Deep Speech 2: End-to-End Speech Recognition in English and Mandarin

Presenter: Saket Gurukar

Most of the slides are borrowed from Adam Coates, Amnon Drory & Matan Karo
Speech recognition

- Many exciting & valuable applications
Speech recognition

• Many exciting & valuable applications

“Speech Is 3x Faster than Typing for English and Mandarin Text Entry on Mobile Devices”

[Ruan et al., 2016]
Speech recognition

• Given speech audio, generate a transcript.

Important goal of AI: historically hard for machines, easy for people.
Traditional ASR pipeline

- Traditional systems usually model phoneme sequences instead of words. This necessitates a dictionary or other model to translate.

\[
W^* = \arg\max_W P(W|X) = \arg\max_W \sum_Q P(O|Q)P(Q|W)P(W)
\]
Traditional ASR pipeline

• Traditional pipeline is highly tweak-able, but also hard to get working well.
• Historically, each part of system has own set of challenges.
  – E.g., choosing feature representation.
End to End Learning

- Goal: Replace all individual components in pipeline with a single Deep Learning Framework
Speech Basics  (short intro)
Audio Input

Spectrogram

• Take a small window (e.g., 20ms) of waveform.
  – Compute FFT and take magnitude. (i.e., power)
  – Describes frequency content in local window.
Audio Input

Spectrogram

- Concatenate frames from adjacent windows to form “spectrogram”.

![Spectrogram Diagram](image)
Acoustic Model

• Goal: create a neural network (DNN/RNN) from which we can extract transcription, $y$.
  – Train from labeled pairs $(x, y^*)$

\[ y = "Hello" \]
How to classify whether the prediction is right?

• Problem: Different audio sound of same word leads to different transcriptions.

• Solution: Connectionist Temporal Classification (CTC)

Define a mapping $\beta(c) \rightarrow y$.

– Given a specific character sequence $c$, squeeze out duplicates + blanks to yield transcription:

$$y = \beta(c) = \beta(\text{HHH}_E\_\text{LL}_L\text{LO}\_\_\_\_) = "\text{HELLO}"$$
Architecture
Model Architecture

- 11 layers
- The chosen architecture:
  - 3 x 2D conv,
  - 7 x RNN,
  - 1 x FC
- Batch Normalization along the DNN.
Convolution

Convolutional Layers: Audio

Frequency

Time
(mini)Batch Normalization

• A typical feed-forward layer consists of an affine transformation followed by a non-linearity $f(\ W^*\ h + \ b)$

• BatchNorm transformation $f(B(Wh))$

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

• The terms $E$ and $Var$ are the empirical mean and variance over a minibatch.
(mini)Batch Normalization

**Figure**: Training curves of two models trained with and without BatchNorm.

\[ N - M \Rightarrow \text{Total } N \text{ layers, } M \text{ RNN layers, 1 Convolution layer, Rest FC layers} \]
Recurrent neural networks

Bi-Directional RNN
Accuracy Optimizations
Preprocessing: SortaGrad

- Dealing with different lengths of utterances
- Try to keep similar – length utterances together

“In first epoch, train on simple samples first.”

Bad minibatch:

Good minibatch:

(LibriSpeech clean data.)
Accuracy Optimizations : SortaGrad
Accuracy Optimizations: Language Models

• Probabilistic models of a language
  • whether a given sentence makes sense or not
  • which word is reasonable in the current context

• Models learnt using KenLM toolkit.
• Helpful in disambiguate homophones.

• Example: *he expects the agent to sell it for two hundred dollars.*
Accuracy Optimizations: Language Models

<table>
<thead>
<tr>
<th>RNN output</th>
<th>Decoded Transcription</th>
</tr>
</thead>
<tbody>
<tr>
<td>what is the weather like in bostin right now</td>
<td>what is the weather like in boston right now</td>
</tr>
<tr>
<td>prime miniter nerenr modi</td>
<td>prime minister narendra modi</td>
</tr>
<tr>
<td>arther n tickets for the game</td>
<td>are there any tickets for the game</td>
</tr>
</tbody>
</table>

From Hannun et al., 2014.

<table>
<thead>
<tr>
<th>Language</th>
<th>Architecture</th>
<th>Dev no LM</th>
<th>Dev LM</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>5-layer, 1 RNN</td>
<td>27.79</td>
<td>14.39</td>
</tr>
<tr>
<td>English</td>
<td>9-layer, 7 RNN</td>
<td>14.93</td>
<td>9.52</td>
</tr>
<tr>
<td>Mandarin</td>
<td>5-layer, 1 RNN</td>
<td>9.80</td>
<td>7.13</td>
</tr>
<tr>
<td>Mandarin</td>
<td>9-layer, 7 RNN</td>
<td>7.55</td>
<td>5.81</td>
</tr>
</tbody>
</table>

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.
Accuracy Optimizations: Striding

• Longer stride leads to faster computation
  • Penalty in accuracy

• Trick: Predict bigrams instead of characters (unigrams).

• Example: Audio of “the cat sat” will be transcripted as 
  \[th\, e\, space\, ca\, t\, space\, sa\, t\].

<table>
<thead>
<tr>
<th></th>
<th>Dev no LM</th>
<th>Dev LM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Stride</td>
<td>Bigrams</td>
</tr>
<tr>
<td>2</td>
<td>14.93</td>
<td>14.56</td>
</tr>
<tr>
<td>3</td>
<td>15.01</td>
<td>15.60</td>
</tr>
<tr>
<td>4</td>
<td>18.86</td>
<td>14.84</td>
</tr>
</tbody>
</table>

Table 5: Comparison of WER with different amounts of striding for unigram and bigram outputs on a model with 1 layer of 1D-invariant convolution, 7 recurrent layers, and 1 fully connected layer. All models have BatchNorm, SortaGrad, and 35 million parameters. The models are compared on a development set with and without the use of a 5-gram language model.
Other Accuracy Optimizations

• Replace RNN cell with Gated Recurrent Units (GRU) cell
  • Update and forget gates
  • GRUs can retain long-term dependencies.

• Frequency Convolutions
  • Use of 2D convolution instead of 1D
System Optimizations
Scalability and Data-Parallelism

• Proposed architecture have tens of millions of parameters!
• Problem : Training algorithm takes tens of exaFLOPs to converge.

• Solution: Optimized software + more GPUs
  • 8 Titan X GPUs
  • 45% of the theoretical peak computational throughput.

• Synchronous SGD
  • Reproducible and deterministic.
  • Easy to Debug
Scalability and Data-Parallelism

- Bind one process to each GPU
- Processes exchange gradient matrices during the backpropagation by using all-reduce

*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.
Bottleneck

- Processes exchange gradient matrices
- New OpenMPI implementation
  - GPUdirect: avoids extraneous copies between CPU and GPU
  - Optimized all-reduce.

<table>
<thead>
<tr>
<th>GPU</th>
<th>OpenMPI all-reduce</th>
<th>Our all-reduce</th>
<th>Performance Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>55359.1</td>
<td>2587.4</td>
<td>21.4</td>
</tr>
<tr>
<td>8</td>
<td>48881.6</td>
<td>2470.9</td>
<td>19.8</td>
</tr>
<tr>
<td>16</td>
<td>21562.6</td>
<td>1393.7</td>
<td>15.5</td>
</tr>
<tr>
<td>32</td>
<td>8191.8</td>
<td>1339.6</td>
<td>6.1</td>
</tr>
<tr>
<td>64</td>
<td>1395.2</td>
<td>611.0</td>
<td>2.3</td>
</tr>
<tr>
<td>128</td>
<td>1602.1</td>
<td>422.6</td>
<td>3.8</td>
</tr>
</tbody>
</table>

*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

• CTC loss computation requires
  • transfer of activations from the GPUs to the CPU

• GPU implementation
  • Refactored code to simplify the dependencies in CTC calculation using shared memory.

<table>
<thead>
<tr>
<th>Language</th>
<th>Architecture</th>
<th>CPU CTC Time</th>
<th>GPU CTC Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>5-layer, 3 RNN</td>
<td>5888.12</td>
<td>203.56</td>
<td>28.9</td>
</tr>
<tr>
<td>Mandarin</td>
<td>5-layer, 3 RNN</td>
<td>1688.01</td>
<td>135.05</td>
<td>12.5</td>
</tr>
</tbody>
</table>

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.
Memory allocation

• Long speech utterances results in large allocations of memory
  • Bottleneck: standard cudaMalloc and std::malloc
  • Allocates non contiguous memory blocks for different utterences
  • This allocation strategy is good for multiple different applications

• Solution: Own implementation of memory allocator
  • Avoids fragmentation
  • 2X faster
Deployment Optimizations
Batch Dispatch

Throughput

• Large DNN/RNN models hard to deploy on CPUs.
• Large DNN/RNN models run great on GPUs.
  – But only if “batch size” is high enough.
  ➢ Processing 1 audio stream at a time is inefficient.

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>FLOPs</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.065 TFLOPs</td>
<td>1x</td>
</tr>
<tr>
<td>10</td>
<td>0.31 TFLOPs</td>
<td>5x</td>
</tr>
<tr>
<td>32</td>
<td>0.92 TFLOPs</td>
<td>14x</td>
</tr>
</tbody>
</table>

Performance for K1200 GPU:

[Chris Fougner]
Batch Dispatch

Throughput

• Batch packets together as data comes in.

- Arrows represent packet of speech data (e.g., 100ms).
- Idea: Process packets that arrive at similar times in parallel.

[Chris Fougner]
Latency

• Many acoustic model structures hard to serve in practice.
  – E.g., Bi-directional RNNs.

Must wait for all audio to arrive before computing.
Latency

- Fix: bake limited context into model structure.
Results
Training Data: Annotated Audio

Thousands of hours of annotated speech for training: in **English** and **Mandarin**.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Speech Type</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSJ</td>
<td>read</td>
<td>80</td>
</tr>
<tr>
<td>Switchboard</td>
<td>conversational</td>
<td>300</td>
</tr>
<tr>
<td>Fisher</td>
<td>conversational</td>
<td>2000</td>
</tr>
<tr>
<td>LibriSpeech</td>
<td>read</td>
<td>960</td>
</tr>
<tr>
<td>Baidu</td>
<td>read</td>
<td>5000</td>
</tr>
<tr>
<td>Baidu</td>
<td>mixed</td>
<td>3600</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>11940</strong></td>
</tr>
</tbody>
</table>

*Table 9*: Summary of the datasets used to train DS2 in English. The Wall Street Journal (WSJ), Switchboard and Fisher [13] corpora are all published by the Linguistic Data Consortium. The LibriSpeech dataset [46] is available free on-line. The other datasets are internal Baidu corpora.
# Results: Sometimes better than Humans

## Read Speech

<table>
<thead>
<tr>
<th>Test set</th>
<th>DS1</th>
<th>DS2</th>
<th>Human</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSJ eval’92</td>
<td>4.94</td>
<td>3.60</td>
<td>5.03</td>
</tr>
<tr>
<td>WSJ eval’93</td>
<td>6.94</td>
<td>4.98</td>
<td>8.08</td>
</tr>
<tr>
<td>LibriSpeech test-clean</td>
<td>7.89</td>
<td>5.33</td>
<td>5.83</td>
</tr>
<tr>
<td>LibriSpeech test-other</td>
<td>21.74</td>
<td>13.25</td>
<td>12.69</td>
</tr>
</tbody>
</table>

## Noisy Speech

<table>
<thead>
<tr>
<th>Test set</th>
<th>DS1</th>
<th>DS2</th>
<th>Human</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHiME eval clean</td>
<td>6.30</td>
<td>3.34</td>
<td>3.46</td>
</tr>
<tr>
<td>CHiME eval real</td>
<td>67.94</td>
<td>21.79</td>
<td>11.84</td>
</tr>
<tr>
<td>CHiME eval sim</td>
<td>80.27</td>
<td>45.05</td>
<td>31.33</td>
</tr>
</tbody>
</table>

19/12/2017
Results: Sometimes better than Humans

<table>
<thead>
<tr>
<th>Test set</th>
<th>DS1</th>
<th>DS2</th>
<th>Human</th>
</tr>
</thead>
<tbody>
<tr>
<td>VoxForge American-Canadian</td>
<td>15.01</td>
<td>7.55</td>
<td>4.85</td>
</tr>
<tr>
<td>VoxForge Commonwealth</td>
<td>28.46</td>
<td>13.56</td>
<td>8.15</td>
</tr>
<tr>
<td>VoxForge European</td>
<td>31.20</td>
<td>17.55</td>
<td>12.76</td>
</tr>
<tr>
<td>VoxForge Indian</td>
<td>45.35</td>
<td>22.44</td>
<td>22.15</td>
</tr>
</tbody>
</table>

Accented Speech
A Benchmark for Machine Learning from an Academic/Industry Cooperative

Researchers from:
Baidu, Google, Harvard, Stanford, and UC Berkeley
Companies:

AMD  Baidu  Google  Intel  Sambanova  Wave Computing

Researchers from these educational institutions:

Harvard University  Stanford University  University of California, Berkeley  University of Minnesota  University of Toronto
Goals for MLPerf

1. Accelerate progress in ML via fair and useful measurement
2. Encourage innovation across state-of-the-art ML systems
3. Serve both industrial and research communities
4. Enforce replicability to ensure reliable results
5. Keep benchmark effort affordable so all can play
Difficulties of ML Benchmarking

1. Diversity in deep learning models used
   a. Problem domain
   b. Models
   c. Datasets

2. Pace of field
   a. State-of-the-art models evolve every few months

3. Lack of evaluation metric
   a. Accuracy
   b. Time to train, latency of inference

4. Multi-disciplinary field
   a. Algorithms, Systems, Hardware
Outline

- Model diversity
- Agile benchmark development
- Evaluation metrics
- Open and closed divisions
- Contributing to MLPerf
Fathom suite showed breadth in ML benchmarking

- Collection of 8 diverse learning models
- Clear, tested implementations in TensorFlow
- Training and inference modes provided
- Provided broad view and coverage
- Models have drastically changed and greatly advanced since 2015
## Benchmarks Considered for MLPerf

<table>
<thead>
<tr>
<th>Area</th>
<th>Vision</th>
<th>Language</th>
<th>Audio</th>
<th>Commerce</th>
<th>Action / RL</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem</strong></td>
<td>Image Classification</td>
<td>Translation</td>
<td>Speech Recognition</td>
<td>Rating Recommendations</td>
<td>Games</td>
<td>GANs 3D point clouds</td>
</tr>
<tr>
<td></td>
<td>Object Detection / Segmentation</td>
<td>Language Model</td>
<td>Text-to-Speech</td>
<td>Sentiment Analysis</td>
<td>Go</td>
<td>Word embeddings</td>
</tr>
<tr>
<td></td>
<td>Face ID</td>
<td>Word Embedding</td>
<td>Question Answering</td>
<td>Keyword Spotting</td>
<td>Robotics</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HealthCare (Radiology)</td>
<td></td>
<td>Language Modeling</td>
<td>Language Modeling</td>
<td>Healthcare (EHR)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Video Detection</td>
<td></td>
<td>Chatbots</td>
<td>Speaker ID</td>
<td>Fraud detection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Self-Driving</td>
<td></td>
<td>Graph embeddings</td>
<td>Graph embeddings</td>
<td>Anomaly detection</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Content ID</td>
<td></td>
<td>Time series prediction</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Large scale regression</td>
<td></td>
</tr>
<tr>
<td><strong>Datasets</strong></td>
<td>ImageNet</td>
<td>WMT</td>
<td>LibriSpeech</td>
<td>MovieLens-20M</td>
<td>Atari</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COCO</td>
<td>English-German</td>
<td>SQuAD</td>
<td>Amazon</td>
<td>Go</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LM-Benchmark</td>
<td>IMDB</td>
<td>Chess</td>
<td></td>
</tr>
<tr>
<td><strong>Models</strong></td>
<td>ResNet-50</td>
<td>Transformer</td>
<td>Deep Speech 2</td>
<td>Neural Collaborative Filtering</td>
<td>DQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TF Object Detection</td>
<td>OpenNMT</td>
<td>SQuAD Explorer</td>
<td>CNNs</td>
<td>PPO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Detectron</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Accuracy</strong></td>
<td>COCO mAP</td>
<td>BLEU</td>
<td>WER</td>
<td>Prediction accuracy</td>
<td>Prediction accuracy</td>
<td></td>
</tr>
<tr>
<td><strong>Metrics</strong></td>
<td>Prediction accuracy</td>
<td></td>
<td>Perplexity</td>
<td></td>
<td>Win/Loss</td>
<td></td>
</tr>
</tbody>
</table>

MLPerf
Outline

- Model diversity
- **Agile benchmark development**
- Evaluation metrics
- Open and closed divisions
- Contributing to MLPerf
Agile Benchmark Development

- Rapidly iterate the benchmark suite:
  - Remain relevant in the very fast moving ML field

AlexNet (2012)

VGG16 (2014)

Agile Benchmark Development

- Rapidly iterate the benchmark suite:
  - Remain relevant in the very fast moving ML field
  - Scale problems to match faster hardware

A **300,000x Increase in Compute** since 2012

From OpenAI Blog “AI and Compute”
Agile Benchmark Development

- Rapidly iterate the benchmark suite:
  - Remain relevant in the very fast moving ML field
  - Scale problems to match faster hardware
  - Correct inevitable mistakes in the formulation
- At least initially, revise annually? MLPerf18, MLPerf19, ...
- Like SPEC, have quarterly deadlines and then publish results for that quarter via searchable database
Outline

- Model diversity
- Agile benchmark development
- **Evaluation metrics**
- Open and closed divisions
- Contributing to MLPerf
Metrics Should Capture Performance and Quality

- **Performance**: how fast is a model for training, inference?
  - Focus of benchmarks like DeepBench, Fathom
- **Quality**: how good are a model’s predictions?
  - Focus of benchmarks like ImageNet, MS COCO
Performance and Quality *aren’t always correlated*

End-to-end training of a ResNet56 CIFAR10 model on a Nvidia P100 machine with 512 GB of memory and 28 CPU cores, using TensorFlow 1.2 compiled from source with CUDA 8.0 and CuDNN 5.1.
Metrics Should Capture Performance and Quality

- **Performance**: how fast is a model for training, inference?
- **Quality**: how good are a model's predictions?

Important for benchmark to capture **both** performance and quality
MLPerf metric: **Training time** to reach quality target + cost or power

- Quality target is *specific for each benchmark* and *close to state-of-the-art*
  - Updated with each release to keep up with the state-of-the-art
  - Median of 5 runs
- Time includes preprocessing and validation
- Reference implementations that achieve quality target

**In addition, either:**
- *Cost* of public cloud resources (no spot/preemptible instances)
- *Power utilization* for on-premise hardware
Summary result combines benchmark metrics

**Why?**

- Provide a concise indicator of “general purpose ML” performance
- Encourage the field to move in a common direction, ultimately leading to greater performance across the board

**How?** For participants that submit to each benchmark category:

- For each benchmark task, normalize the time result to the reference implementation on baseline hardware
- Summary score computed via geometric mean of results
Outline

- Model diversity
- Agile benchmark development
- Evaluation metrics
- Open and closed divisions
- Contributing to MLPerf
Goal: **Encourage Innovation** and fair comparison

- ML algorithms are under active development
- Many models with different trade-offs
Open/Closed Divisions + Replication

- **Closed** division requires using the specified model
  - Limits overfitting
  - Enables apples-to-apples comparison
  - Simplifies work for HW groups

- **Open** division allows using any model
  - Encourages innovation
  - Ensures closed division does not stagnate
Outline

● Model diversity
● Agile benchmark development
● Evaluation metrics
● Open and closed divisions
● **Contributing to MLPerf**
Ways to support and be involved

- **Github**: reference code
- **Submissions**: data points
- **Google group**: discussion of the benchmark and changes
- **Meetings**: community building and focused discussion towards action
- **Working groups**: targeted groups to flesh out specific areas
  - Inference
  - Reinforcement learning
  - Summary score
  - Measuring power and cost
Questions?