Achieving Human Parity in Conversational Speech Recognition using CNTK and a GPU Farm

Andreas Stolcke and Frank Seide
Microsoft AI + Research
anstolck@microsoft.com, fseide@microsoft.com

Roadmap

- Task and history
- System overview and results
- Human versus machine
- Cognitive Toolkit (CNTK)
- Summary and outlook

Acknowledgments

Fil Alleva
Jasha Droppo
Xuedong Huang
Mike Seltzer
Lingfeng Wu
Wayne Xiong
Dong Yu
Geoff Zweig

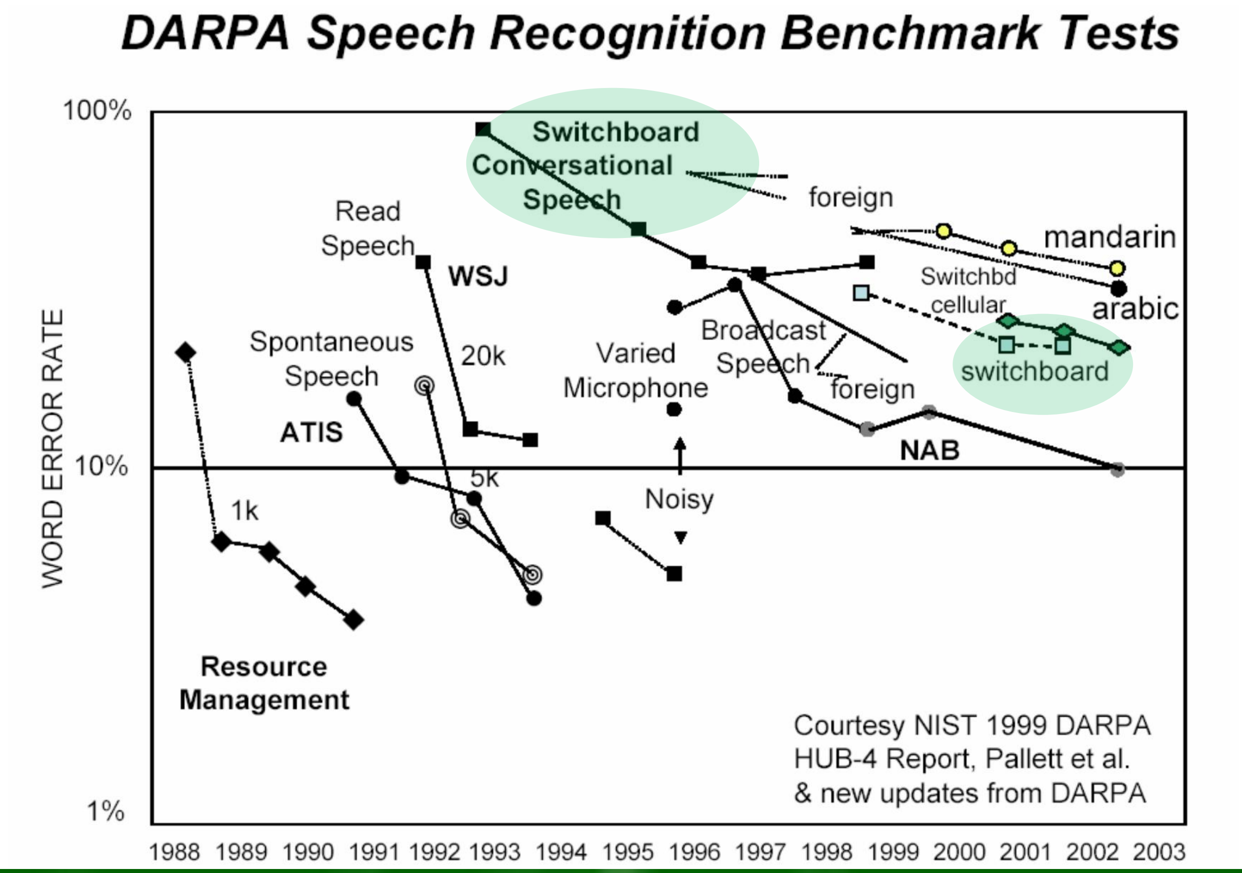
Introduction:
Task and History

The Human Parity Experiment

- Conversational telephone speech has been a benchmark in the research community for 20 years
  - Focus here: strangers talking to each other via telephone, given a topic
  - Known as the “Switchboard” task in speech community
- Can we achieve human-level performance?
- Top-level tasks:
  - Measure human performance
  - Build the best possible recognition system
  - Compare and analyze

30 Years of Speech Recognition Benchmarks

For many years, DARPA drove the field by defining public benchmark tasks



Read and planned speech:

RM

ATIS

WSJ

Conversational Telephone Speech (CTS):

Switchboard (SWB)
(strangers, on-topic)

CallHome (CH)
(friends & family, unconstrained)

History of Human Error Estimates for SWB

- Lippman (1997): 4%
  - based on "personal communication" with NIST, no experimental data cited
- LDC LREC paper (2010): 4.1-4.5%
  - Measured on a different dataset (but similar to our NIST eval set, SWB portion)
- Microsoft (2016): 5.9%
  - Transcribers were blind to experiment
  - 2-pass transcription, isolated utterances (no "transcriber adaptation")
- IBM (2017): 5.1%
  - Using multiple independent transcriptions, picked best transcriber
  - Vendor was involved in experiment and aware of NIST transcription conventions

Note: Human error will vary depending on

- Level of effort (e.g., multiple transcribers)
- Amount of context supplied (listening to short snippets vs. entire conversation)

Recent ASR Results on Switchboard

Group	2000 SWB WER	Notes	Reference
Microsoft	16.1%	DNN applied to LVCSR for the first time	Seide et al, 2011
Microsoft	9.9%	LSTM applied for the first time	A.-R. Mohammed et al, IEEE ASRU 2015
IBM	6.6%	Neural Networks and System Combination	Saon et al., Interspeech 2016
Microsoft	5.8%	First claim of "human parity"	Xiong et al., arXiv 2016, IEEE Trans. SALP 2017
IBM	5.5%	Revised view of "human parity"	Saon et al., Interspeech 2017
Capio	5.3%		Han et al., Interspeech 2017
Microsoft	5.1%	Current Microsoft research system	Xiong et al., MSR-TR-2017-39, ICASSP 2018

System Overview and Results

System Overview

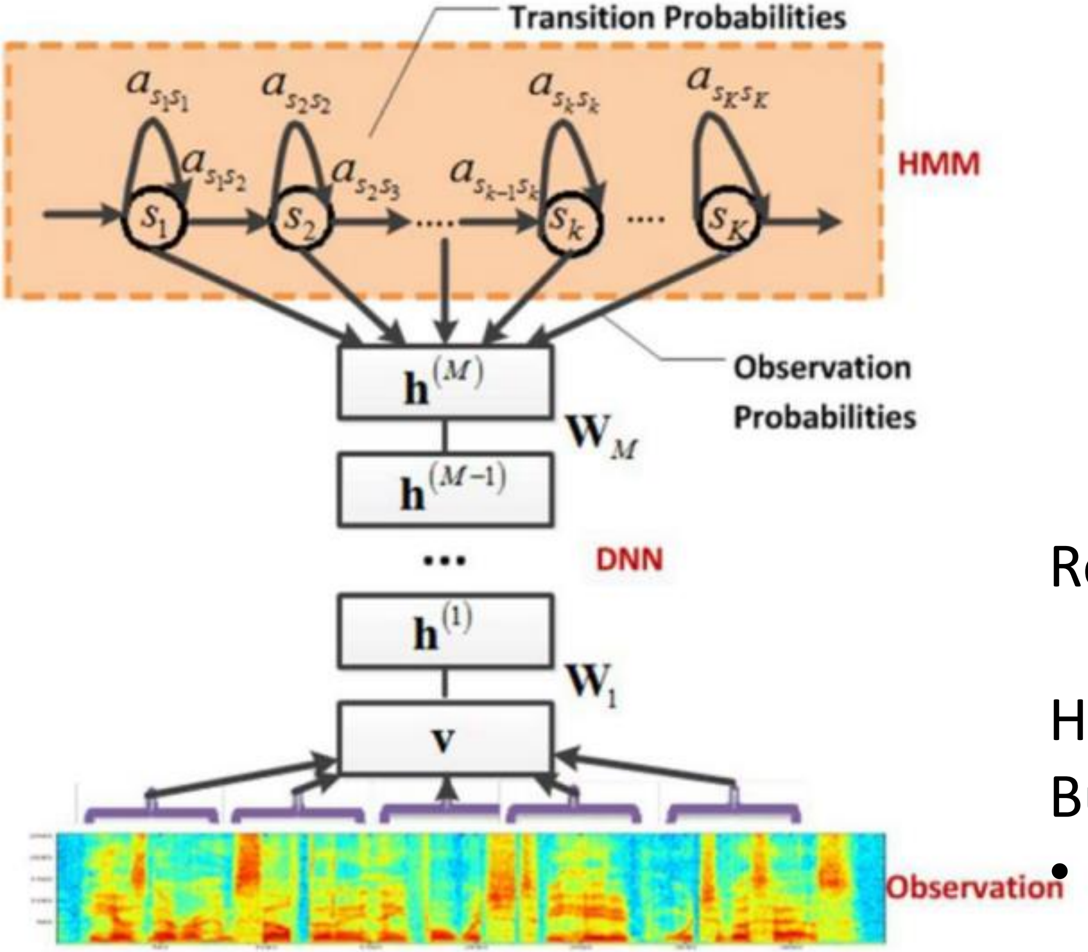
- Hybrid HMM/deep neural net architecture
- Multiple acoustic model types
  - Different architectures (convolutional and recurrent)
  - Different acoustic model unit clusterings
- Multiple language models
  - All based on LSTM recurrent networks
  - Different input encodings
  - Forward and backward running
- Model combination at multiple levels

For details, see our upcoming paper in ICASSP-2018

Data used

- Acoustic training: 2000 hours of conversational telephone data
- Language model training:
  - Conversational telephone transcripts
  - Web data collected to be conversational in style
  - Broadcast news transcripts
- Test on NIST 2000 SWB+CH evaluation set
- *Note:* data chosen to be compatible with past practice
  - NOT using proprietary sources

Acoustic Modeling Framework: Hybrid HMM/DNN



	CallHome	Switchboard
DNN	21.9%	13.4%

1st pass decoding

Record performance in 2011 [Seide et al.]

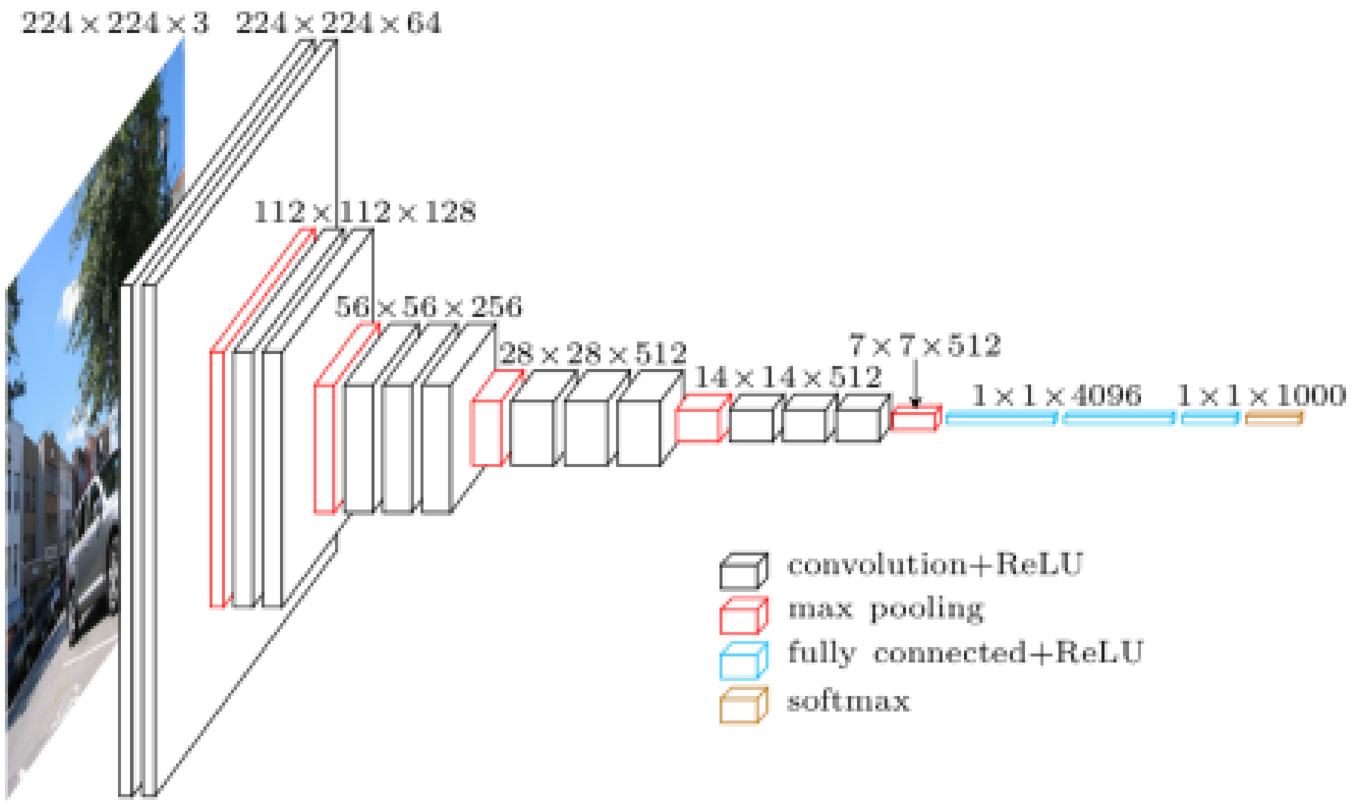
Hybrid HMM/NN approach still standard

But DNN model now obsolete (!)

- Poor spatial/temporal invariance

[Yu et al., 2010; Dahl et al., 2011]

Acoustic Modeling: Convolutional Nets



Adapted from image processing

Robust to temporal and frequency shifts

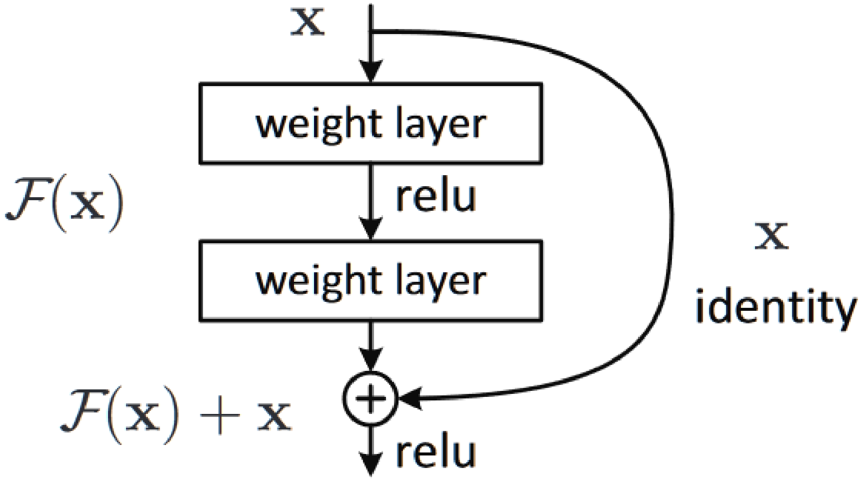
[Simonyan & Zisserman, 2014; Frossard 2016, Saon et al., 2016, Krizhevsky et al., 2012]

Acoustic Modeling: ResNet

Add a non-linear offset to linear transformation of features
Similar to fMPE in Povey et al., 2005
See also Ghahremani & Droppo, 2016

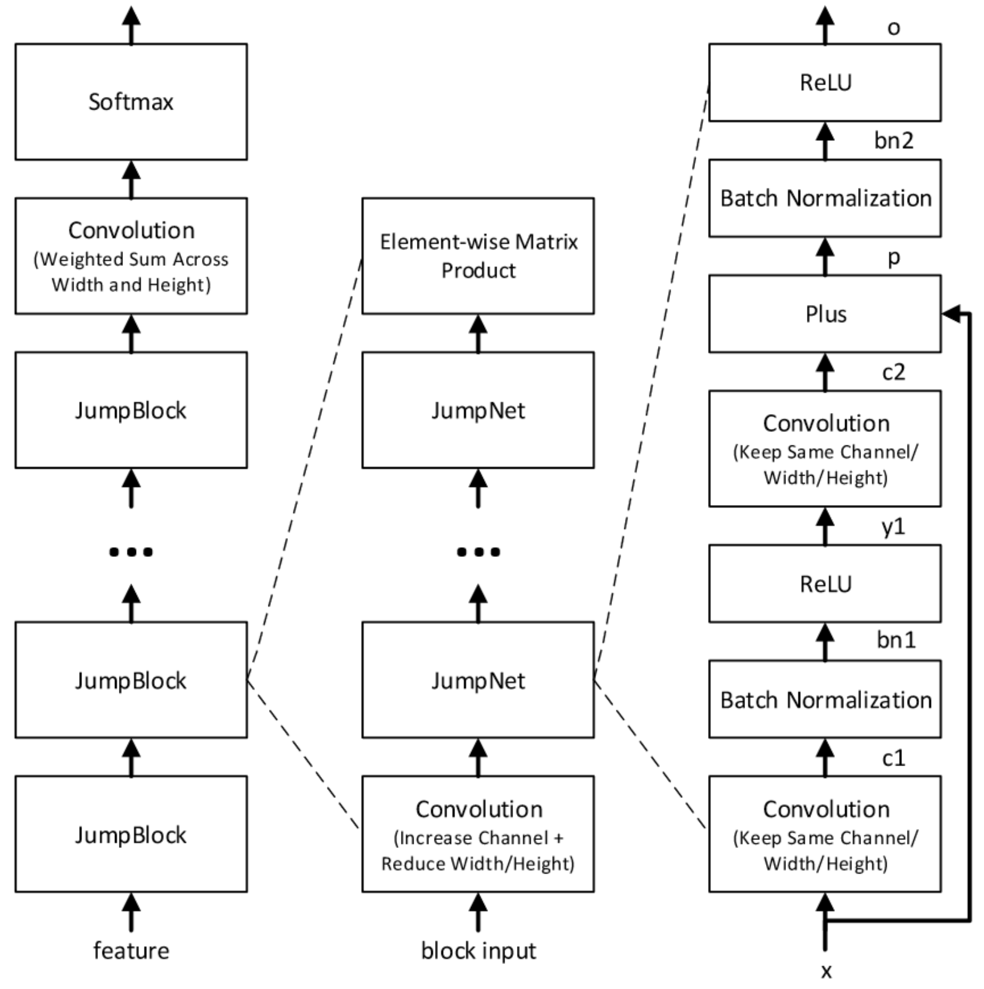
	CallHome	Switchboard
DNN	21.9%	13.4%
ResNet	17.3%	11.1%

1st pass decoding



[He et al., 2015]

Acoustic Modeling: LACE CNN

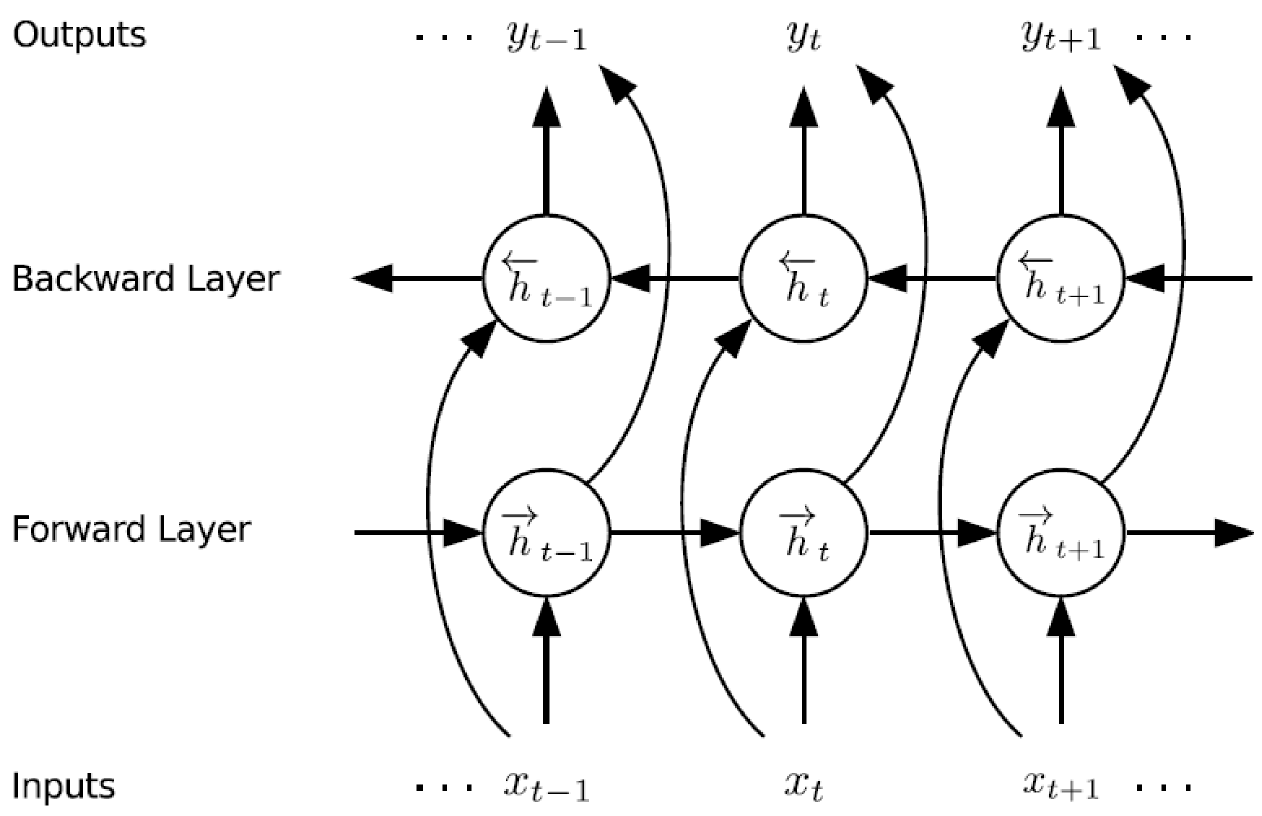


	CallHome	Switchboard
DNN	21.9%	13.4%
ResNet	17.3%	11.1%
LACE	16.9%	10.4%

1st pass decoding

CNNs with **batch normalization, Resnet jumps**, and **attention masks** [Yu et al., 2016]

Acoustic Modeling: Bidirectional LSTMs



	CallHome	Switchboard
DNN	21.9%	13.4%
ResNet	17.3%	11.1%
LACE	16.9%	10.4%
BLSTM	17.3%	10.3%

Stable form of recurrent neural net
Robust to temporal shifts

[Hochreiter & Schmidhuber, 1997,
Graves & Schmidhuber, 2005; Sak et al., 2014]

Acoustic Modeling: CNN-BLSTM

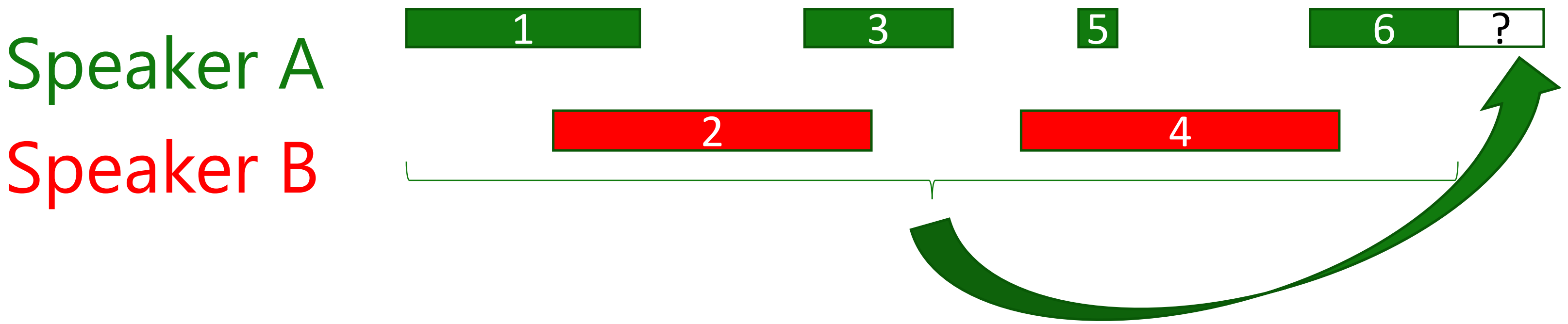
- Combination of convolutional and recurrent net model
  [Sainath et al., 2015]
- Three convolutional layers
- Six BLSTM recurrent layers

Language Modeling: Multiple LSTM variants

- Decoder uses a word 4-gram model
- N-best hypotheses are rescored with multiple LSTM recurrent network language models
- LSTMs differ by
  - Direction: forward/backward running
  - Encoding: word one-hot, word letter trigram, character one-hot
  - Scope: utterance-level / **session-level**

Session-level Language Modeling

- Predict next word from full conversation history, not just one utterance:



LSTM language model	Perplexity
Utterance-level LSTM (standard)	44.6
+ session word history	37.0
+ speaker change history	35.5
+ speaker overlap history	35.0

Acoustic model combination

Step 0: create 4 different versions of each acoustic model by clustering phonetic model units (**senones**) differently

Step 1: combine **different models** for **same senone** set at the **frame level** (posterior probability averaging)

Step 2: after LM rescoring, combine **different senone** systems at the **word level** (confusion network combination)

Results

Word error rates (WER)

Senone set	Acoustic models	SWB WER	CH WER
1	BLSTM	6.4	12.1
2	BLSTM	6.3	12.1
3	BLSTM	6.3	12.0
4	BLSTM	6.3	12.8
1	BLSTM + Resnet + LACE + CNN-BLSTM	5.4	10.2
2	BLSTM + Resnet + LACE + CNN-BLSTM	5.4	10.2
3	BLSTM + Resnet + LACE + CNN-BLSTM	5.6	10.2
4	BLSTM + Resnet + LACE + CNN-BLSTM	5.5	10.3
1+2+3+4	BLSTM + Resnet + LACE + CNN-BLSTM	5.2	9.8
	+ Confusion network rescoring	**5.1**	**9.8**

Frame-level combination

Word-level combination

Human vs. Machine

Microsoft Human Error Estimate (2015)

- Skype Translator has a weekly transcription contract
  - For quality control, training, etc.
- Initial transcription followed by a second checking pass
  - Two transcribers on each speech excerpt
- One week, we added NIST 2000 CTS evaluation data to the pipeline
  - Speech was pre-segmented as in NIST evaluation

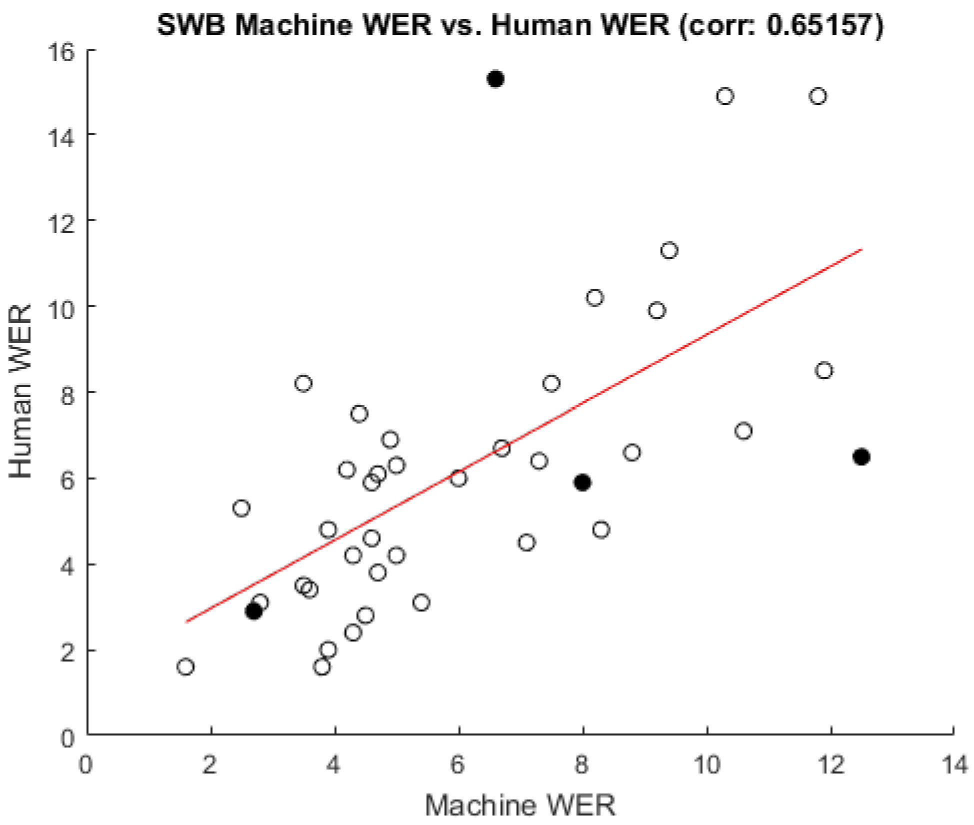
Human Error Estimate: Results

- Applied NIST scoring protocol (same as ASR)
- Switchboard: **5.9%** error rate
- CallHome: **11.3%** error rate
- SWB in the 4.1% - 9.6% range expected based on NIST study
- CH is *difficult for both people and machines*
  - Machine error about 2x higher
  - High ASR error not just because of mismatched conditions

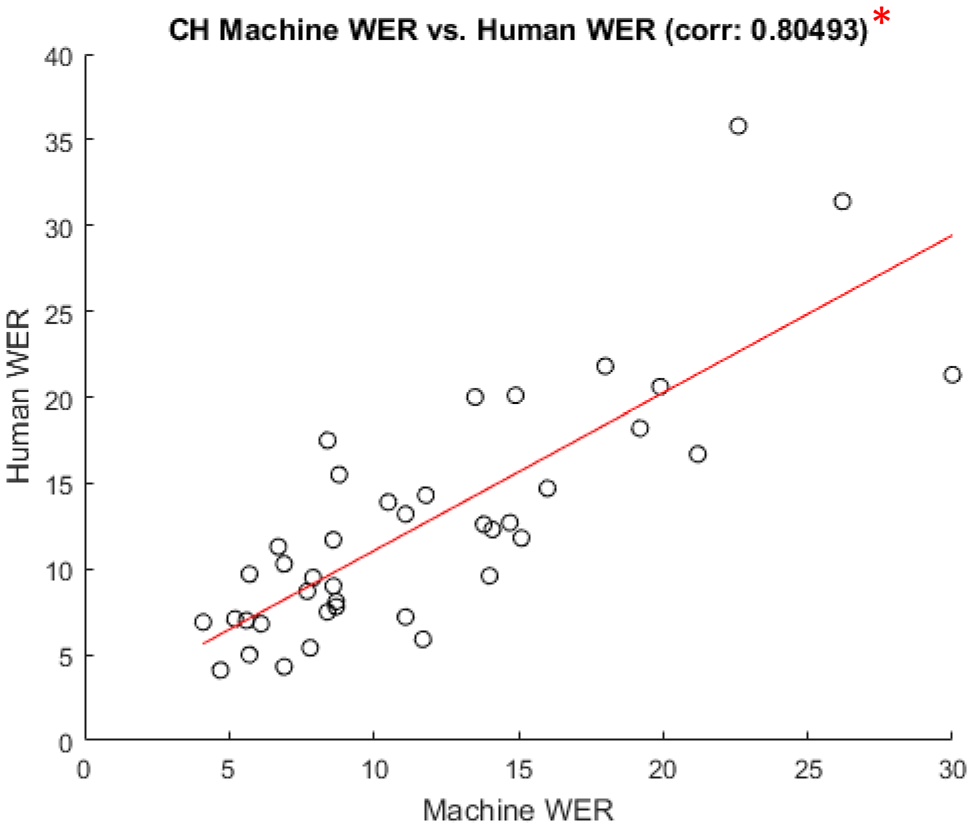
New questions:

- Are human and machine errors correlated?
- Do they make the same type of errors?
- Can humans tell the difference?

Correlation between human and machine errors?



$\rho = 0.65$



$\rho = 0.80$

*Two CallHome conversations with multiple speakers per conversation side removed, see paper for full results

Humans and machines: different error types?

Top word substitution errors (≈ 21k words in each test set)

CH		SWB	
ASR	**Human**	**ASR**	**Human**
45: (%hesitation) / %bcack	12: a / the	29: (%hesitation) / %bcack	12: (%hesitation) / hmm
12: was / is	10: (%hesitation) / a	9: (%hesitation) / oh	10: (%hesitation) / oh
9: (%hesitation) / a	10: was / is	9: was / is	9: was / is
8: (%hesitation) / oh	7: (%hesitation) / hmm	8: and / in	8: (%hesitation) / a
8: a / the	7: bentsy / bensi	6: (%hesitation) / i	5: in / and
7: and / in	7: is / was	6: in / and	4: (%hesitation) / %bcack
7: it / that	6: could / can	5: (%hesitation) / a	4: and / in
6: in / and	6: well / oh	5: (%hesitation) / yeah	4: is / was

Overall similar patterns: short function words get confused (also: inserted/deleted)

One outlier: machine falsely recognizes backchannel "uh-huh" for filled pause "uh"

- These words are acoustically confusable, have opposite pragmatic functions in conversation
- Humans can disambiguate by prosody and context

Can humans tell the difference?

- Attendees at a major speech conference played “Spot the Bot”
- Showed them human and machine output side-by-side in random order, along with reference transcript
- Turing-like experiment: tell which transcript is human/machine
- Result: it was hard to beat a random guess
  - 53% accuracy (188/353 correct)
  - Not statistically different from chance ($p \approx 0.12$, one-tailed)

CNTK

Intro - Microsoft Cognitive Toolkit (CNTK)

Microsoft
Cognitive
Toolkit
aka.ms/CognitiveToolkit
Microsoft

- Microsoft's open-source deep-learning toolkit
  - https://github.com/Microsoft/CNTK Star 14,094 Fork 3,741

Intro - Microsoft Cognitive Toolkit (CNTK)

Microsoft Cognitive Toolkit
aka.ms/CognitiveToolkit
Microsoft

- Microsoft's open-source deep-learning toolkit
  - https://github.com/Microsoft/CNTK Star 14,094 Fork 3,741
- Designed for ease of use
  - — think "what", not "how"
- Runs over 80% Microsoft internal DL workloads
- Interoperable:
  - ONNX format
  - WinML
  - Keras backend
  - 1st-class on Linux and Windows, docker support

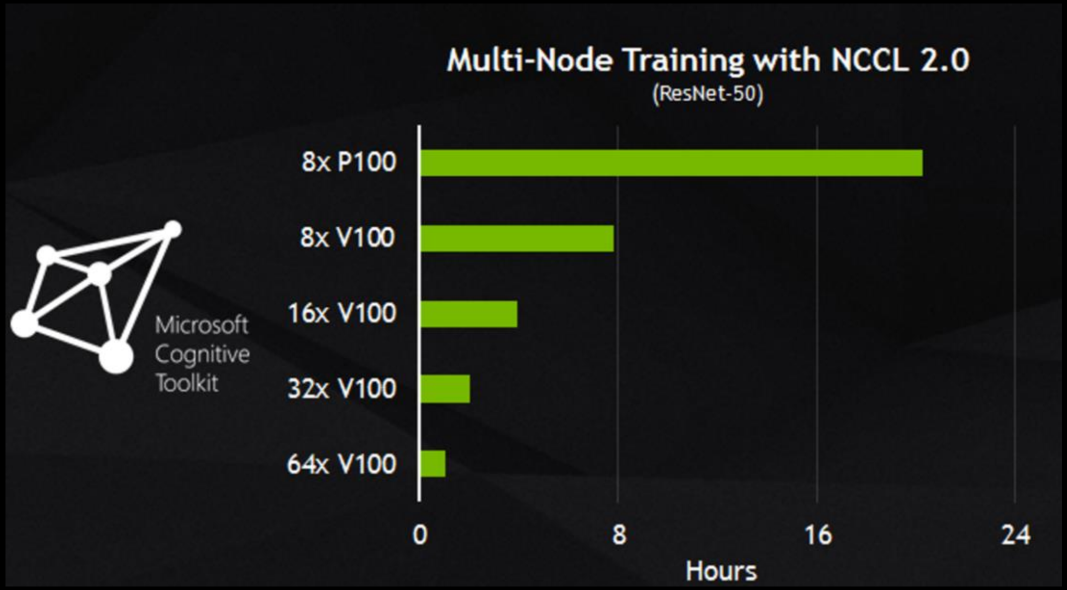
CNTK – The Fastest Toolkit

http://dlbench.comp.hkbu.edu.hk/
Benchmarking by HKBU, Version 8
Single Tesla K80 GPU, CUDA: 8.0 CUDNN: v5.1

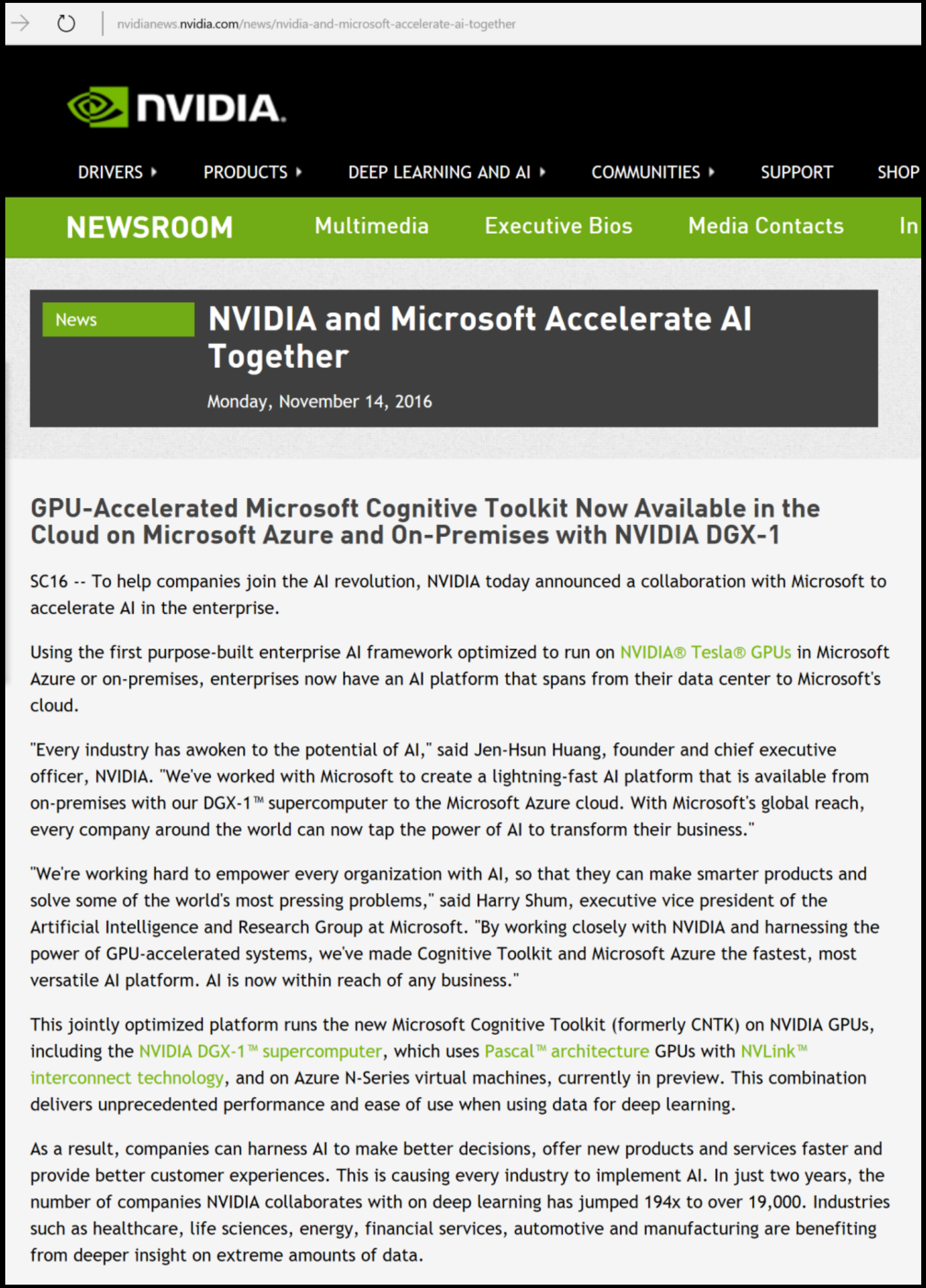
Caffe: 1.0rc5(39f28e4)
CNTK: 2.0 Beta10(1ae666d)
MXNet: 0.93(32dc3a2)
TensorFlow: 1.0(4ac9c09)
Torch: 7(748f5e3)

	Caffe	CNTK	MxNet	TensorFlow	Torch
FCN5 (1024)	55.329ms	51.038ms	60.448ms	62.044ms	52.154ms
AlexNet (256)	36.815ms	27.215ms	28.994ms	103.960ms	37.462ms
ResNet (32)	143.987ms	81.470ms	84.545ms	181.404ms	90.935ms
LSTM (256) (v7 benchmark)	-	43.581ms (44.917ms)	288.142ms (284.898ms)	- (223.547ms)	1130.606ms (906.958ms)

Superior performance



GTC, May 2017



nvidianews.nvidia.com/news/nvidia-and-microsoft-accelerate-ai-together

NVIDIA

DRIVERS PRODUCTS DEEP LEARNING AND AI COMMUNITIES SUPPORT SHOP

NEWSROOM Multimedia Executive Bios Media Contacts In

News

NVIDIA and Microsoft Accelerate AI Together

Monday, November 14, 2016

GPU-Accelerated Microsoft Cognitive Toolkit Now Available in the Cloud on Microsoft Azure and On-Premises with NVIDIA DGX-1

SC16 -- To help companies join the AI revolution, NVIDIA today announced a collaboration with Microsoft to accelerate AI in the enterprise.

Using the first purpose-built enterprise AI framework optimized to run on NVIDIA® Tesla® GPUs in Microsoft Azure or on-premises, enterprises now have an AI platform that spans from their data center to Microsoft's cloud.

"Every industry has awoken to the potential of AI," said Jen-Hsun Huang, founder and chief executive officer, NVIDIA. "We've worked with Microsoft to create a lightning-fast AI platform that is available from on-premises with our DGX-1™ supercomputer to the Microsoft Azure cloud. With Microsoft's global reach, every company around the world can now tap the power of AI to transform their business."

"We're working hard to empower every organization with AI, so that they can make smarter products and solve some of the world's most pressing problems," said Harry Shum, executive vice president of the Artificial Intelligence and Research Group at Microsoft. "By working closely with NVIDIA and harnessing the power of GPU-accelerated systems, we've made Cognitive Toolkit and Microsoft Azure the fastest, most versatile AI platform. AI is now within reach of any business."

This jointly optimized platform runs the new Microsoft Cognitive Toolkit (formerly CNTK) on NVIDIA GPUs, including the NVIDIA DGX-1™ supercomputer, which uses Pascal™ architecture GPUs with NVLink™ interconnect technology, and on Azure N-Series virtual machines, currently in preview. This combination delivers unprecedented performance and ease of use when using data for deep learning.

As a result, companies can harness AI to make better decisions, offer new products and services faster and provide better customer experiences. This is causing every industry to implement AI. In just two years, the number of companies NVIDIA collaborates with on deep learning has jumped 194x to over 19,000. Industries such as healthcare, life sciences, energy, financial services, automotive and manufacturing are benefiting from deeper insight on extreme amounts of data.

Deep-learning toolkits must address two questions:

- **How to author neural networks?** ← user's job
- **How to execute them efficiently? (training/test)** ← *tool*'s job*!!*

Deep-learning toolkits must address two questions:

- **How to author neural networks?** ← user's job
- **How to execute them efficiently? (training/test)** ← *tool's job!!*

Deep Learning Process

Script configures and executes through CNTK Python APIs…

data → collect → corpus

reader
- minibatch source
- task-specific deserializer
- automatic randomization
- distributed reading

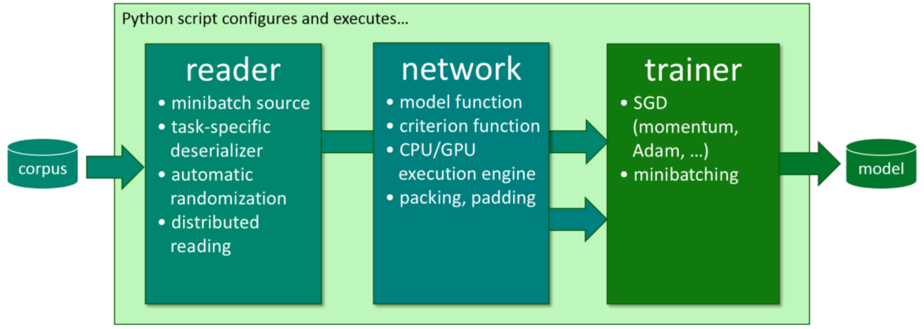
network
- model function
- criterion function
- CPU/GPU execution engine
- packing, padding

trainer
- SGD (momentum, Adam, …)
- minibatching

model → deploy → app

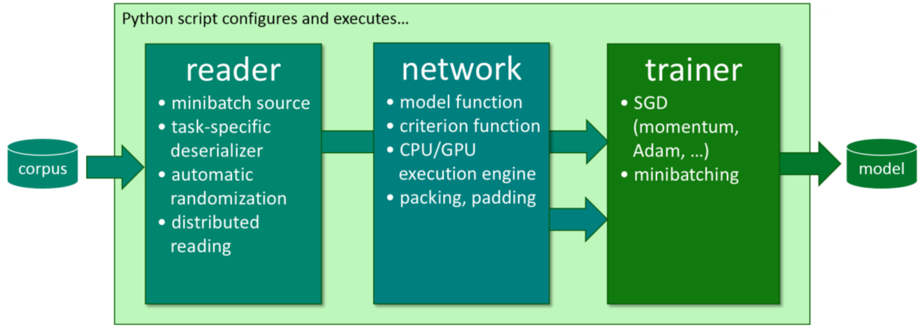
As easy as 1-2-3

```
from cntk import *

# reader
def create_reader(path, is_training):
    ...

# network
def create_model_function():
    ...
def create_criterion_function(model):
    ...

# trainer (and evaluator)
def train(reader, model):
    ...
def evaluate(reader, model):
    ...

# main function
model = create_model_function()

reader = create_reader(..., is_training=True)
train(reader, model)

reader = create_reader(..., is_training=False)
evaluate(reader, model)
```



As easy as 1-2-3

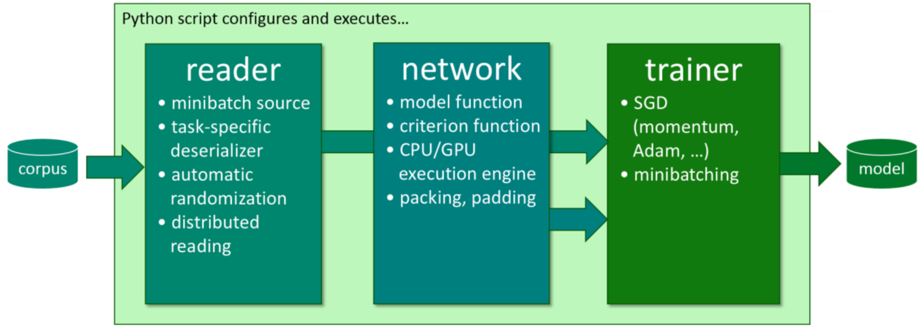
```python
from cntk import *

# reader
def create_reader(path, is_training):
    ...

# network
def create_model_function():
    ...
def create_criterion_function(model):
    ...

# trainer (and evaluator)
def train(reader, model):
    ...
def evaluate(reader, model):
    ...

# main function
model = create_model_function()

reader = create_reader(..., is_training=True)
train(reader, model)

reader = create_reader(..., is_training=False)
evaluate(reader, model)
```




```
python my_cntk_script.py
```

As easy as 1-2-3

```python
from cntk import *

# reader
def create_reader(path, is_training):
    ...

# network
def create_model_function():
    ...
def create_criterion_function(model):
    ...

# trainer (and evaluator)
def train(reader, model):
    ...
def evaluate(reader, model):
    ...

# main function
model = create_model_function()

reader = create_reader(..., is_training=True)
train(reader, model)

reader = create_reader(..., is_training=False)
evaluate(reader, model)
```




```
mpiexec --np 16 --hosts server1,server2,server3,server4 \
python my_cntk_script.py
```

neural networks as graphs

Microsoft Cognitive Toolkit
Microsoft

neural networks as graphs

example: 2-hidden layer feed-forward NN

$$
\begin{aligned}
h_1 &= \sigma(\mathbf{W}_1\, x + b_1) \\
h_2 &= \sigma(\mathbf{W}_2\, h_1 + b_2) \\
P &= \mathrm{softmax}(\mathbf{W}_{\mathrm{out}}\, h_2 + b_{\mathrm{out}})
\end{aligned}
$$

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

neural networks as graphs

example: 2-hidden layer feed-forward NN

$$
\begin{aligned}
h_1 &= \sigma(\mathbf{W}_1\, x + b_1) \\
h_2 &= \sigma(\mathbf{W}_2\, h_1 + b_2) \\
P &= \mathrm{softmax}(\mathbf{W}_{\mathrm{out}}\, h_2 + b_{\mathrm{out}})
\end{aligned}
$$

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

$$
\begin{aligned}
ce &= \log P_{\mathrm{label}} \\
\textstyle\sum_{\mathrm{corpus}} ce &= \max
\end{aligned}
$$

neural networks as graphs

example: 2-hidden layer feed-forward NN

$$
\begin{aligned}
h_1 &= \sigma(\mathbf{W}_1\, x + b_1) \\
h_2 &= \sigma(\mathbf{W}_2\, h_1 + b_2) \\
P &= \mathrm{softmax}(\mathbf{W}_{\mathrm{out}}\, h_2 + b_{\mathrm{out}})
\end{aligned}
$$

```
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

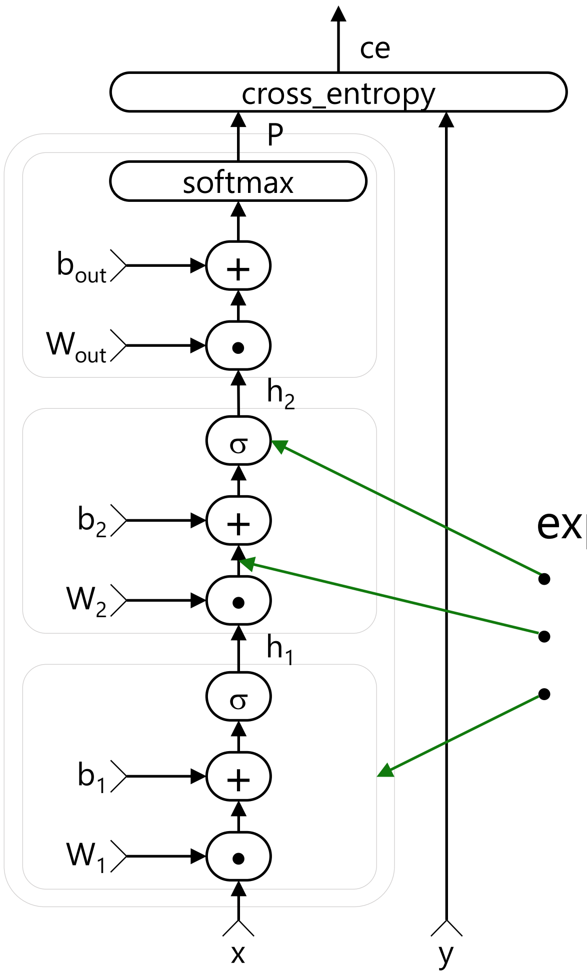
$$
\begin{aligned}
ce &= \log P_{\mathrm{label}} \\
\textstyle\sum_{\mathrm{corpus}} ce &= \max
\end{aligned}
$$

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

neural networks as graphs

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

neural networks as graphs

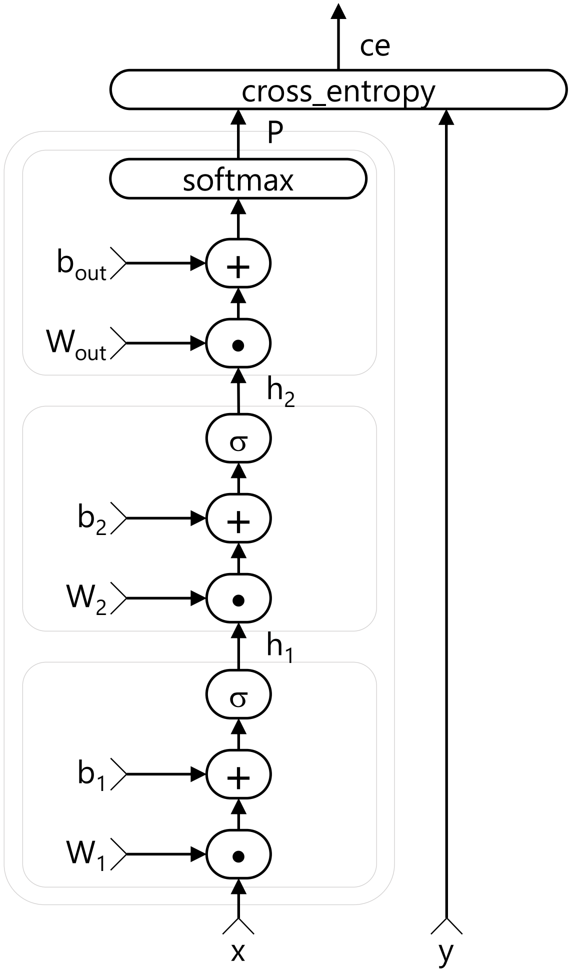


expression tree with

- primitive ops
- values (tensors)
- composite ops

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

neural networks as graphs



why graphs?

- automatic differentiation!!
  - chain rule: $\partial\mathcal{F} / \partial\mathrm{in} = \partial\mathcal{F} / \partial\mathrm{out} \cdot \partial\mathrm{out} / \partial\mathrm{in}$
  - run graph backwards

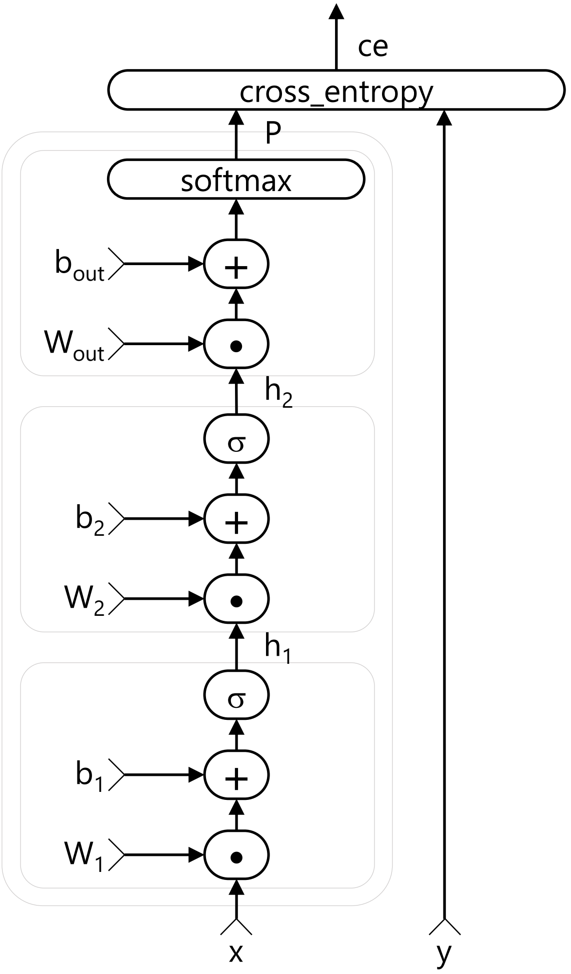
→ **"back propagation"**

graphs are the "assembly language" of DNN tools

authoring networks as functions

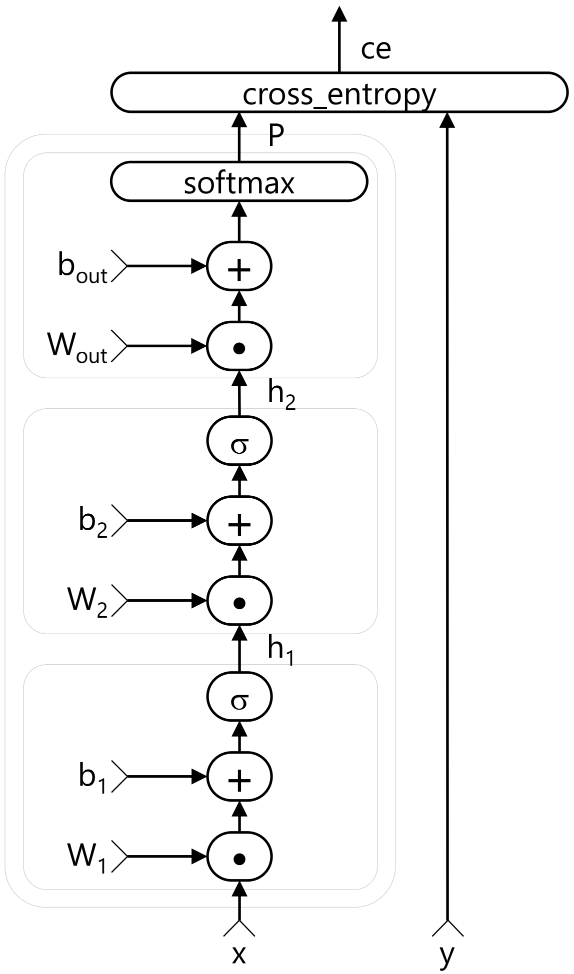
- **CNTK** model: neural networks are functions
  - pure functions
  - with “special powers”:
    - can compute a gradient w.r.t. any of its nodes
    - external deity can update model parameters
- user specifies network as function objects:
  - formula as a Python function (low level, e.g. LSTM)
  - function composition of smaller sub-networks (layering)
  - higher-order functions (equiv. of scan, fold, unfold)
  - model parameters held by function objects
- “compiled” into the static execution graph under the hood
- inspired by Functional Programming
- becoming standard: Chainer, Keras, PyTorch, Sonnet, Gluon

authoring networks as functions



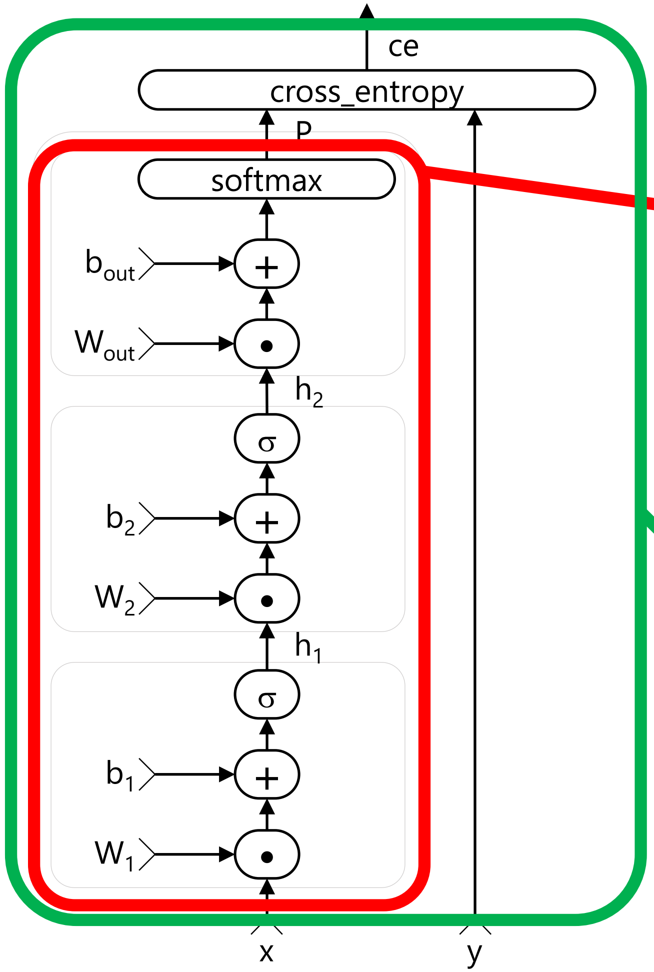

```
# --- graph building ---
M = 40 ; H = 512 ; J = 9000  # feat/hid/out dim
# define learnable parameters
W1   = Parameter((M,H)); b1   = Parameter(H)
W2   = Parameter((H,H)); b2   = Parameter(H)
Wout = Parameter((H,J)); bout = Parameter(J)
# build the graph
x = Input(M) ; y = Input(J)  # feat/labels
h1 = sigmoid(x  @ W1   + b1)
h2 = sigmoid(h1 @ W2   + b2)
P  = softmax(h2 @ Wout + bout)
ce = cross_entropy(P, y)
```

authoring networks as functions




```
# --- graph building with function objects ---
M = 40 ; H = 512 ; J = 9000  # feat/hid/out dim
#  - function objects own the learnable parameters
#  - here used as blocks in graph building
x = Input(M) ;  y = Input(J)  # feat/labels
h1 = Dense(H, activation=sigmoid)(x)
h2 = Dense(H, activation=sigmoid)(h1)
P  = Dense(J, activation=softmax)(h2)
ce = cross_entropy(P, y)
```

authoring networks as functions




```
M = 40 ; H = 512 ; J = 9000  # feat/hid/out dim
# compose model from function objects
model = Sequential([Dense(H, activation=sigmoid),
                    Dense(H, activation=sigmoid),
                    Dense(J, activation=softmax)])
# criterion function (invokes model function)
@Function
def criterion(x: Tensor[M], y: Tensor[J]):
    P = model(x)
    return cross_entropy(P, y)
# function is passed to trainer
tr = Trainer(criterion, Learner(model.parameters), …)
```

relationship to Functional Programming

- fully connected (FCN) `map`
  - describes objects through probabilities of "**class membership.**"
- convolutional (CNN) `windowed >> map` FIR filter
  - repeatedly applies a little FCN over images or other **repetitive structures**
- recurrent (RNN) `scanl, foldl, unfold` IIR filter
  - repeatedly applies a FCN over a **sequence**, using its **own previous output**

composition

- stacking layers:

```
model = Sequential([Dense(H, activation=sigmoid),
                    Dense(H, activation=sigmoid),
                    Dense(J)])
```

- recurrence:

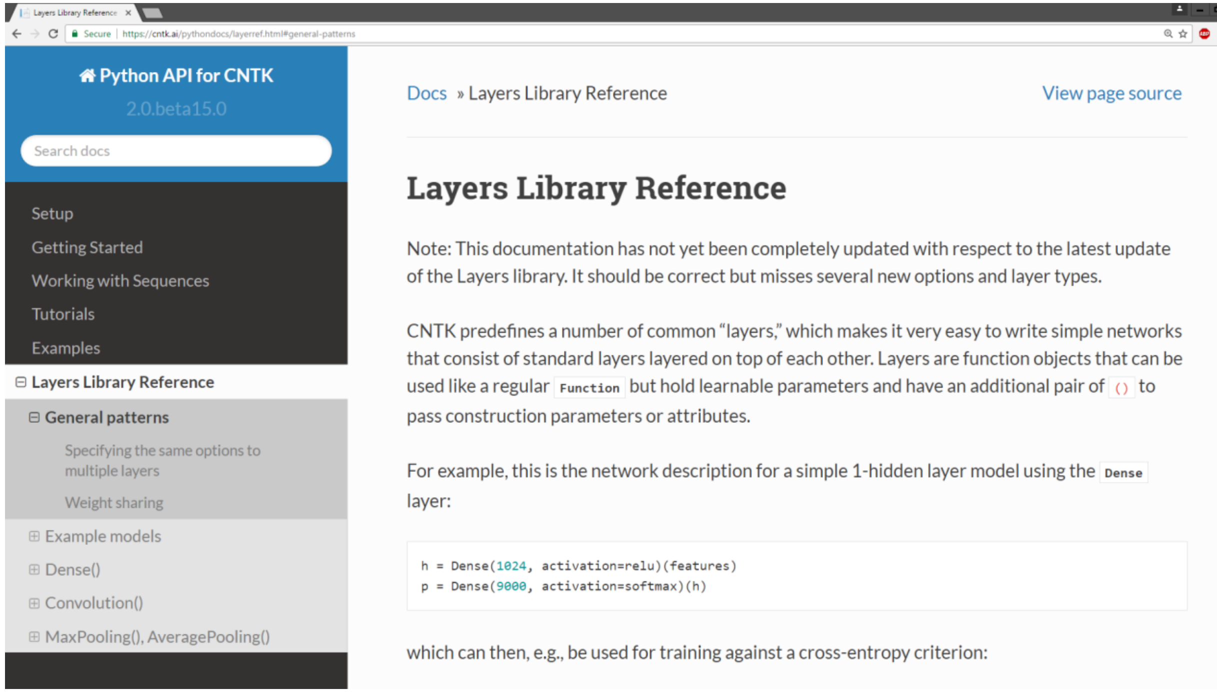
```
model = Sequential([Embedding(emb_dim),
                    Recurrence(GRU(hidden_dim)),
                    Dense(num_labels)])
```

- unfold:

```
model = UnfoldFrom(lambda history: s2smodel(history, input) >> hardmax,
                   until_predicate=lambda w: w[...,sentence_end_index],
                   length_increase=length_increase)
output = model(START_SYMBOL)
```

Layers API

- basic blocks:
  - `LSTM(), GRU(), RNNUnit()`
  - `Stabilizer(), identity`
- layers:
  - `Dense(), Embedding()`
  - `Convolution(), Deconvolution()`
  - `MaxPooling(), AveragePooling(), MaxUnpooling(), GlobalMaxPooling(), GlobalAveragePooling()`
  - `BatchNormalization(), LayerNormalization()`
  - `Dropout(), Activation()`
  - `Label()`
- composition:
  - `Sequential(), For(), operator >>, (function tuples)`
  - `ResNetBlock(), SequentialClique()`
- sequences:
  - `Delay(), PastValueWindow()`
  - `Recurrence(), RecurrenceFrom(), Fold(), UnfoldFrom()`
- models:
  - `AttentionModel()`



Extensibility

- Core interfaces can be implemented in user code
  - UserFunction
  - UserLearner
  - UserMinibatchSource

Extending CNTK

CNTK provides extension possibilities through

- custom operators in pure Python as so-called 'user functions'
- custom learning algorithms (like SGD or Adam) as 'user learners'
- custom minibatch sources as 'user minibatch sources'

User defined functions

Implementing a custom operator in pure Python is simple matter of

- inheriting from `UserFunction`
- implementing `forward()` and `backward()`, whose signatures depe
  inputs and outputs
- specifying the outputs' shape, data type and dynamic axes in `in`
- providing a static `deserialize()` method to inflate previously sav

In the simplest case, just only one input and output, `forward()` takes ar
tuple of a state and the result. The state can be used to pass data from
backward pass, but can be set to None if not needed.

Let's consider the example of a sigmoid. This is just for demonstration p
computation better use `sigmoid()`.

As the derivative of $\mathrm{sigmoid}(x)$ is $\mathrm{sigmoid}(x) * (1 - \mathrm{sigmoid}(x))$
value as the state variable, which is then later fed into backward(). Note
Python value (including tuple, strings, etc.):

deep-learning toolkits must address two questions:

- how to author neural networks? ← user's job
- how to execute them efficiently? (training/test) ← *tool*'s job*!!*

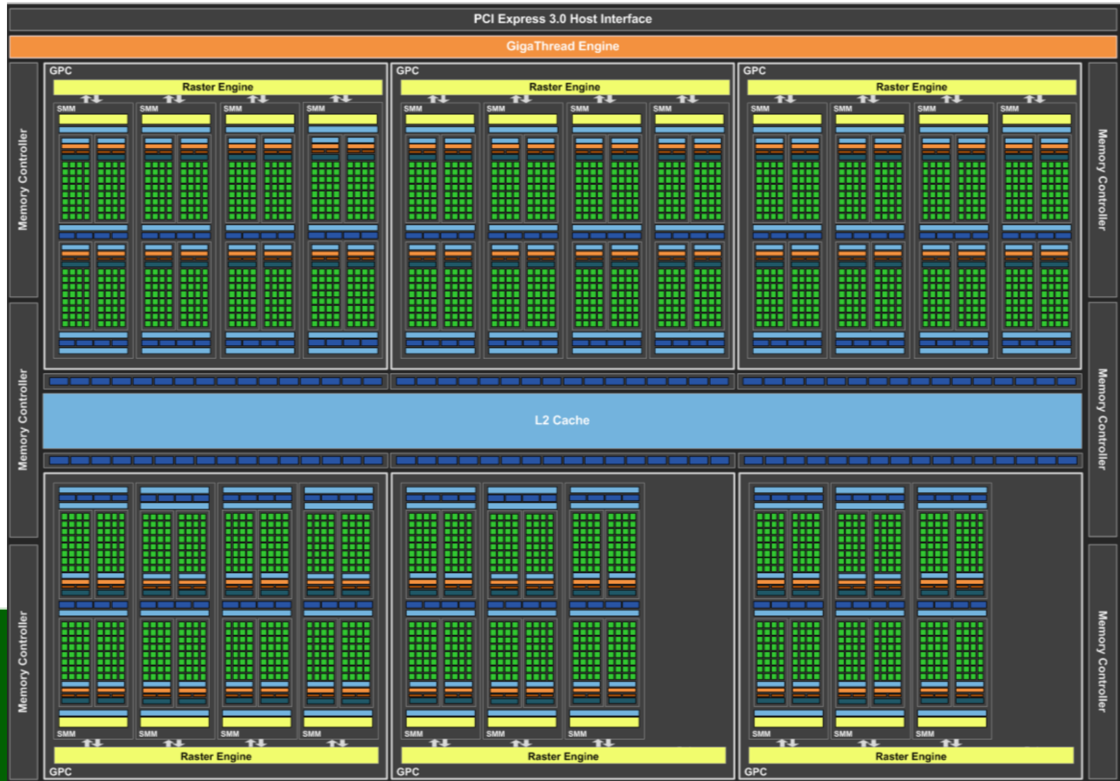
high performance with GPUs

- GPUs are massively parallel super-computers
  - NVidia Titan X: 3583 parallel Pascal processor cores
  - GPUs made NN research and experimentation productive

- CNTK must turn DNNs into **parallel programs**

- two main priorities in GPU computing:
  1. make sure all CUDA cores are always busy
  2. read from GPU RAM as little as possible

[Jacob Devlin, NLPCC 2016 Tutorial]



minibatching

- **minibatching :=** batch N samples, e.g. N=256; execute in lockstep

minibatching

- **minibatching :=** batch N samples, e.g. N=256; execute in lockstep
  - turns N matrix-vector products into one matrix-matrix product → peak GPU performance
  - element-wise ops and reductions benefit, too
  - has limits (convergence, dependencies, memory)
- critical for GPU performance
  - difficult to get right

→ CNTK makes batching fully transparent

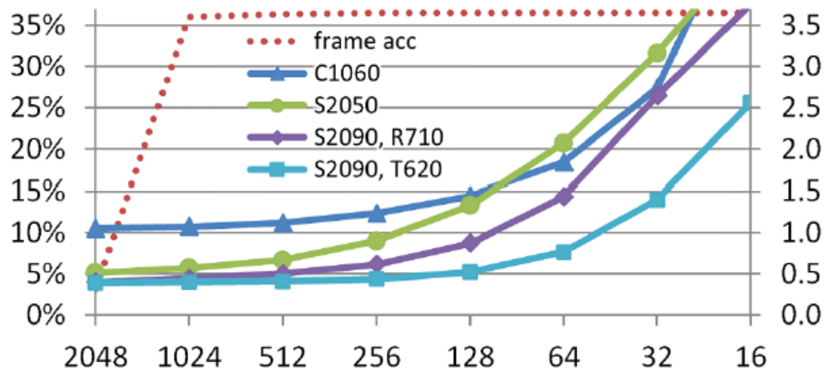


Figure 1: *Relative runtime for different minibatch sizes and GPU/server model types, and corresponding frame accuracy measured after seeing 12 hours of data.*[7]

symbolic loops over sequential data

extend our example to a recurrent network (RNN)

$$h_1 = \sigma(\mathbf{W}_1\, x + b_1)$$

$$h_2 = \sigma(\mathbf{W}_2\, h_1 + b_2)$$

$$P = \mathrm{softmax}(\mathbf{W}_{\mathrm{out}}\, h_2 + b_{\mathrm{out}})$$

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

$$\sum_{\mathrm{corpus}} ce = \max$$

symbolic loops over sequential data

extend our example to a recurrent network (RNN)

$$h_1(t) = \sigma(\mathbf{W}_1 x(t) + b_1)$$

$$h_2(t) = \sigma(\mathbf{W}_2 h_1(t) + b_2)$$

$$P(t) = \mathrm{softmax}(\mathbf{W}_{\mathrm{out}} h_2(t) + b_{\mathrm{out}})$$

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

$$\sum\nolimits_{\mathrm{corpus}} ce(t) = \max$$

symbolic loops over sequential data

extend our example to a recurrent network (RNN)

$$h_1(t) = \sigma(\mathbf{W}_1\, x(t) + \mathbf{R}_1\, h_1(t\text{-}1) + b_1)$$

$$h_2(t) = \sigma(\mathbf{W}_2\, h_1(t) + \mathbf{R}_2\, h_2(t\text{-}1) + b_2)$$

$$P(t) = \text{softmax}(\mathbf{W}_{\text{out}}\, h_2(t) + b_{\text{out}})$$

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

$$\sum\nolimits_{\text{corpus}} ce(t) = \max$$

symbolic loops over sequential data

extend our example to a recurrent network (RNN)

$$h_1(t) = \sigma(\mathbf{W}_1\, x(t) + \mathbf{R}_1\, h_1(t\text{-}1) + b_1)$$

$$h_2(t) = \sigma(\mathbf{W}_2\, h_1(t) + \mathbf{R}_2\, h_2(t\text{-}1) + b_2)$$

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

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

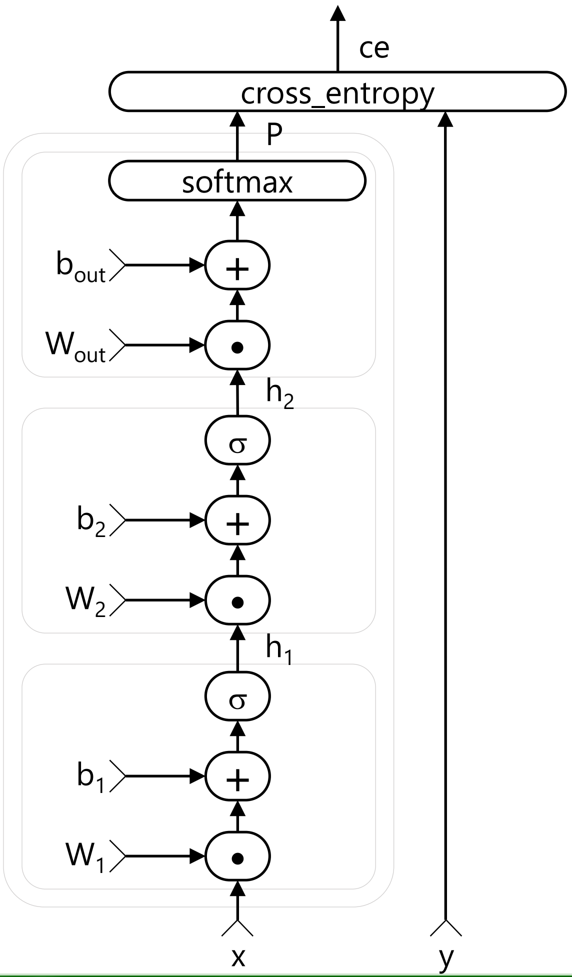
$$\textstyle\sum_{\mathrm{corpus}} ce(t) = \max$$

```
h1 = sigmoid(x  @ W1 + past_value(h1) @ R1 + b1)
h2 = sigmoid(h1 @ W2 + past_value(h2) @ R2 + b2)
P  = softmax(h2 @ Wout + bout)
ce = cross_entropy(P, L)
```

symbolic loops over sequential data

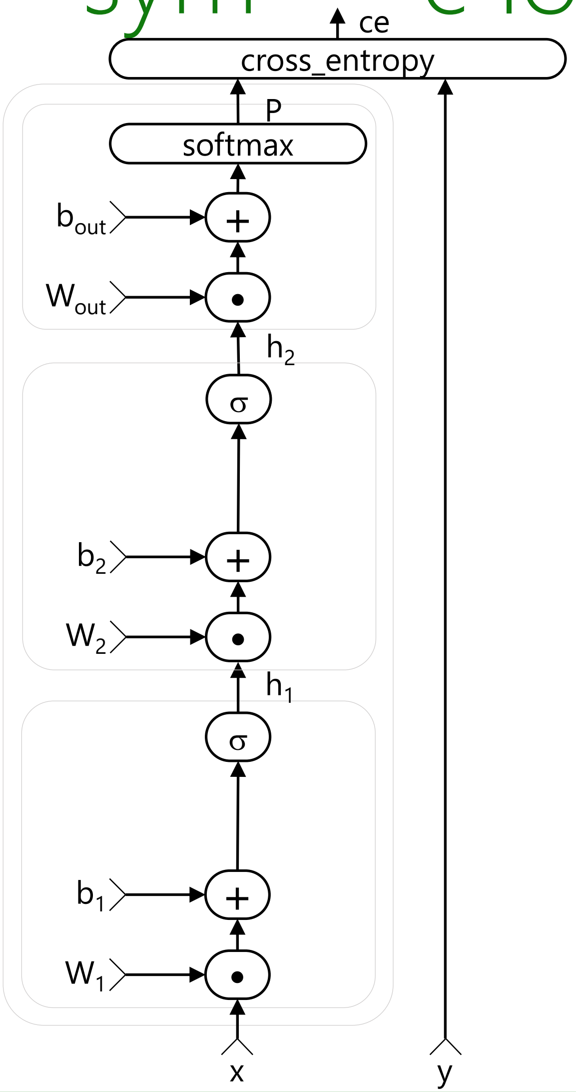
```
h1 = sigmoid(x  @ W1 + past_value(h1) @ R1 + b1)
h2 = sigmoid(h1 @ W2 + past_value(h2) @ R2 + b2)
P  = softmax(h2 @ Wout + bout)
ce = cross_entropy(P, L)
```

symbolic loops over sequential data



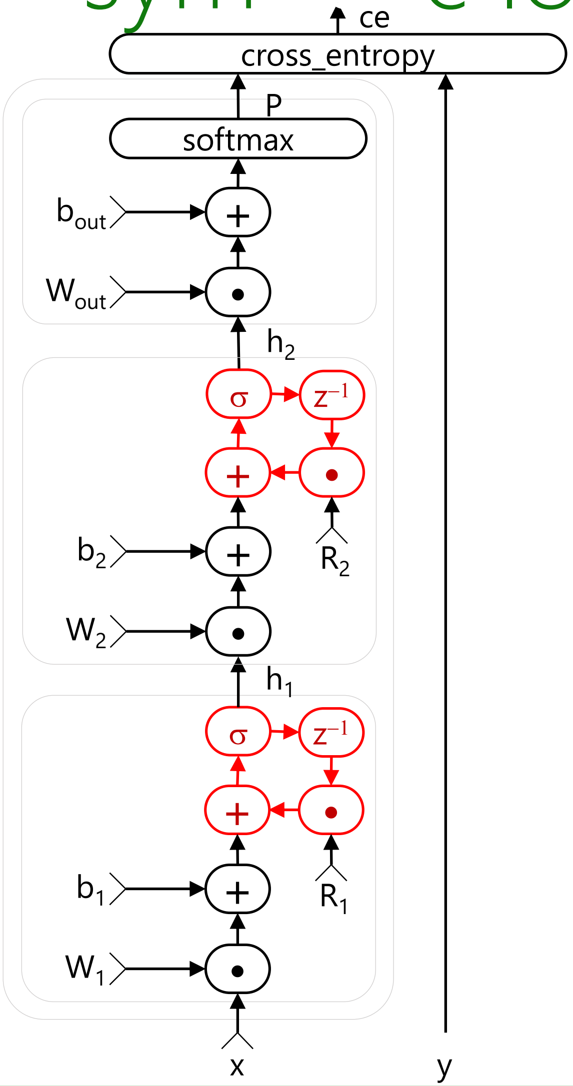

```
h1 = sigmoid(x  @ W1 + past_value(h1) @ R1 + b1)
h2 = sigmoid(h1 @ W2 + past_value(h2) @ R2 + b2)
P  = softmax(h2 @ Wout + bout)
ce = cross_entropy(P, L)
```

symbolic loops over sequential data




```
h1 = sigmoid(x  @ W1 + past_value(h1) @ R1 + b1)
h2 = sigmoid(h1 @ W2 + past_value(h2) @ R2 + b2)
P  = softmax(h2 @ Wout + bout)
ce = cross_entropy(P, L)
```

symbolic loops over sequential data




```
h1 = sigmoid(x  @ W1 + past_value(h1) @ R1 + b1)
h2 = sigmoid(h1 @ W2 + past_value(h2) @ R2 + b2)
P  = softmax(h2 @ Wout + bout)
ce = cross_entropy(P, L)
```

- CNTK automatically unrolls cycles at *execution time*
- non-cycles (black) are still executed in parallel
  - cf. TensorFlow: has to be manually coded

batching variable-length sequences

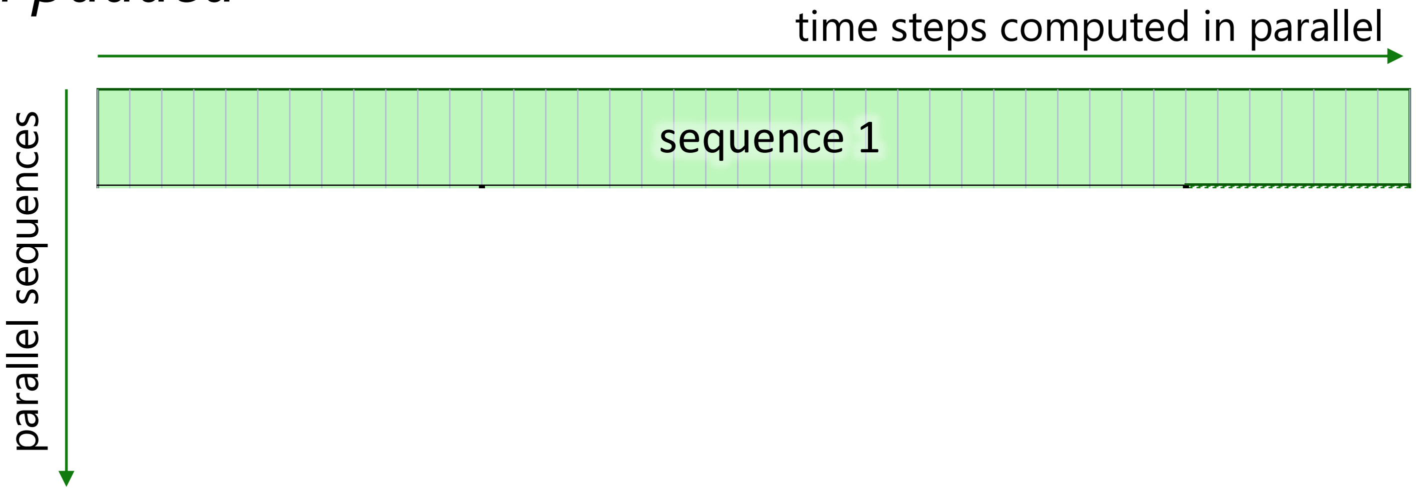
- minibatches containing sequences of different lengths are automatically packed *and padded*

Microsoft
Cognitive
Toolkit

Microsoft

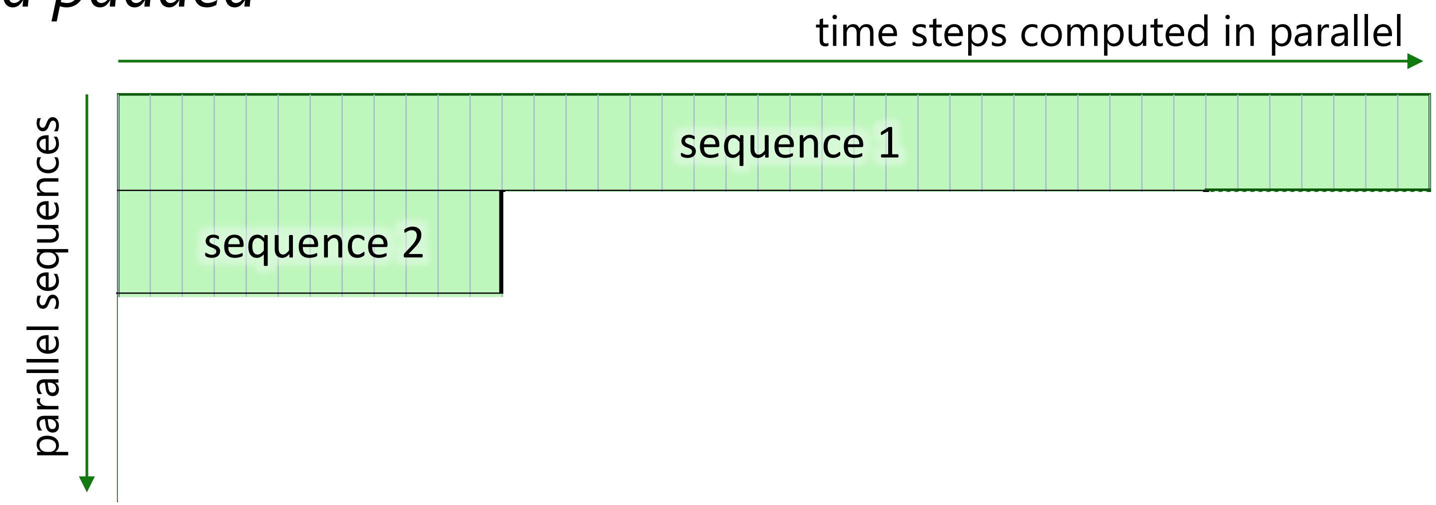
batching variable-length sequences

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



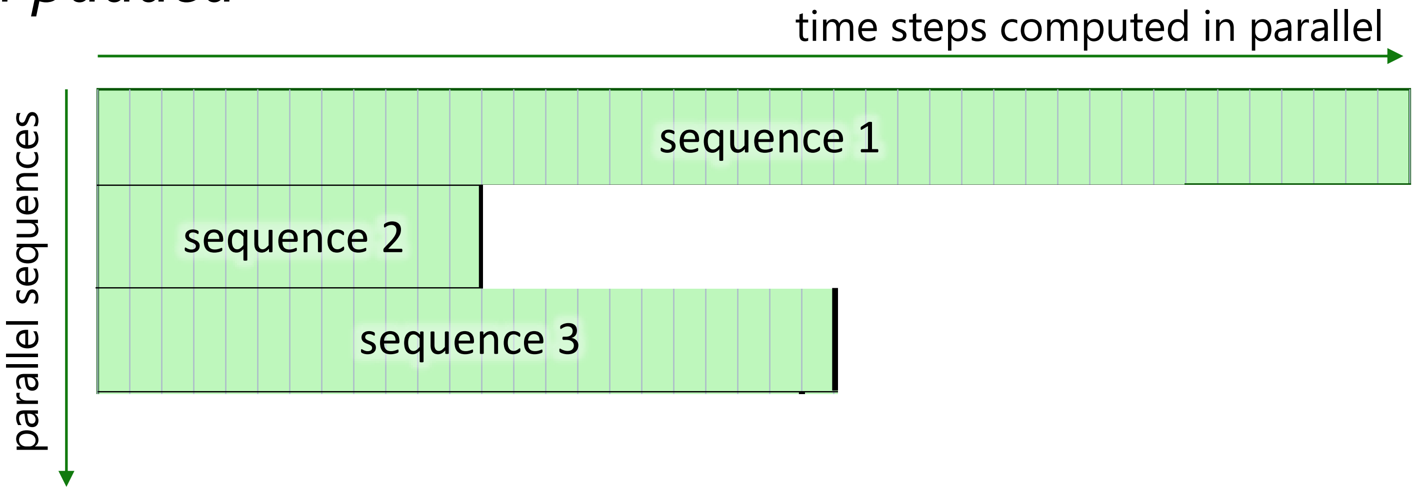
batching variable-length sequences

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



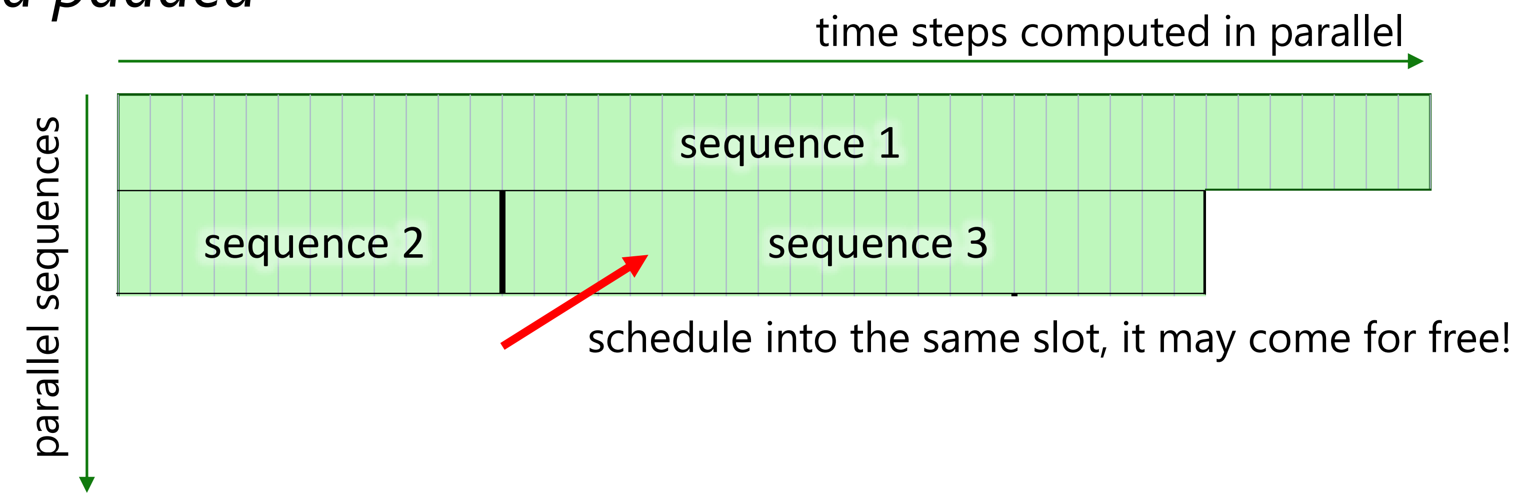
batching variable-length sequences

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



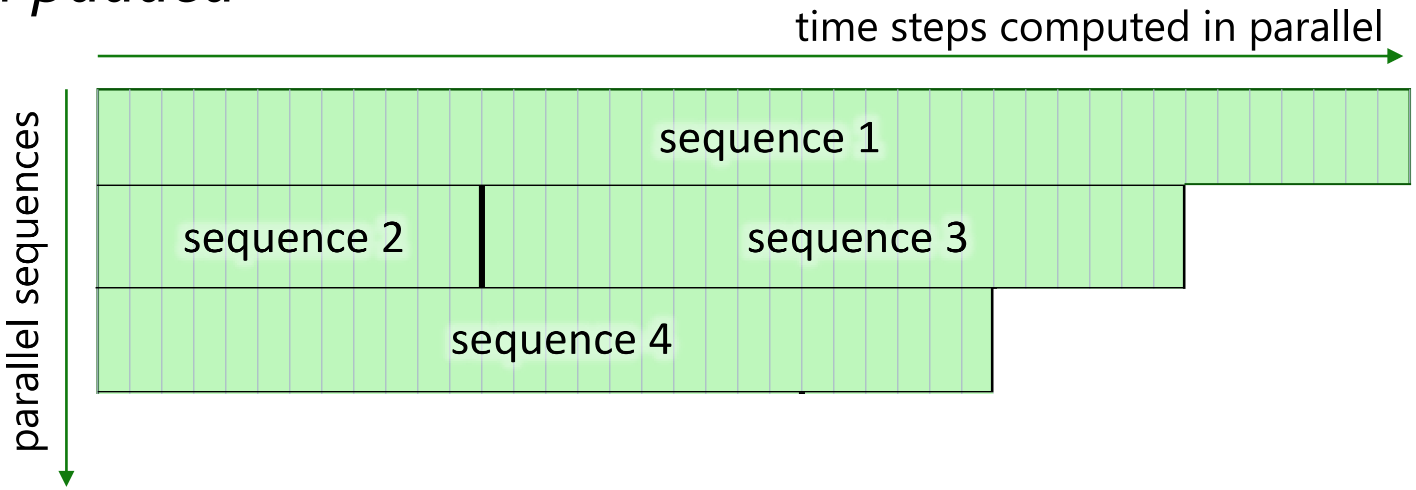
batching variable-length sequences

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



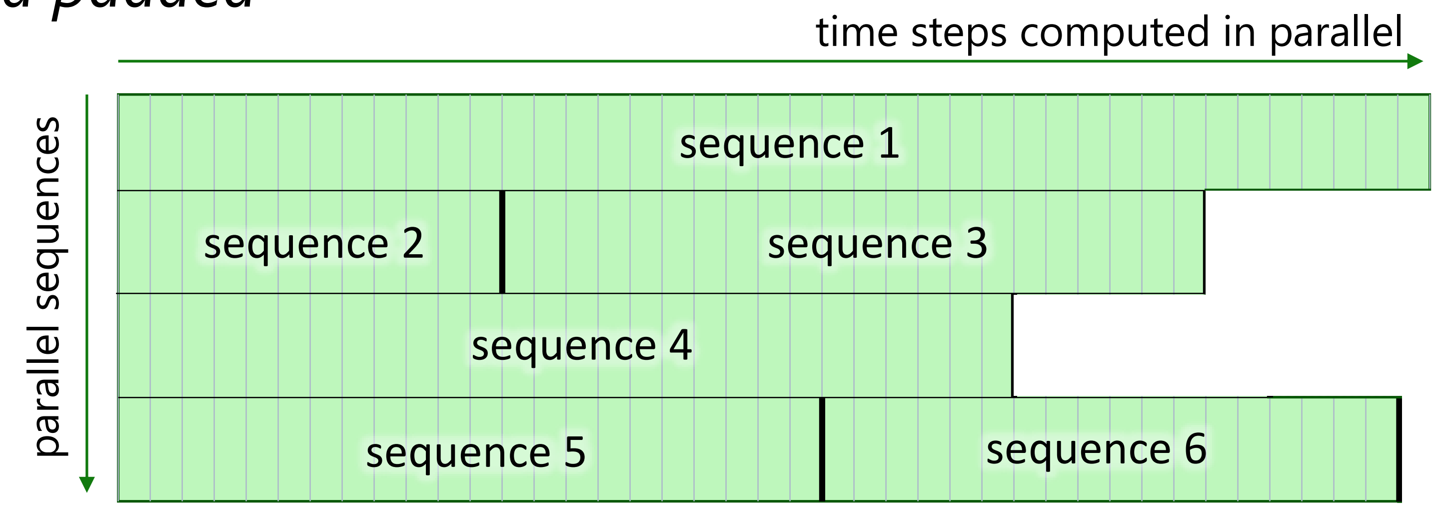
batching variable-length sequences

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



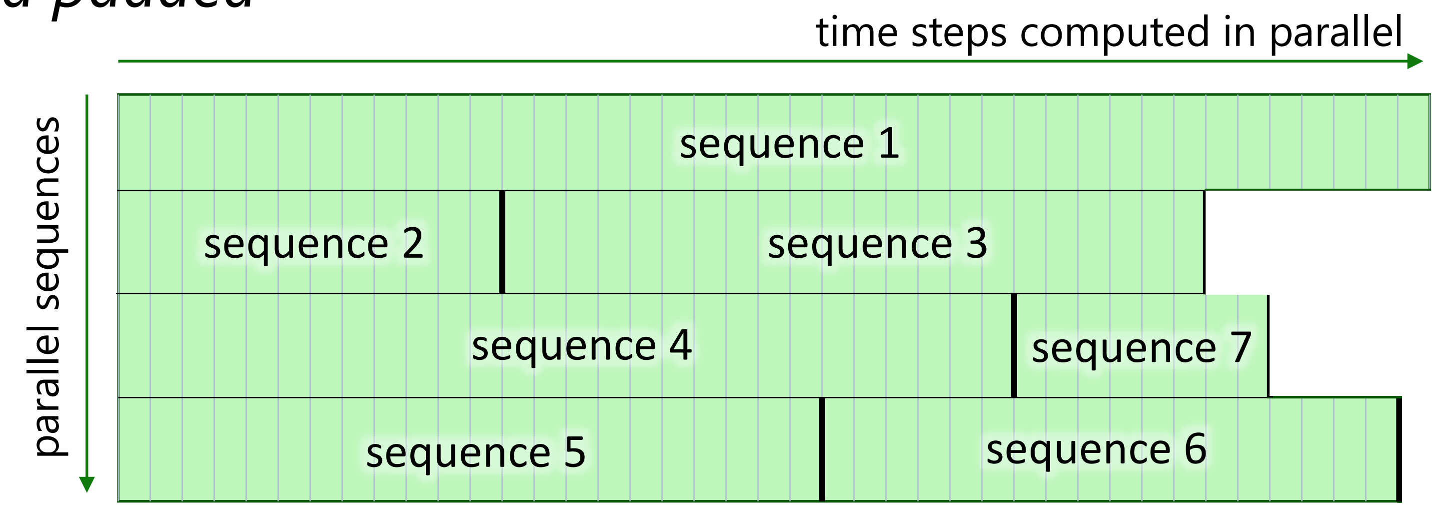
batching variable-length sequences

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



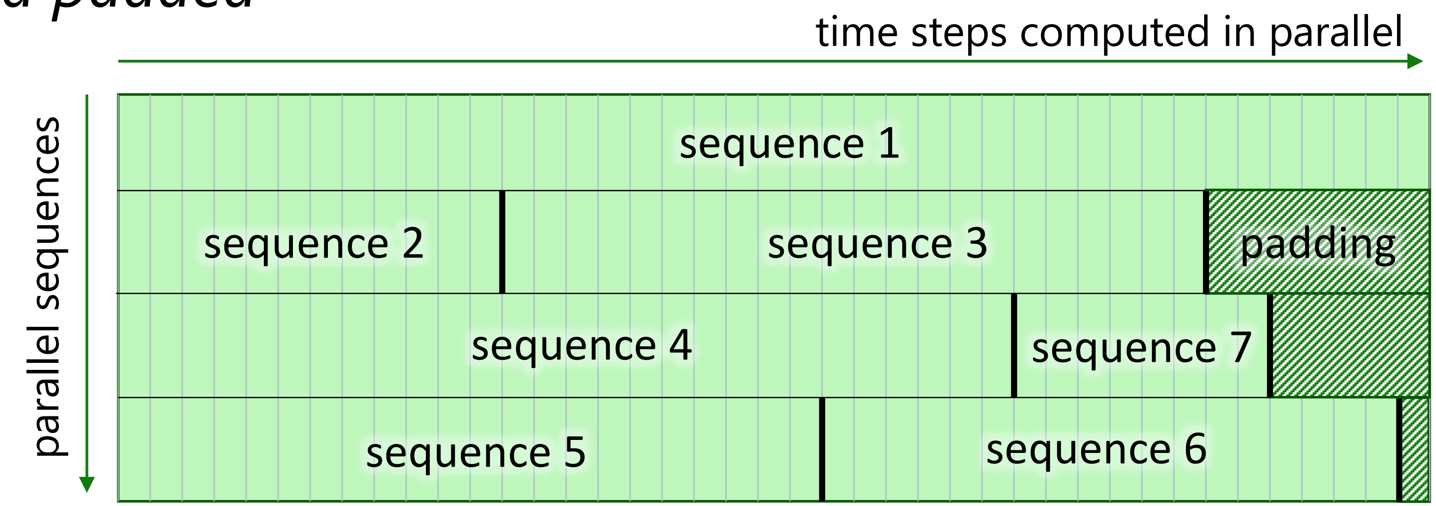
batching variable-length sequences

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



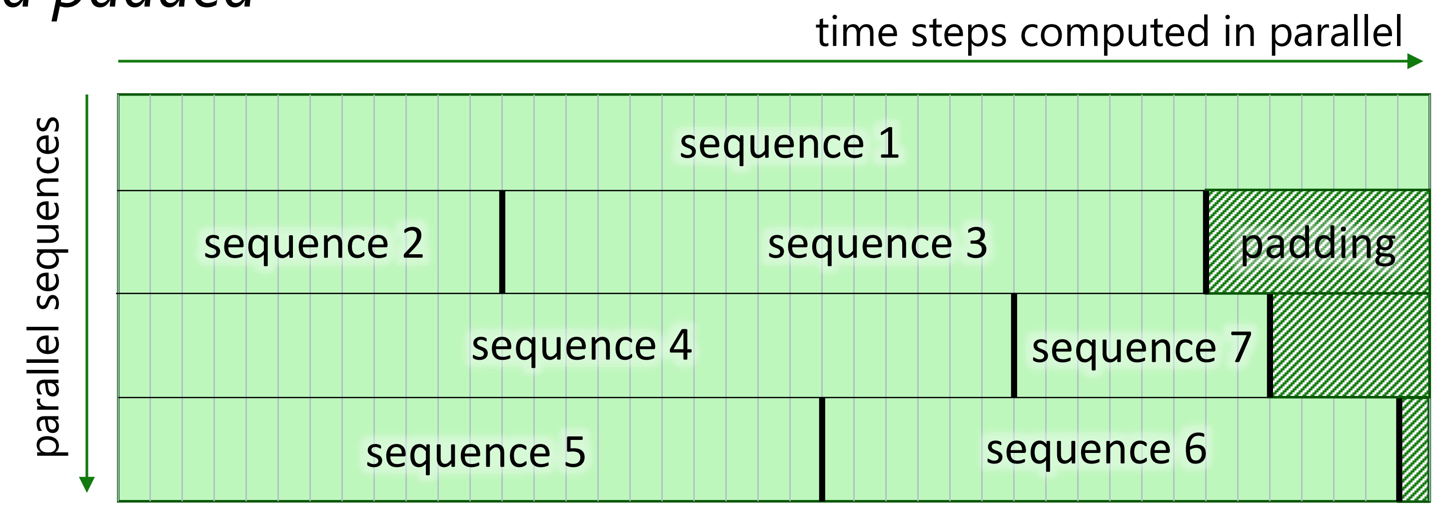
batching variable-length sequences

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



batching variable-length sequences

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



- fully transparent batching
  - recurrent → CNTK unrolls, handles sequence boundaries
  - non-recurrent operations → parallel
  - sequence reductions → mask

data-parallel training

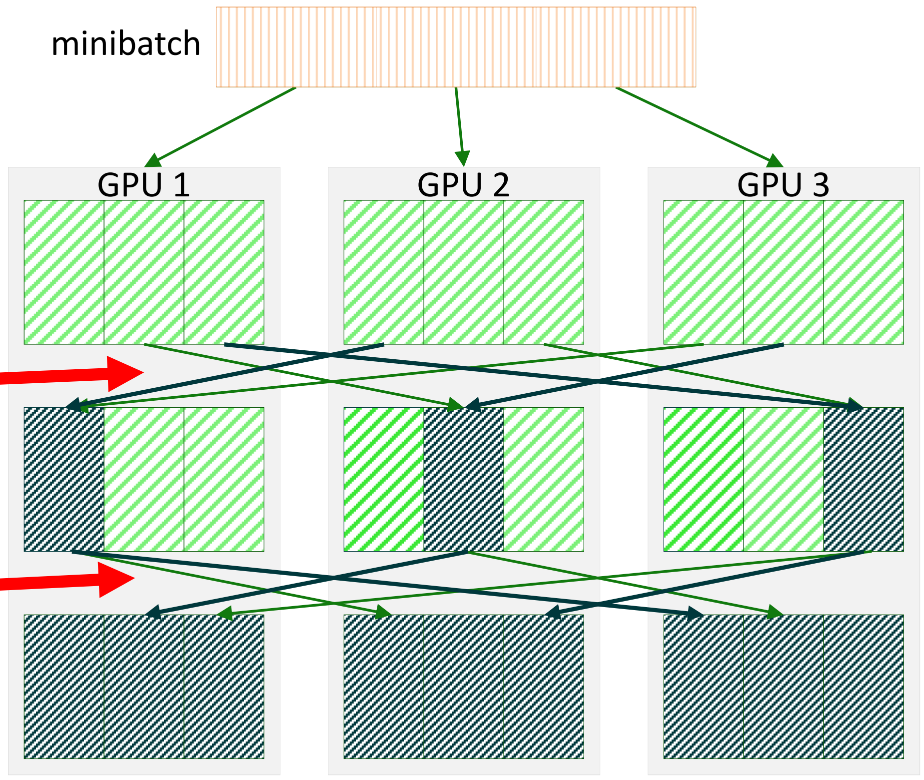
how to reduce communication cost:

communicate less each time

- 1-bit SGD:
  [F. Seide, H. Fu, J. Droppo, G. Li, D. Yu: "1-Bit Stochastic Gradient Descent... Distributed Training of Speech DNNs", Interspeech 2014]
  - quantize gradients to 1 bit per value
  - trick: carry over quantization error to next minibatch

1-bit quantized with residual

1-bit quantized with residual



data-parallel training

how to reduce communication cost:

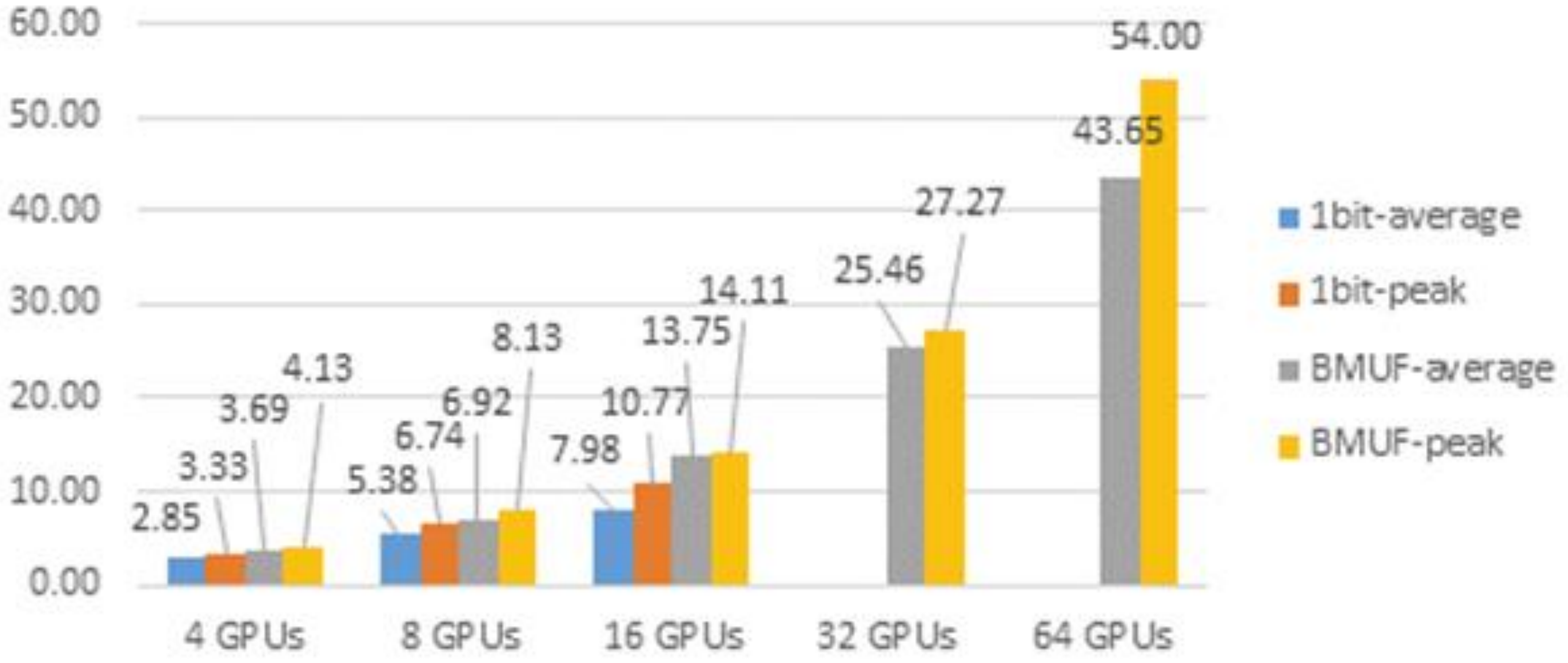
communicate less each time

- 1-bit SGD: [F. Seide, H. Fu, J. Droppo, G. Li, D. Yu: "1-Bit Stochastic Gradient Descent...Distributed Training of Speech DNNs", Interspeech 2014]
  - quantize gradients to 1 bit per value
  - trick: carry over quantization error to next minibatch

communicate less often

- automatic MB sizing [F. Seide, H. Fu, J. Droppo, G. Li, D. Yu: "ON Parallelizability of Stochastic Gradient Descent...", ICASSP 2014]
- block momentum [K. Chen, Q. Huo: "Scalable training of deep learning machines by incremental block training...," ICASSP 2016]
  - very effective parallelization method
  - combines model averaging with error-residual idea

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]

runtimes in Human Parity project

- data parallel training with 1-bit SGD:
  - up to 32 Maxwell GPUs per job (total farm had several hundred)
  - key enabler for this project
  - reduced training times from months to weeks
    - BLSTM: 8 GPUs (one box); rough CE AMs: ~1 day; fully converged after ~5 days; discriminative training: another ~5 days
    - CNNs and LACE: 16 GPUs (4 boxes); single GPU would take 50 days per data pass*!*
- model size on the order of 50M parameters
- perf (one GPU):

Processing step	Hardware	DNN	ResNet-CNN	BLSTM	LACE
AM training	GPU	0.012	0.60	0.022	0.23
AM evaluation	GPU	0.0064	0.15	0.0081	0.081
AM evaluation	CPU	0.052	11.7	n/a	8.47
Decoding	GPU	1.04	1.19	1.40	1.38

CNTK's approach to the two key questions:

- **efficient network authoring**
  - **networks as function objects**, well-matching the nature of DNNs
  - focus on **what, not how**
  - familiar syntax and flexibility **in Python**
- **efficient execution**
  - graph → parallel program through **automatic minibatching**
  - **symbolic loops** with dynamic scheduling
  - unique **parallel training algorithms** (1-bit SGD, Block Momentum)

Cognitive Toolkit: deep learning like Microsoft product groups

- ease of use
  - *what*, not *how*
  - powerful library
  - minibatching is automatic
- fast
  - optimized for NVidia GPUs & libraries
  - easy yet best-in-class multi-GPU/multi-server support
- flexible
  - Python and C++ API, powerful & composable
  - integrates with ONNX, WinML, Keras, R, C#, Java
- 1st-class on Linux and Windows
- train like MS product groups: internal=external version

Microsoft Cognitive Toolkit

aka.ms/CognitiveToolkit

Microsoft

Summary and Outlook

Summary

- Reached a significant milestone in automatic speech recognition
- Human and ASR are similar in
  - overall accuracy
  - types of errors
  - dependence on inherent speaker difficulty
- Achieved via
  - Deep convolutional and recurrent networks
  - Trained efficiently, in parallel on large matched speech corpus
  - Combining complementary models using different architectures
- CNTK's efficiency & data-parallel operation was critical enabler

Outlook

- Speech recognition is not solved!
- Need to work on
  - Robustness to acoustic environment (e.g., far-field mics, overlap)
  - Speaker mismatch (e.g., accented speech)
  - Style mismatch (e.g., planned vs. spontaneous, single vs. multiple spkrs)
- Computational challenges
  - Inference too expensive for mobile devices
  - Static graph limits what can be expressed → Dynamic networks

Thank You!

Questions?

anstolck@microsoft.com *fseide@microsoft.com*