CNTK—Microsoft's Open Source Deep Learning Toolkit

Taifeng Wang

Lead Researcher, Microsoft Research Asia

2016 GTC China

Deep learning in Microsoft

- Cognitive Services
  - https://how-old.net
  - http://www.captionbot.ai
- Skype Translator
- Bing
  - Cortana
  - ads
  - relevance
  - multimedia
  - …
- HoloLens
- Microsoft Research
  - speech, image, text

CNTK - Computational Network Toolkit

- CNTK is Microsoft's **open-source**, **cross-platform** toolkit for learning and evaluating **deep neural networks**.

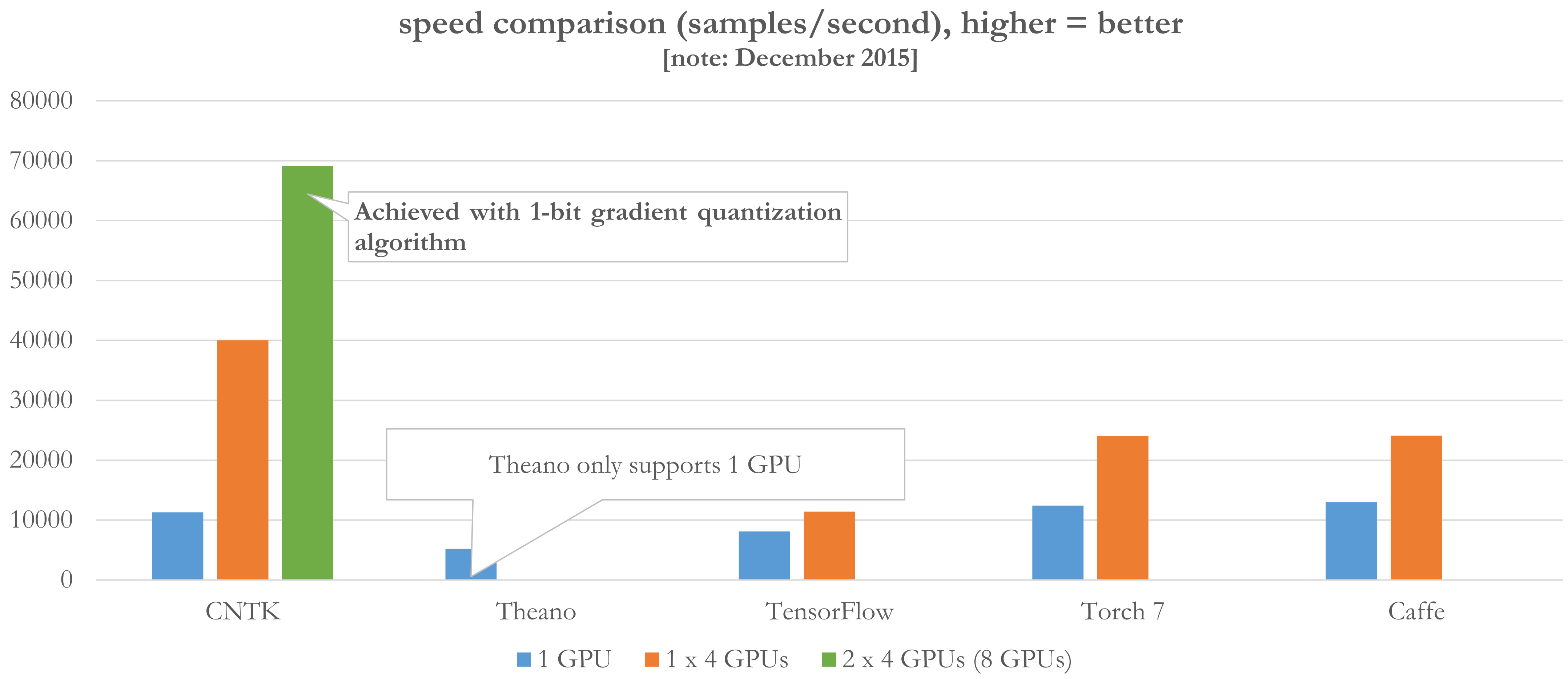
- CNTK expresses (nearly) **arbitrary neural networks** by composing simple building blocks into complex **computational networks**, supporting relevant network types and applications.

- CNTK is **production-ready**: State-of-the-art accuracy, efficient, and scales to multi-GPU/multi-server.

CNTK is Microsoft's open-source, cross-platform toolkit for learning and evaluating deep neural networks.

- open-source model inside and outside the company
  - created by Microsoft Speech researchers (Dong Yu et al.) 4 years ago; open-sourced (CodePlex) in early 2015
  - on GitHub since Jan 2016 under permissive license
  - nearly all development is out in the open
- growing use by Microsoft product groups
  - all have full-time employees on CNTK that actively contribute
  - CNTK trained models are already being tested in production, receiving real traffic
- external contributions e.g. from MIT and Stanford
- Linux, Windows, .Net, docker, cudnn5
  - Python, C++, and C# APIs coming soon

CNTK is production-ready: State-of-the-art accuracy, efficient, and scales to multi-GPU/multi-server.



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

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

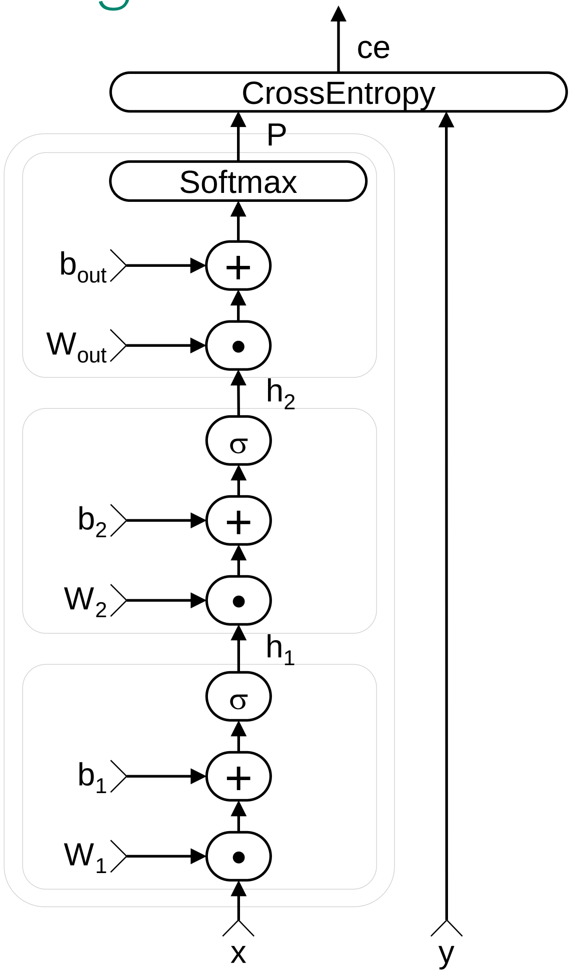
```
h1 = Sigmoid (W1   * x  + b1)
h2 = Sigmoid (W2   * h1 + b2)
P  = Softmax (Wout * h2 + bout)
```

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

$$ce = y^{\mathrm{T}} \log P$$
$$\sum_{\mathrm{corpus}} ce = \max$$

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

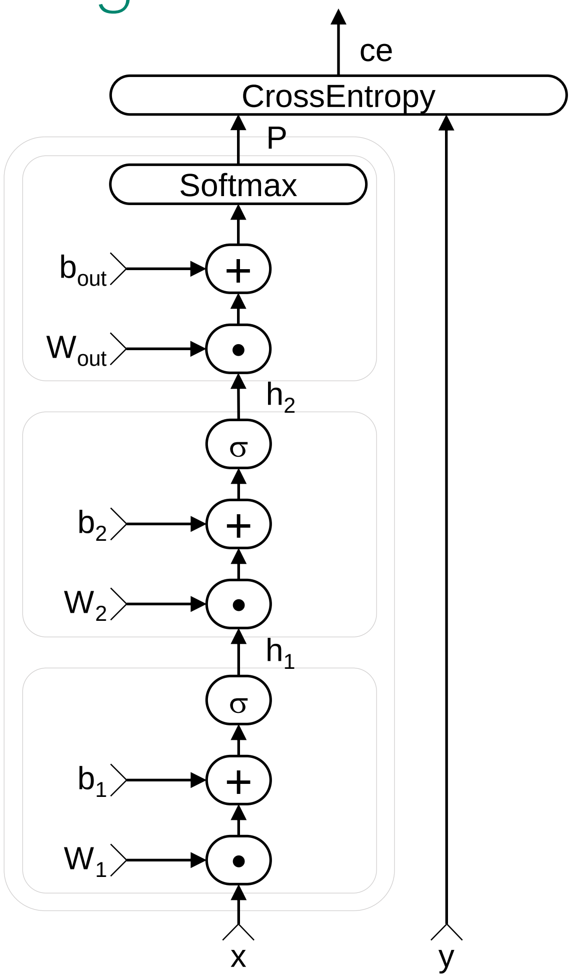
CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications.




```
h1 = Sigmoid (W1   * x  + b1)
h2 = Sigmoid (W2   * h1 + b2)
P  = Softmax (Wout * h2 + bout)
ce = CrossEntropy (y, P)
```

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
  - arbitrary-rank tensors with static and dynamic axes
  - automatic dimension inference
  - sparse-matrix support for inputs and labels
- automatic differentiation
  - $\partial \mathcal{F} / \partial \text{in} = \partial \mathcal{F} / \partial \text{out} \cdot \partial \text{out} / \partial \text{in}$
- deferred computation → execution engine
  - optimized execution
  - memory sharing
- editable

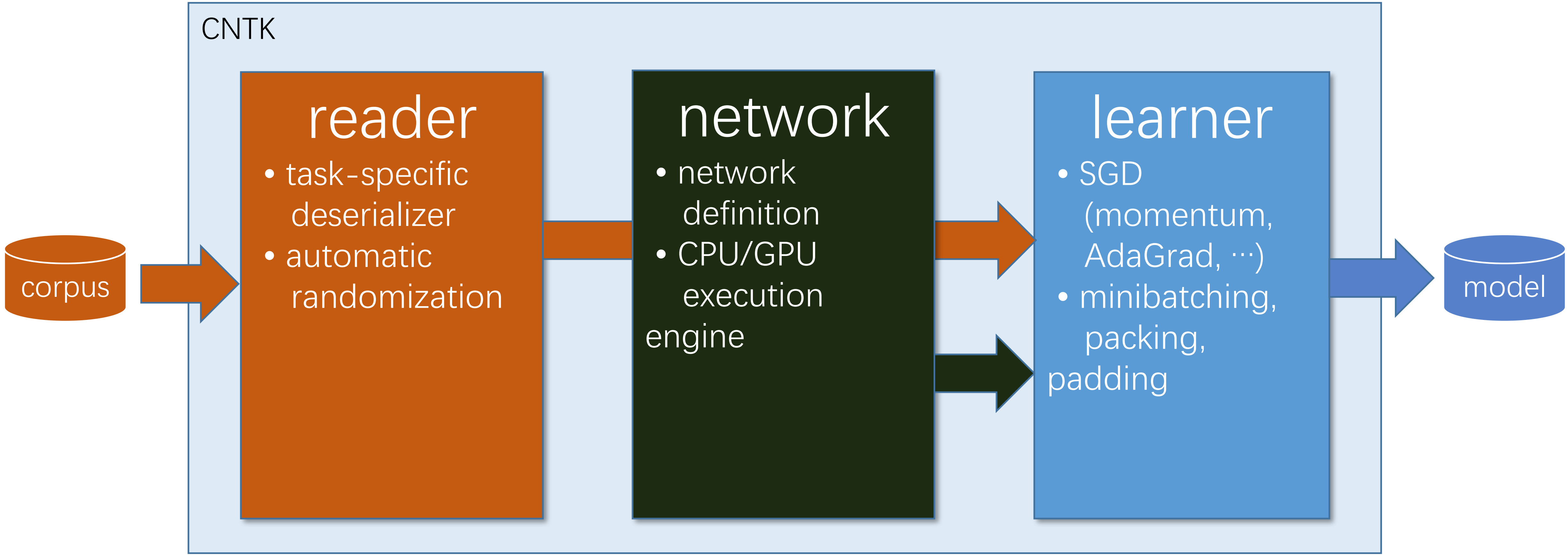
CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications.

- Lego-like composability allows CNTK to support a wide range of networks, e.g.
  - feed-forward DNN
  - RNN, LSTM
  - convolution
  - DSSM
  - sequence-to-sequence
- for a range of applications including
  - speech
  - vision
  - text
- and combinations

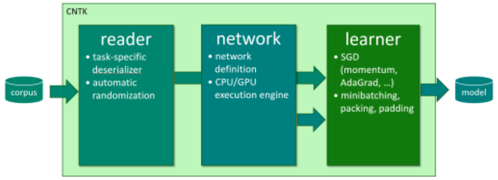
CNTK is production-ready: State-of-the-art accuracy, efficient, and scales to multi-GPU/multi-server.

- state-of-the-art accuracy on benchmarks and production models
- optimized for GPU
- multi-GPU/multi-server parallel training on production-size corpora

CNTK architecture



how to: top-level configuration

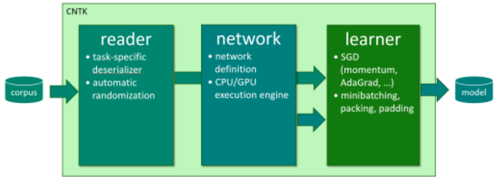



```
cntk configFile=yourConfig.cntk command="train:eval"
```

```
# content of yourConfig.cntk:
train = {
    action = "train"
    deviceId = "auto"
    modelPath = “$root$/models/model.dnn"

    reader = { … }
    BrainScriptNetworkBuilder = { … }
    SGD = { … }
}
eval = { … }
```

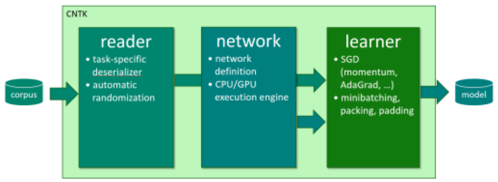
how to: reader

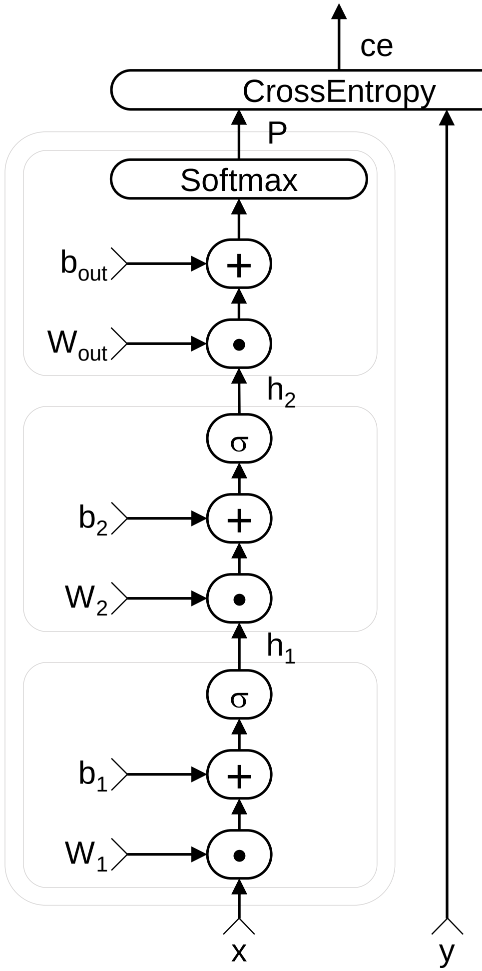



```
reader = {
    readerType = "ImageReader"
    file = "$ConfigDir$/train_map.txt"
    randomize = "auto"
    features = { width=224; height=224; channels=3; cropRatio=0.875 }
    labels = { labelDim=1000 }
}
```

- stock readers for images, speech (HTK), plain text, UCI
  - readers can be combined (e.g. image captioning)
  - custom format: implement IDeserializer
- automatic on-the-fly randomization
  - randomizes data in chunks, then runs rolling window
  - no need to pre-randomize; important for large data sets

how to: network

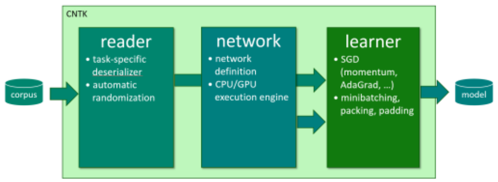


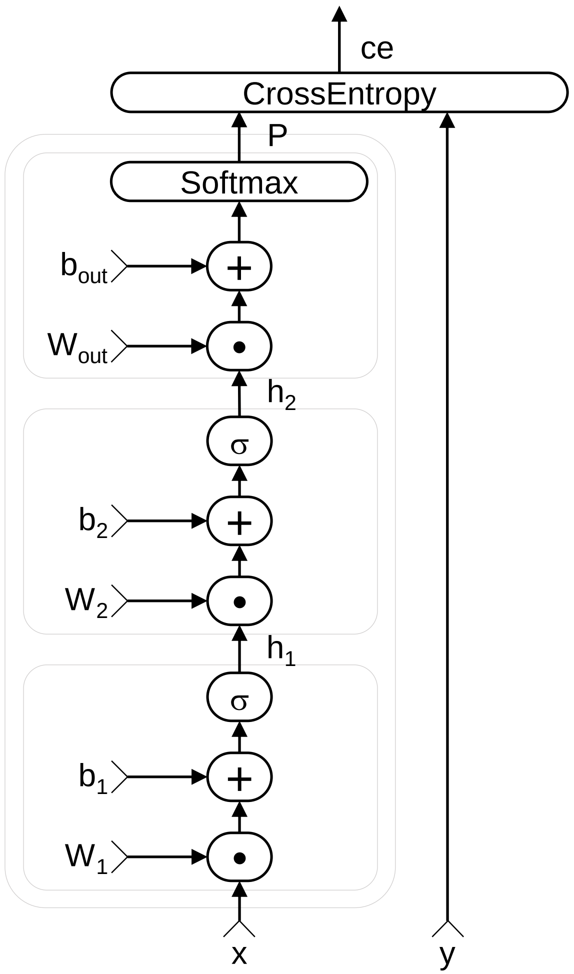



```
M = 40 ; N = 512 ; J = 9000 // feat/hid/out dim
x = Input{M} ; y = Input{J} // feat/labels
W1   = Parameter{N, M} ; b1   = Parameter{N}
W2   = Parameter{N, N} ; b2   = Parameter{N}
Wout = Parameter{J, N} ; bout = Parameter{J}

h1 = Sigmoid(W1   * x  + b1)
h2 = Sigmoid(W2   * h1 + b2)
P  = Softmax(Wout * h2 + bout)
ce = CrossEntropy(y, P)
```

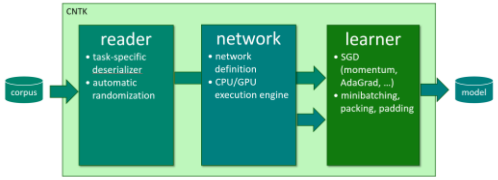
how to: network






```
M = 40 ; N = 512 ; J = 9000 // feat/hid/out dim
x = Input{M} ; y = Input{J} // feat/labels
Layer (x, out, in, act) = {   // reusable block
    W = Parameter{out, in} ; b = Parameter{out}
    h = act(W * x + b)
}.h
h1 = Layer(x,  N, M, Sigmoid)
h2 = Layer(h1, N, N, Sigmoid)
P  = Layer(h2, J, N, Softmax)
ce = CrossEntropy(y, P)
```

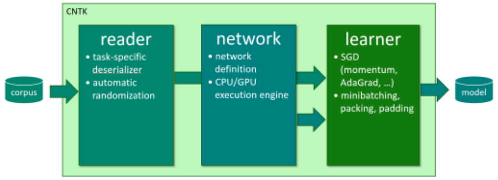
how to: learner




```
SGD = {
    maxEpochs = 50
    minibatchSize = $mbSizes$
    learningRatesPerSample = 0.007*2:0.0035
    momentumAsTimeConstant = 1100
    AutoAdjust = { … }
    ParallelTrain = { … }
}
```

- various model-update types like momentum, RmsProp, AdaGrad, …
- learning rate and momentum can be specified in MB-size agnostic way
- auto-adjustment of learning rate (e.g. “newbob”) and minibatch size
- multi-GPU/multi-server

how: typical workflow



- configure reader, network, learner
- train & evaluate , with parallelism:

```
mpiexec --np 16 --hosts server1,server2,server3,server4    \
CNTK configFile=myTask.cntk  command=MyTrain:MyTest parallelTrain=true deviceId=auto
```

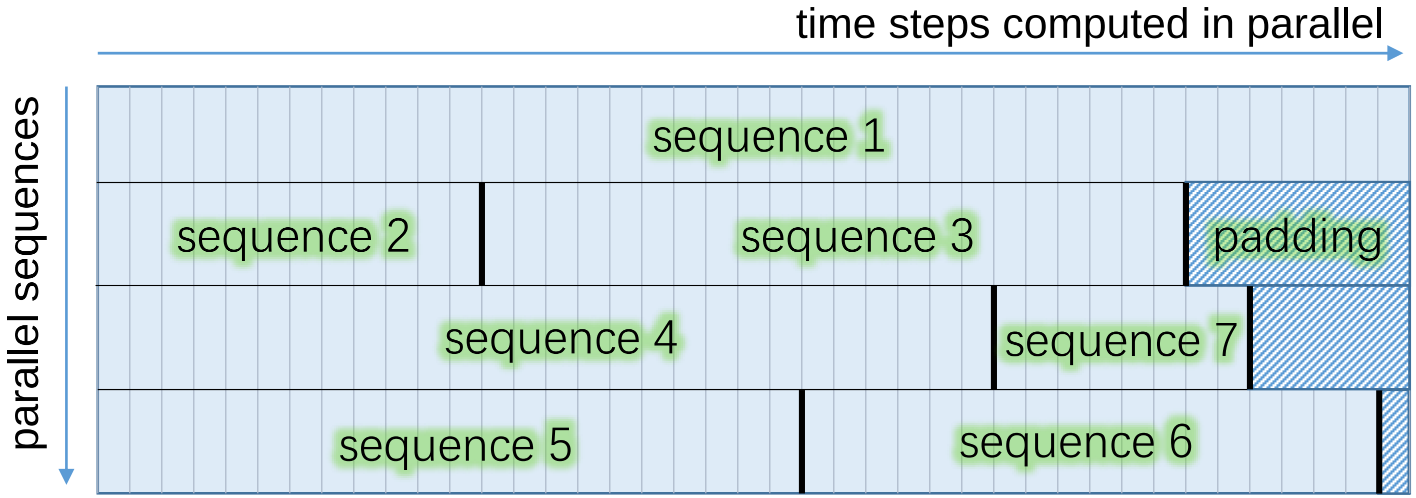
- modify models, e.g. for layer-building discriminative pre-training:
  - `CNTK configFile=myTask.cntk  command=MyTrain1:AddLayer:MyTrain2`
- apply model file-to-file:
  - `CNTK configFile=myTask.cntk  command=MyRun`
- use model from code: EvalDll.dll/.so (C++) or EvalWrapper.dll (.Net)

deep dive

- base features:
  - SGD with momentum, AdaGrad, Nesterov, etc.
  - computation network with automatic gradient
- higher-level features:
  - auto-tuning of learning rate and minibatch size
  - memory sharing
  - implicit handling of time
  - minibatching of variable-length sequences
  - data-parallel training
- you can do all this with other toolkits, but must write it yourself

deep dive: variable-length sequences

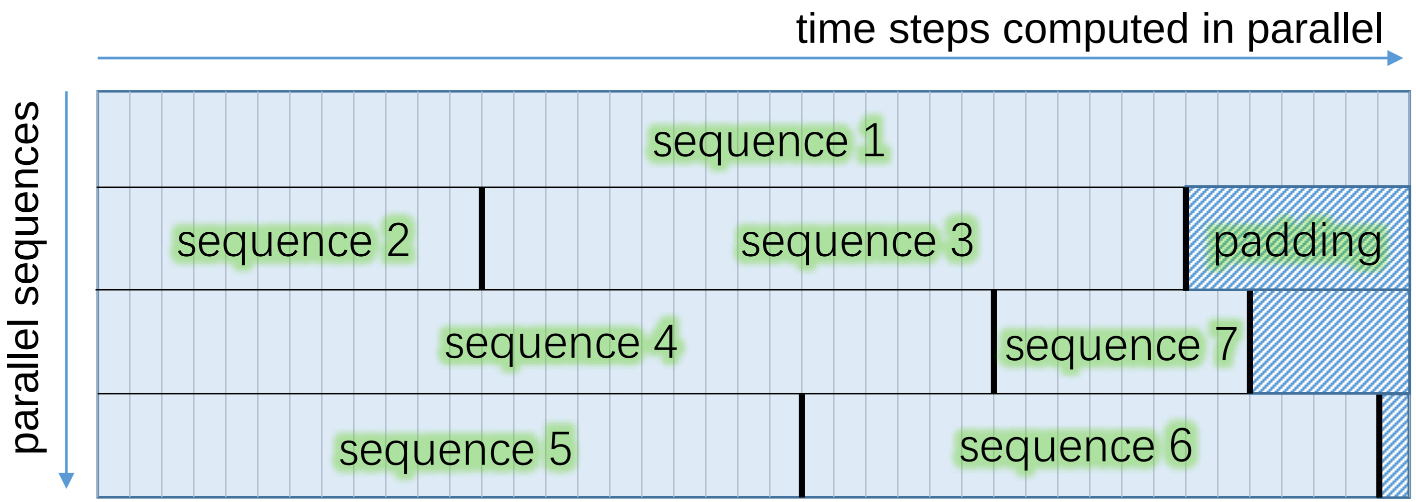
- minibatches containing sequences of different lengths are automatically packed *and padded*



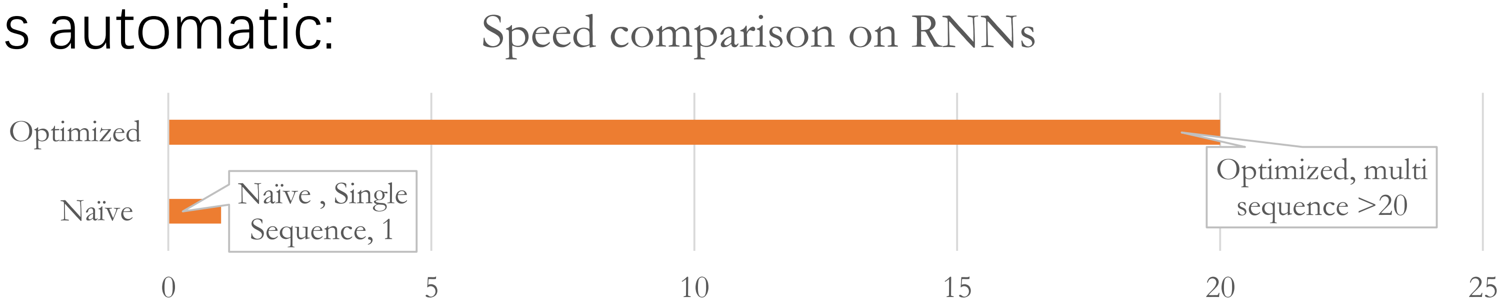
- CNTK handles the special cases:
  - PastValue operation correctly resets state and gradient at sequence boundaries
  - non-recurrent operations just pretend there is no padding (“garbage-in/garbage-out”)
  - sequence reductions

deep dive: variable-length sequences

- minibatches containing sequences of different lengths are automatically packed *and padded*



- speed-up is automatic:



Microsoft

deep dive: data-parallel training

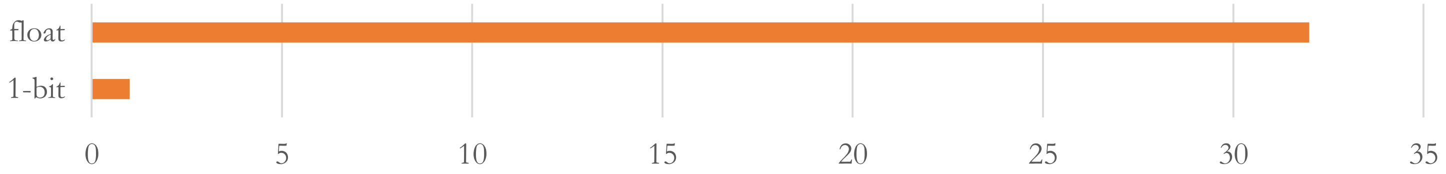
- data-parallelism: distribute each minibatch over workers, then aggregate
- challenge: communication cost
  - optimal iff
    compute and communication time per minibatch is equal (assuming overlapped processing)
- example: DNN, MB size 1024, 160M model parameters
  - compute per MB: 1/7 second
  - communication per MB: 1/9 second (640M over 6 GB/s)
  - can't even parallelize to 2 GPUs: communication cost already dominates!
- approach:
  - **communicate less** → 1-bit SGD
  - **communicate less often** → automatic MB sizing; Block Momentum

deep dive: 1-bit SGD

- quantize **gradients** to but **1 bit per value** with **error feedback**
  - carries over quantization error to next minibatch

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

Transferred Gradient (bits/value), smaller is better



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

deep dive: automatic minibatch scaling

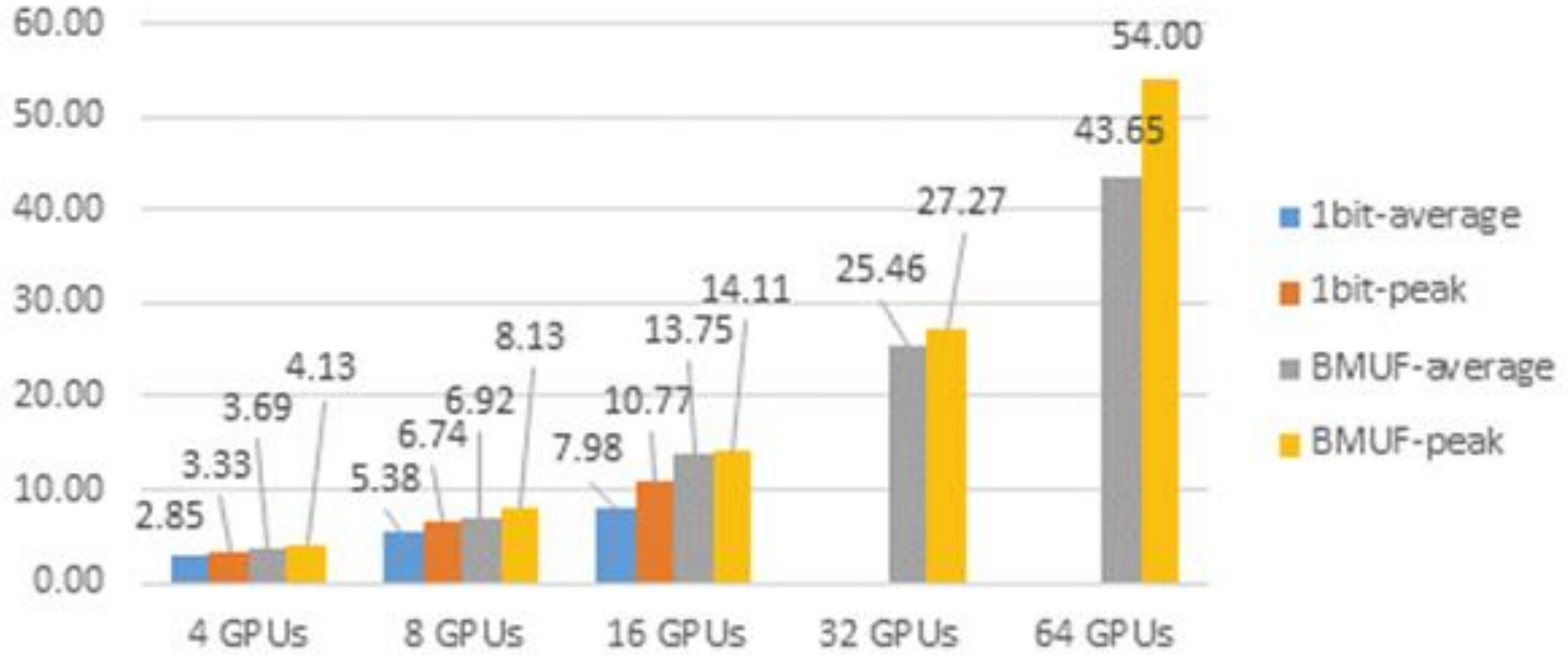
- goal: communicate less often
- every now and then try to grow MB size on small subset
  - important: keep contribution per sample and momentum effect constant
  - hence define learning rate and momentum in a MB-size agnostic fashion
- quickly scales up to MB sizes of 3k; runs at up to 100k samples

deep dive: Block Momentum

- very recent, very effective parallelization method
- goal: avoid to communicate after every minibatch
  - run a block of many minibatches without synchronization
  - then exchange and update with “block gradient”
- problem: taking such a large step causes divergence
- approach:
  - only add 1/K-th of the block gradient (K=#workers)
  - and carry over the missing (1-1/K) *to the next block update* (error residual like 1-bit SGD)
  - same as the common momentum formula

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

deep dive: data-parallel training



LSTM SGD baseline	11.08				
Parallel Algorithms	4-GPU	8-GPU	16-GPU	32-GPU	64-GPU
1bit	10.79	10.59	11.02		
BMUF	10.82	10.82	10.85	10.92	11.08

Table 2: WERs (%) of parallel training for LSTMs

[Yongqiang Wang, IPG; internal communication]

Microsoft

conclusion

- CNTK is Microsoft's **open-source**, **cross-platform** toolkit for learning and evaluating **deep neural networks**.
  - Linux, Windows, docker, .Net
  - growing use and contribution by various product teams
- 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有关材料

http://www.cntk.ai

https://github.com/microsoft/cntk/wiki

Thanks!

taifengw@microsoft.com

https://www.microsoft.com/en-us/research/people/taifengw/