Introduction to Speech Recognition

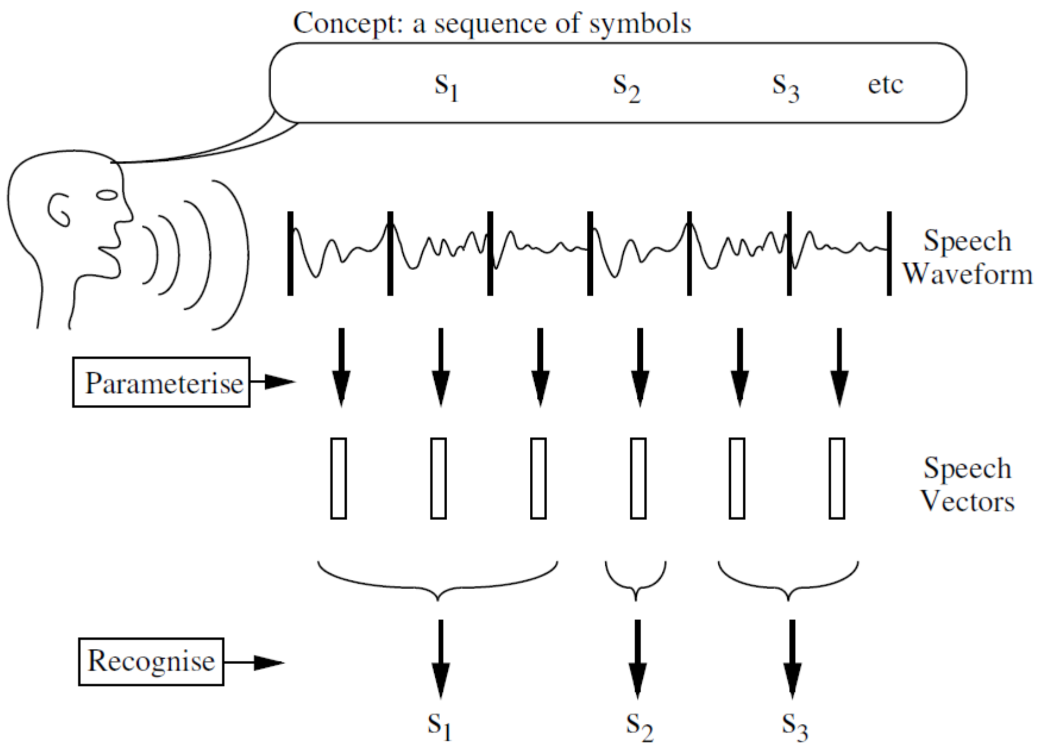
by LuoPingfeng

2020.03.01

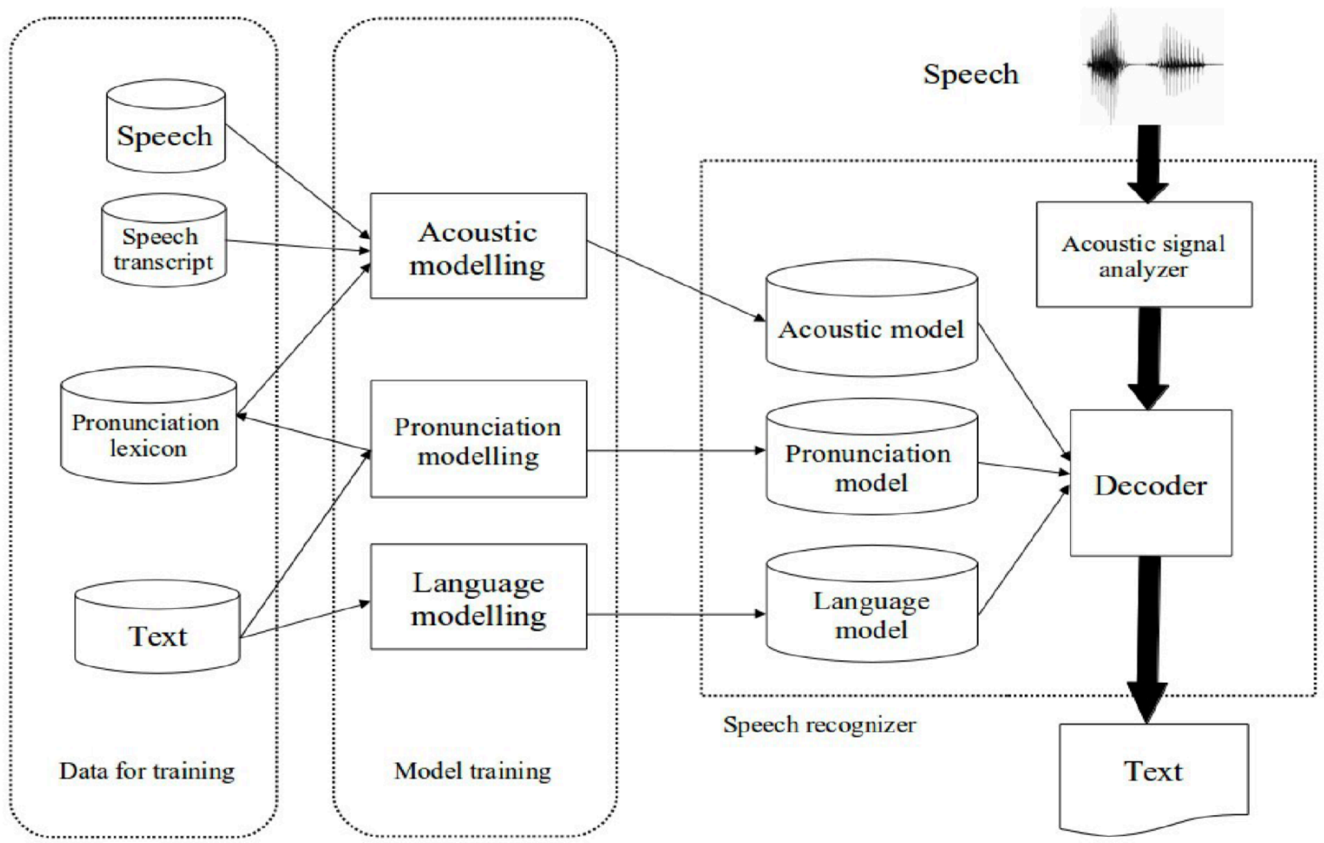
Outline

What?

The task of speech recognition is to convert speech into a sequence of words by a computer program



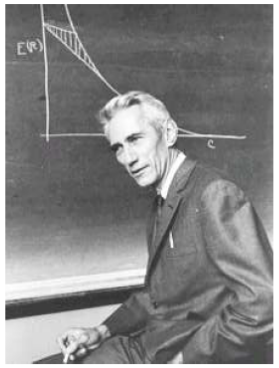
How?


Speech
Speech transcript
Pronunciation lexicon
Text
Data for training
Acoustic modelling
Pronunciation modelling
Language modelling
Model training
Speech
Acoustic signal analyzer
Acoustic model
Pronunciation model
Language model
Decoder
Speech recognizer
Text

Rough history?

Thomas Bayes (1701-1761)

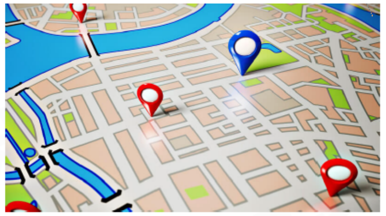
AA Markov (1856–1922)


Claude Shannon (1916–2001)

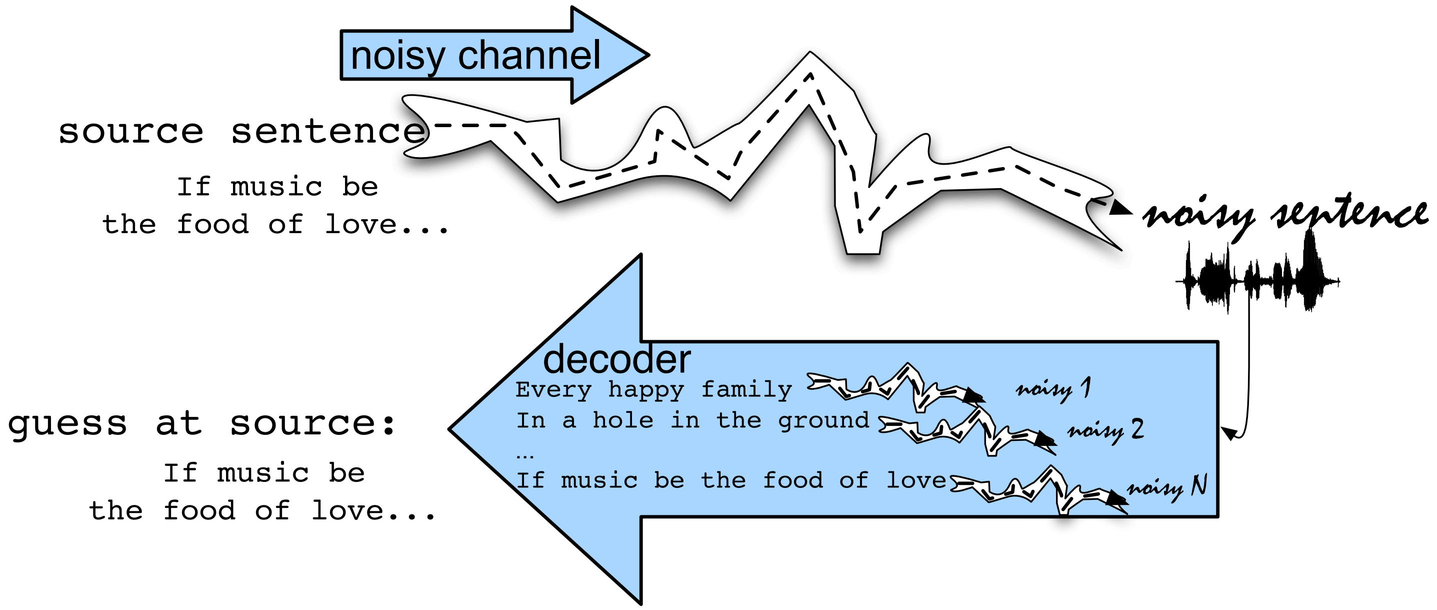
- HMM (Baum, 1972; Baker, 1975;Jelinek, 1976) is one of the most common types of acoustic models
- Segmental models including neural networks (Lippman,1987; Morgan et al., 2005), maximum entropy models (Gao and Kuo, 2006), and (hidden) conditional random fields (Gunawardana et al., 2006)
- Deep learning break through (DNN-HMM, CTC, Encoder-Decoder, Attention…, 2009-now)

Application?

- dictation
- voice input
- voice dialing
- voice user interface
- intelligent assistant
- computer-aided language learning



Noisy Channel Model



Noisy Channel Model

How to find the most likely sentence W out of all sentences L given acoustic input O?

- Treat acoustic input O as sequence of individual observations
  - $O = o_1,o_2,o_3,\ldots,o_t$
- Define a sentence as a sequence of words
  - $W = w_1,w_2,w_3,\ldots,w_n$

Noisy Channel Model

- Probabilistic implication:

$$\hat{W} = \underset{W \in L}{\arg\max}\, P(W \mid O)$$

- We can use Bayes rule:

$$\hat{W} = \underset{W \in L}{\arg\max} \frac{P(O \mid W)P(W)}{P(O)}$$

- Since denominator is the same for each candidate sentence W, we can ignore it for the argmax:

$$\hat{W} = \underset{W \in L}{\arg\max}\, \underbrace{P(O \mid W)}_{\text{Likelihood}}\, \underbrace{P(W)}_{\text{Prior}}$$

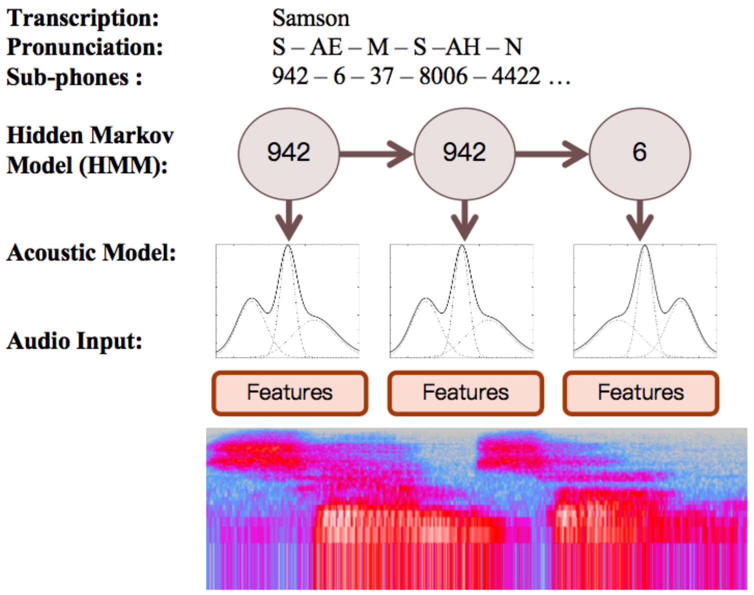
Acoustic Model

The likelihood P(O|W) is computed by the acoustic model, this model includes the representation of knowledge about acoustics, phonetics, microphone and environment variability, etc.

For the HMM-based acoustic model rewrite

$$P(O|W) = \sum_{S \in S_1^T} P(O, S|W)$$

S is a hidden state sequence in HMM.



Transcription: Samson
Pronunciation: S – AE – M – S –AH – N
Sub-phones : 942 – 6 – 37 – 8006 – 4422 …
Hidden Markov Model (HMM):
942
942
6
Acoustic Model:
Audio Input:
Features
Features
Features

EM Iteratively to solve HMM-based acoustic model

- E : Generate a forced alignment with existing model
  - Viterbi decoding with a very constrained prior (the transcript)
  - Assigns observations to HMM states
- M : Create new observation models from update alignments

Language Model

The prior P(W) is computed by the language model, this model includes knowledge of what constitutes a possible word, what words are likely to co-occur, and in what sequence.

$$\begin{aligned} P(W) &= P(w_1)P(w_2|w_1)\dots P(w_M|w_1^{M-1}) \\ &= \prod_{m=1}^{M} P(w_m|w_1^{m-1}). \end{aligned}$$

In the n-gram model, the conditional probability is approximated by truncating the history into the previous n − 1 words.

$$P(W) = \prod_{m=1}^{M} P(w_m|w_{m-n+1}^{m-1})$$

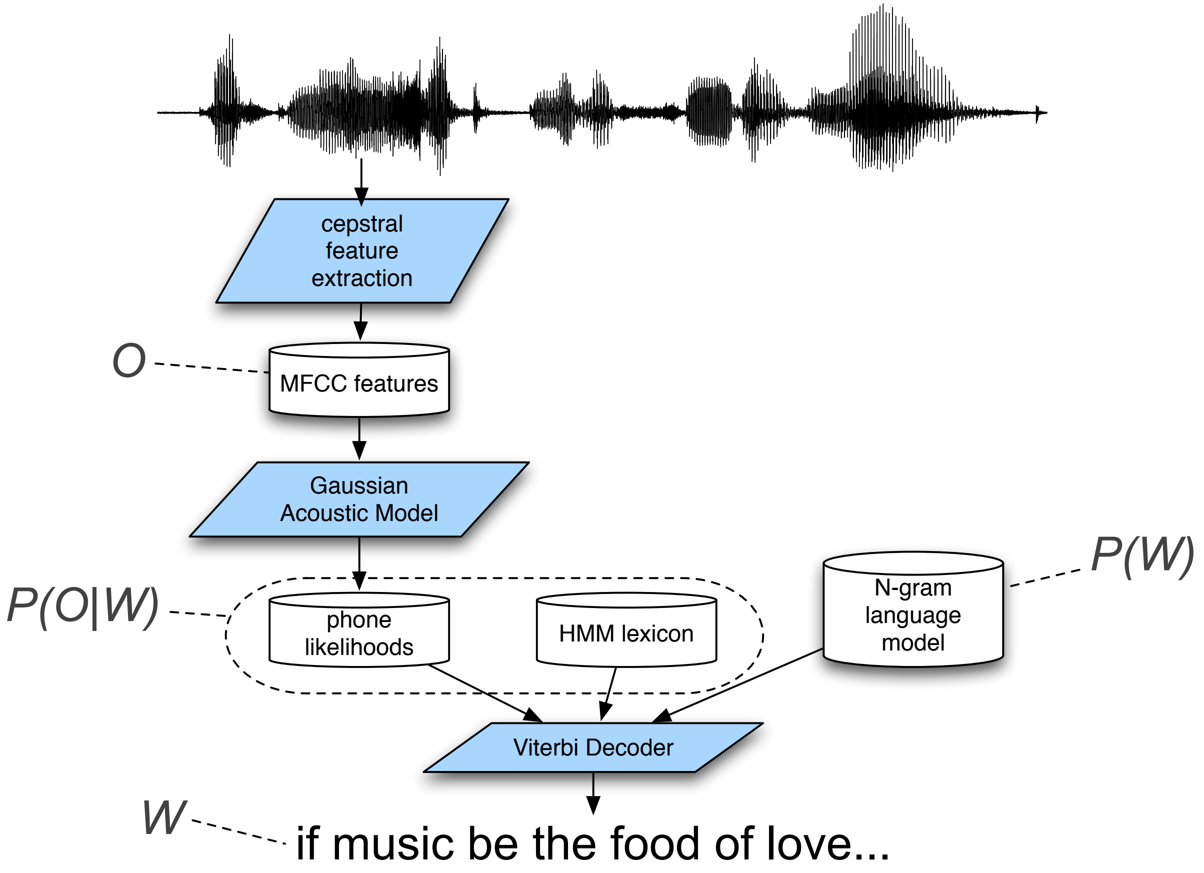
The maximum likelihood estimate of the n-gram probability can be obtained using a text corpus as

$$P(w_n|w_1^{n-1}) = \frac{C(w_1^n)}{C(w_1^{n-1})}$$

Speech Recognition Architecture

- Feature Extraction:
  - “MFCC/PLP” spectra features
- Acoustic Model:
  - HMM for computing P(O|W)
- Lexicon/Pronunciation Model
  - what phones this W pronounces
- Language Model
  - N-grams for computing P(W)
- Decoder
  - Viterbi or A*stack algorithm to find the most likely word sequence W

Speech Recognition Architecture



Decoder

The task of decoder is given a observation O = ($o_1 o_2 \ldots o_T$) finding the most likely words W = ($w_1, w_2 \ldots w_n$) by acoustic and language models

$$\hat{W} = \underset{W \in L}{\arg\max}\, \underbrace{P(O \mid W)}_{\text{Acoustic}} \underbrace{P(W)}_{\text{Language}}$$

Decoder

- Represent components in speech recognition as WFSTs:
  - H: HMM structure (Acoustic model)
  - C: Phonetic context dependency (Context dependence phone)
  - L: Lexicon (Pronunciation dictionary)
  - G: Grammar (Language model)
- The decoding network is composition of these models : $H \circ C \circ L \circ G$
  - Successively determinize and combine the component transducers, then minimize the final network

Decoder - WFST Algorithm

Composition: combine transducers at different levels

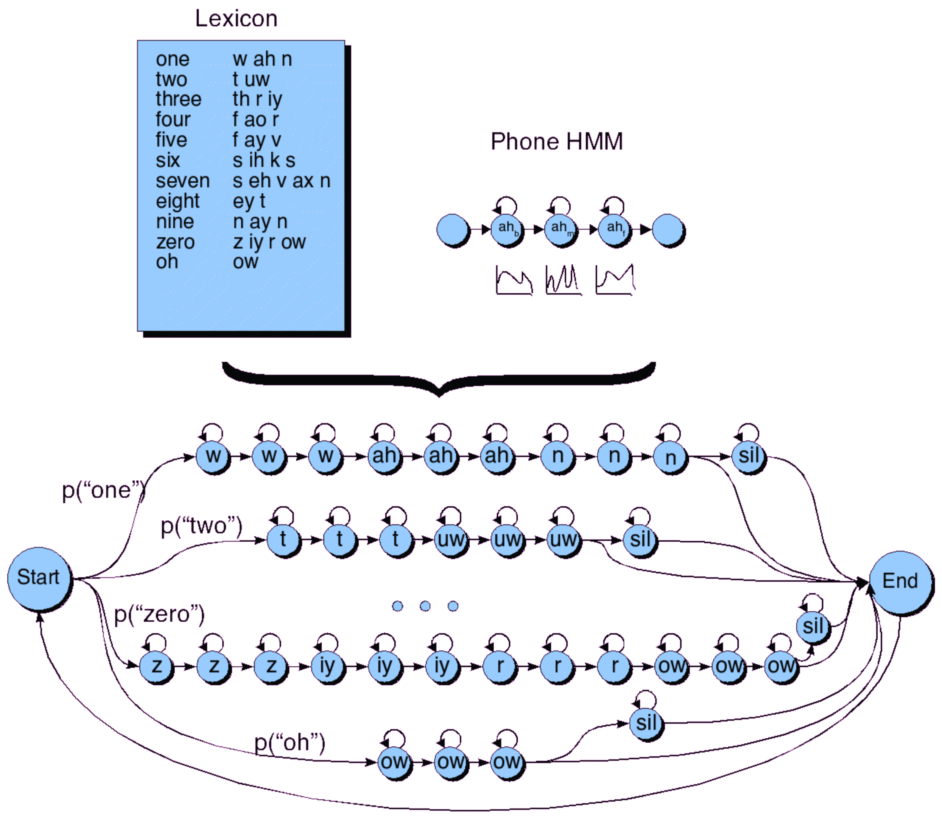
If G is a finite state grammar and L is a pronunciation dictionary, L ◦ G transduces a phone string to word strings allowed by the grammar

Determinization: Ensures each state has no more than one output transition for a given input label

Minimization: transforms a transducer to an equivalent transducer with the fewest possible states and transitions

Decoder - WFST Algorithm

combining cascade wfst-based models into one


Lexicon

one	w ah n
two	t uw
three	th r iy
four	f ao r
five	f ay v
six	s ih k s
seven	s eh v ax n
eight	ey t
nine	n ay n
zero	z iy r ow
oh	ow

Decoder

- One possibility:
  - For each state sequence S in WFST
  - Compute P(W | O)
  - Pick the highest sequence
- Why not?
  - $O(N^T)$
- Instead:
  - The Viterbi algorithm $O(N^2T)$
  - Is a **dynamic programming** algorithm

Decoder - Viterbi Algorithm

- The posterior P(W | O) can be factored into word-level score as

$$
\begin{aligned}
\hat{W} &= \underset{W \in \mathcal{W}}{\operatorname{argmax}} \left\{ \sum_{S \in \mathcal{S}_W} p(O, S|W) P(W) \right\} \\
&= \underset{W \in \mathcal{W}}{\operatorname{argmax}} \left\{ \sum_{S \in \mathcal{S}_W} \prod_{m=1}^{M_W} p(o_{t_{m-1}+1}^{t_m}, s_{t_{m-1}+1}^{t_m} | w_m) P(w_m | w_1^{m-1}) \right\}
\end{aligned}
$$

- Make the “Viterbi Approximation”

$$
\begin{aligned}
\hat{W} &= \underset{W \in \mathcal{W}}{\operatorname{argmax}} \left\{ \max_{S \in \mathcal{S}_W} p(O, S|W) P(W) \right\} \\
&= \underset{W \in \mathcal{W}}{\operatorname{argmax}} \left\{ \max_{S \in \mathcal{S}_W} \prod_{m=1}^{M_W} p(o_{t_{m-1}+1}^{t_m}, s_{t_{m-1}+1}^{t_m} | w_m) P(w_m | w_1^{m-1}) \right\}
\end{aligned}
$$

where $p(o_t^\tau, s_t^\tau | w)$ denotes the likelihood that the model of word w generates the speech segment $o_t \ldots o_\tau$ along the state transition process $s_t \ldots s_\tau$. $\mathcal{S}_W$ denotes a set of possible state sequences for W. t_m represents the ending frame of word w_m, which is determined by the state sequence S, i.e., the requirement that s_{t_m} is a final state in the model of w_m and s_{t_m+1} is an initial state in the model of w_{m+1} is satisfied. Here we assume $t_0 = 0$.

Decoder - Viterbi intuition

- Define S = (q0, q1, ... qT), O = (o0, o1,...oT)
- Process observation sequence O by time frame t
- Filling out the trellis
- Each cell:

$$v_t(j) = \max_{q_0,q_1,\ldots,q_{t-1}} P(q_0,q_1\ldots q_{t-1},o_1,o_2\ldots o_t,q_t=j|\lambda)$$

$$v_t(j) \ = \ \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

$v_{t-1}(i)$	the **previous Viterbi path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j

Decoder - Viterbi Recursion

1. **Initialization:**

$$\begin{aligned} v_1(j) &= a_{0j} b_j(o_1) \;\; 1 \le j \le N \\ bt_1(j) &= 0 \end{aligned}$$

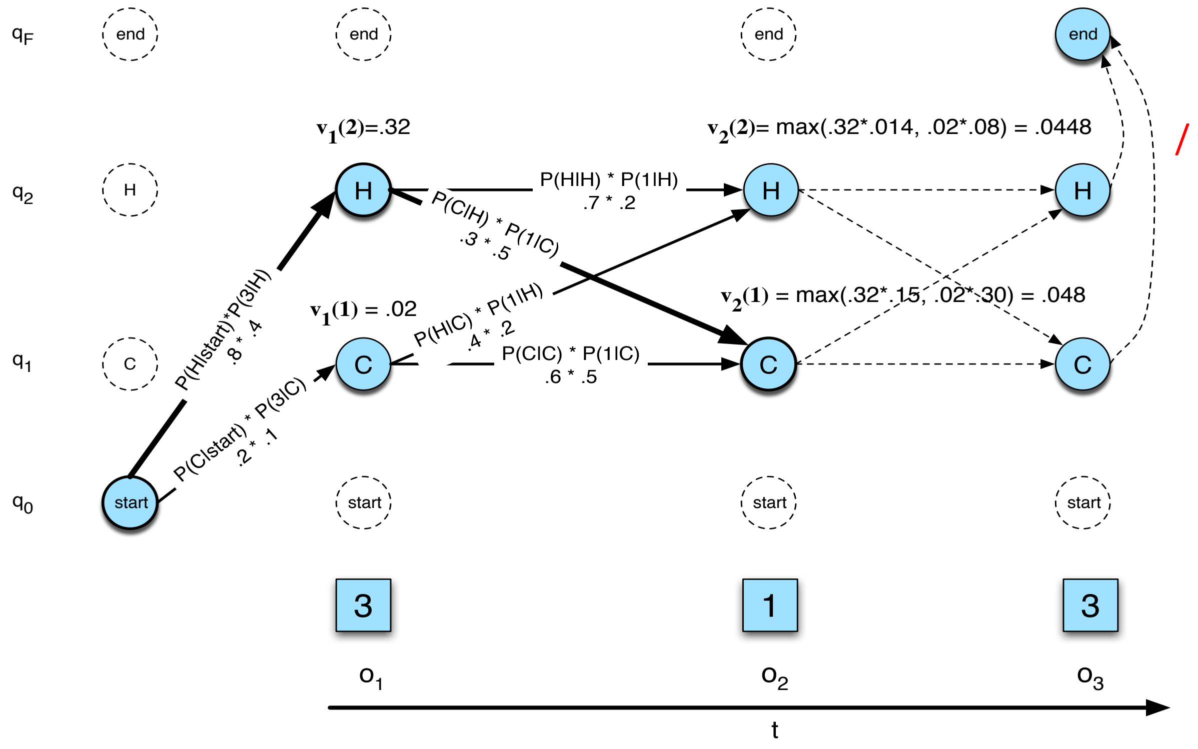
2. **Recursion** (recall that states 0 and q_F are non-emitting):

$$\begin{aligned} v_t(j) &= \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \\ bt_t(j) &= \operatorname*{argmax}_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \end{aligned}$$

3. **Termination:**

$$\begin{aligned} \text{The best score:} \quad P* = v_t(q_F) &= \max_{i=1}^{N} \; v_T(i) * a_{i,F} \\ \text{The start of backtrace:} \quad q_T* = bt_T(q_F) &= \operatorname*{argmax}_{i=1}^{N} \; v_T(i) * a_{i,F} \end{aligned}$$

Decoder - The Viterbi trellis

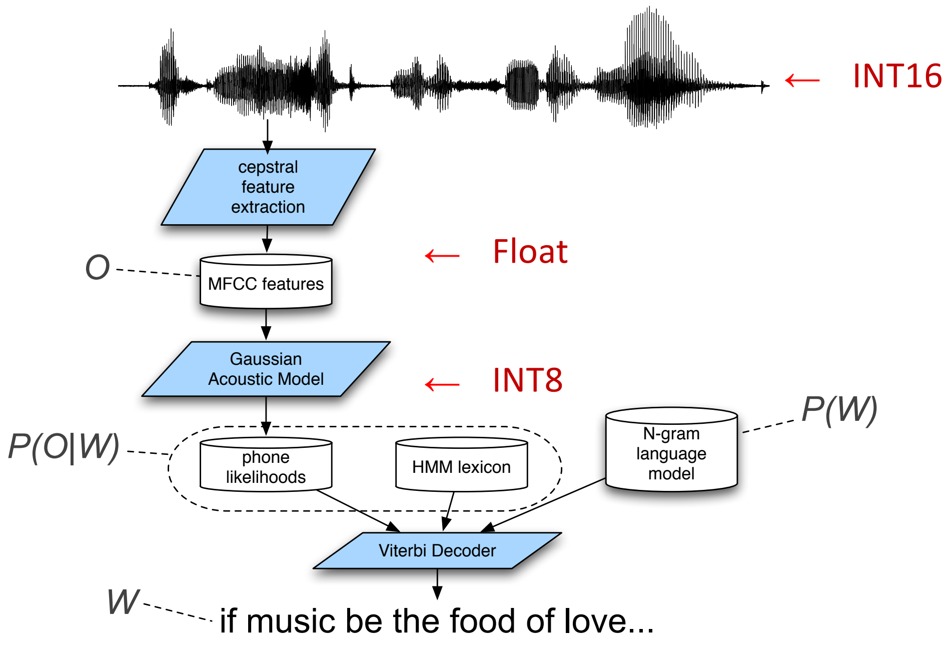


Engineering optimization

Rule of thumb for data-intensive computing is to place computation where the data is, instead of moving the data to the point of computation.

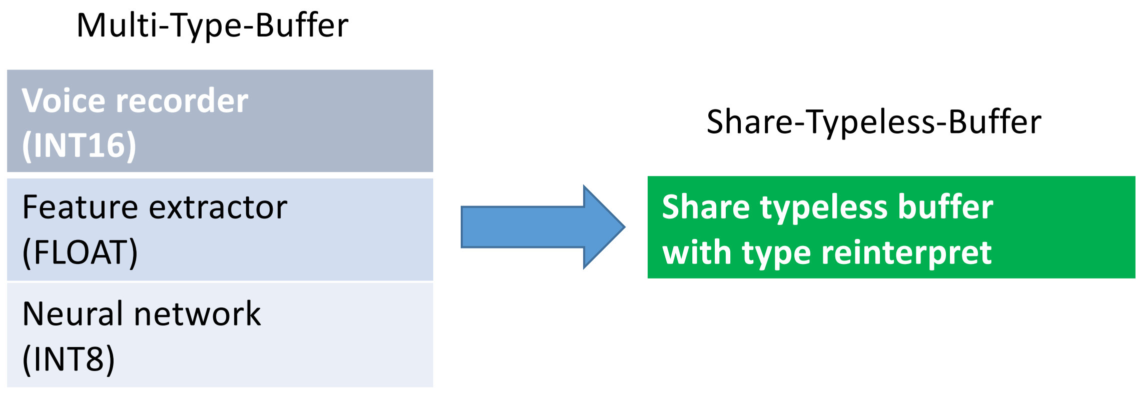
Time division multiplex buffer

Recorder, Feature extractor, Neural network... these modules use different buffer, if they can share same buffer, we can save much memory for mobile or embed devices.
But the problem is these modules own different data type (voice is int16, feature is float, neural network is float/int8).



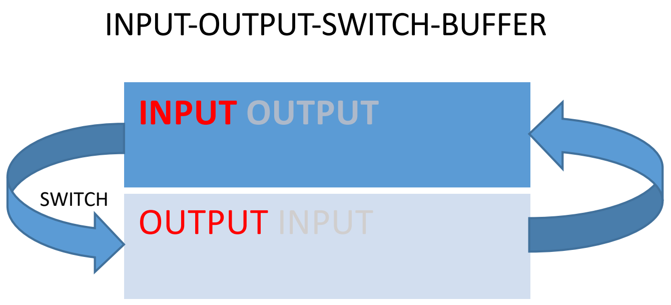
Time division multiplex buffer

So we create a typeless buffer and share between all modules, and make buffer typed only when special module use it, by this way we can merge original multi-buffer for many modules into one and save times memory.
And it also computation efficient, because when doing computation the previous module already put the data to where the current module computation performing (**Rule of thumb for data-intensive computing**).



Time division multiplex buffer (II)

In neural network forward computation, ordinary, each layer of neural network own its computation-buffer for input/output/cache, but when layer's number is large this will take much memory. To save memory, we use a input-output-switch-buffer that is when doing forward computation layer by layer we switch the input and output role for the buffer and shared between all layers in the netwrok.
By this way we also save much memory, and when doing computation the previous layer already put the data to where the current layer computation performing, so the input-output-switch-buffer also computation efficient (**Rule of thumb for data-intensive computing again**).



Quantized Math

Quantize a math number

$$x_{float} = x_{scale} \times (x_{quantized} - x_{zero\_point})$$

Calculate the scale

$$x_{float} \in [x_{float}^{min}, x_{float}^{max}]$$

$$x_{scale} = \frac{x_{float}^{max} - x_{float}^{min}}{x_{quantized}^{max} - x_{quantized}^{min}}$$

$$x_{zero\_point} = x_{quantized}^{max} - x_{float}^{max} \div x_{scale}$$

$$x_{quantized} = x_{float} \div x_{scale} + x_{zero\_point}$$

Quantized Arithmetic

$$
\begin{aligned}
z_{float} &= x_{float} \cdot y_{float} \\
z_{scale} \cdot (z_{quantized} - z_{zero\_point}) &= (x_{scale} \cdot (x_{quantized} - x_{zero\_point})) \cdot (y_{scale} \cdot (y_{quantized} - y_{zero\_point})) \\
&= x_{scale} \cdot y_{scale} \cdot (x_{quantized} - x_{zero\_point}) \cdot (y_{quantized} - y_{zero\_point}) \\
z_{quantized} - z_{zero\_point} &= \frac{x_{scale} \cdot y_{scale}}{z_{scale}} \cdot (x_{quantized} - x_{zero\_point}) \cdot (y_{quantized} - y_{zero\_point}) \\
z_{quantized} &= \frac{x_{scale} \cdot y_{scale}}{z_{scale}} \cdot (x_{quantized} - x_{zero\_point}) \cdot (y_{quantized} - y_{zero\_point}) + z_{zero\_point} \\
Multiplier_{x,y,z} &= \frac{x_{scale} \cdot y_{scale}}{z_{scale}} \\
z_{quantized} &= Multiplier_{x,y,z} \cdot (x_{quantized} - x_{zero\_point}) \cdot (y_{quantized} - y_{zero\_point}) + z_{zero\_point}
\end{aligned}
$$

Quantize neural network model

Goal: Convert FP32 Neural network to INT8 Neural network without significant accuracy loss

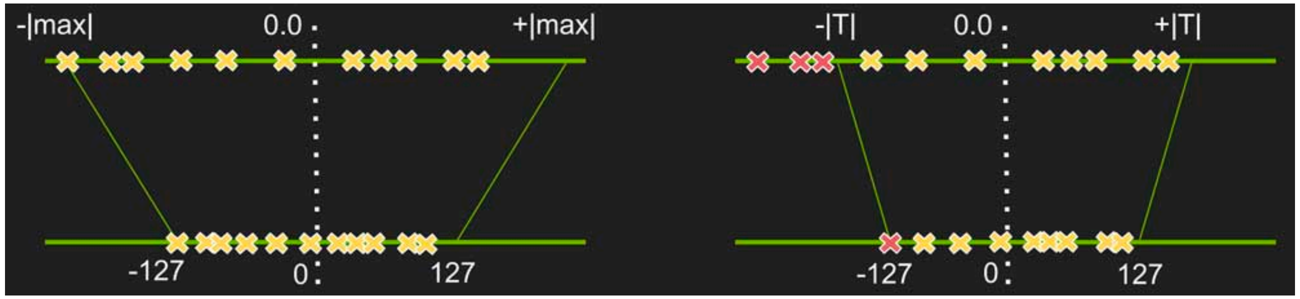
Why: INT8 math has higher throughput, and lower memory requirements

Challenge to Quantize neural network model

1.INT8 has significantly lower precision and dynamic range

	Dynamic Range	**Min Positive Value**
FP32	$-3.4 \times 10^{38} \sim +3.4 \times 10^{38}$	1.4×10^{-45}
FP16	$-65504 \sim +65504$	5.96×10^{-8}
INT8	$-128 \sim +127$	1

2.Number distribution is unsymmetric, loss accuracy when encode FP32 to INT8

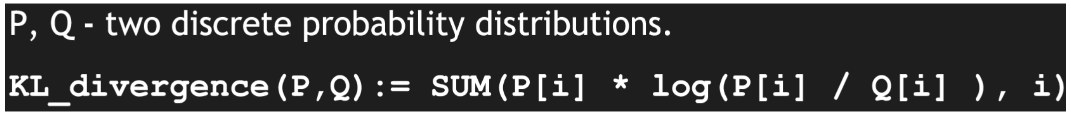


3.Not all operations in neural network can be quantized, need quantizing/dequantizing between layers

Quantize neural network model

Since FP32 → INT8 is re-encoding information, how to optimize the quantizing threshold than minimize information loss?

Using KL_divergence to measure loss of information.



P, Q - two discrete probability distributions.

```
KL_divergence(P,Q):= SUM(P[i] * log(P[i] / Q[i] ), i)
```

Solution: Run FP32 inference on a Dataset, collect histograms of activations and generate quantized distributions with different saturation thresholds, than pick threshold which minimizes KL_divergence(ref_distr, quant_distr).

Quantize neural network model

Entropy Calibration - pseudocode

```
Input: FP32 histogram H with 2048 bins: bin[ 0 ], …, bin[ 2047 ]

For i in range( 128 , 2048 ):
    P = [ bin[ 0 ] , ..., bin[ i-1 ] ] // reference_distribution
    outliers_count = sum( bin[ i ] , bin[ i+1 ] , … , bin[ 2047 ] )
    P[ i-1 ] += outliers_count
    P /= sum(P)                                            // normalize distribution P
    Q = quantize [ bin[ 0 ], …, bin[ i-1 ] ] into 128 levels // candidate_distribution expand Q to ‘ i ’ bins
    Q /= sum(Q)                                            // normalize distribution Q
    divergence[ i ] = KL_divergence( P, Q)
End For

Find index ‘m’ for which divergence[ m ] is minimal threshold = ( m + 0.5 ) * ( width of a bin )
```

Optimize Viterbi algorithm

- Viterbi is $O(N^2T)$
  - N is number of WFST states, T is frames length
- Methods to make Viterbi search faster
  - Beam search (pruning)
  - Weight push (early pruning)
  - Tree-based lexicons (reduce search space)
  - Cache Wfst operations

Beam search

- Instead of retaining all candidates (cells) at every time frame
- Use a threshold T to keep subset:
  - At each time t
  - Identify state with lowest cost Dmin
  - Each state with cost > Dmin+ T is discarded (“pruned”) before moving on to time t+1
  - Unpruned states are called the active states

Key results of our work

ASR engine performance overview

	base model	total memory	RTF	ACC
kaldi	410KB	3.5MB	0.035	94.75%
petrel_lite (our)	410KB	460KB	0.017	94.74%

petrel_lite is efficient by using Time-Division-Multiplex-Buffer, Model Compression and Viterbi Path recycling

Remaining problems

- Low resource ASR – with no significant ACC loss compare to high resource ASR
- Robustness and Adaptability – noise, distance, accent
- Confidence Measures – better methods to evaluate the correctness of hypotheses
- Out-of-Vocabulary (OOV) Words – dealing OOV in a sensible way
- Dialect and Mixed language

Any questions ?

Thank you for your attention!