Deep Speech

Scaling up end-to-end speech recognition

prepared by Sabina Kasaeva

AGENDA

01

METHOD MOTIVATION

- Previous techniques
- Aims of the DeepSpeech model

02

MODEL ARCHITECTURE

- RNN
- CTC Loss
- Regularization
- N-gram LM as a decoder

03

GPU OPTIMIZATIONS

- Data parallelism
- Model parallelism
- Striding

04

TRAINING DATA

- noisy data synthesis by superposition
- capturing lombard effect

05

EXPERIMENTS

- conversational speech
- noisy speech

MOTIVATION BEHIND THE METHOD

SLIDE TITLE HERE

PREVIOUS MODELS

- Top speech recognition systems rely on sophisticated pipelines composed of multiple algorithms and hand-engineered processing stages

AIMS

- lessen the number of extra steps and instruments needed to organise speech recognition
- perform great in noisy environments

SPECIFICS

- one must find innovative ways to build large, labeled training sets and must be able to train networks that are large enough to effectively utilize all of this data

MODEL ARCHITECTURE

ARCHITECTURE

INPUT DATA

x_i, t, p - x_i is a time-series of length T_i, with each time slice as vector of powers of p'th frequency

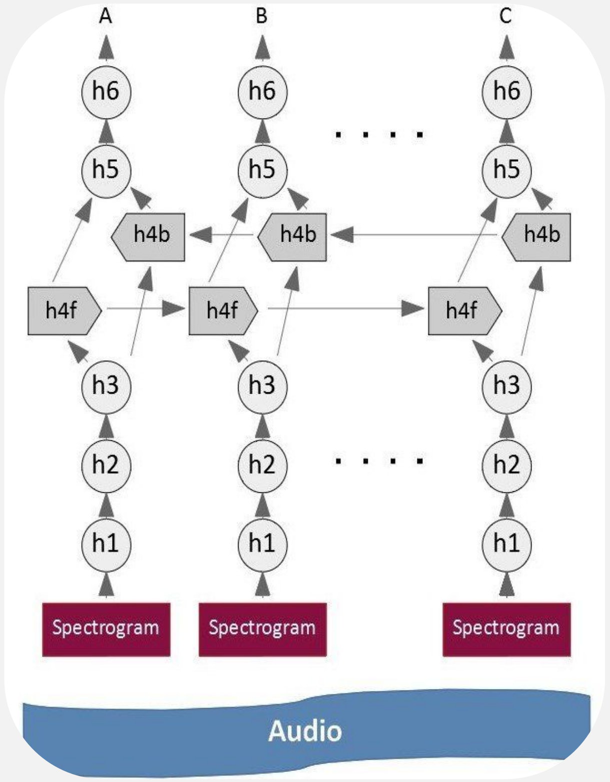
3 NON-RECURRENT LAYERS

$$h_t^{(l)} = g(W^{(l)} h_t^{(l-1)} + b^{(l)})$$

h is set with x at first, l denotes the layer, g(z) = min{max(0, z}, 20} is clipped ReLU

4 - BIDIRECTIONAL RECURRENT LAYER

$$h_t^{(f)} = g(W^{(4)} h_t^{(3)} + W_r^{(f)} h_{t-1}^{(f)} + b^{(4)})$$
$$h_t^{(b)} = g(W^{(4)} h_t^{(3)} + W_r^{(b)} h_{t+1}^{(b)} + b^{(4)})$$



ARCHITECTURE

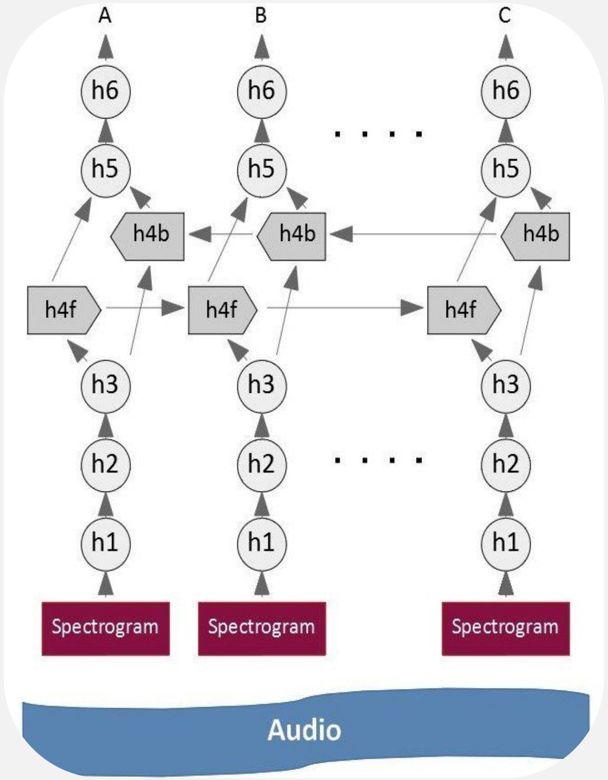
5 - NON-RECURRENT LAYER

$$h_t^{(5)} = g(W^{(5)} h_t^{(4)} + b^{(5)})$$

where $h_t^{(4)} = h_t^{(f)} + h_t^{(b)}$.

6 - SOFTMAX OUTPUT

$$h_{t,k}^{(6)} = \hat{y}_{t,k} \equiv \mathbb{P}(c_t = k|x) = \frac{\exp(W_k^{(6)} h_t^{(5)} + b_k^{(6)})}{\sum_j \exp(W_j^{(6)} h_t^{(5)} + b_j^{(6)})}.$$



CONNECTIONIST TEMPORAL CLASSIFICATION LOSS. PROBLEM STATEMENT

We have a dataset of audio clips and corresponding transcripts. We don't know how the characters in the transcript align to the audio.

We try to map X – audio sequences to Y – transcriptions. There are challenges which get in the way of us using simpler supervised learning algorithms. In particular:

- Both X and Y can vary in length.
- We don't have an accurate alignment (correspondence of the elements) of X and Y.

For a given X model gives us an output distribution over all possible Y's.

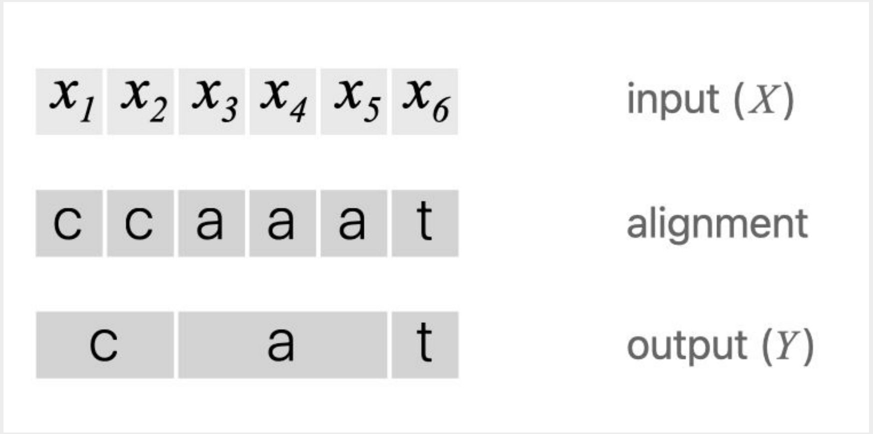
Loss Function: For a given input, we'd like to train our model to maximize the probability it assigns to the right answer.

$$Y^* = \underset{Y}{\operatorname{argmax}} \; p(Y \mid X).$$

CTC LOSS. INTRODUCING “BLANK” SYMBOL

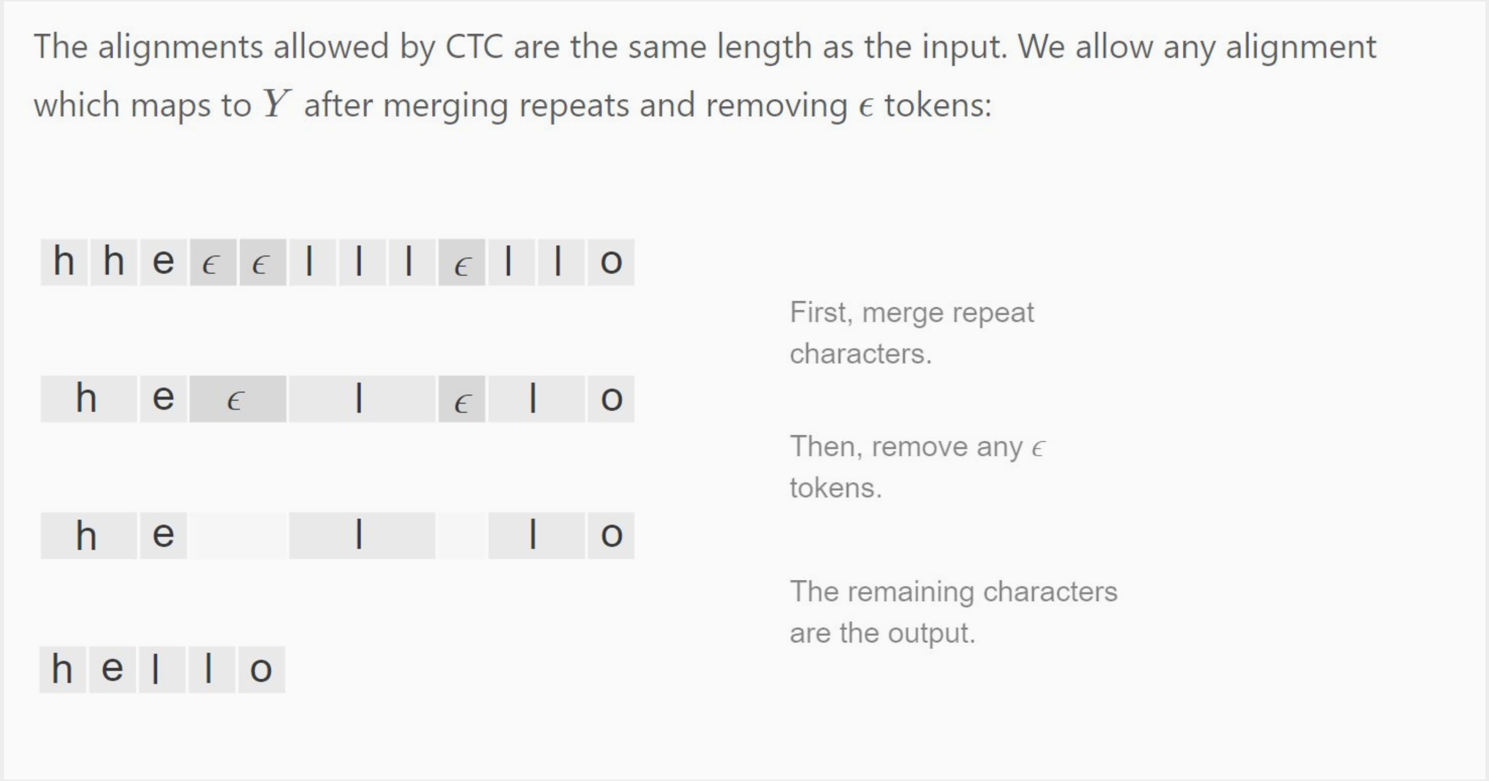
Naive approach has two problems.

- It doesn’t make sense to force every input step to align to some output, cause the input can have stretches of silence with no corresponding output.
- There is no way to produce outputs with multiple characters in a row



CTC introduces a new token called the *blank* token ϵ. The ϵ token doesn’t correspond to anything and is simply removed from the output.

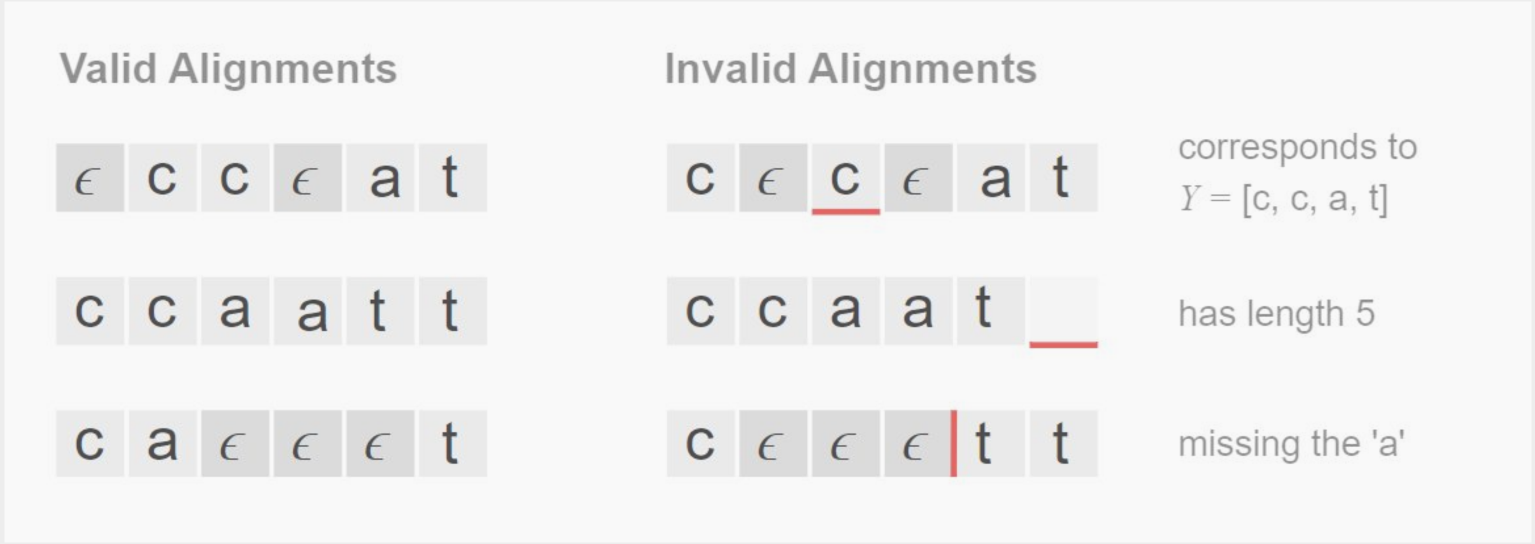
CTC LOSS. INTRODUCING "BLANK" SYMBOL



The alignments allowed by CTC are the same length as the input. We allow any alignment which maps to Y after merging repeats and removing ϵ tokens:

h h e ϵ ϵ l l l ϵ l l o
First, merge repeat characters.
h e ϵ l ϵ l o
Then, remove any ϵ tokens.
h e l l o
The remaining characters are the output.
h e l l o

CTC LOSS. INTRODUCING “BLANK” SYMBOL

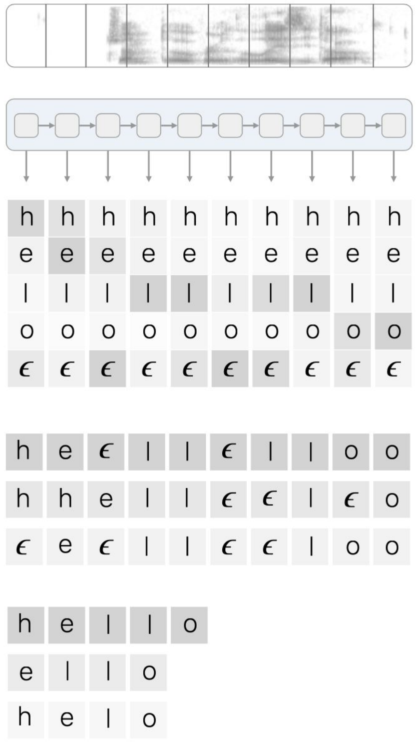


CTC LOSS. BASIC STATEMENTS

The CTC alignments have a few notable properties:

- the allowed alignments between *X* and *Y* are monotonic.
- the alignment of *X* to *Y* is many-to-one. One or more input elements can align to a single output element but not vice-versa.
- the length of *Y* cannot be greater than the length of X.

CTC LOSS. COUNTING THE LOSS



We start with an input sequence, like a spectrogram of audio.

The input is fed into an RNN, for example.

The network gives $p_t(a \mid X)$, a distribution over the outputs $\{h, e, l, o, \epsilon\}$ for each input step.

With the per time-step output distribution, we compute the probability of different sequences

By marginalizing over alignments, we get a distribution over outputs.

$$p(Y \mid X) \quad = \quad \sum_{A \in \mathcal{A}_{X,Y}} \quad \prod_{t=1}^{T} p_t(a_t \mid X)$$

The CTC conditional **probability**

marginalizes over the set of valid alignments

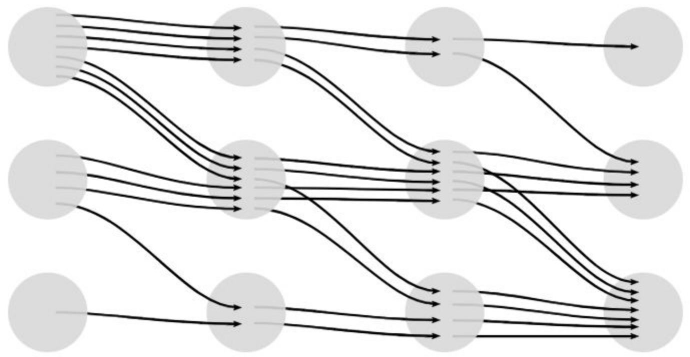
computing the **probability** for a single alignment step-by-step.

It can be very expensive to to count the CTC conditional probability using a straightforward approach, cause there can be a massive number of alignments.

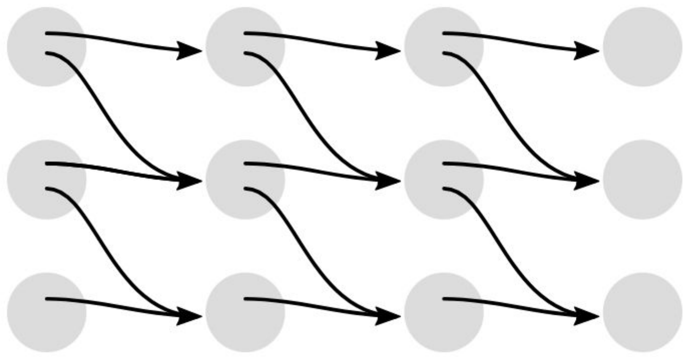
To do that efficiently we can use dynamic programming.

CTC LOSS. COUNTING THE LOSS

The key insight is that if two alignments have reached the same output at the same step, then we can merge them.



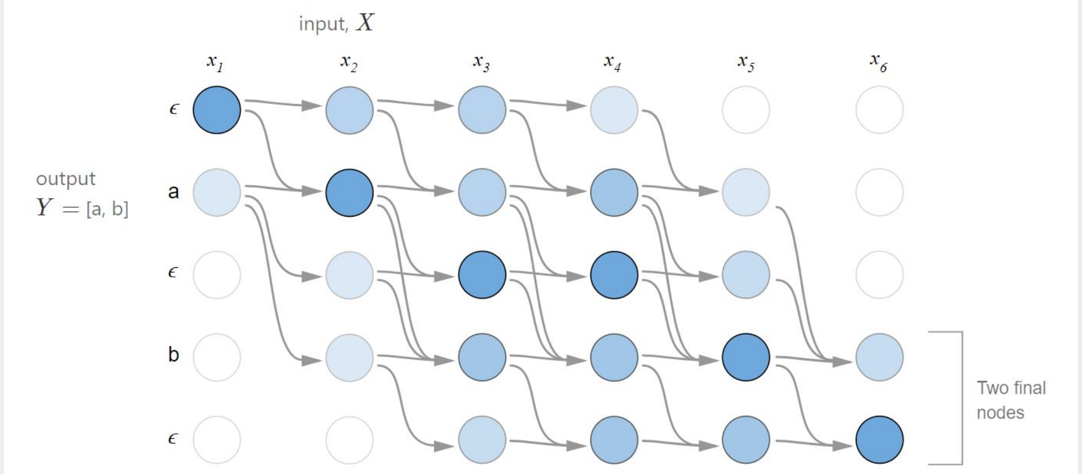
Summing over all alignments can be very expensive.



Dynamic programming merges alignments, so it's much faster.

CTC LOSS. COUNTING THE LOSS

There are two valid starting nodes and two valid final nodes since the ϵ at the beginning and end of the sequence is optional. The complete probability is the sum of the two final nodes.



Node (s, t) in the diagram represents $\alpha_{s,t}$ – the CTC score of the subsequence $Z_{1:s}$ after t input steps.

Let's let *α* be the score of the merged alignments at a given node. More precisely, *α_s,t* is the CTC score of the subsequence *Z[1:s]* after *t* input steps.

We'll compute the final CTC score, *P*(*Y*|*X*), from the *α*'s at the last time-step.

.

CTC LOSS. LOSS OPTIMIZATION

The CTC loss function is differentiable as sums and products of the per time-step output probabilities.

We can analytically compute the gradient of the loss function with respect to the (unnormalized) output probabilities and from there run backpropagation as usual.

For a training set D, the model's parameters are tuned to minimize the negative log-likelihood.

$$\sum_{(X,Y)\in\mathcal{D}} -\log\ p(Y \mid X)$$

CTC LOSS. INFERENCE

TAKE MOST LIKELY OUTPUT AT EACH TIME-STEP

$$A^* = \underset{A}{\operatorname{argmax}} \prod_{t=1}^{T} p_t(a_t \mid X)$$

VS

BEAM SEARCH TO TAKE INTO ACCOUNT "SIMILAR" OUTPUTS

[a, a, a] and [a, a, ϵ]

REGULARIZATION

DROPOUT

The dropout with rate 5%-10% is applied to feed-forward layers, not the recurrent.

JITTERING

Audios are shifted by 5 ms to the left and right, the forward propagation is performed and output probabilities are averaged.

RNN'S ENSEMBLE

At test time RNN'S ensemble is used with averaging the output probabilities.

N-GRAM LM AS A DECODER

N-GRAM LM

An n-gram LM is a statistical model used to estimate the likelihood of a sequence of words.

N-GRAM

An n-gram is a contiguous sequence of n items from a given sample of words.

PROBABILITY ESTIMATION

The probability of a word is estimated based on the conditional probability of that word given the preceding (n-1) words.

N-GRAM LM AS A DECODER

The errors made by the RNN in this case tend to be phonetically plausible renderings of English words. Now we optimize Q with beam search.

$$Q(c) = \log(\mathbb{P}(c|x)) + \alpha \log(\mathbb{P}_{\text{lm}}(c)) + \beta \ \textbf{word\_count}(c)$$

RNN output	Decoded Transcription
what is the weather like in bostin right now	what is the weather like in boston right now
prime miniter nerenr modi	prime minister narendra modi
arther n tickets for the game	are there any tickets for the game

Table 1: Examples of transcriptions directly from the RNN (left) with errors that are fixed by addition of a language model (right).

GPU OPTIMIZATIONS

GPU OPTIMIZATIONS

FEED-FORWARD

DATA PARALLELISM

Feed-forward layers are computed by batches rather than samples. Batch size is maximum size possible with each GPU

BIDIRECTIONAL RECURRENT

MODEL PARALLELISM

Recurrent layers are parallelised by the time axis. Each GPU computing [1;T/2] or [T/2;T] time slices (forward and backward hidden vectors respectively). Then exchanging vectors T/2 and swapping functions (now backward and forward on the same data chunk)

FEATURE EXTRACTION

STRIDING

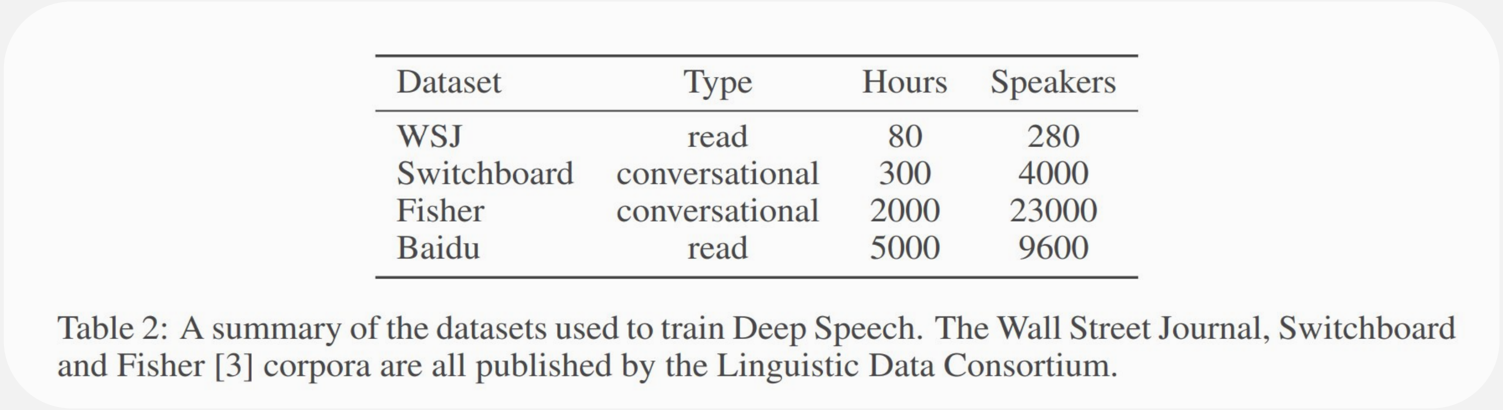
Setting stride (window size) between frames in a spectrogram

TRAINING DATA

TRAINING DATA

Noisy data is synthesized with superposition principle, thus using linear combination of a clear sound and some noises. The average powers for each frequency must remain statistically close to real-life noisy sound.

To capture Lombard Effect (when the speaker adapts its voice to the noise) and still have clean audios, speakers were offered the earphones with background noise.

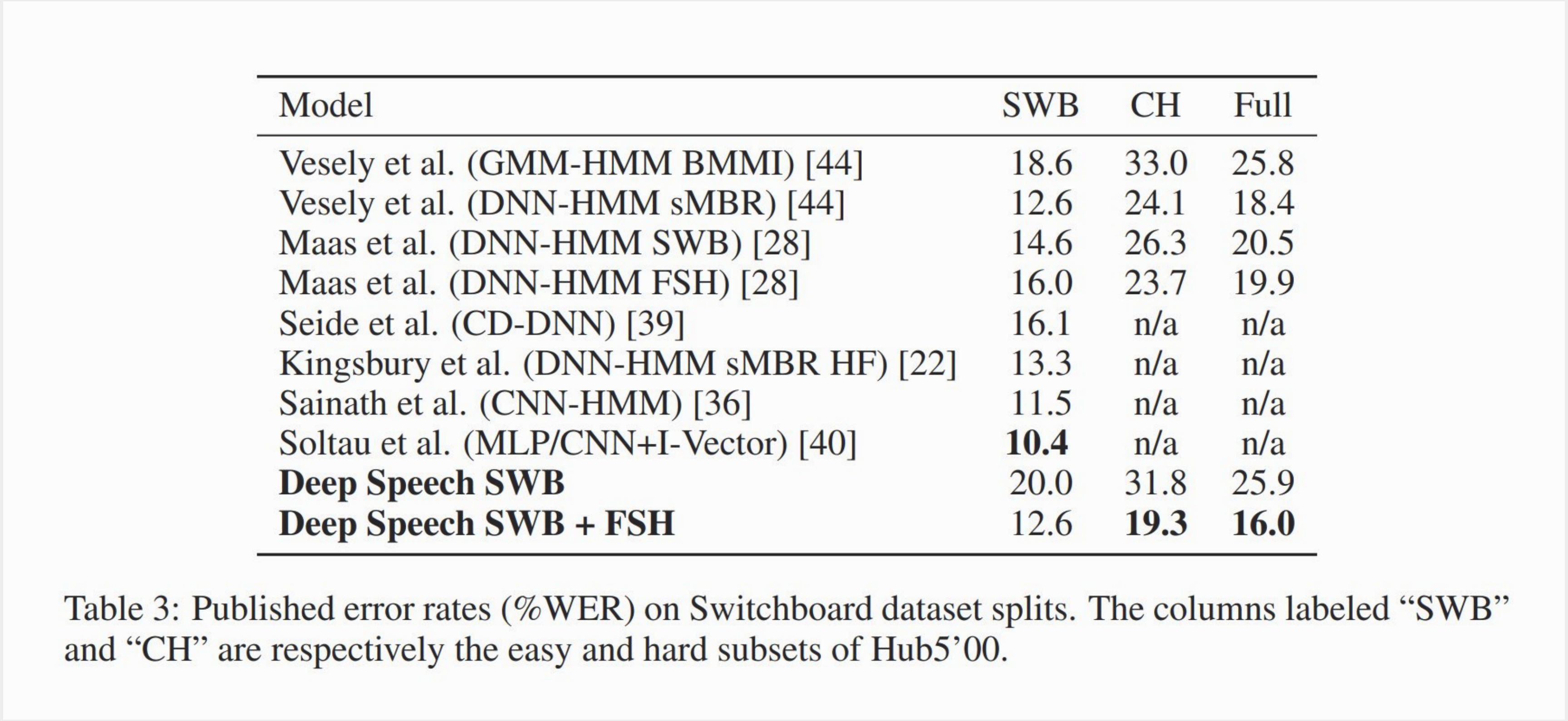


Dataset	Type	Hours	Speakers
WSJ	read	80	280
Switchboard	conversational	300	4000
Fisher	conversational	2000	23000
Baidu	read	5000	9600

Table 2: A summary of the datasets used to train Deep Speech. The Wall Street Journal, Switchboard and Fisher [3] corpora are all published by the Linguistic Data Consortium.

EXPERIMENTS

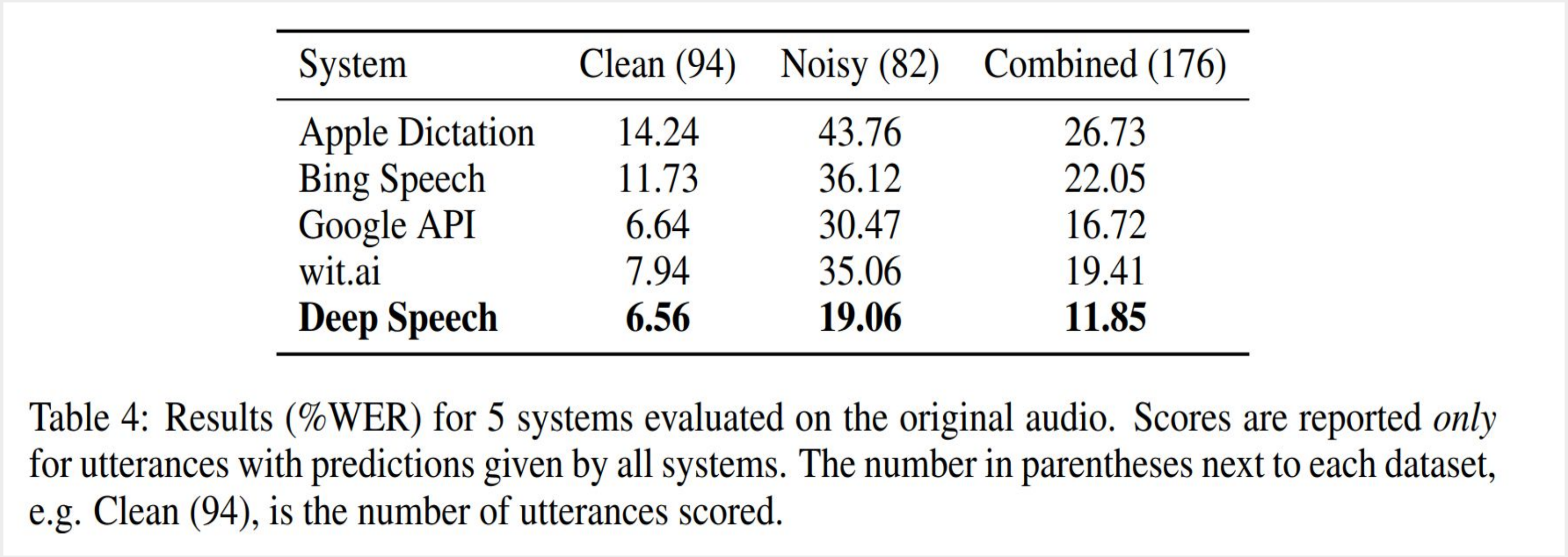
Conversational Speech: Switchboard Hub5'00 (Full)



Model	SWB	CH	Full
Vesely et al. (GMM-HMM BMMI) [44]	18.6	33.0	25.8
Vesely et al. (DNN-HMM sMBR) [44]	12.6	24.1	18.4
Maas et al. (DNN-HMM SWB) [28]	14.6	26.3	20.5
Maas et al. (DNN-HMM FSH) [28]	16.0	23.7	19.9
Seide et al. (CD-DNN) [39]	16.1	n/a	n/a
Kingsbury et al. (DNN-HMM sMBR HF) [22]	13.3	n/a	n/a
Sainath et al. (CNN-HMM) [36]	11.5	n/a	n/a
Soltau et al. (MLP/CNN+I-Vector) [40]	**10.4**	n/a	n/a
Deep Speech SWB	20.0	31.8	25.9
Deep Speech SWB + FSH	12.6	**19.3**	**16.0**

Table 3: Published error rates (%WER) on Switchboard dataset splits. The columns labeled "SWB" and "CH" are respectively the easy and hard subsets of Hub5'00.

NOISY SPEECH



System	Clean (94)	Noisy (82)	Combined (176)
Apple Dictation	14.24	43.76	26.73
Bing Speech	11.73	36.12	22.05
Google API	6.64	30.47	16.72
wit.ai	7.94	35.06	19.41
Deep Speech	**6.56**	**19.06**	**11.85**

Table 4: Results (%WER) for 5 systems evaluated on the original audio. Scores are reported *only* for utterances with predictions given by all systems. The number in parentheses next to each dataset, e.g. Clean (94), is the number of utterances scored.

LITERATURE OVERVIEW

Awni Hannun∗ , Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, Andrew Y. Ng. Deep Speech: Scaling up end-to-end speech recognition. 2014

A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: ´ Labelling unsegmented sequence data with recurrent neural networks. In ICML, pages 369–376. ACM, 2006.

https://distill.pub/2017/ctc/