Speech Technology: Frontiers and Applications

On-device Speech Algorithms

Xiangang Li, Guoguo Chen

Outline

- From server to edge
- On-device speech recognition
- On-device wake word detection
- Homework

Outline

- From server to edge
- On-device speech recognition
- On-device wake word detection
- Homework

From server to edge: old days

- DragonDictate
  - DOS

- Dragon NaturallySpeaking
  - Windows and Mac

- Google Voice Search / Apple Siri
  - Server side ASR

From server to edge: concerns

- Latency
- Privacy
- Reliability
- Scalability

From server to edge: difficulties

- Computing resource
  - Limited MIPS
  - Limited RAM
  - Limited flash

- Data pipeline
  - No data feedback loop. Federated learning?

Outline

- From server to edge
- On-device speech recognition
- On-device wake word detection
- Homework

On-device ASR: traditional ASR

- PocketSphinx
  - LVCSR optimized for mobile devices
- From server side ASR to on-device ASR
  - Computation/resource optimization
- Traditional ASR components
  - Language model
  - Acoustic model

On-device ASR: traditional ASR

- Language model optimization
  - Vocabulary
  - N-gram models
  - Small v.s. large models
    - Small model on device
    - Large model on server
    - LM rescoring

On-device ASR: traditional ASR

- Acoustic model optimization (neural network models)
  - Model compression
    - SVD decomposition
  - Model quantization
    - Quantization aware training

On-device ASR: traditional ASR

- SVD decomposition
  - Parameter matrices are sparse
  - Reconstructs matrices to reduce parameters

$$A_{m\times n} = U_{m\times n}\Sigma_{n\times n}V_{n\times n}^{T}$$

$$A_{m\times n} = U_{m\times k}\Sigma_{k\times k}V_{k\times k}^{T} = U_{m\times k}N_{k\times k}$$

On-device ASR: traditional ASR

Acoustic Model		WER	Number of parameters
Baseline, GMM model		29.1%	11M
Original DNN model		25.6%	29M
SVD (1024)		25.6%	25M
SVD (512)		25.7%	21M
SVD (256)	Before fine-tune	28.6%	19M
	After fine-tune	25.6%	
All hidden layers (512)	Before fine-tune	26.0%	21M
	After fine-tune	25.6%	
All hidden layers (256)	Before fine-tune	27.0%	17M
	After fine-tune	25.8%	
All hidden and output layers (256)	Before fine-tune	29.7%	7M
	After fine-tune	25.4%	
All hidden and output layer (192)	Before fine-tune	36.7%	5.6M
	After fine-tune	25.5%	

WER (%) Results of baseline, original DNN and SVD restructuring on output/hidden layers.

Photo credit: [Restructuring of Deep Neural Network Acoustic Models with Singular Value Decomposition]

On-device ASR: traditional ASR

- Quantization aware training
  - 8 bit quantization
  - Includes quantization noise in training

Algorithm 1 Quantization aware SGD training. Where L is the number of layers. C is the cost function, infer-and-recover$(\cdot)$ is a function that performs the inference computation in integer form but returns the results in recovered floating point. error$(\cdot)$, wgradient$(\cdot)$, bgradient$(\cdot)$, adjust$(\cdot)$ are functions that perform the typical backpropagation operations in floating point.

Require: a mini-batch of (inputs, outputs), parameters w_{t-1}, and b_{t-1} (weights and biases) in floating point precision, from previous training step $t-1$.

procedure TRAININGSTEP
    $w^q_{t-1} \leftarrow$ quantize(w_{t-1})
    for k=1 to L **do**
        $a_k \leftarrow$ infer-and-recover$(a_{k-1}, w^q_{t-1}, b_{t-1})$
    end for
    Compute output error δ_L
    for k=L-1 to 2 **do**
        $\delta_k \leftarrow$ error$(w_{k+1,t-1}, \delta_{k+1}, a_{k+1})$
        $\frac{\partial C}{\partial w_{k,t-1}} \leftarrow$ wgradient$(a_{k-1}, delta_k)$
        $\frac{\partial C}{\partial b_{k,t-1}} \leftarrow$ bgradient$(delta_k)$
        $w_{k,t} \leftarrow$ adjust$(w_{t-1}, \frac{\partial C}{\partial w_{k,t-1}})$
        $b_{k,t} \leftarrow$ adjust$(b_{t-1}, \frac{\partial C}{\partial b_{k,t-1}})$
    end for
end procedure

Pseudocode of quantization aware SGD training.

Photo credit: [On the efficient representation and execution of deep acoustic models]

On-device ASR: traditional ASR

深蓝学院 shenlanxueyuan.com

System (Params.)	WER (%) on Clean Eval Set				WER (%) on Noisy Eval Set			
	match	mismatch	quant	quant-all	match	mismatch	quant	quant-all
4×300 (~2.9M)	13.6	14.3 (5.1%)	13.5 (-0.7%)	13.6 (0.0%)	26.3	28.2 (7.2%)	26.5 (0.8%)	26.5 (0.8%)
5×300 (~3.7M)	12.5	13.1 (4.8%)	12.6 (0.8%)	12.7 (1.6%)	24.6	26.6 (8.1%)	24.8 (0.8%)	25.0 (1.6%)
4×400 (~5.0M)	12.1	12.5 (3.3%)	12.3 (1.7%)	12.3 (1.7%)	23.2	25.0 (7.8%)	23.7 (2.2%)	23.8 (2.6%)
5×400 (~6.3M)	11.4	11.7 (2.6%)	11.5 (0.9%)	11.7 (2.6%)	22.3	23.5 (5.4%)	22.6 (1.3%)	22.7 (1.8%)
4×500 (~7.7M)	11.7	12.0 (2.6%)	11.7 (0.0%)	11.7 (0.0%)	22.6	23.6 (4.4%)	22.6 (0.0%)	22.7 (0.4%)
5×500 (~9.7M)	10.9	11.1 (1.8%)	11.2 (2.8%)	11.1 (1.8%)	20.9	21.7 (3.8%)	21.4 (2.4%)	21.5 (2.9%)
$P = 100$ (~2.7M)	11.6	12.1 (4.3%)	11.8 (1.7%)	11.9 (2.6%)	22.6	23.8 (5.3%)	23.1 (2.2%)	23.3 (3.1%)
$P = 200$ (~4.8M)	10.6	10.8 (1.9%)	10.6 (0.0%)	10.7 (0.9%)	20.5	21.4 (4.4%)	20.6 (0.5%)	20.7 (1.0%)
$P = 300$ (~6.8M)	10.3	10.5 (1.9%)	10.5 (1.9%)	10.6 (2.9%)	19.8	20.3 (2.5%)	20 (1.0%)	20.4 (3.0%)
$P = 400$ (~8.9M)	10.3	10.5 (1.9%)	10.3 (0.0%)	10.5 (1.9%)	19.6	20.2 (3.1%)	19.8 (1.0%)	19.9 (1.5%)
Avg. Relative Loss	-	3.0%	0.9%	1.6%	-	5.2%	1.2%	1.9%

Word error rates on ‘clean’ and ‘noisy’ evaluation sets for various model architectures. Numbers in parentheses represent the loss relative to the ‘matched’ condition where models are trained and evaluated using floating point arithmetic.

match: float-point training, float-point inference
mismatch: float-point training, fixed-point inference
quant: quantization aware training without softmax layer
quant-all: quantization aware training on all layers

Photo credit:
[On the efficient representation and execution of deep acoustic models]

On-device ASR: E2E ASR

- Traditional ASR for on-device
  - Optimizes/Shrinks each component
  - Performs worse than server side ASR

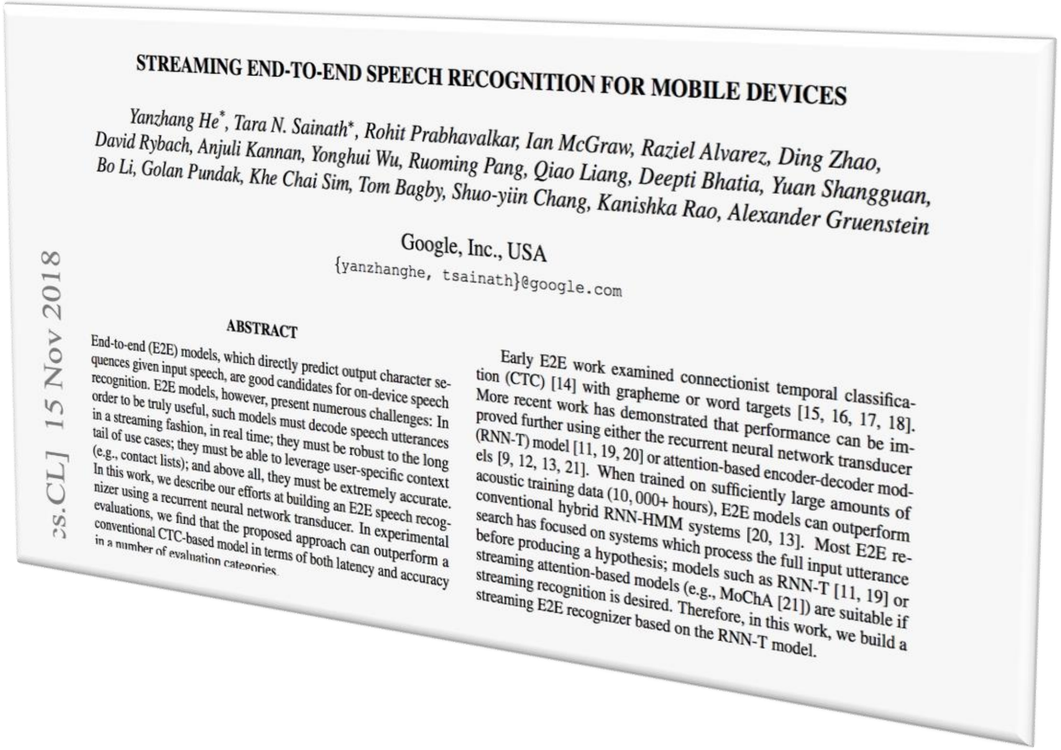
- What about E2E ASR?
  - One component to optimize
  - One component to implement
  - Very appealing candidate for on-device ASR

On-device ASR: E2E ASR

- E2E ASR challenges
  - Real time factor
  - Streaming decoder
  - Long tail use cases
  - User specific context (LM)

On-device ASR: E2E ASR

- RNN-T for on-device ASR
  - 2019 Google IO release
  - Pure neural solution
  - 80M model size
  - Faster then real time on a single core
  - Comparable results to server side ASR



STREAMING END-TO-END SPEECH RECOGNITION FOR MOBILE DEVICES

Yanzhang He*, Tara N. Sainath*, Rohit Prabhavalkar, Ian McGraw, Raziel Alvarez, Ding Zhao, David Rybach, Anjuli Kannan, Yonghui Wu, Ruoming Pang, Qiao Liang, Deepti Bhatia, Yuan Shangguan, Bo Li, Golan Pundak, Khe Chai Sim, Tom Bagby, Shuo-yiin Chang, Kanishka Rao, Alexander Gruenstein

Google, Inc., USA
{yanzhanghe, tsainath}@google.com

cs.CL] 15 Nov 2018

ABSTRACT

End-to-end (E2E) models, which directly predict output character sequences given input speech, are good candidates for on-device speech recognition. E2E models, however, present numerous challenges: In order to be truly useful, such models must decode speech utterances in a streaming fashion, in real time; they must be robust to the long tail of use cases; they must be able to leverage user-specific context (e.g., contact lists); and above all, they must be extremely accurate. In this work, we describe our efforts at building an E2E speech recognizer using a recurrent neural network transducer. In experimental evaluations, we find that the proposed approach can outperform a conventional CTC-based model in terms of both latency and accuracy in a number of evaluation categories.

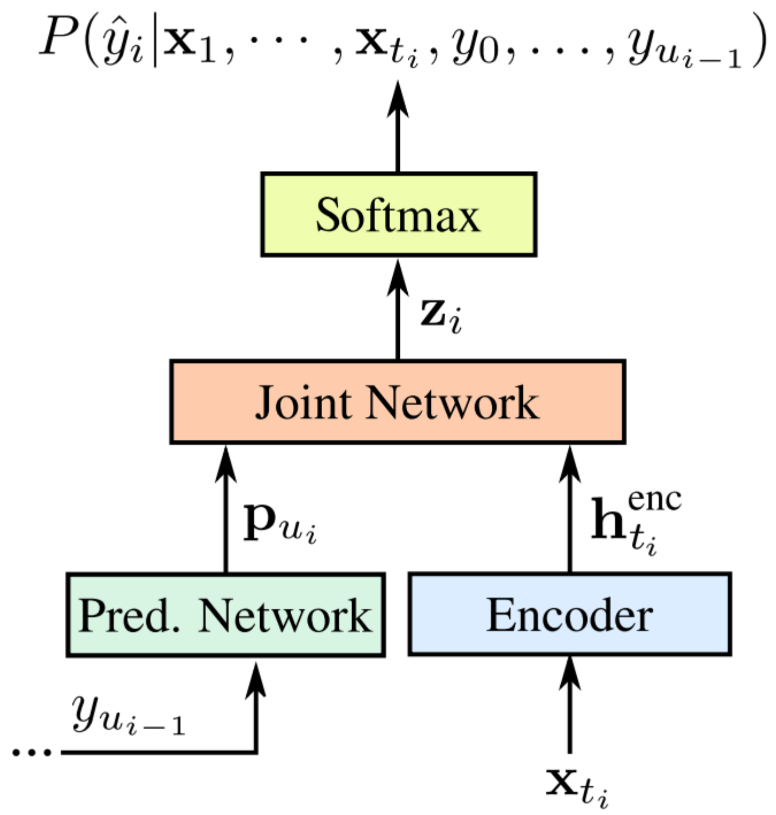
Early E2E work examined connectionist temporal classification (CTC) [14] with grapheme or word targets [15, 16, 17, 18]. More recent work has demonstrated that performance can be improved further using either the recurrent neural network transducer (RNN-T) model [11, 19, 20] or attention-based encoder-decoder models [9, 12, 13, 21]. When trained on sufficiently large amounts of acoustic training data (10,000+ hours), E2E models can outperform conventional hybrid RNN-HMM systems [20, 13]. Most E2E research has focused on systems which process the full input utterance before producing a hypothesis; models such as RNN-T [11, 19] or streaming attention-based models (e.g., MoChA [21]) are suitable if streaming recognition is desired. Therefore, in this work, we build a streaming E2E recognizer based on the RNN-T model.

Paper to read:

[Streaming End-to-end Speech Recognition For Mobile Devices]

On-device ASR: E2E ASR

- Detail matters
    - Project layer after each LSTM layer
    - Time-reduction layer to reduce frame rate
    - State caching in prediction network (like RNNLM)
    - Quantization
    - Contextual biasing
    - Text normalization



RNN-Transducer.

Photo credit:
[Streaming End-to-end Speech Recognition For Mobile Devices]

On-device ASR: E2E ASR

ID	Model	*VS*	*IME*	*RT90*
E2	RNN-T Grapheme (Float)	7.5%	4.4%	1.58
E7	RNN-T Word-piece (Float)	7.0%	4.1%	1.43
E8	+ Asymmetric Quantization	7.1%	4.2%	1.03
E9	+ Symmetric Quantization	**7.3%**	**4.2%**	**0.51**
B2	CTC + Symmetric Quantization	9.2%	5.4%	0.86

RNN-T performance and real-time factor at 90 percentile.

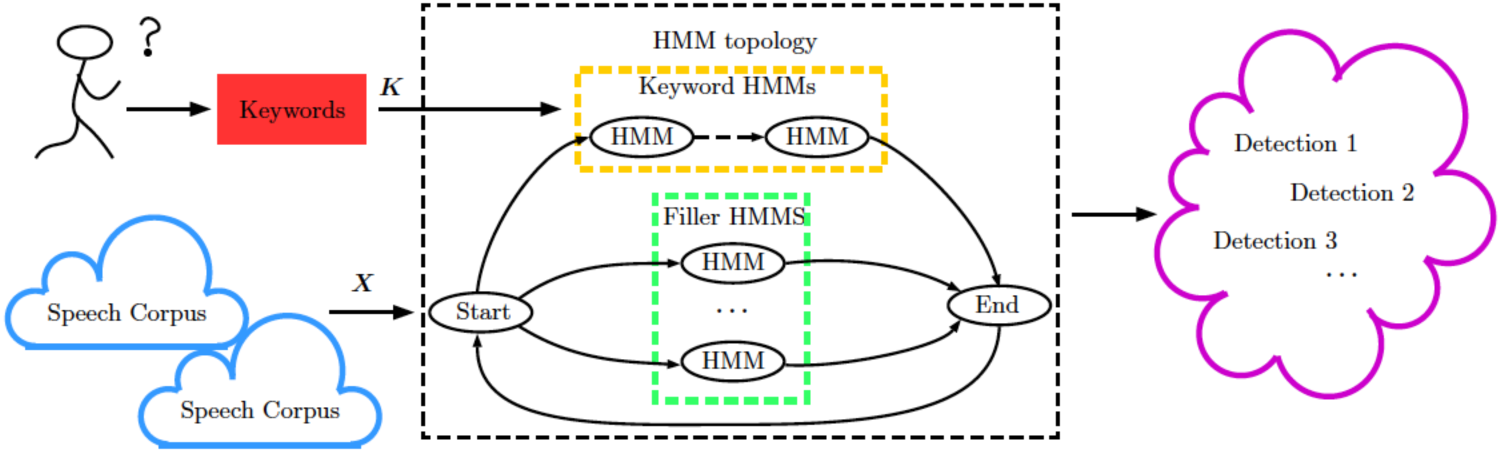
Photo credit:
Streaming End-to-end Speech Recognition For Mobile Devices

Outline

- From server to edge
- On-device speech recognition
- On-device wake word detection
- Homework

On-device WW: applications

On-device WW: keyword/filler model



- HMM densities can be computed use a neural network trained on general speech recognition data or keyword specific data
- Wake word detection is done by running Viterbi decoding with the given HMM topology
- Works reasonably well, but model size tends to be large, as it usually models a few hundred/thousand sub-word unites

On-device WW: improvements?

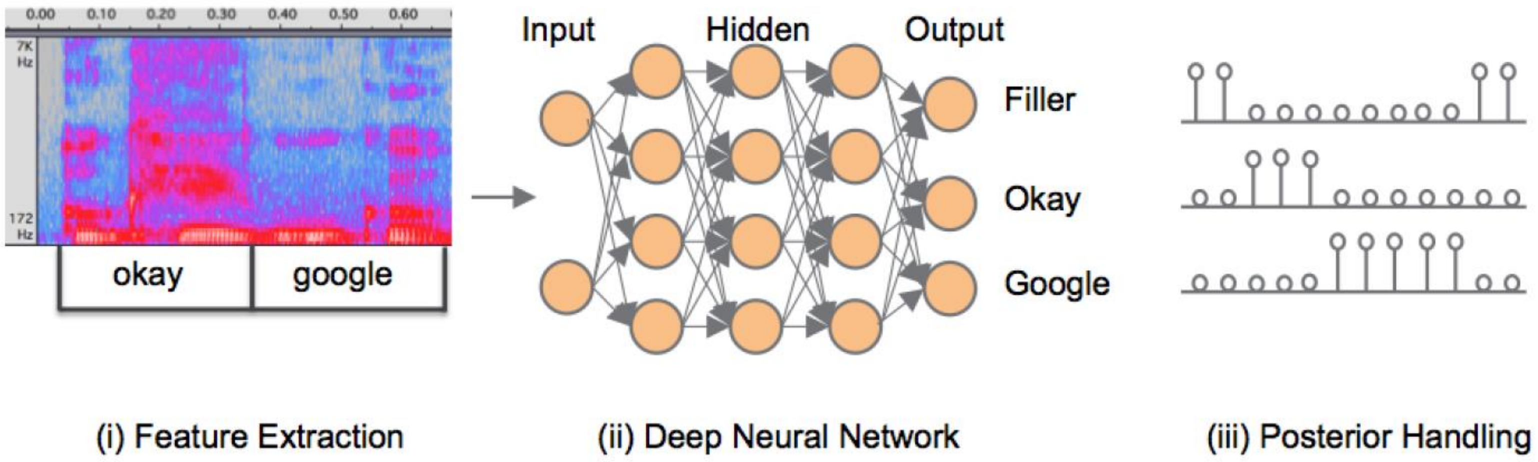
- In wake word applications, typically only one word/phrase needs to be detected
  - Collecting large amount of keyword specific data is possible

- Whole word modeling is effective when training data is adequate (Lee, 1989)
  - Reduces the number of modeling units
  - HMMs can be dropped

- Whole word modeling with DNN

On-device WW: Deep KWS

深蓝学院
shenlanxueyuan.com



Deep KWS framework.

Photo credit: [Small-footprint Keyword Spotting Using Deep Neural Networks]

- 40-dimensional log-filterbank features, stacks 10 future frames and 30 frames in the past
- Deep neural network module takes the stacked features, and generate frame-level posteriors for each label
- Posterior handling module takes posteriors from the DNN, and makes a final detection decision

On-device WW: Deep KWS

- Posterior smoothing

$$p'_{ij} = \frac{1}{j - h_{smooth} + 1} \sum_{k=h_{smooth}}^{j} p_{ik}$$

- Confidence

$$confidence = \sqrt[n-1]{\prod_{i=1}^{n-1} \max_{h_{max} \le k \le j} p'_{ik}}$$

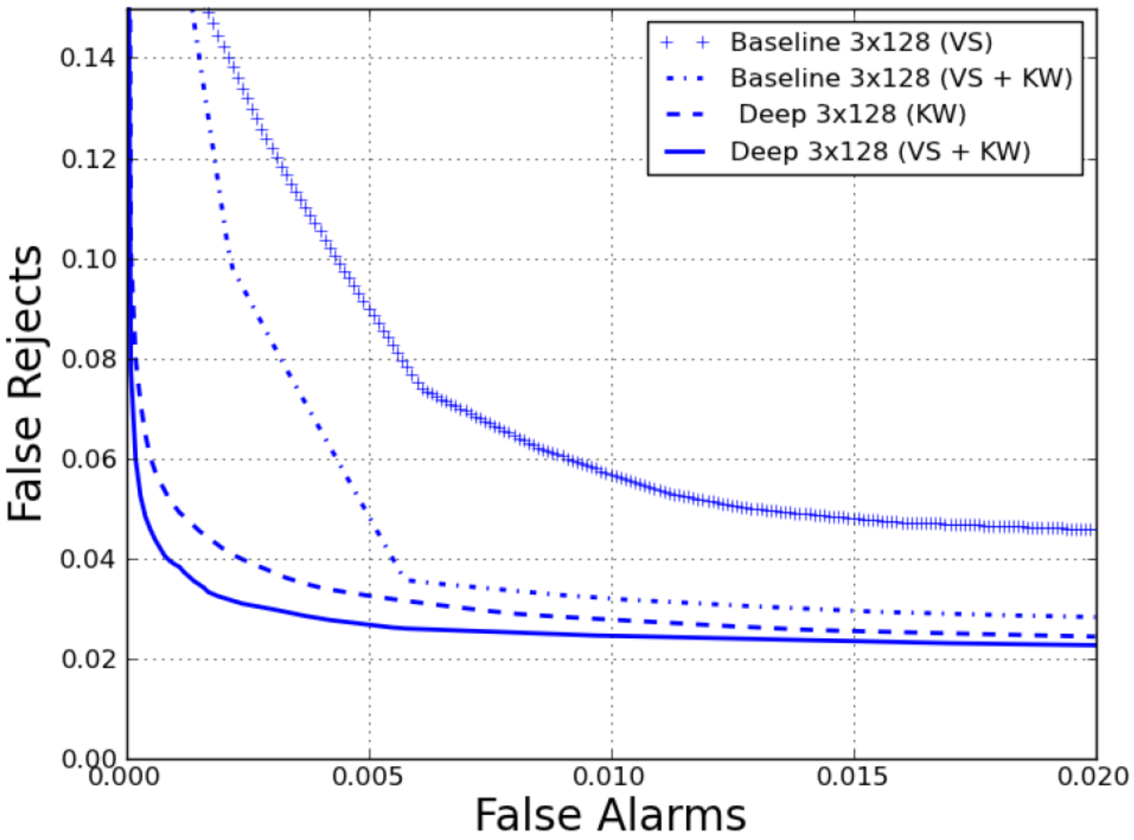
On-device WW: Deep KWS

answer call	dismiss alarm
go back	ok google
read aloud	record a video
reject call	show more commands
snooze alarm	take a picture

Keywords used in evaluation

- VS data: 3000 hours of manually transcribed voice search data
- KW data: 2.3K positive examples, 133K negative examples for each keyword; for “Okay Google” the number of positive examples is 40K
- Testing: 1K positive examples, 70K negative examples for each keyword; for “Okay Google” the number of positive examples is 2.2K

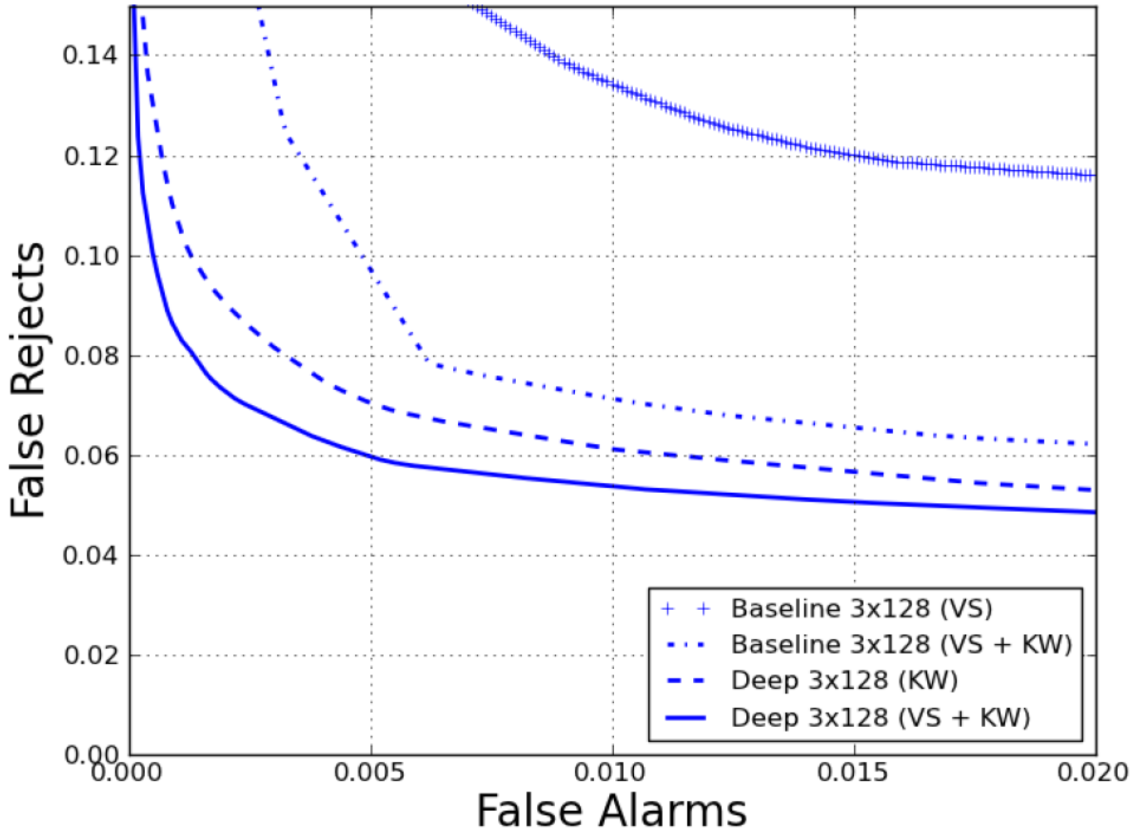
On-device WW: Deep KWS



HMM vs. Deep KWS system with 3 hidden layers, 128 hidden nodes neural network on clean data.

Photo credit:
[Small-footprint Keyword Spotting Using Deep Neural Networks]

On-device WW: Deep KWS



HMM vs. Deep KWS system with 3 hidden layers, 128 hidden nodes neural network on noisy data.

Photo credit:
[Small-footprint Keyword Spotting Using Deep Neural Networks]

On-device WW: issues?

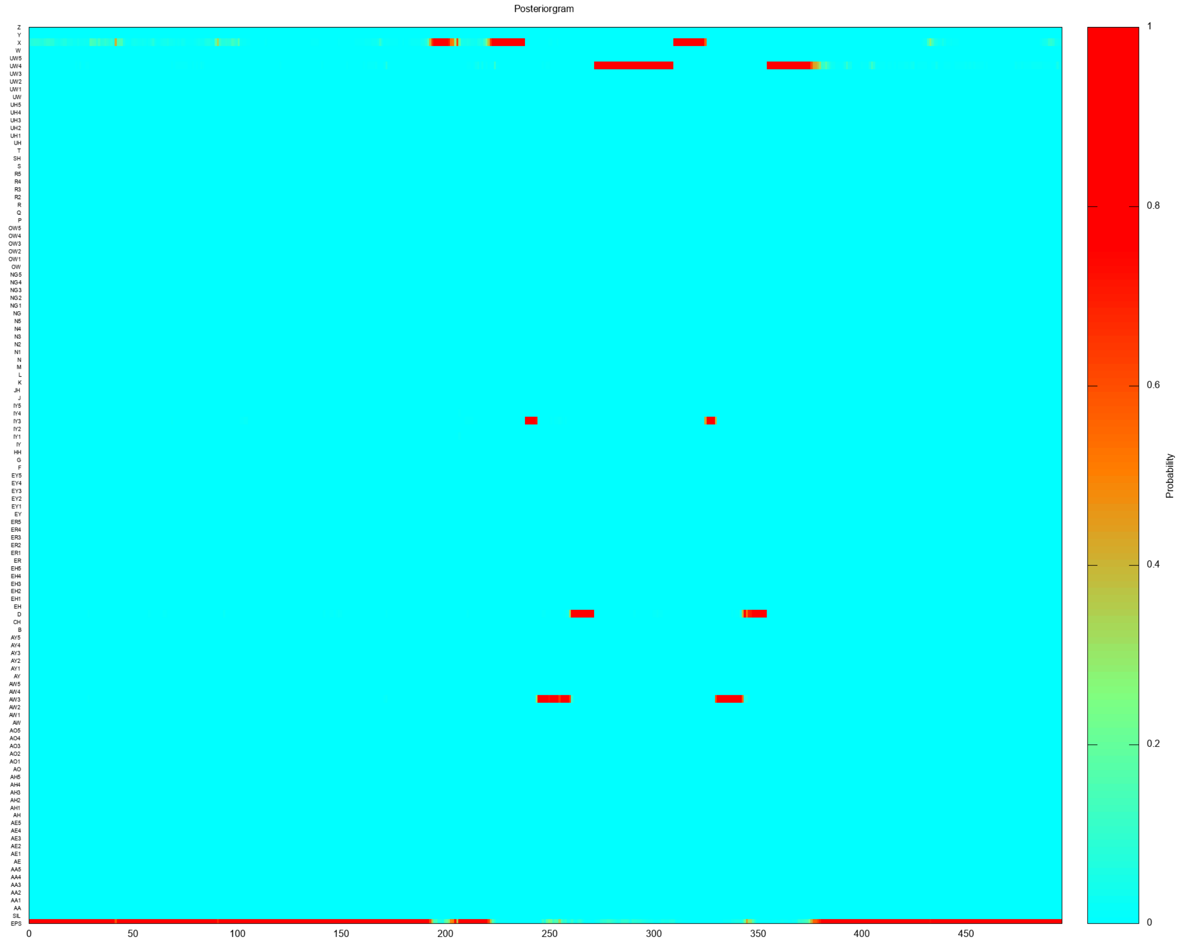
- Short wake words, e.g., Alexa
- Multiple wake words
- Wake word selection
- Data requirement

On-device WW: improvement 1

- Modeling unit
  - Monophone
  - Works for short wake word
  - Takes advantage of non-keyword data

- Decoding
  - Sliding window
  - Viterbi search on the window posteriorgram

On-device WW: improvement 1

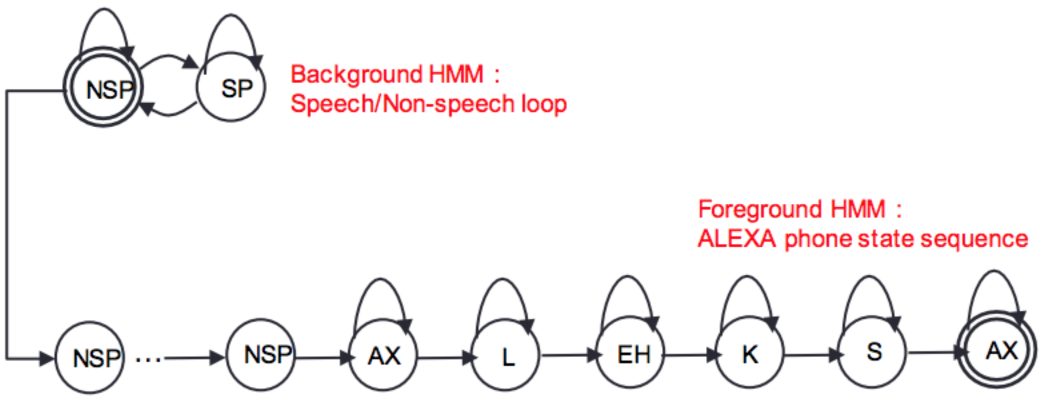


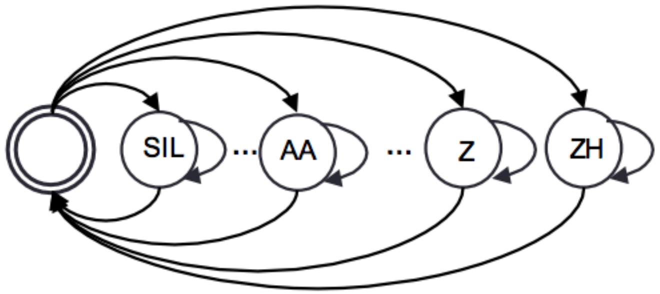
On-device WW: improvement 2

- Modeling unit
  - Monophone
  - Works for short wake word
  - Takes advantage of non-keyword data

- Decoding
  - Keyword/filler model
  - Adds HMM back
  - Uses monophones to model filler

On-device WW: improvement 2

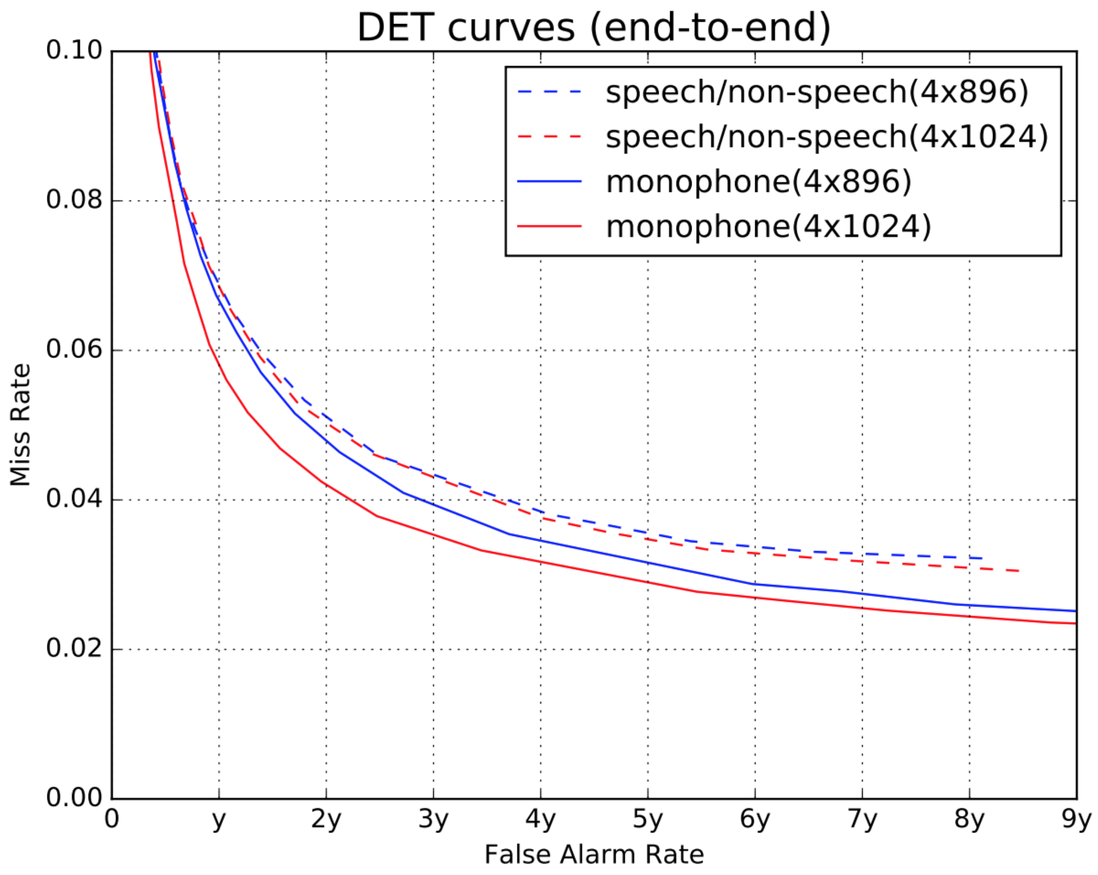




One the left, simplified HMM decoding graph for wake word “Alexa”, with a speech/non-speech loop as the filler model. On the right, a simplified monophone-based filler model.

Photo credit:
[Monophone-based Background Modeling for Two-stage On-device Wake Word Detection]

On-device WW: improvement 2



Wake word detection performance of monophone-based background model.

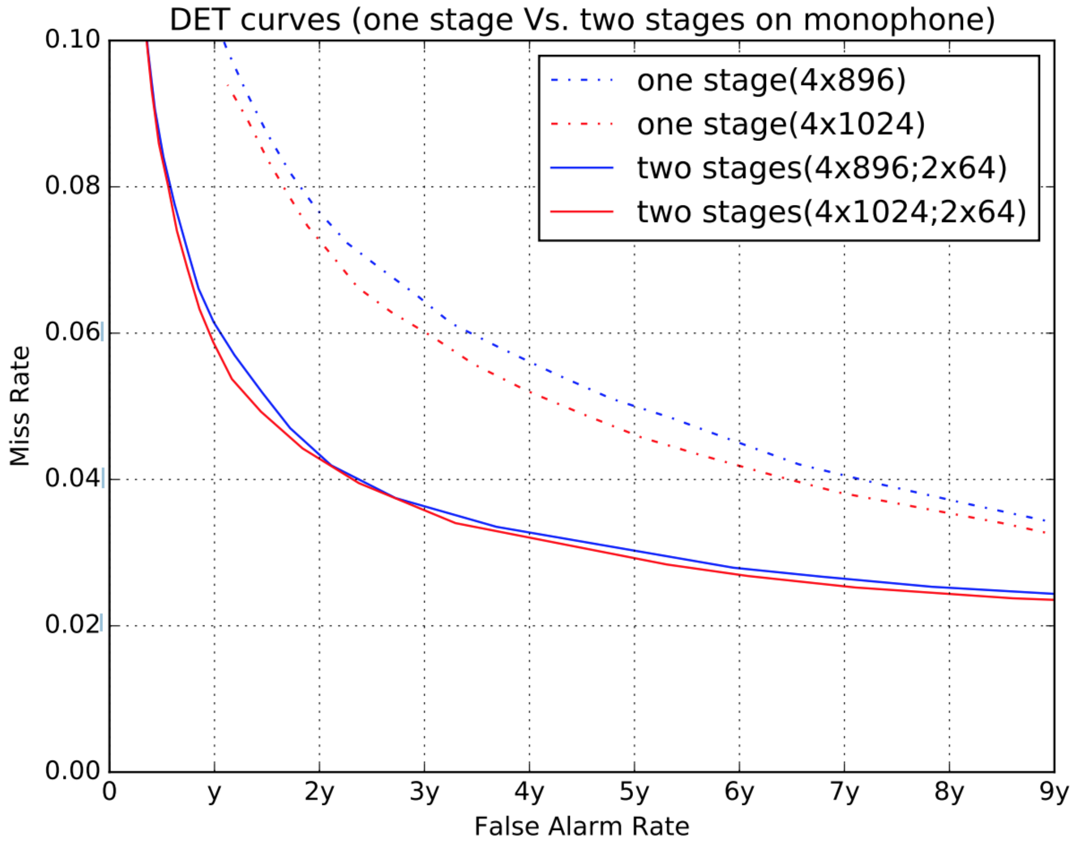
Photo credit:
[Monophone-based Background Modeling for Two-stage On-device Wake Word Detection]

On-device WW: improvement 3

- Two-stage detection
  - False alarm supression

- Options
  - Second stage on server
    - Transmit wake word audio to server for verification
    - ASR (tuned) as second stage
  - Second stage on device
    - Simple classifier
    - Classification features: confidence, alignment, etc

On-device WW: improvement 2



Wake word detection performance of two-stage wake word detection based on classifier.

Photo credit:
[Monophone-based Background Modeling for Two-stage On-device Wake Word Detection]

Outline

- From server to edge
- On-device speech recognition
- On-device wake word detection
- Homework

Homework

深蓝学院
shenlanxueyuan.com

- Task
  - Train a command detection system which detects all commands in the dataset
  - Feel free to use your favorite platform/toolkit
  - Feel free to pick one or more methods, or come up with your own
  - Simple report on the method(s) you use and the performance

- Data
  - Google's Speech Commands dataset: https://storage.cloud.google.com/download.tensorflow.org/data/speech_commands_v0.02.tar.gz

Thanks!