Deep Speech 2

Presented by
Oscar & Siddharth

0) High level overview

What is Deep Speech 2

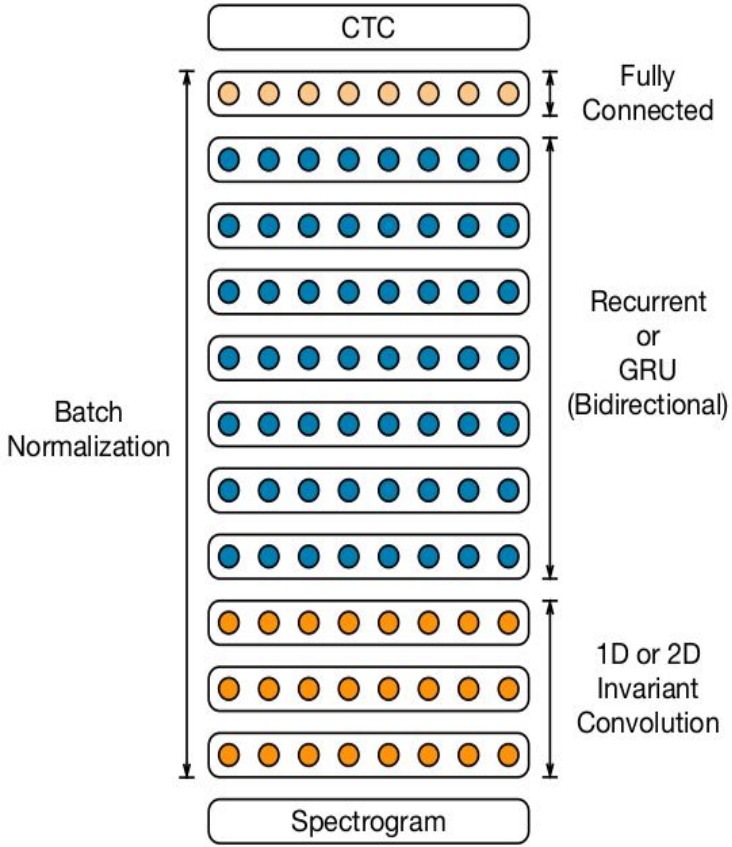
- End-to-End Deep learning based speech recognition system
- No need for hand engineered components
- Is able to handle noisy environments, different accents and languages
- By exploiting HPC, Deep Speech 2 is around 7x faster than Deep Speech 1, up to 43% more accurate
- Possible to deploy the system in online setting
- Able to beat Humans on some benchmarks

Deep Speech 2 - Components

- Model Architectures
  - Combination of convolutional, recurrent and fully connected layers
  - Different number of recurrent layers ranging from 1-7
- Large Training Datasets
  - Around 11,940 hrs of English speech and 9,400 hrs of Mandarin
  - Augmenting Data using noise
- Computational Scale - Hardware, Optimizations using HPC techniques
  - Multiple (8-16) GPUs training single model in parallel using synchronous SGD
  - Custom implementation of CTC loss function, Memory allocator, All-reduce function
  - Training time cut down from weeks to days using above optimizations

1) Deep Learning Model

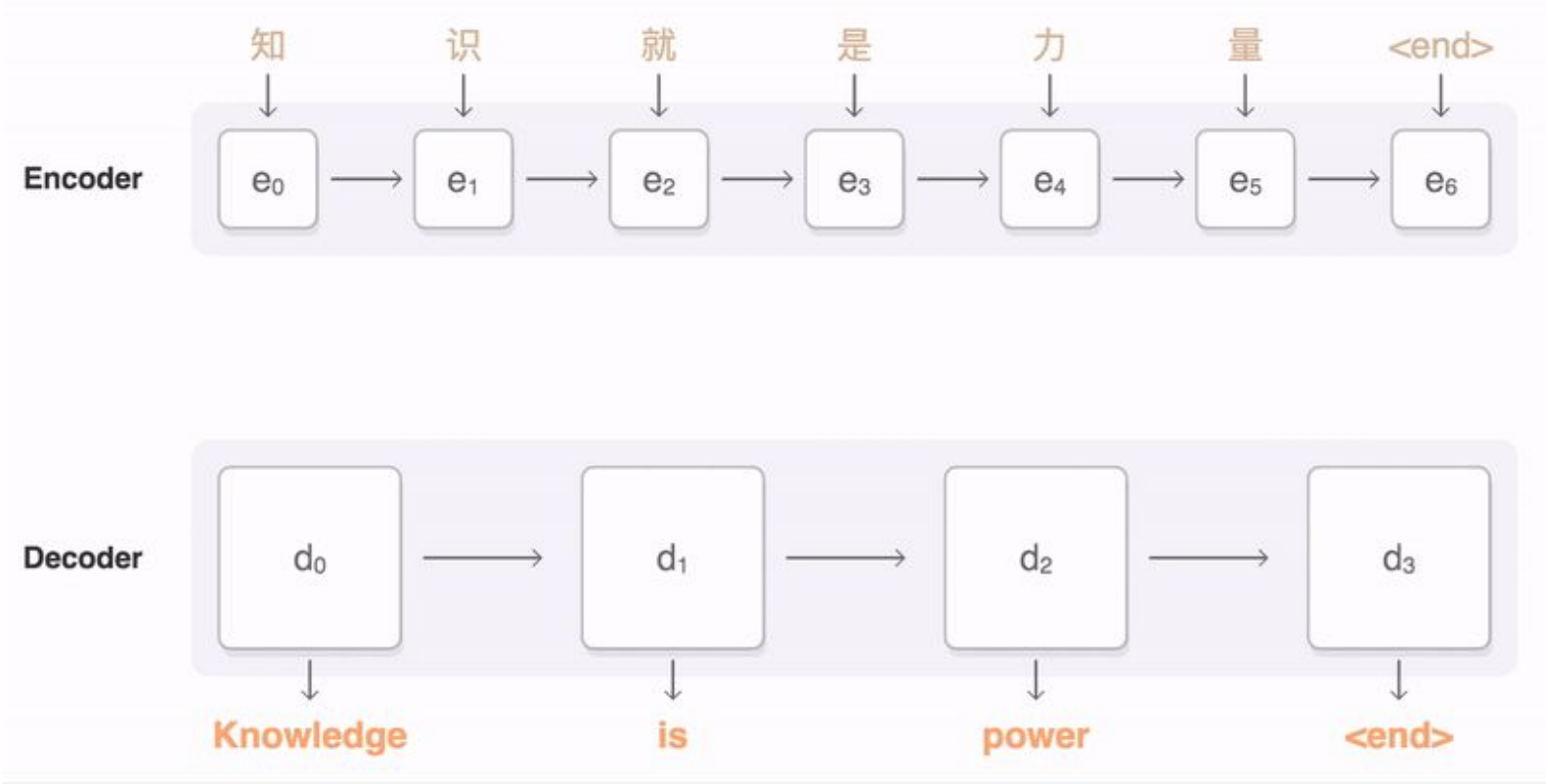
Overview of the Architecture



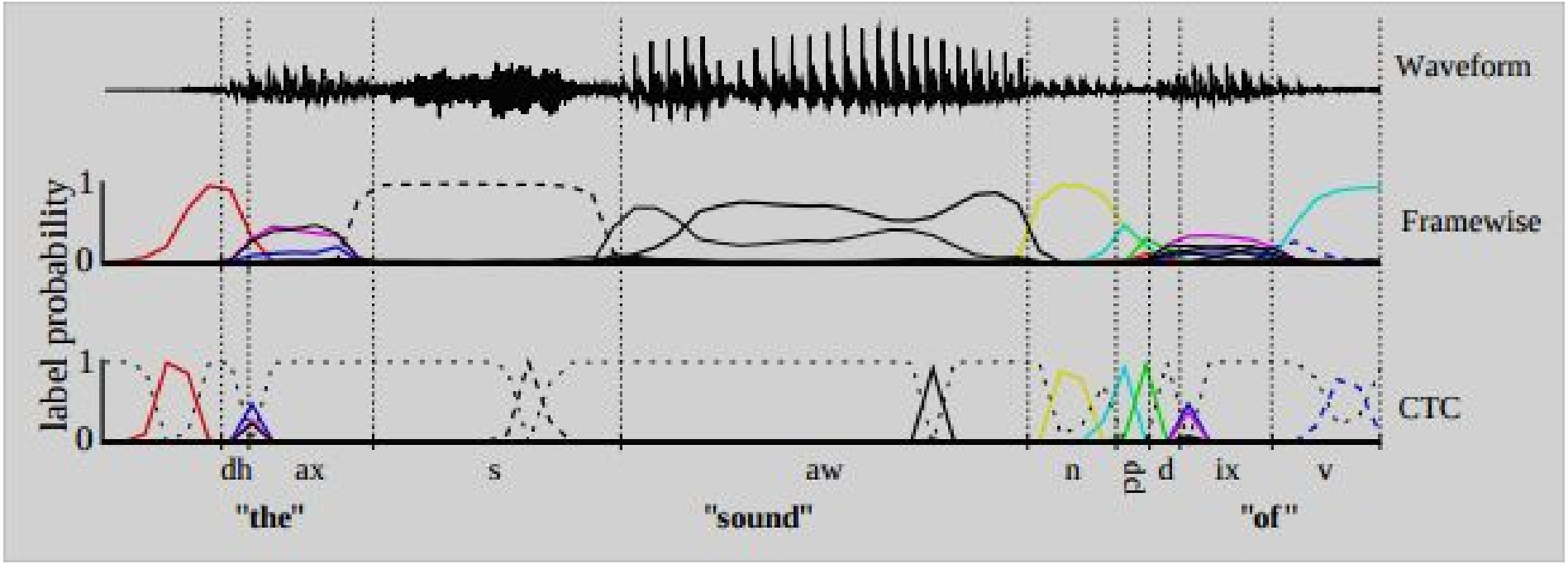
End-to-end Speech Recognition

- First method
  - Encoder RNN maps input to a fixed length vector
  - Decoder RMM expands the fixed length vector into a sequence of output predictions
  - Attention mechanism helps with long inputs or outputs
  - Works well with both phonemes or graphemes
- Second method (works better)
  - RNN with a CTC loss function
  - Works well with graphemes
  - Works well with phonemes if we have a lexicon
  - CTC-RNN network can be trained from scratch without the need of frame-wise alignments from a GMM-HMM model for pre-training
- Both methods map variable length audio input to variable length output

Attention Mechanism



Connectionist Temporal Classification (CTC)



Recurrent BatchNorm

In a typical feed-forward layer containing an affine transformation followed by a non-linearity $f(\cdot)$, we insert a BatchNorm transformation by applying $f(\mathcal{B}(Wh))$ instead of $f(Wh + b)$, where

$$\mathcal{B}(x) = \gamma \frac{x - \mathrm{E}[x]}{(\mathrm{Var}[x] + \epsilon)^{1/2}} + \beta. \tag{6}$$

$$\overrightarrow{h}_t^l = f(\mathcal{B}(W^l h_t^{l-1} + \overrightarrow{U}^l \overrightarrow{h}_{t-1}^l)).$$

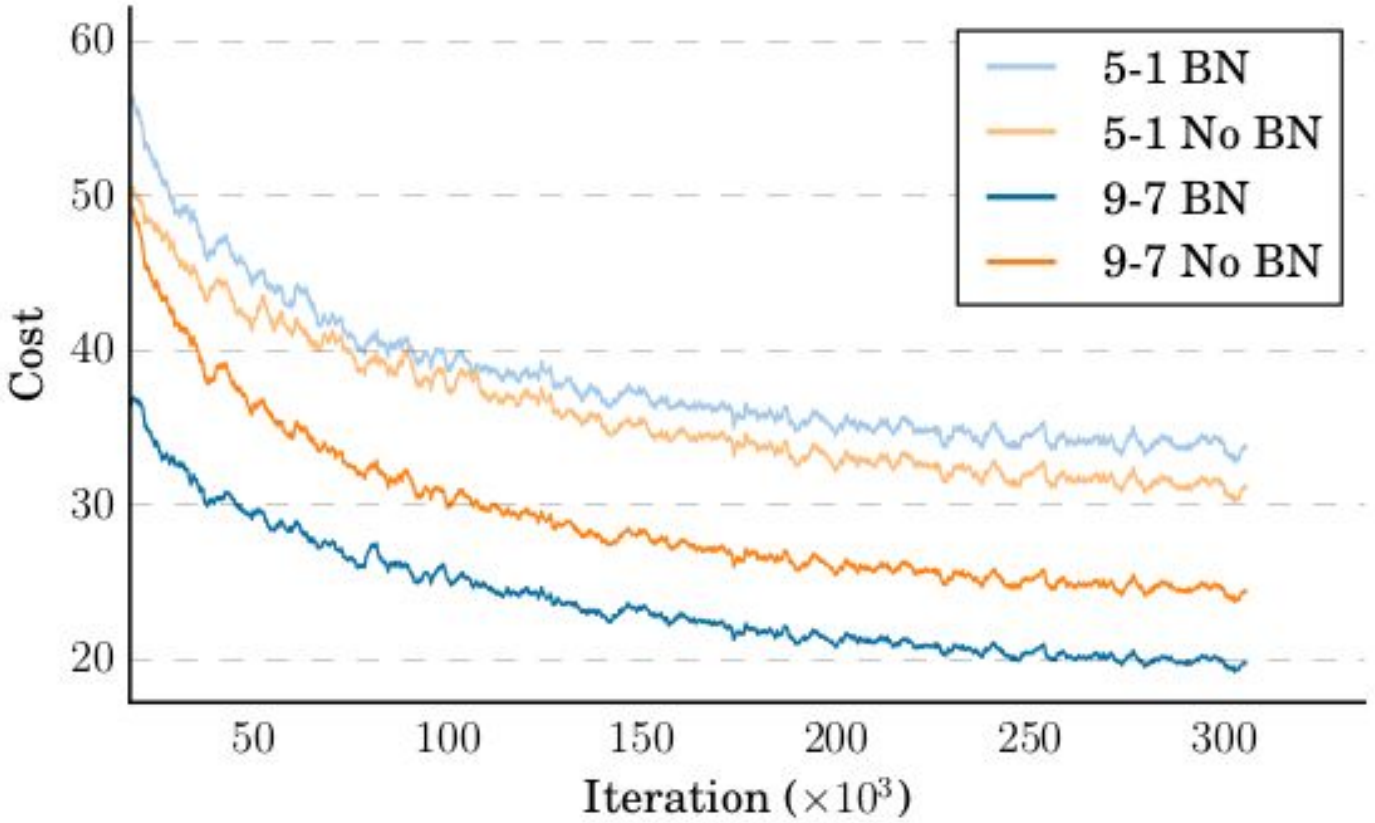
Sample-wise normalization

$$\overrightarrow{h}_t^l = f(\mathcal{B}(W^l h_t^{l-1}) + \overrightarrow{U}^l \overrightarrow{h}_{t-1}^l).$$

Sequence-wise normalization

Architecture	Hidden Units	Train		Dev	
		Baseline	BatchNorm	Baseline	BatchNorm
1 RNN, 5 total	2400	10.55	11.99	13.55	14.40
3 RNN, 5 total	1880	9.55	8.29	11.61	10.56
5 RNN, 7 total	1510	8.59	7.61	10.77	9.78
7 RNN, 9 total	1280	8.76	7.68	10.83	9.52

Table 1: Comparison of WER on a training and development set for various depths of RNN, with and without BatchNorm. The number of parameters is kept constant as the depth increases, thus the number of hidden units per layer decreases. All networks have 38 million parameters. The architecture "M RNN, N total" implies 1 layer of 1D convolution at the input, M consecutive bidirectional RNN layers, and the rest as fully-connected layers with N total layers in the network.


60
50
40
30
20
Cost
50
100
150
200
250
300
Iteration ($\times 10^3$)
5-1 BN
5-1 No BN
9-7 BN
9-7 No BN

SortaGrad

- Longer examples are more challenging
  - Solution one: Truncating Backpropagation through time
  - Solution two: Curriculum learning
- SortaGrad is a curriculum learning strategy

The CTC cost function that we use implicitly depends on the length of the utterance,

$$\mathcal{L}(x, y; \theta) = -\log \sum_{\ell \in \text{Align}(x,y)} \prod_{t}^{T} p_{\text{ctc}}(\ell_t | x; \theta).$$

	Train		Dev	
	Baseline	BatchNorm	Baseline	BatchNorm
Not Sorted	10.71	8.04	11.96	9.78
Sorted	8.76	7.68	10.83	9.52

Table 2: Comparison of WER on a training and development set with and without SortaGrad, and with and without batch normalization.

Simple RNNs vs GRUs

Architecture	Simple RNN	GRU
5 layers, 1 Recurrent	14.40	10.53
5 layers, 3 Recurrent	10.56	8.00
7 layers, 5 Recurrent	9.78	7.79
9 layers, 7 Recurrent	9.52	8.19

Table 3: Comparison of development set WER for networks with either simple RNN or GRU, for various depths. All models have batch normalization, one layer of 1D-invariant convolution, and approximately 38 million parameters.

Convolutions

- 1D: Time-only domain
- 2D: Time-and-frequency domain

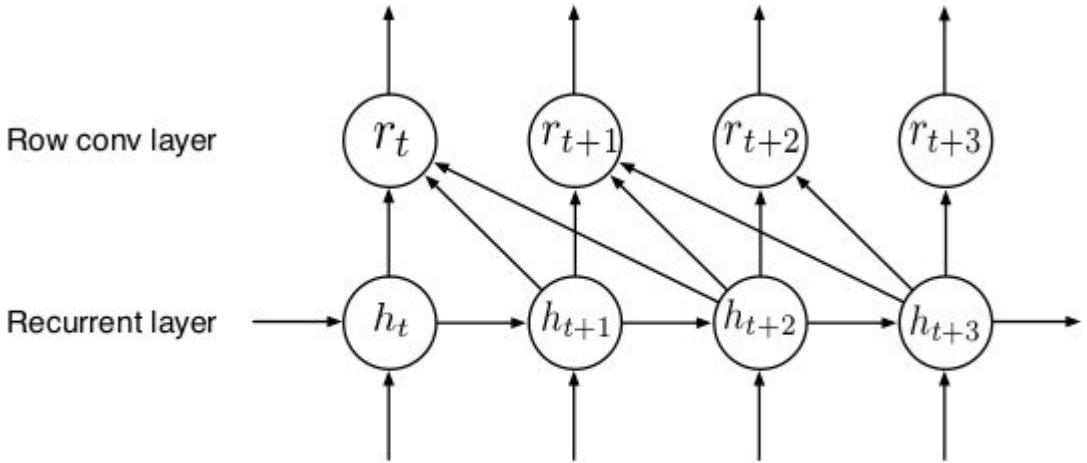


Figure 3: Row convolution architecture with future context size of 2

Architecture	Channels	Filter dimension	Stride	Regular Dev	Noisy Dev
1-layer 1D	1280	11	2	9.52	19.36
2-layer 1D	640, 640	5, 5	1, 2	9.67	19.21
3-layer 1D	512, 512, 512	5, 5, 5	1, 1, 2	9.20	20.22
1-layer 2D	32	41x11	2x2	8.94	16.22
2-layer 2D	32, 32	41x11, 21x11	2x2, 2x1	9.06	15.71
3-layer 2D	32, 32, 96	41x11, 21x11, 21x11	2x2, 2x1, 2x1	8.61	14.74

Table 4: Comparison of WER for various arrangements of convolutional layers. In all cases, the convolutions are followed by 7 recurrent layers and 1 fully connected layer. For 2D-invariant convolutions the first dimension is frequency and the second dimension is time. All models have BatchNorm, SortaGrad, and 35 million parameters.

Using Language Model

- With large networks like the ones used in this system, given enough training data, the network learns an implicit language model.
- However since the labeled training data is still small compared to unlabeled text corpora, the use of explicit LM improves WER

$$Q(y) = \log(p_{\text{ctc}}(y|x)) + \alpha \log(p_{\text{lm}}(y)) + \beta \,\text{word\_count}(y)$$

- Goal is to find y that maximizes Q(y). Here α and β are tunable parameters found using development set.
- The optimal y is found using Beam Search

Effect of using Language Models

Language	Architecture	Dev no LM	Dev LM
English	5-layer, 1 RNN	27.79	14.39
English	9-layer, 7 RNN	14.93	9.52
Mandarin	5-layer, 1 RNN	9.80	7.13
Mandarin	9-layer, 7 RNN	7.55	5.81

Table 6: Comparison of WER for English and CER for Mandarin with and without a language model. These are simple RNN models with only one layer of 1D invariant convolution.

2) HPC Optimizations

System Optimizations

- System Hardware
  - Dense compute nodes with 8 Titan X GPUs per Node
- System Software
  - a deep learning library written in C++
  - high-performance linear algebra library written in both CUDA and C++
- Data parallelism
  - A minibatch of 512 is distributed across 8 GPUs such that each GPU processes 64 out of 512
  - One process per GPU
  - Gradient matrices exchanged during back propagation using inter process communication - all reduce operation
  - Custom implementation of all reduce function - avoids copying between CPU & GPU

Dense Compute Node

- 2 intel CPUs, 8 Titan X GPUs, 384 GB of CPU memory and an 8 TB storage volume

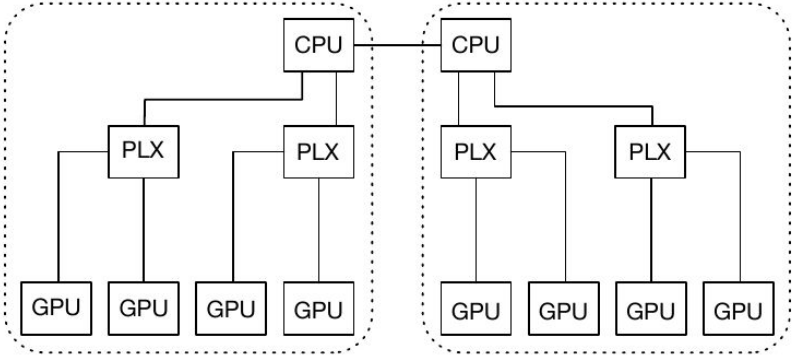


Figure 8: Schematic of our training node where PLX indicates a PCI switch and the dotted box includes all devices that are connected by the same PCI root complex.

Effect of Data parallelism

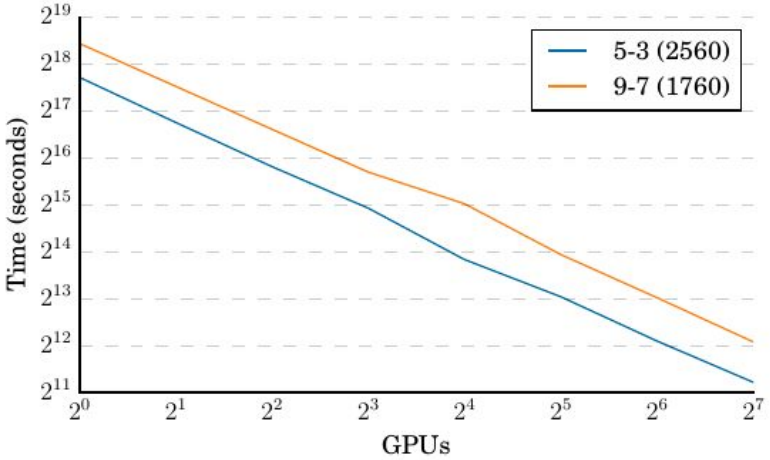


Figure 4: Scaling comparison of two networks—a 5 layer model with 3 recurrent layers containing 2560 hidden units in each layer and a 9 layer model with 7 recurrent layers containing 1760 hidden units in each layer. The times shown are to train 1 epoch. The 5 layer model trains faster because it uses larger matrices and is more computationally efficient.

Performance Gains (their all-reduce v/s OpenMPI's)

GPU	OpenMPI all-reduce	Our all-reduce	Performance Gain
4	55359.1	2587.4	21.4
8	48881.6	2470.9	19.8
16	21562.6	1393.7	15.5
32	8191.8	1339.6	6.1
64	1395.2	611.0	2.3
128	1602.1	422.6	3.8

Table 7: Comparison of two different all-reduce implementations. All times are in seconds. Performance gain is the ratio of OpenMPI all-reduce time to our all-reduce time.

GPU implementation of CTC loss function

- OpenMP parallel implementation of CTC on CPU was not scalable for two reasons
  - It turned out to be computationally expensive for deeper RNNs
  - Transferring large activation matrices between CPUs and GPUs wasted lot of interconnect bandwidth for CTC computation. This bandwidth could be used for other purposes like exchanging gradient matrices for more data parallelism.
- To overcome this limitations, they implemented GPU version of CTC loss function

Language	Architecture	CPU CTC Time	GPU CTC Time	Speedup
English	5-layer, 3 RNN	5888.12	203.56	28.9
Mandarin	5-layer, 3 RNN	1688.01	135.05	12.5

Table 8: Comparison of time spent in seconds in computing the CTC loss function and gradient in one epoch for two different implementations. Speedup is the ratio of CPU CTC time to GPU CTC time.

Custom Memory Allocator

- cudaMalloc and std::malloc are optimized for situations in which multiple applications are running on the system and sharing memory resources
- In this case, it is only the model that is running on the system, thus the default allocators cause 2x slow down.
- To overcome this, they implement their own custom memory allocators for both CPUs and GPUs.
- Also added fallback mechanism where if the GPU runs out of memory, GPU will use CPU memory.
- In nutshell, custom allocator along with fall back mechanism makes their system more robust

Training Data

- For English, Total of 11,940 hours of labeled speech data (around 8 million utterances)
- For Mandarin, 9,400 hours of labeled speech data (around 11 million utterances)
- Above data is pre-processed to convert it into short utterances by performing:
  - Alignment - align the audio and the transcript
  - Segmentation - segment long audio into 7 sec long utterances
  - Filtering - remove erroneous alignments using linear classification
- Training data also augmented by adding noise to 40% of utterances chosen at random

Results - Amount of training data

Fraction of Data	Hours	Regular Dev	Noisy Dev
1%	120	29.23	50.97
10%	1200	13.80	22.99
20%	2400	11.65	20.41
50%	6000	9.51	15.90
100%	12000	8.46	13.59

Table 10: Comparison of English WER for Regular and Noisy development sets on increasing training dataset size. The architecture is a 9-layer model with 2 layers of 2D-invariant convolution and 7 recurrent layers with 68M parameters.

40% decrease in WER for each 10x increase in training data

More Results - Model size

Increasing the size of layers keeping number of layers and other parameters fixed

Model size	Model type	Regular Dev	Noisy Dev
18×10^6	GRU	10.59	21.38
38×10^6	GRU	9.06	17.07
70×10^6	GRU	8.54	15.98
70×10^6	RNN	8.44	15.09
100×10^6	GRU	7.78	14.17
100×10^6	RNN	7.73	13.06

Table 11: Comparing the effect of model size on the WER of the English speech system on both the regular and noisy development sets. We vary the number of hidden units in all but the convolutional layers. The GRU model has 3 layers of bidirectional GRUs with 1 layer of 2D-invariant convolution. The RNN model has 7 layers of bidirectional simple recurrence with 3 layers of 2D-invariant convolution. Both models output bigrams with a temporal stride of 3. All models contain approximately 35 million parameters and are trained with BatchNorm and SortaGrad.

More Results - Performance on Read speech

Read Speech			
Test set	DS1	DS2	Human
WSJ eval'92	4.94	3.60	5.03
WSJ eval'93	6.94	4.98	8.08
LibriSpeech test-clean	7.89	5.33	5.83
LibriSpeech test-other	21.74	13.25	12.69

Table 13: Comparison of WER for two speech systems and human level performance on read speech.

As we can see the system surpasses human performance on 3 out 4 datasets.

More Results - Performance on Accented speech

Accented Speech			
Test set	DS1	DS2	Human
VoxForge American-Canadian	15.01	7.55	4.85
VoxForge Commonwealth	28.46	13.56	8.15
VoxForge European	31.20	17.55	12.76
VoxForge Indian	45.35	22.44	22.15

Table 14: Comparing WER of the DS1 system to the DS2 system on accented speech.

Results on a total of 4096 test examples with 1024 per group

More Results - Performance on Noisy Speech

Noisy Speech			
Test set	DS1	DS2	Human
CHiME eval clean	6.30	3.34	3.46
CHiME eval real	67.94	21.79	11.84
CHiME eval sim	80.27	45.05	31.33

Table 15: Comparison of DS1 and DS2 system on noisy speech. "CHiME eval clean" is a noise-free baseline. The "CHiME eval real" dataset is collected in real noisy environments and the "CHiME eval sim" dataset has similar noise synthetically added to clean speech. Note that we use only one of the six channels to test each utterance.

Online Deployment - Batch Dispatch

- Challenges
  - Bidirectional RNNs require the entire utterance to be presented before transcribing can be done
  - Serving individual requests is memory bandwidth bound as the system must load all the weights of the network for each individual request
  - Serving individual requests limits the amount of parallelism that can be exploited in multi-core systems
- With Batch Dispatch, there is a tradeoff between increased batch size and better efficiency and increased latency.
- They use eager batching scheme - process each batch as soon as the previous batch is processed.
- Eager batching scheme is best for end-user but not computationally most efficient

Online Deployment - Latencies

As the system load increases, the batch size increases as per the Eager batching scheme which helps to keep the latency low

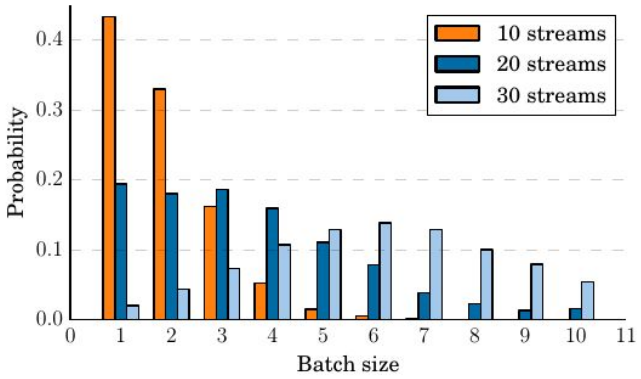


Figure 5: Probability that a request is processed in a batch of given size

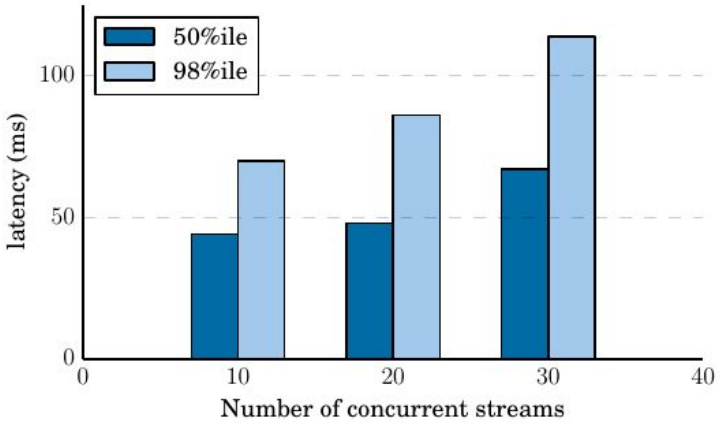


Figure 6: Median and 98 percentile latencies as a function of server load

On a held out set of 2000 utterances, their research system achieves 5.81 % CER whereas the deployed system achieves 6.1 % CER.

Final Online System

- Low deployment latency
- Reduced precision to 16-bit instead of 32-bit
- One row convolutional layer
- 5 Forward-only RNN layers with 2560 units per layer
- One fully-connected layer with 2560 hidden units
- Only 5% relative degradation over their research system

Questions ?