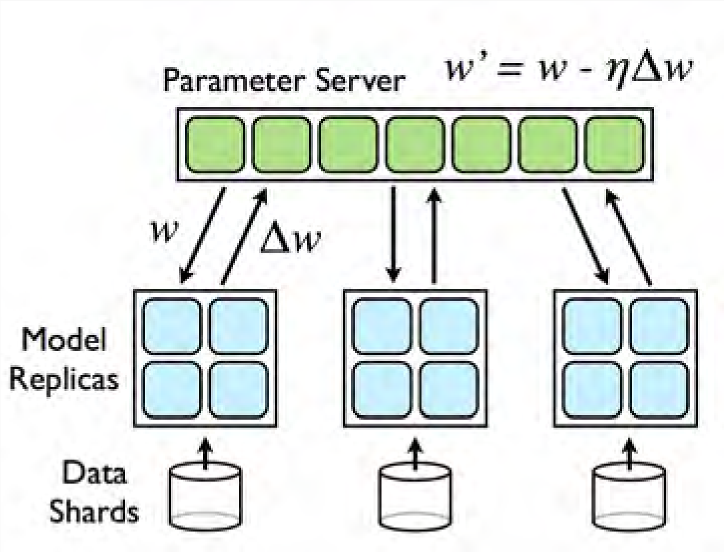
Multiverso parameter server

Communication barrier in synchronous parallelization



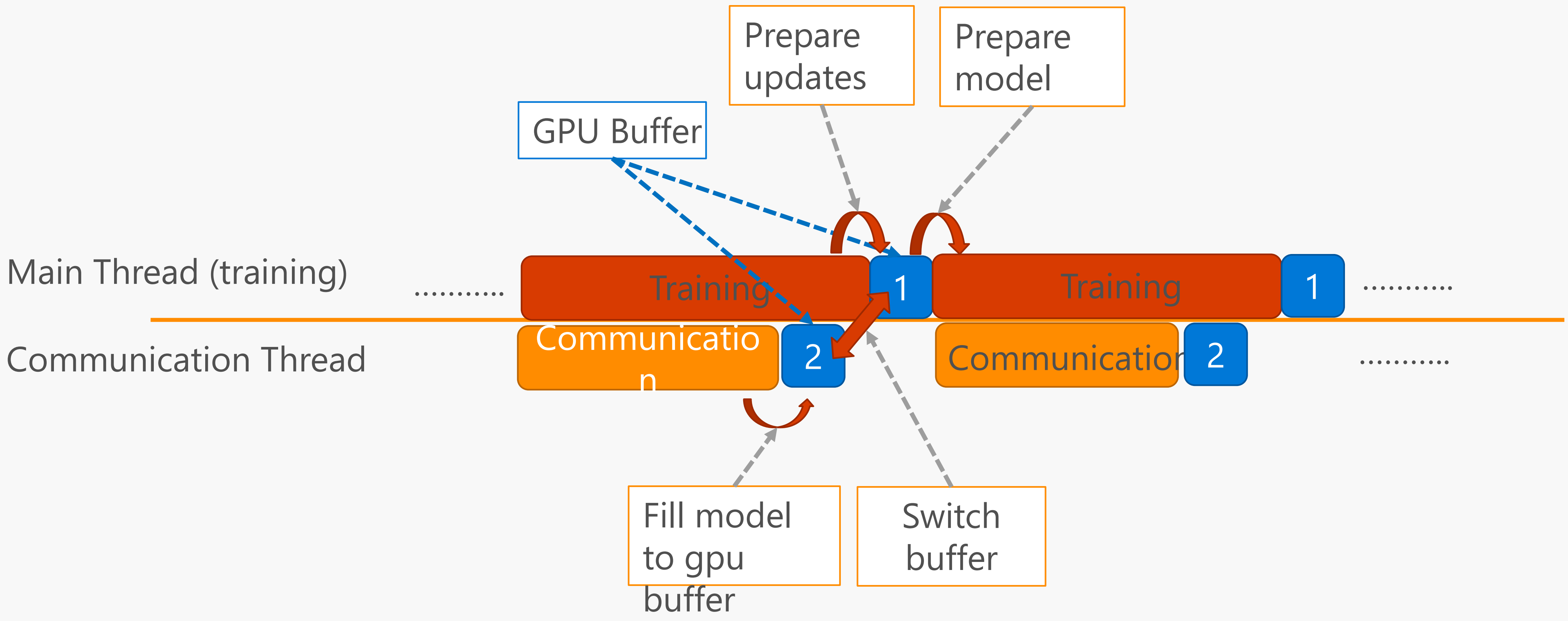


asynchronous parallelization



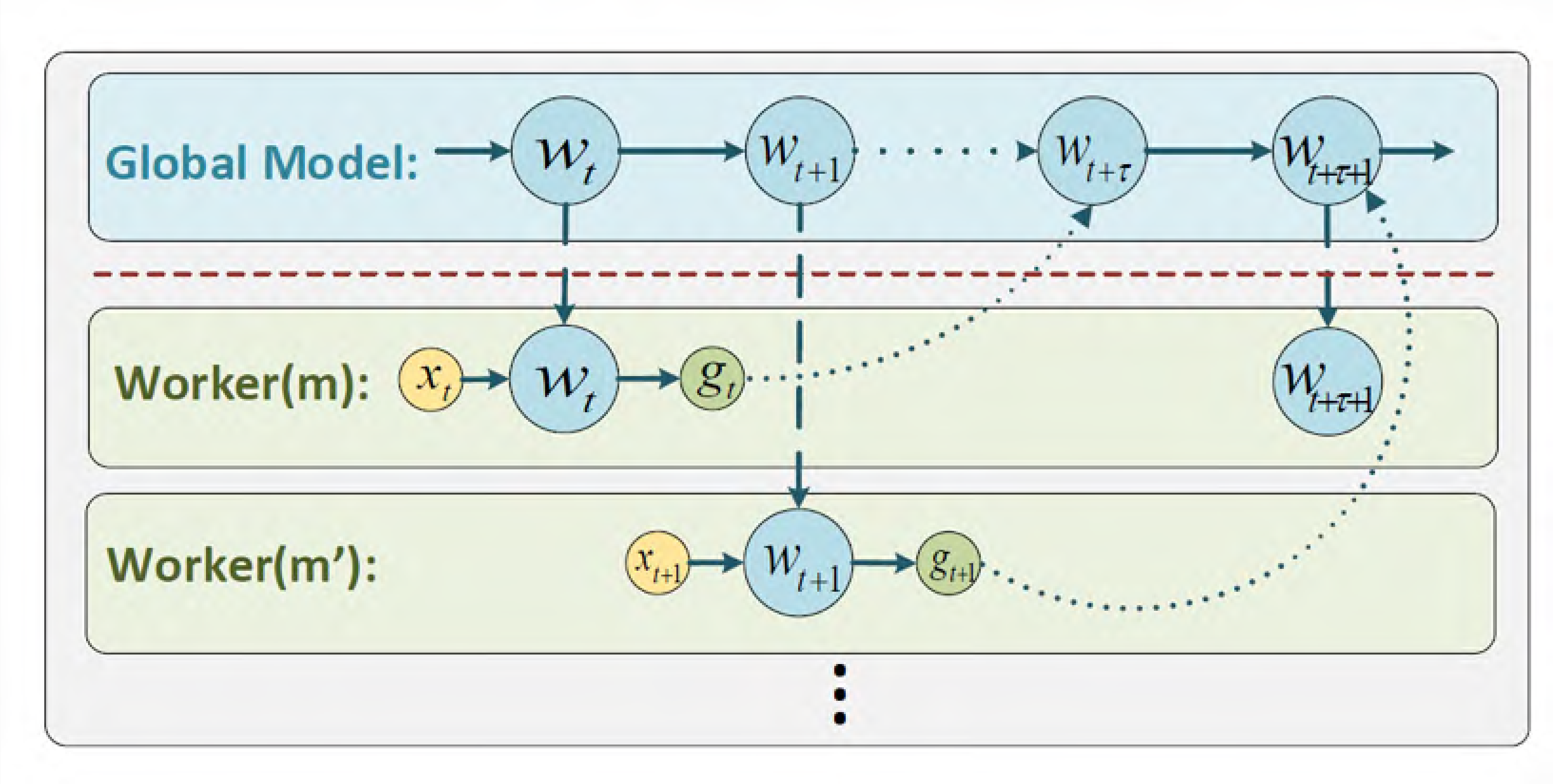
Asynchronous and Pipelined processing

- Client side pipeline on GPU side to enhance throughput
- Hide communication latency.
- Optimized iff the computational cost are equal to communicational cost



A issue for async algorithm

Delayed communication in asynchronous parallelization



- Sequential SGD
  $$w_{t+\tau+1} = w_{t+\tau} - \eta * g(w_{t+\tau})$$
- Async SGD
  $$w_{t+\tau+1} = w_{t+\tau} - \eta * g(w_t)$$

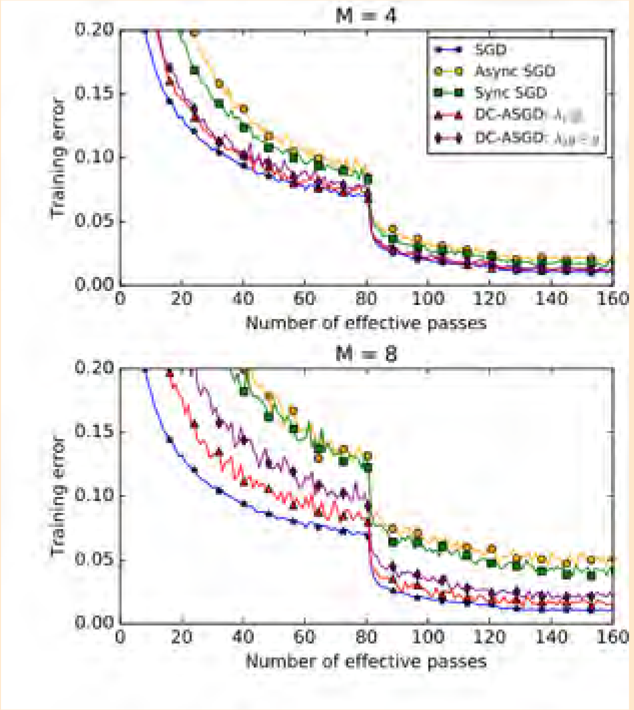
Delay Compensated ASGD (DC-ASGD)

ASGD: $w_{t+\tau+1} = w_{t+\tau} - \eta\, g(w_t)$

DC-ASGD: $w_{t+\tau+1} = w_{t+\tau} - \eta\, g(w_t) - \lambda\phi(g(w_t)) \cdot (w_{t+\tau} - w_t)$

A work that directly targets to handle the delay, and it is experimentally effective and with convergence analysis.

Training curve for a Resnet DNN model for cifra-10

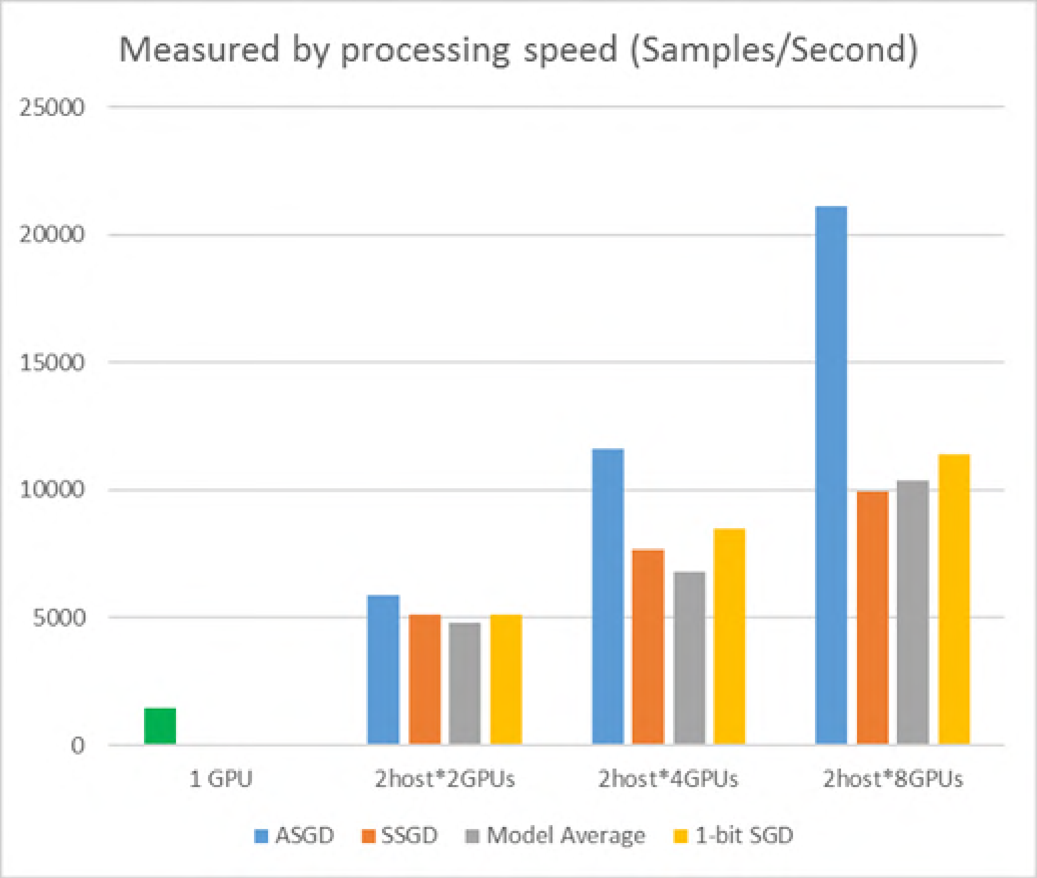


The experimental result

Microsoft

Multiverso parameter server

- High speed infrastructure
- Advanced async algorithm



Using ASGD in CNTK

Available on C++

Brainscript(CNTK receipt)

```
learningRatesPerSample = 0.0005

  ParallelTrain = [
    parallelizationMethod = DataParallelASGD
    distributedMBReading = true
    syncPerfStats = 20
    DataParallelASGD = [
      syncPeriodPerWorker=256
      usePipeline = true
      AdjustLearningRateAtBeginning = [
        adjustCoefficient = 0.2
        adjustNBMiniBatch = 1024
        # Learning rate will be adjusted to original one after ((1 / adjustCoefficient) * adjustNBMiniBatch) samples
        # which is 5120 in this case
      ]]]
```

Summary

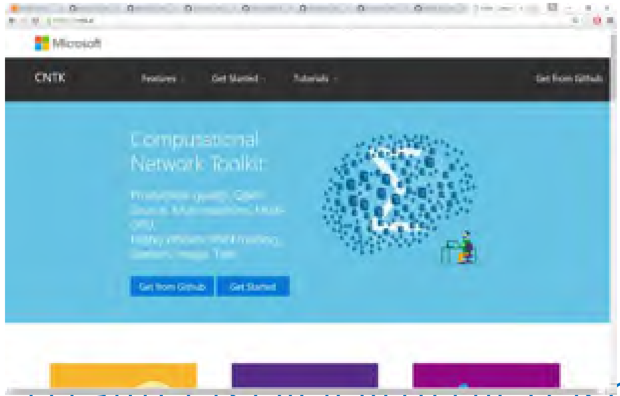
conclusion

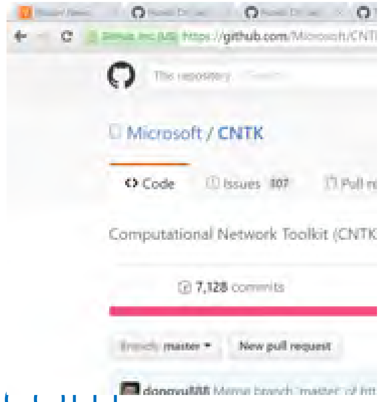
Microsoft

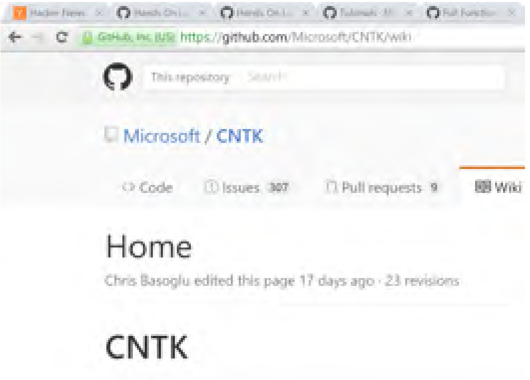
- CNTK is Microsoft's **open-source**, **cross-platform** toolkit for learning and evaluating **deep neural networks**.
  - Linux, Windows, docker, .Net
- CNTK expresses (nearly) **arbitrary neural networks** by composing simple building blocks into complex **computational networks**, supporting relevant network types and applications.
  - automatic differentiation, deferred computation, optimized execution and memory use
  - powerful description language, composability
  - implicit time; efficient static and recurrent NN training through batching
  - data parallelization, GPUs & servers: 1-bit SGD, Block Momentum
  - feed-forward DNN, RNN, LSTM, convolution, DSSM; speech, vision, text
- CNTK is **production-ready**: State-of-the-art accuracy, efficient, and scales to multi-GPU/multi-server.

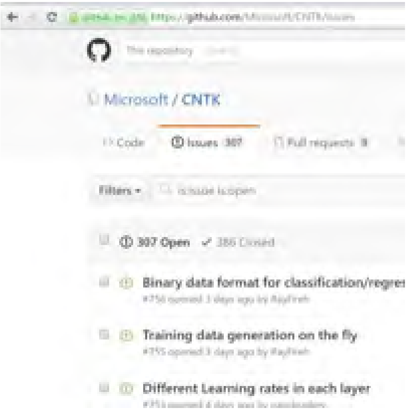
CNTK: democratizing the AI tool chain

- Web site: `https://cntk.ai/`
- Github: `https://github.com/Microsoft/CNTK`
- Wiki: `https://github.com/Microsoft/CNTK/wiki`
- Issues: `https://github.com/Microsoft/CNTK/issues`









Microsoft

Thanks

More information please find in:

http://cntk.ai
http://www.dmtk.io

Contact

Taifeng Wang – taifengw@Microsoft.com