Introduction to CNTK: Microsoft's Open-Source Deep-Learning Toolkit

Frank Seide
Principal Researcher Speech Recognition, CNTK Architect
Microsoft Research

Amit Agarwal
CNTK Principal Software Engineer
Microsoft Technology And Research

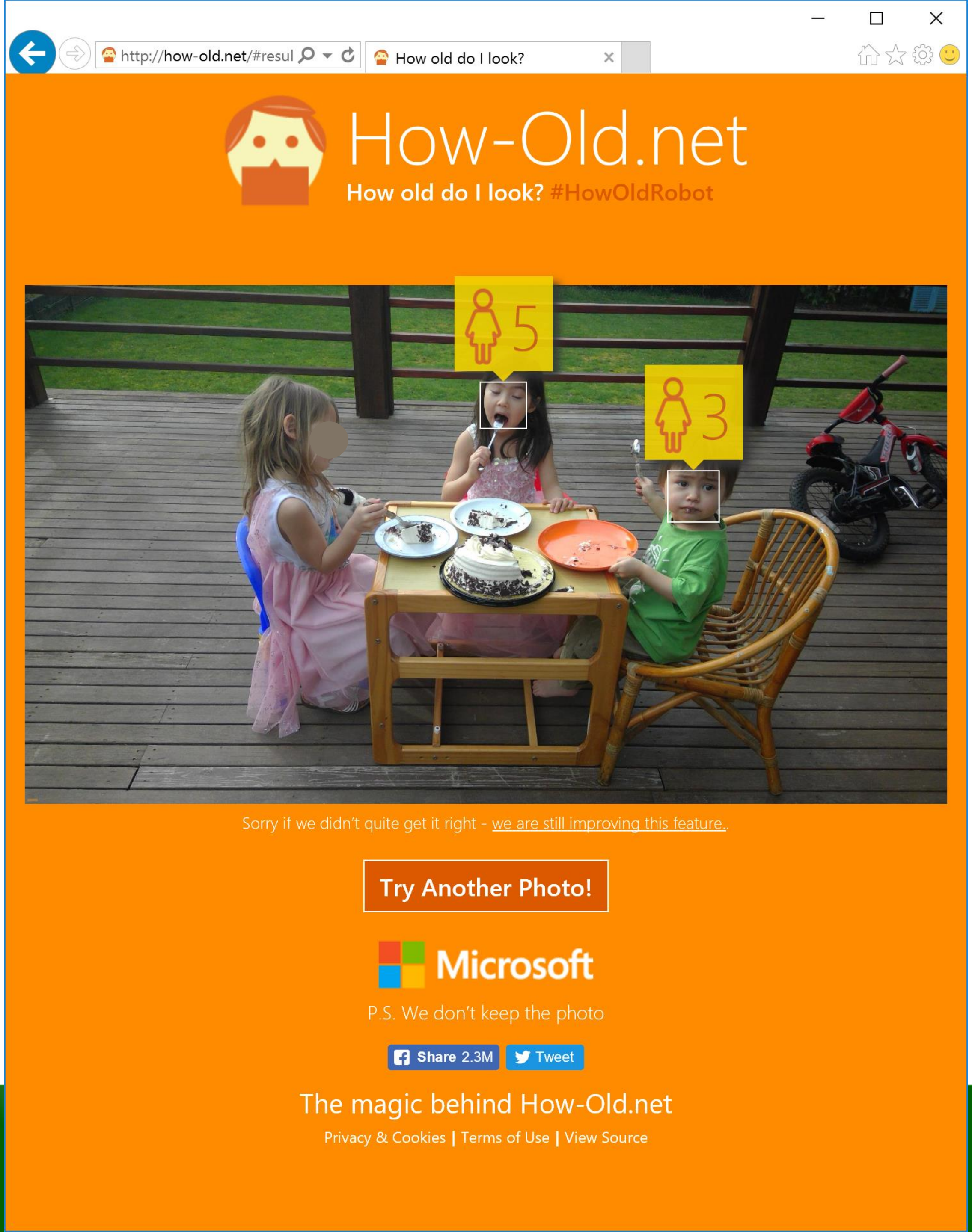
With many contributors:

E. Akchurin, C. Basoglu, G. Chen, S. Cyphers, J. Droppo, A. Eversole, B. Guenter, M. Hillebrand, X. Huang, Z. Huang, V. Ivanov, A. Kamenev, P. Kranen, O. Kuchaiev, W. Manousek, A. May, B. Mitra, O. Nano, G. Navarro, A. Orlov, H. Parthasarathi, B. Peng, M. Radmilac, A. Reznichenko, M. Seltzer, M. Slaney, A. Stolcke, H. Wang, K. Yao, D. Yu, Y. Zhang, G. Zweig

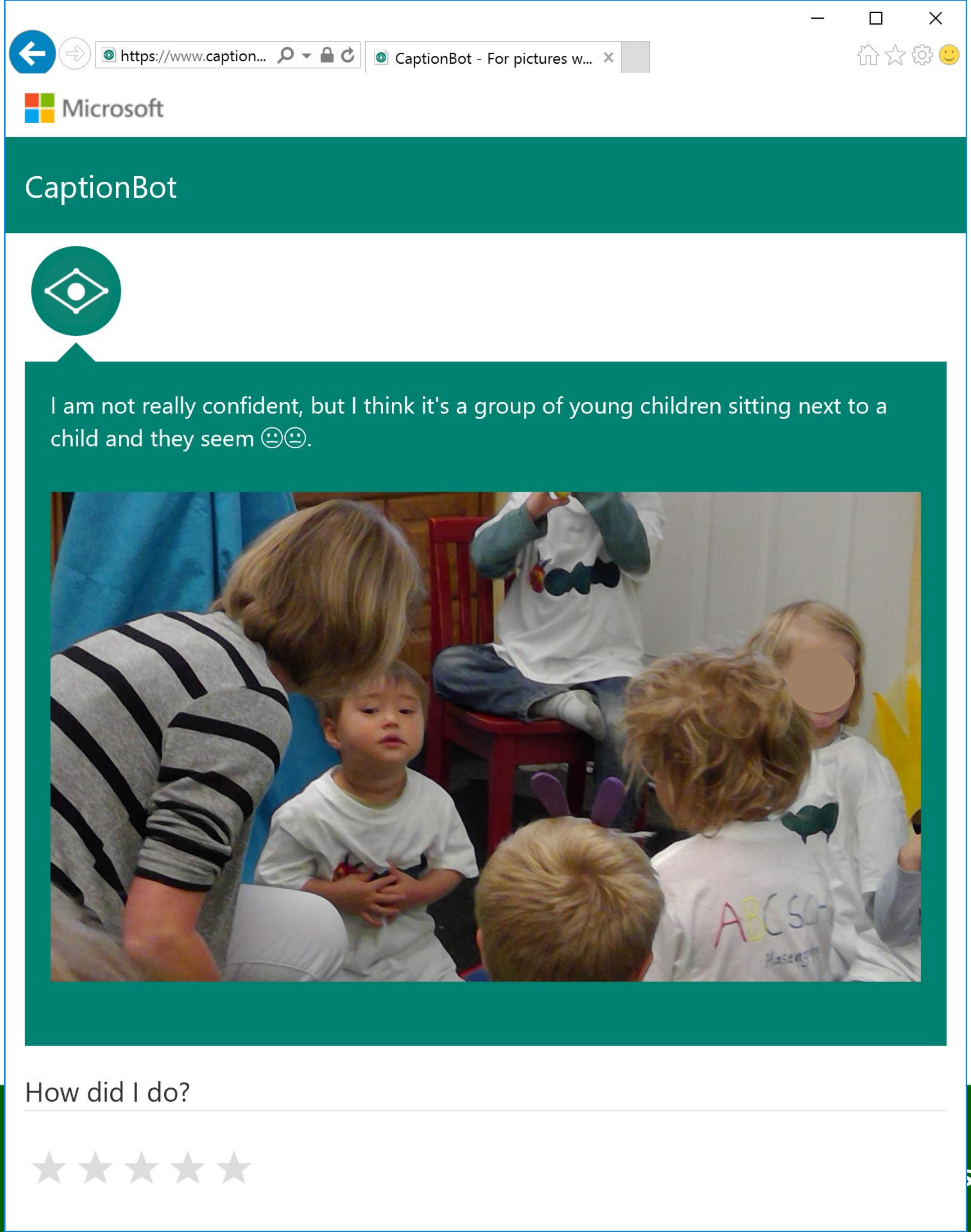
Microsoft

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

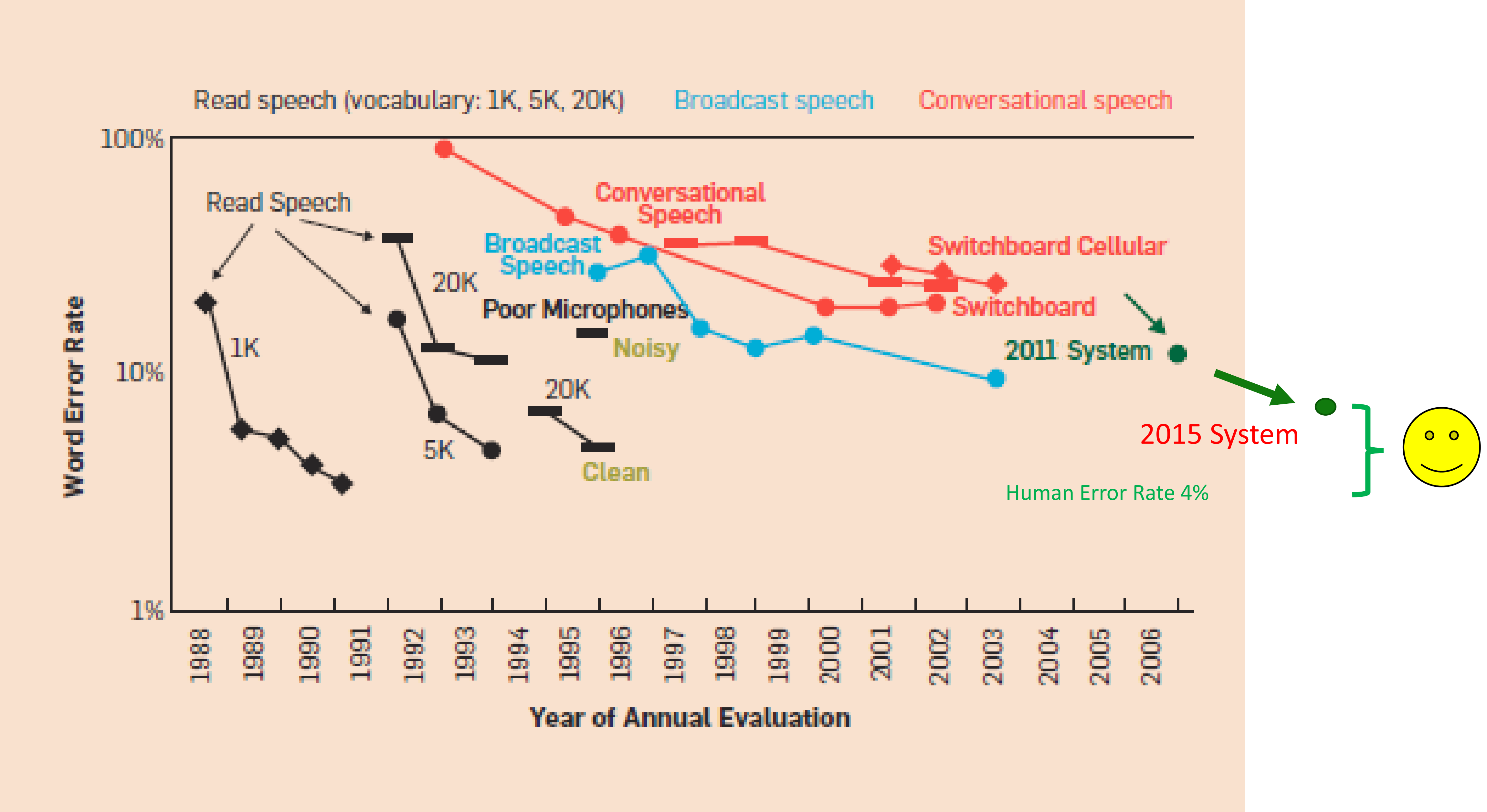


http://how-old.net/#resul
How old do I look?
How-Old.net
How old do I look? #HowOldRobot
5
3
Sorry if we didn't quite get it right - we are still improving this feature..
Try Another Photo!
Microsoft
P.S. We don't keep the photo
Share 2.3M
Tweet
The magic behind How-Old.net
Privacy & Cookies | Terms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

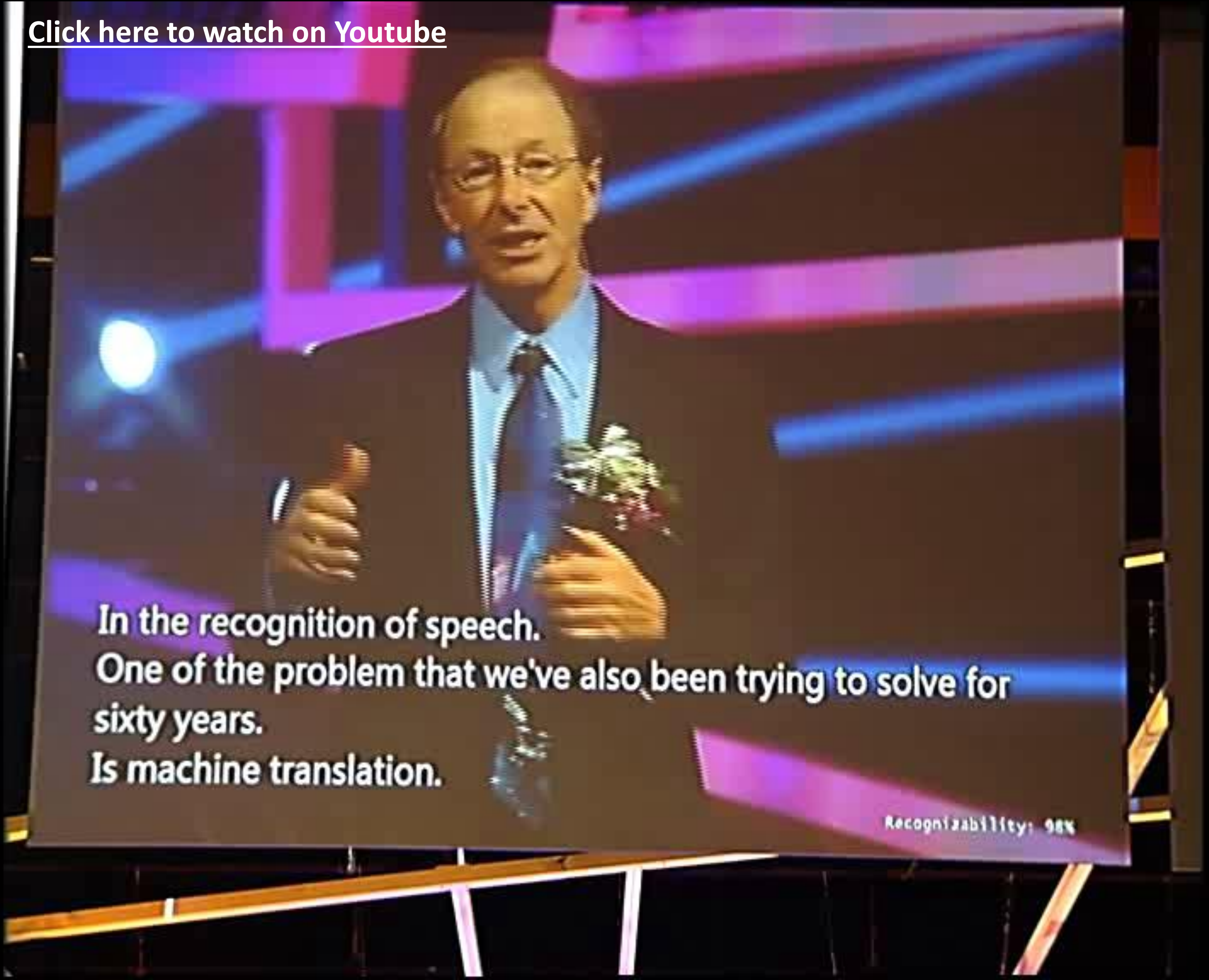


https://www.caption...
CaptionBot - For pictures w...
Microsoft
CaptionBot
I am not really confident, but I think it's a group of young children sitting next to a child and they seem 😐😐.
How did I do?

Figure 1. Historical progress of speech recognition word error rate on more and more difficult tasks.[10] The latest system for the switchboard task is marked with the green dot.

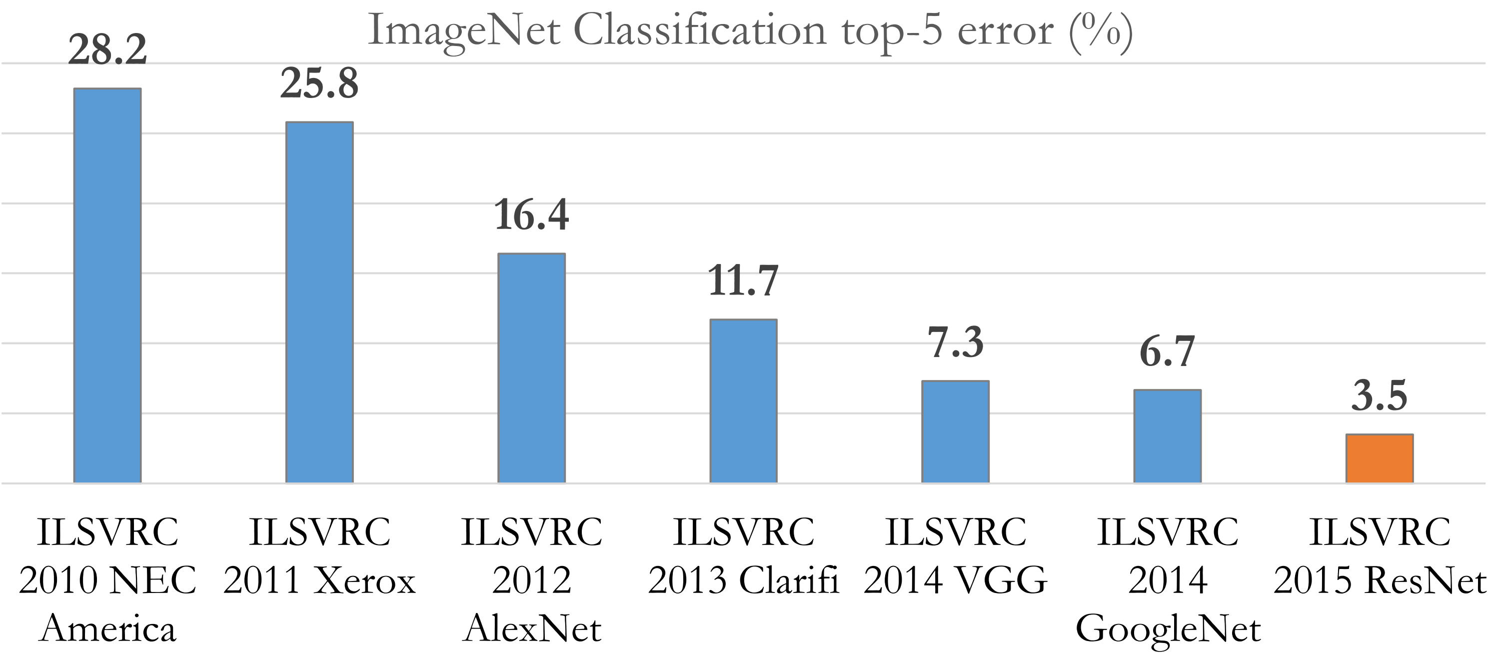


Click here to watch on Youtube



Click here to watch on Youtube

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

Microsoft

I. what is CNTK
II. how to use CNTK
III. deep dive into CNTK technologies

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.

Microsoft

“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 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

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

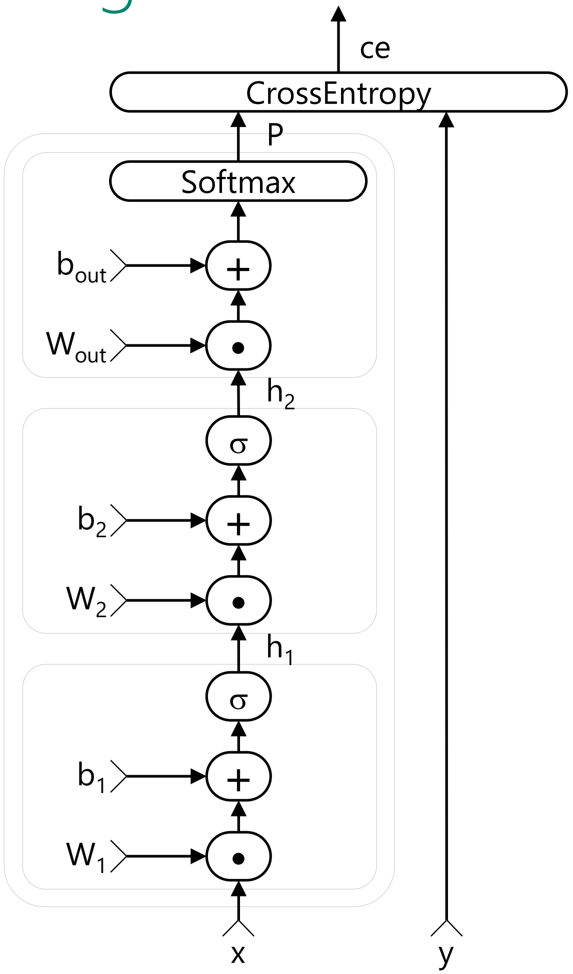
```
h1 = Sigmoid (W1   * x  + b1)
h2 = Sigmoid (W2   * h1 + b2)
P  = Softmax (Wout * h2 + bout)
```

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

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

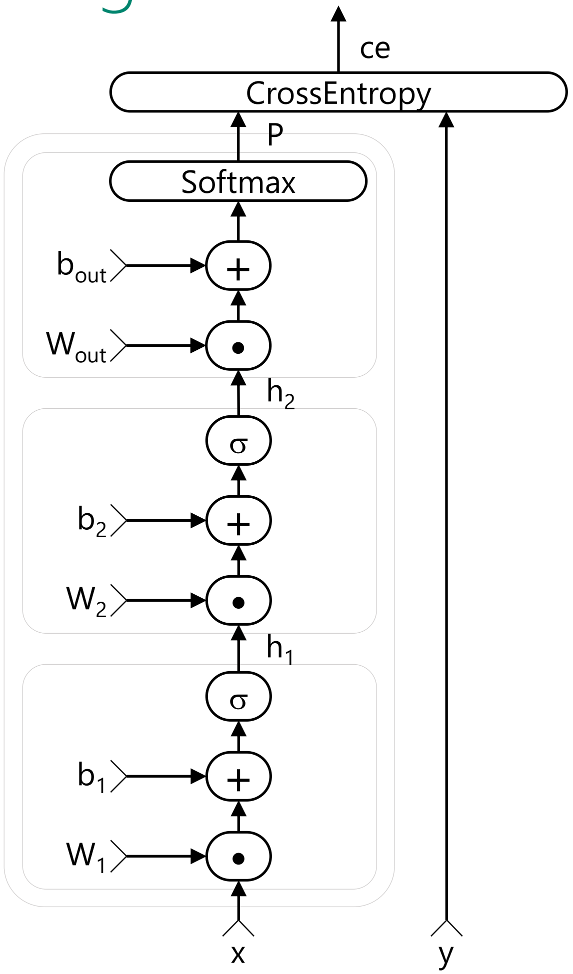
```
ce = CrossEntropy (y, P)
```

“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)
```

“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
  - $\partial\mathcal{F} / \partial in = \partial\mathcal{F} / \partial out \cdot \partial out / \partial in$
- deferred computation → execution engine
  - optimized execution
  - memory sharing
- editable

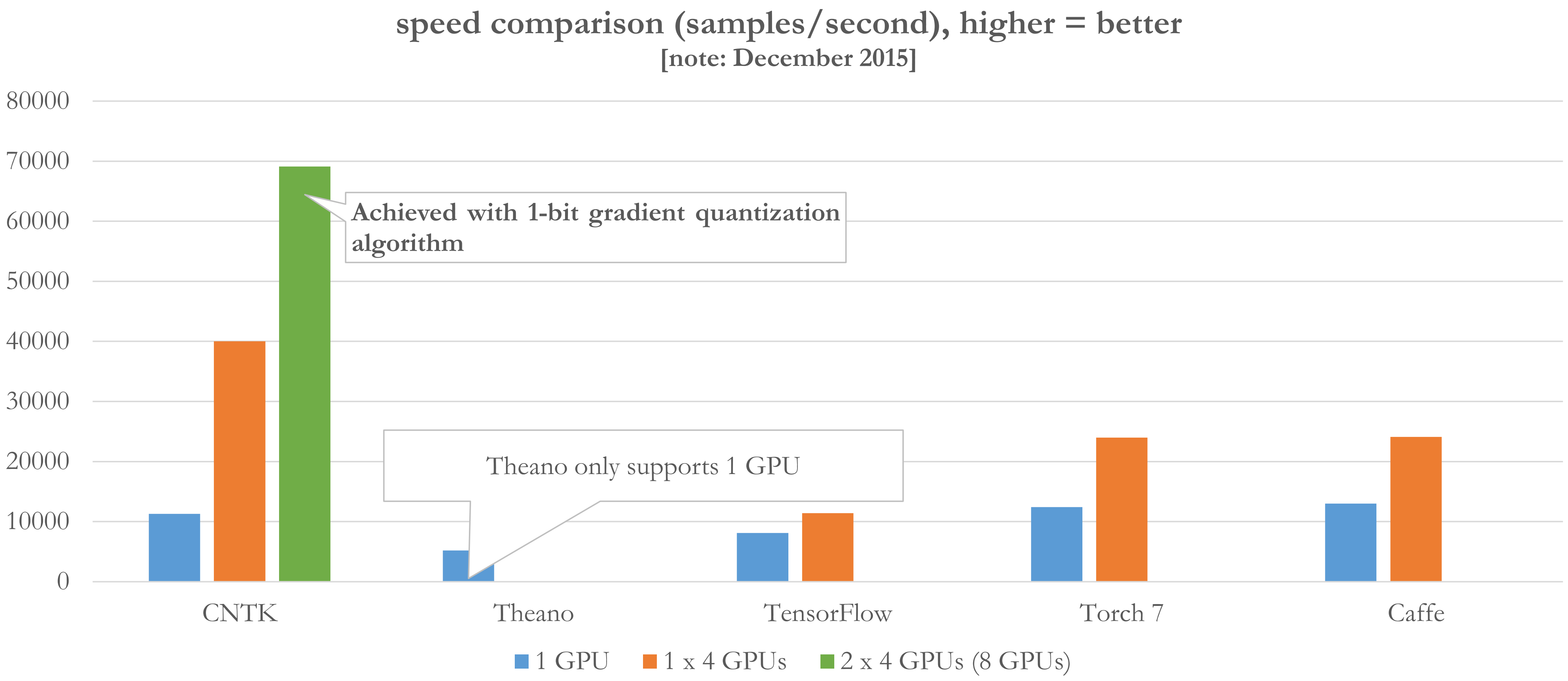
“CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications.”

- 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.”

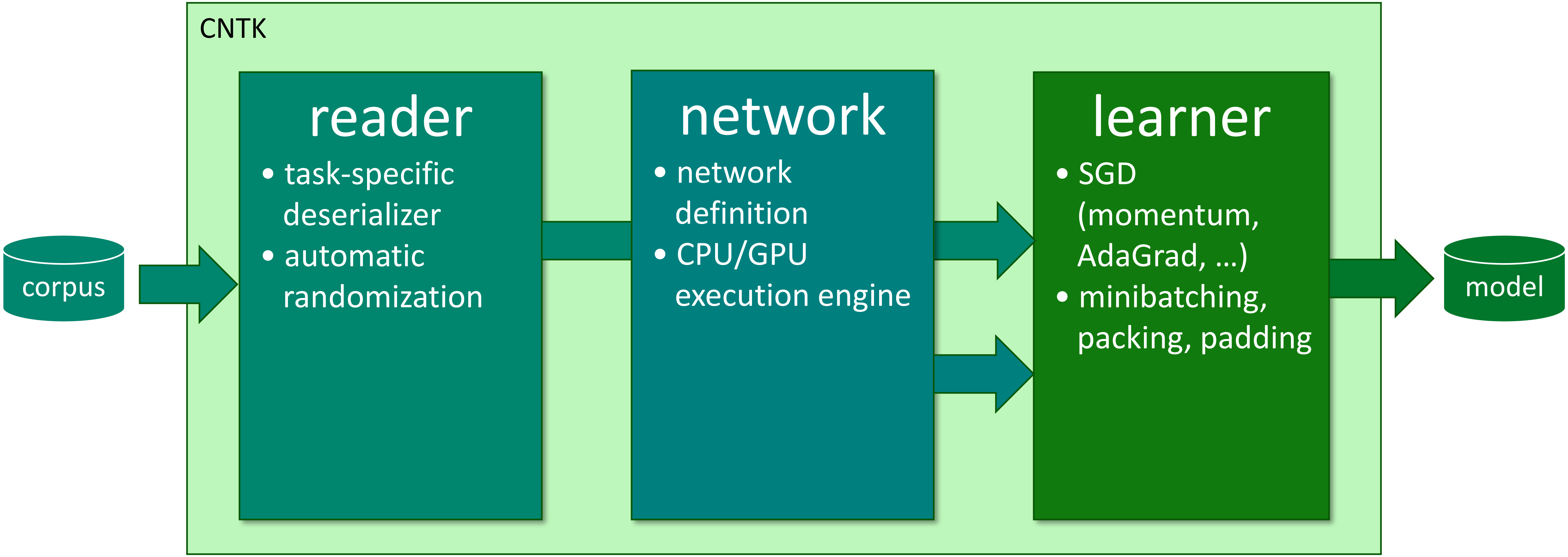
- state-of-the-art accuracy on benchmarks and production models
- optimized for GPU
- multi-GPU/multi-server parallel training on production-size corpora

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

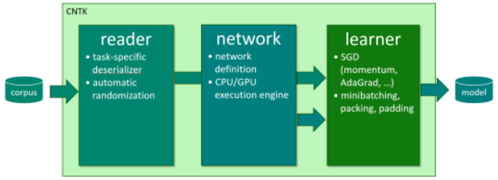


I. what
II. how to
III. deep dive

how to: CNTK architecture



how to: top-level configuration

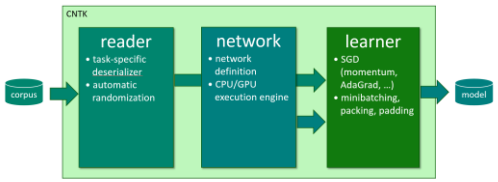



```
cntk configFile=yourConfig.cntk command="train:eval" root="exp-1"
```

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

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

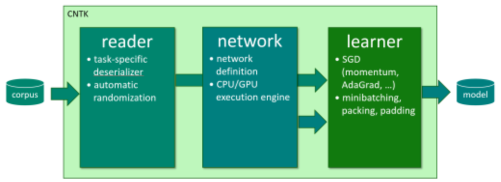
how to: reader




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

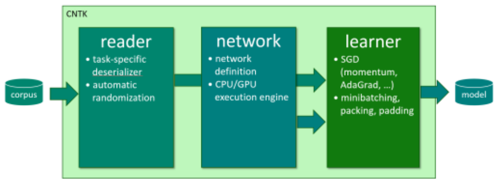
- stock readers for images, speech (HTK), plain text, UCI
  - readers can be combined (e.g. image captioning)
  - custom format: implement IDeserializer
- automatic on-the-fly randomization
  - randomizes data in chunks, then runs rolling window
  - no need to pre-randomize; important for large data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

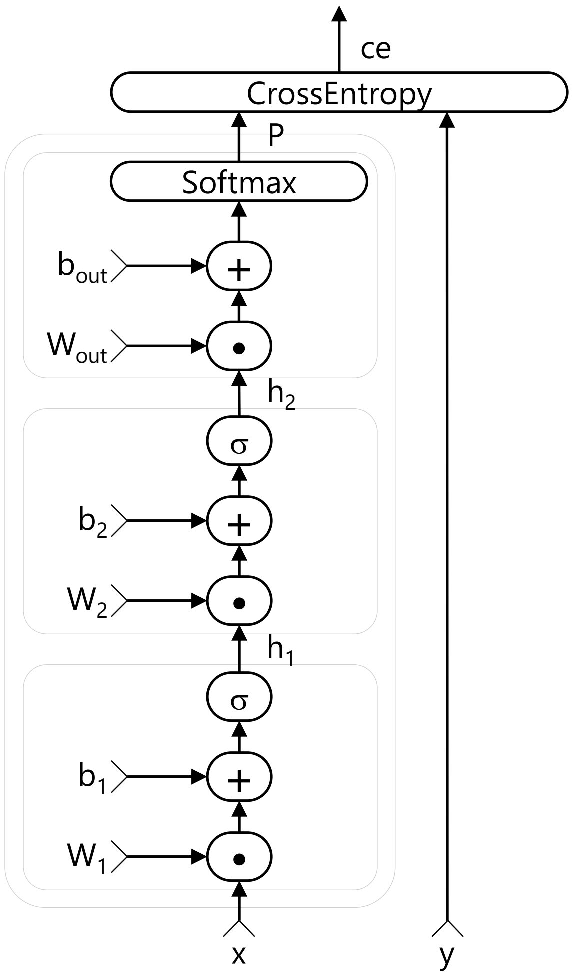
how to: network



- network specification consists of:
  - the network function’s formula
    - including learnable parameters
    - (but no gradients, which are automatically determined by the system)
  - inputs
  - the output(s) and training/evaluation criteria
- network descriptions are called “brain scripts”
  - custom network description language “BrainScript”
  - can soon be done using Python, C++, and C#/.Net

how to: network



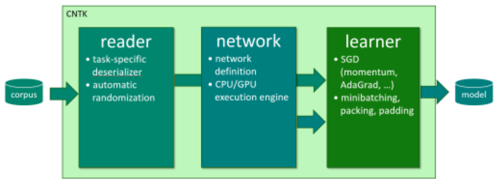


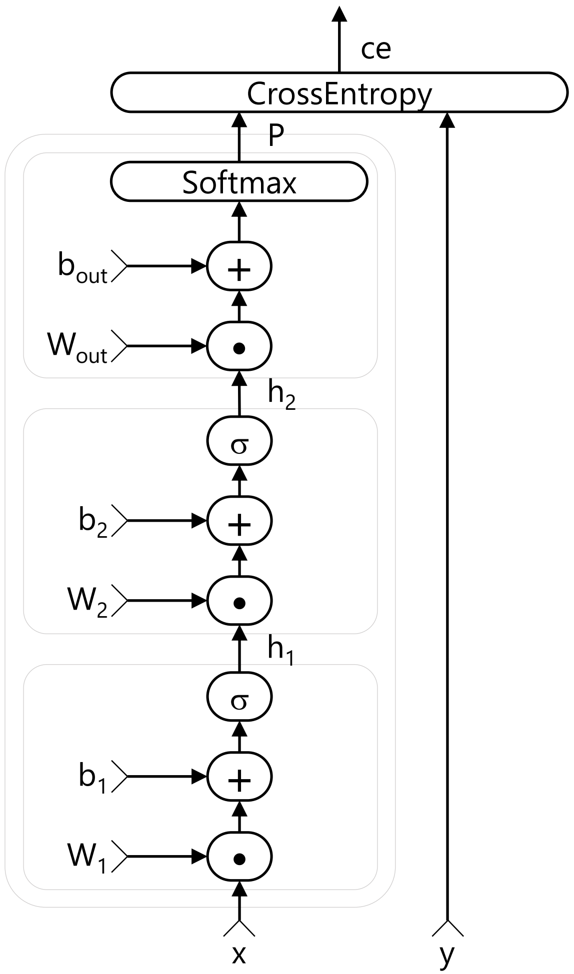

```
M = 40 ; N = 512 ; J = 9000 // feat/hid/out dim
x = Input{M} ; y = Input{J} // feat/labels
W1   = Parameter{N, M} ; b1   = Parameter{N}
W2   = Parameter{N, N} ; b2   = Parameter{N}
Wout = Parameter{J, N} ; bout = Parameter{J}

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

Microsoft

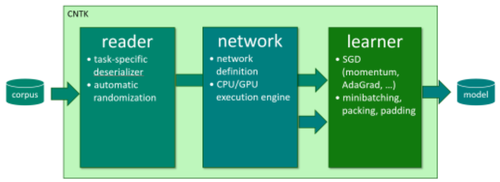
how to: network

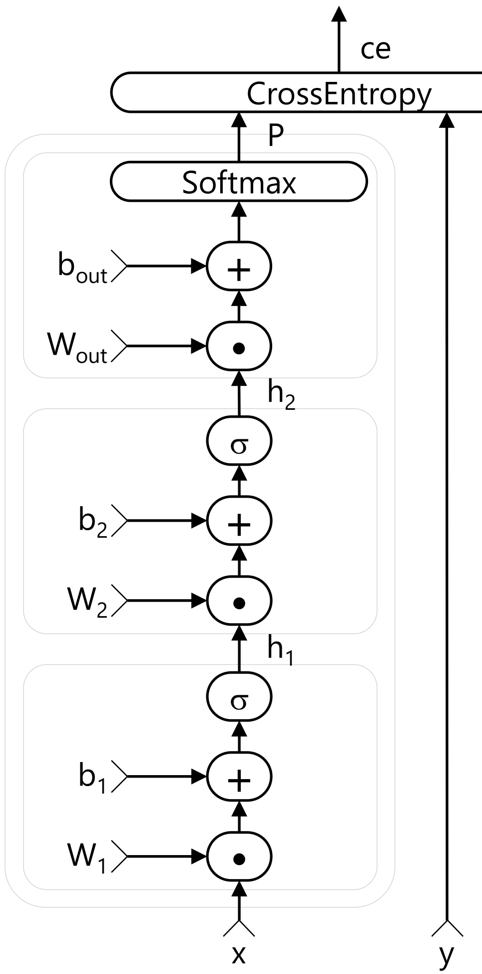





```
M = 40 ; N = 512 ; J = 9000 // feat/hid/out dim
x = Input{M} ; y = Input{J} // feat/labels
Layer (x, out, in, act) = {   // reusable block
    W = Parameter{out, in} ; b = Parameter{out}
    h = act(W * x + b)
}.h
h1 = Layer(x,  N, M, Sigmoid)
h2 = Layer(h1, N, N, Sigmoid)
P  = Layer(h2, J, N, Softmax)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

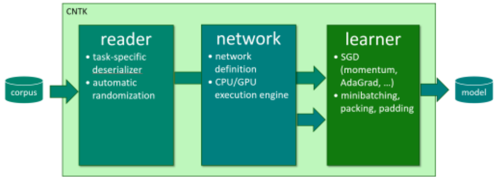
how to: network

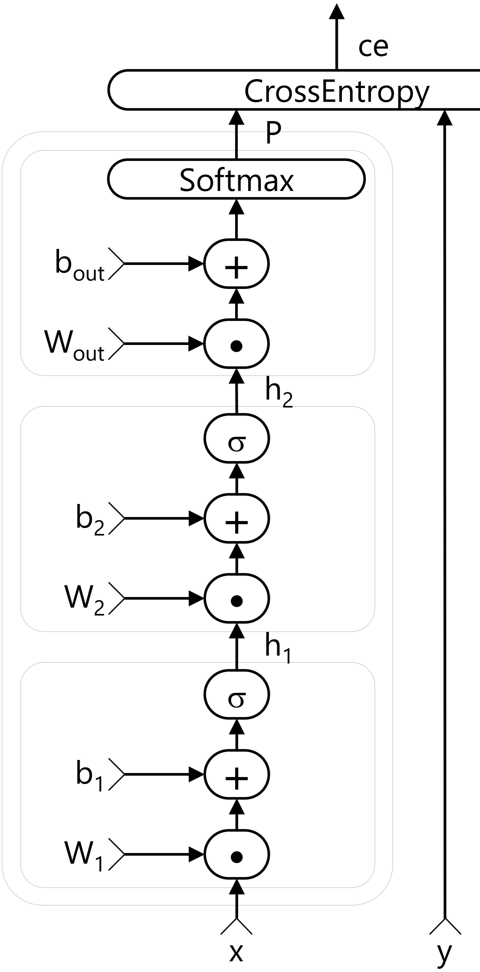





```
M = 40 ; N = 512 ; J = 9000 ; L = 2
x = Input{M} ; y = Input{J} // feat/labels
Layer (x, out, in, act) = { … }
DNNStack (x, out, in, L) =
    if L == 1 then Layer (x, out, in, Sigmoid)
    else Layer (DNNStack (x, out, in, L-1),
                out, out, Sigmoid)
hL = DNNStack(x, M, N, L)   // parameterized
P  = Layer(hL, J, N, Softmax)
ce = CrossEntropy(y, P)
```

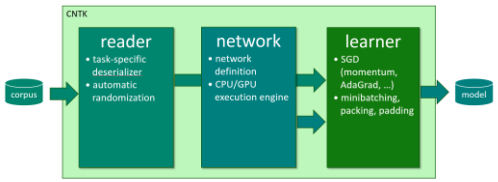
how to: network

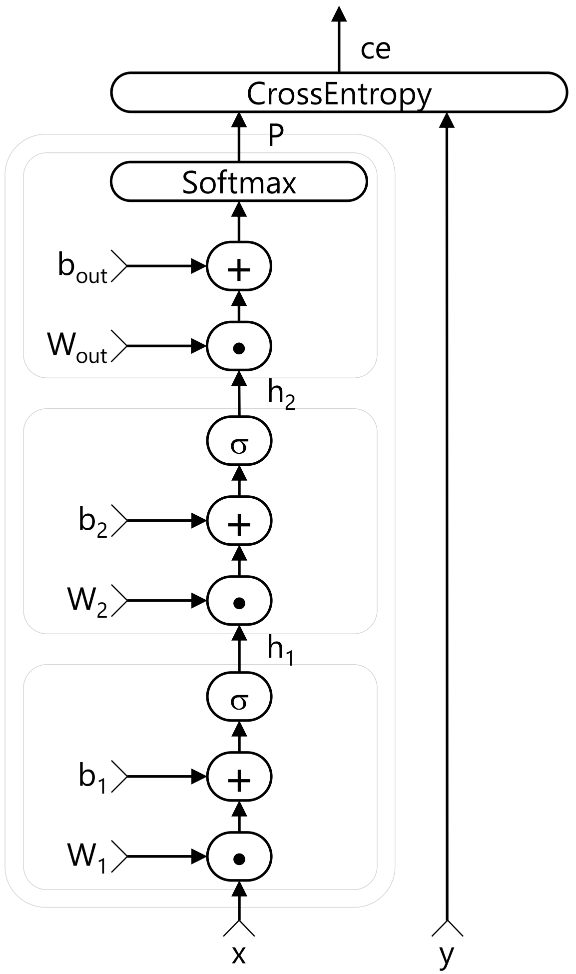





```
M = 40 ; N = 512 ; J = 9000 // feat/hid/out dim
x = Input{M} ; y = Input{J} // feat/labels
DenseLayer {Nout, activation=Identity} = {
    W = Parameter(Nout, 0) ; b = Parameter(Nout)
    apply(x) = activation(W * x + b)
}.apply
h1 = DenseLayer{N, activation=Sigmoid}(x)
h2 = DenseLayer{N, activation=Sigmoid}(h1)
P  = DenseLayer{J, activation=Softmax}(h2)
ce = CrossEntropy(y, P)
```

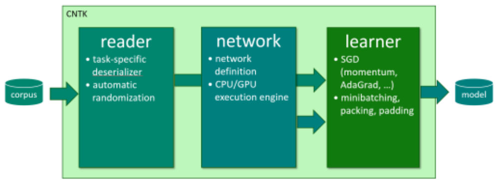
how to: network

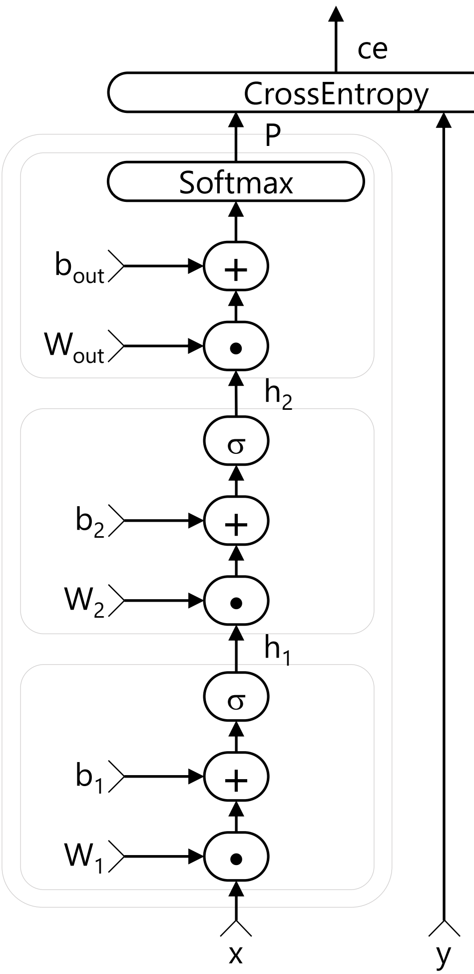





```
M = 40 ; N = 512 ; J = 9000 // feat/hid/out dim
x = Input{M} ; y = Input{J} // feat/labels
DenseLayer {out, activation=Identity} = { … }
Sequential (fnArray) = { … }
model = Sequential (
    DenseLayer{N, activation=Sigmoid} :
    DenseLayer{N, activation=Sigmoid} :
    DenseLayer{J, activation=Softmax}
)
P = model (x)
ce = CrossEntropy(y, P)
```

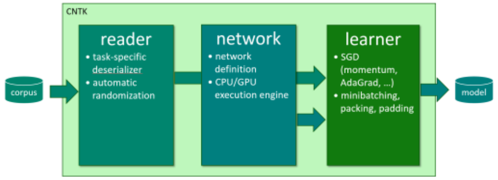
how to: network





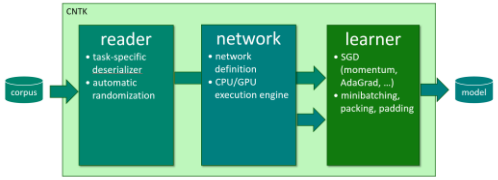

```
M = 40 ; N = 512 ; J = 9000 // feat/hid/out dim
x = Input{M} ; y = Input{J} // feat/labels
DenseLayer {out, activation=Identity} = { … }
Sequential (fnArray) = [ … ]
model = Sequential (
    DenseLayer{N} : Sigmoid :
    DenseLayer{N} : Sigmoid :
    DenseLayer{J} : Softmax
)
P = model (x)
ce = CrossEntropy(y, P)
```

how to: network



- CNTK BrainScript:
  - straight-forward, easily understandable syntax
  - custom functions and function objects for reusable modules, e.g. layers
  - allows high-level composability
- soon, BrainScripts can be written in Python, C++, and .Net

how to: “BrainScript??”

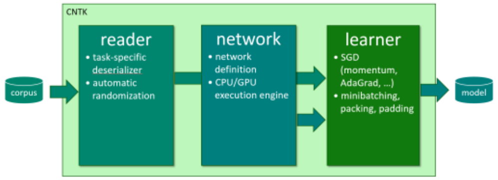


- *full name* perfectly expresses our grand *long-term ambition*

- *two-letter acronym* perfectly expresses *today’s state* of the degree that artificial neural networks actually implement brains

☺

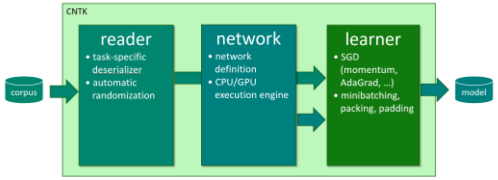
how to: learner




```
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)
    ```

I. what
II. how to
III. deep dive

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: handling of time

extend our example to an RNN

$$h_1(t) = \sigma(W_1\, x(t) \qquad\qquad\qquad + b_1)$$

$$h_2(t) = \sigma(W_2\, h_1(t) \qquad\qquad\qquad + b_2)$$

$$P(t) = \mathrm{softmax}(W_{\mathrm{out}}\, h_2(t) + b_{\mathrm{out}})$$

$$ce(t) = L^{\mathrm{T}}(t) \log P(t)$$

$$\textstyle\sum_{\mathrm{corpus}} ce(t) = \max$$

deep dive: handling of time

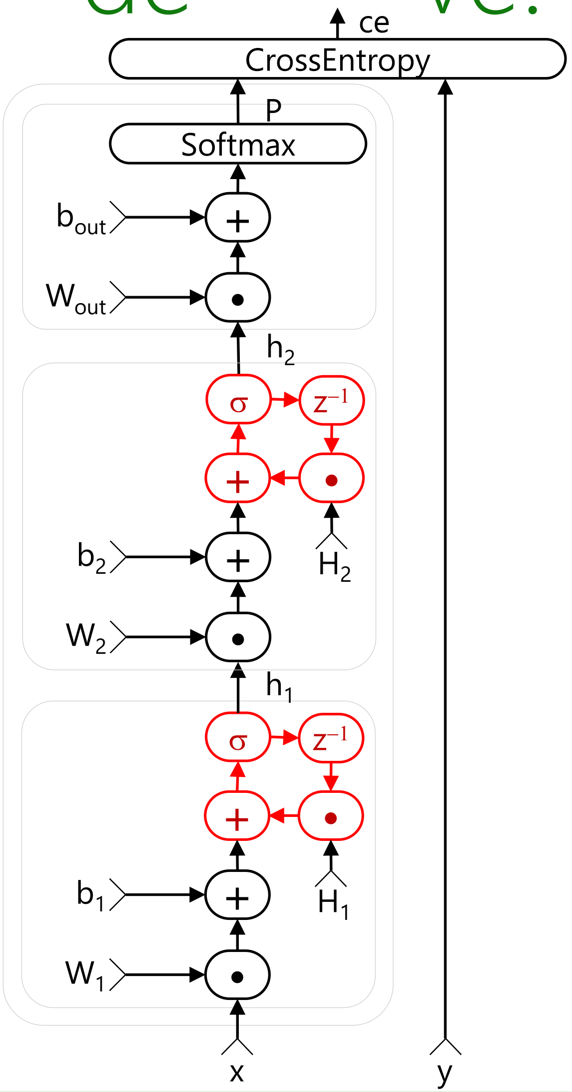
extend our example to an RNN

$$h_1(t) = \sigma(W_1\, x(t) + H_1\, h_1(t\text{-}1) + b_1)$$
$$h_2(t) = \sigma(W_2\, h_1(t) + H_2\, h_2(t\text{-}1) + b_2)$$
$$P(t) = \mathrm{softmax}(W_{out}\, h_2(t) + b_{out})$$
$$ce(t) = L^{T}(t) \log P(t)$$
$$\textstyle\sum_{corpus} ce(t) = \max$$

```
h1 = Sigmoid(W1 * x  + H1 * PastValue(h1) + b1)
h2 = Sigmoid(W2 * h1 + H2 * PastValue(h2) + b2)
P  = Softmax(Wout * h2 + bout)
ce = CrossEntropy(L, P)
```

→ no explicit notion of time

deep dive: handling of time



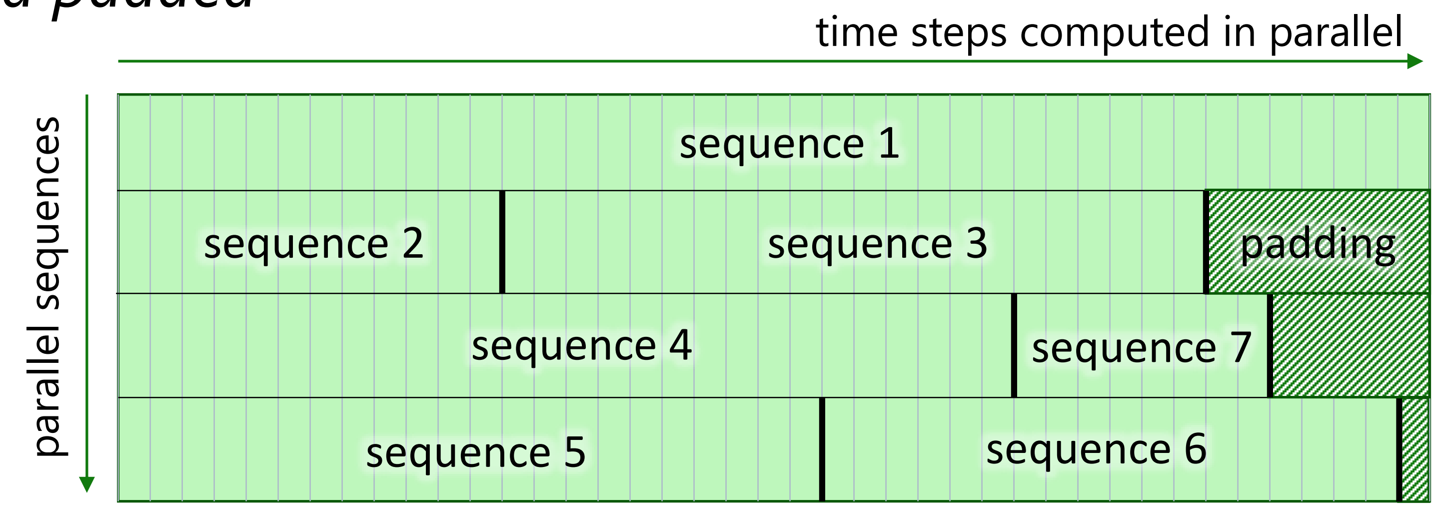

```
h1 = Sigmoid(W1 * x  + H1 * PastValue(h1) + b1)
h2 = Sigmoid(W2 * h1 + H2 * PastValue(h2) + b2)
P  = Softmax(Wout * h2 + bout)
ce = CrossEntropy(L, P)
```

- CNTK automatically unrolls cycles
  - cycles are detected with Tarjan's algorithm
  - loops become part of deferred computation
  - only nodes in cycles are unrolled
- efficient and composable
  - cf. TensorFlow, where recurrence must be manually unrolled in imperative code: [https://www.tensorflow.org/versions/r0.8/tutorials/recurrent/index.html]

```
lstm = rnn_cell.BasicLSTMCell(lstm_size)
state = tf.zeros([batch_size, lstm.state_size])
for current_batch_of_words in words_in_dataset:
    output, state = lstm(current_batch_of_words, state)
```

deep dive: variable-length sequences

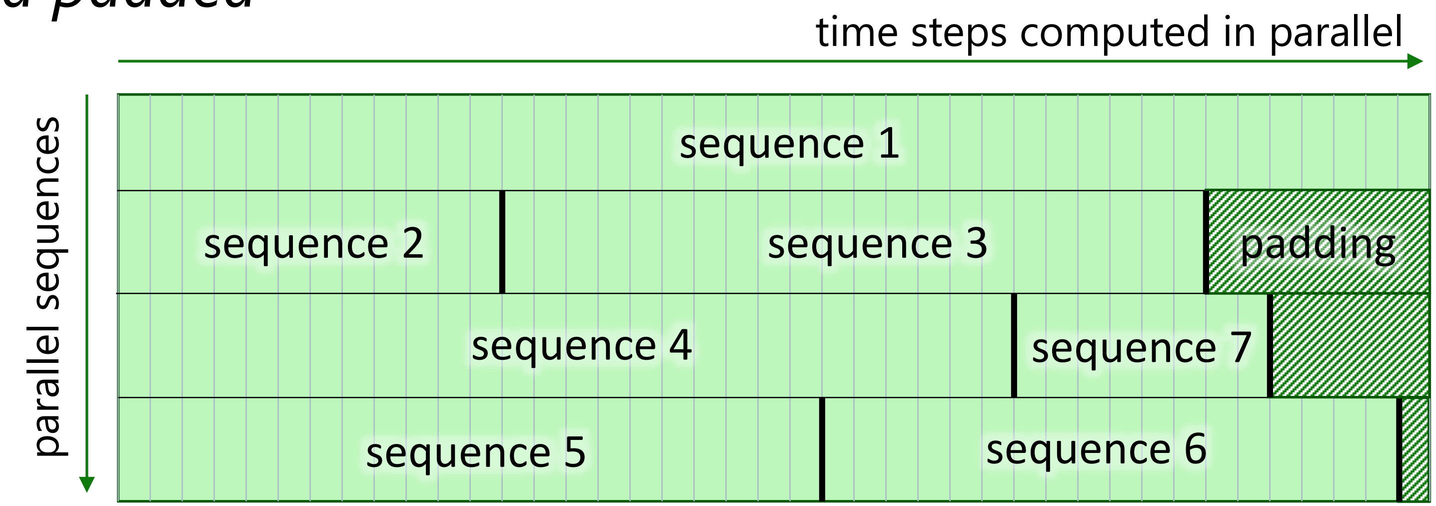
- minibatches containing sequences of different lengths are automatically packed *and padded*



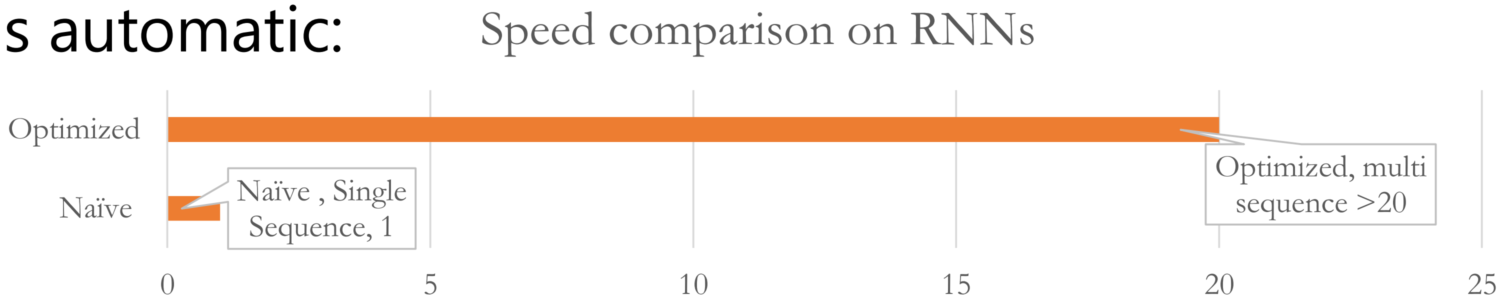
- CNTK handles the special cases:
  - PastValue operation correctly resets state and gradient at sequence boundaries
  - non-recurrent operations just pretend there is no padding (“garbage-in/garbage-out”)
  - sequence reductions

deep dive: variable-length sequences

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



- speed-up is automatic:



recap

- users never explicitly see time axes
- CNTK infers loops from PastValue (and FutureValue) operations
  - graph looks like signal processing
- CNTK automatically batches, packs, and pads sequences into minibatches
  - and computes the right thing

Look, Ma, no bucketing!

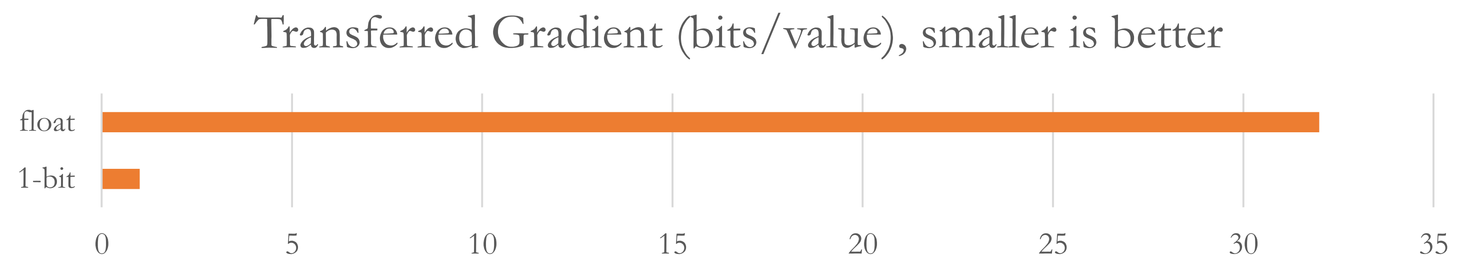
deep dive: data-parallel training

- 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

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



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

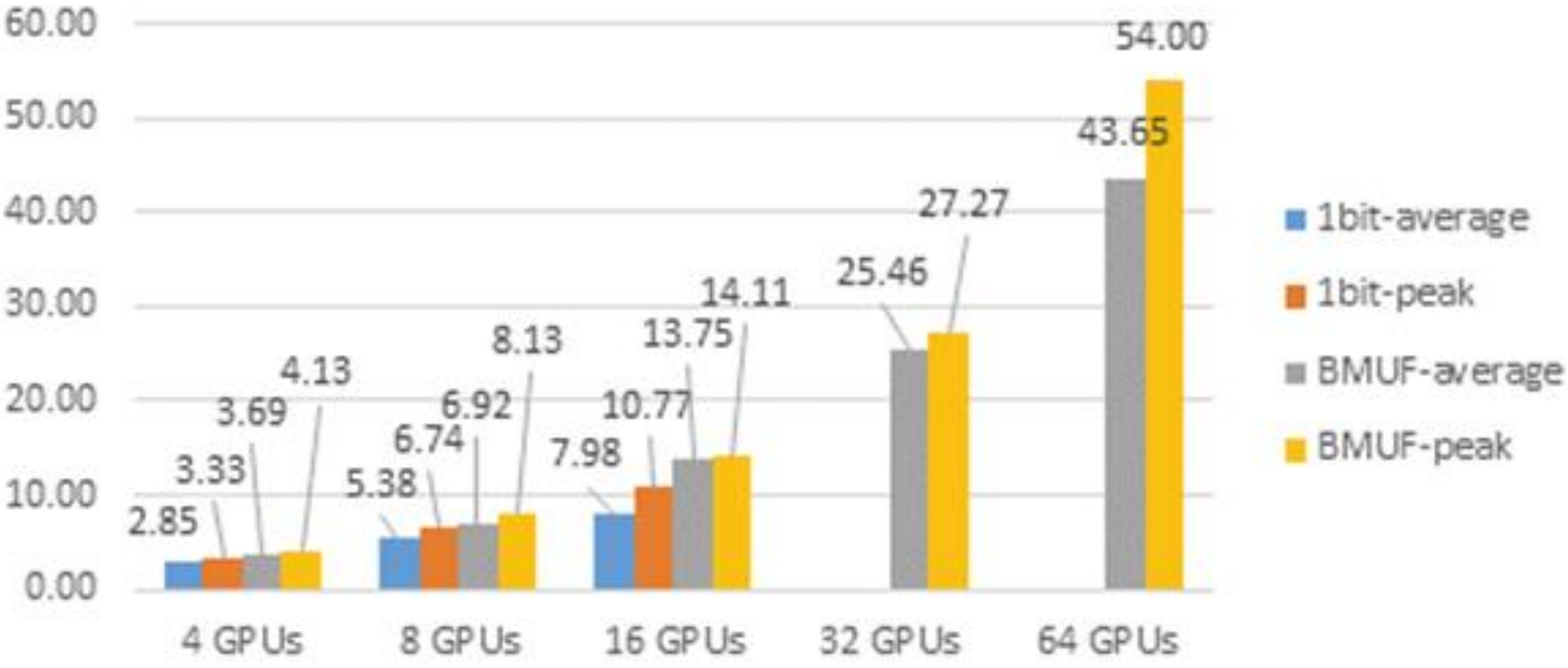
- goal: communicate less often
- every now and then try to grow MB size on small subset
  - important: keep contribution per sample and momentum effect constant
  - hence define learning rate and momentum in a MB-size agnostic fashion
- quickly scales up to MB sizes of 3k; runs at up to 100k samples

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

I. what
II. how to
III. deep dive

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.