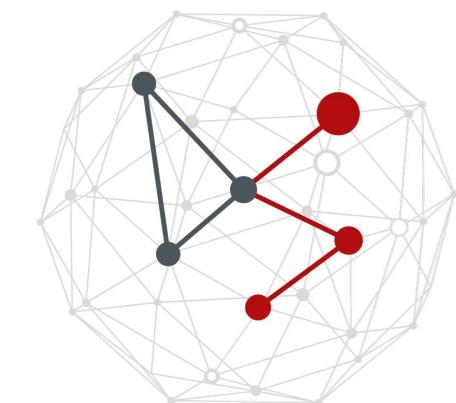


SPEECH MODELS AND AUTOMATIC SPEECH RECOGNITION (ASR) SYSTEMS

Michele Rossi
rossi@dei.unipd.it

Dept. of Information Engineering
University of Padova, IT



Outline (1/2)

- Quick intro to speech recognition systems
- Mel Frequency Cepstral Coefficients (MFCC)
 - What are they for?
- Computing MFCC
 - The 6 basic steps for their computation
 - The Mel scale for human perceived frequency
 - The 6 steps in greater detail
- Additional features
 - Energy
 - Delta coefficients
 - Delta-Delta coefficients

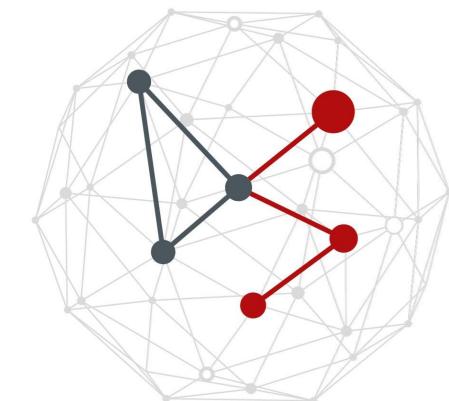
Outline (2/2)

- Automatic Speech Recognition (ASR) systems
 - The noisy channel model
 - The ASR system
 - Language, pronunciation and acoustic models
 - Viterbi decoding – decoding example
 - General ASR architecture
 - HMM training
 - Performance & open issues

Speech recognition: a quick intro

- Raw audio samples are not suitable for statistical modeling
 - We represent them using feature vectors
- Speech recognition
 - Input: sampled & digitized speech data
 - Desired output: sequence of words that were spoken
 - It is a pattern matching problem
 - Combines acoustic and language models
 - Most modern systems use HMM
 - A speech signal is modeled as a sequence of piecewise stationary signals
- HMM
 - Training: forward-backward algorithm with labeled speech data
 - Gaussian Mixture models (GMM) used to model emissions
- CNN+RNN
 - Used in modern systems

SPEECH FEATURES



MFCC – what are they for?

- First step in any speech recognition system
 - MFCC are audio features: should be good to identify the linguistic content (neglecting background noise, emotion, etc.)
- Sounds generated by humans
 - Are filtered by the shape of the vocal tract, including tongue, teeth, etc.
 - This shape determines which sound comes out
 - If we determine this shape accurately, this should give a good representation of the phoneme that is being produced
- Shape of the vocal tract
 - Manifests itself in the envelope of the *short-term power spectrum*
 - The job of MFCCs is to accurately represent this envelope [1]

[1] S. Davis, P. Mermelstein, “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sequences,” *IEEE Transactions on Acoustic, Speech and Signal Processing*, Vol. 28, No. 4, 1980.

Computing MFCC – what is it?

- A highly engineered preprocessing of the audio waveform
- Designed
 - to discard the large amount of information in waveforms that has been found to be **irrelevant for discrimination**
 - to express the remaining information in a form that facilitates discrimination, e.g., with **GMM-HMMs** or **Neural Networks**
- GMM advantages
 - given enough components, (i) they can model PDFs *to any degree level of accuracy* & (ii) they are fairly easy to *train through EM*

MFCC computation involves 6 steps

1. Segment the signal into short frames
2. For each frame: calculate its power spectrum
3. Apply a Mel filterbank to the power spectrum
 - sum the energy within each filter
4. Take the logarithm of all filterbank energies
5. Take the DCT of the log filterbank energies
6. Keep DCT coefficients 2-13, discard the others

Step 1 - framing

- An audio signal is constantly changing (*non-stationary*)
- Dealing with non-stationary data is difficult
- To simplify things
 - We assume that on short time scales the signal does not change much
 - “Doe not change” means it is **statistically stationary**
- This is why we frame the signal into **20-40ms time windows**
 - Standard value is **25ms**
 - **Shorter:** not enough samples to get a reliable spectral estimate
 - **Longer:** the signal (statistics) changes too much within each frame

Step 2 – power spectrum

- Compute the power spectrum of a frame
 - This is motivated by the human *cochlea* (organ in the ear)
 - It vibrates at different spots depending on the frequency of the incoming sound waves
 - Depending on the location of the cochlea that vibrates, different nerves fire informing the brain that certain frequencies are present
 - This is then translated by the brain into words, sentences, etc.
- The periodogram estimate of the power spectrum
 - Does a similar job
 - Informing us on which frequencies are present in the current frame

Step 3 – filterbank

- The periodogram estimate of the power spectrum
 - Still contains a lot of information that is not required by Autonomous Speech Recognition (ASR) systems
 - In fact, the cochlea cannot discern the difference between two closely spaced frequencies
 - This effect becomes more apparent as frequencies increase
- Hence
 - We take clumps of periodogram **bins** and sum them up to get an *idea of how much energy there exists in each region*
 - This is performed using a **Mel filterbank**
 - **First Mel filter is very narrow:** to get an idea of how much energy there is around 0 Hertz
 - **Filters get wider as frequencies get higher:** we become less concerned about variations (**Mel scale** tells us how to space filterbanks)

Step 4 – logarithm

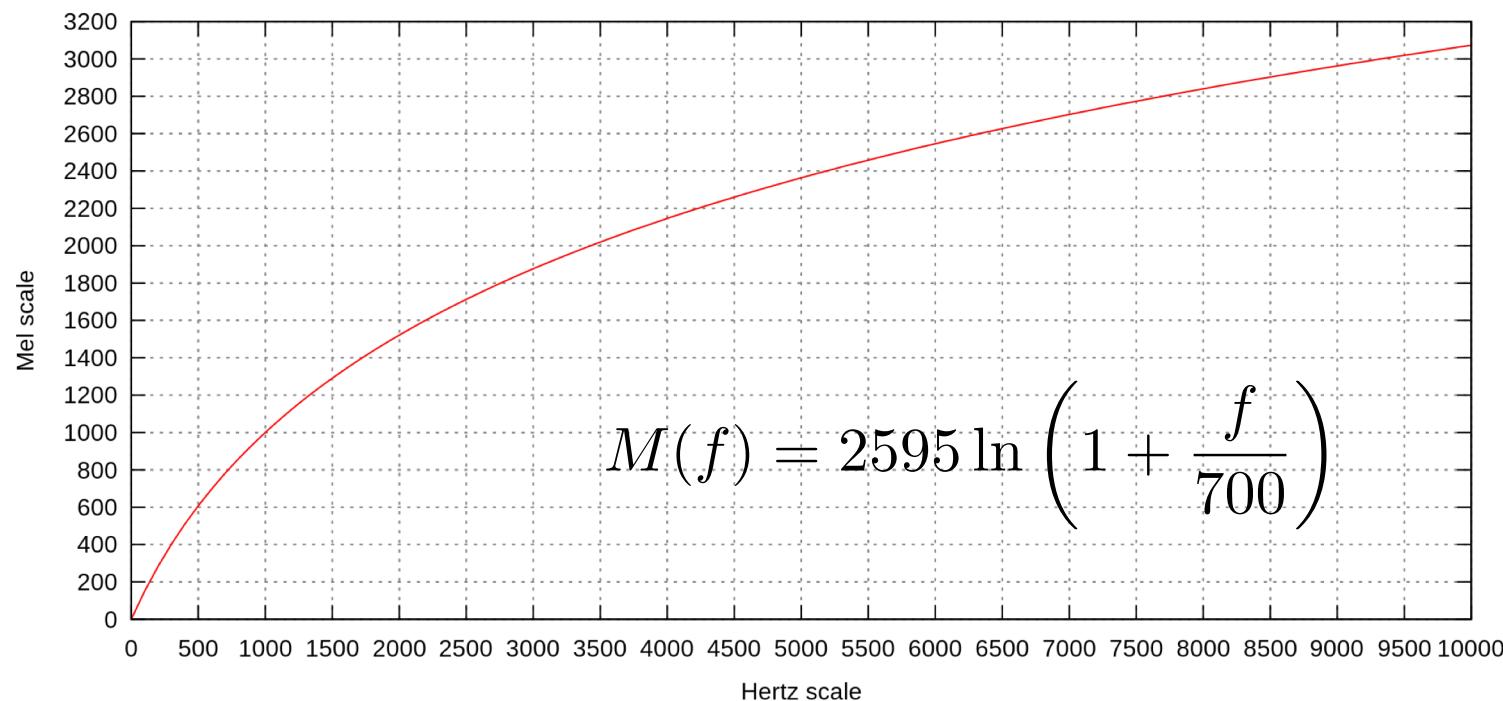
- Once we have the filterbank energies
 - We take their logarithm
 - This is also motivated by human hearing: we do not hear loudness on a linear scale
 - Generally, to double the perceived volume of a sound we need to put 8 times as much energy into it
 - This means that large variations in energy *may not sound that different* if the sound is loud
- Hence
 - This compression operation (logarithm) makes our features match more closely what humans actually hear

Steps 5 and 6 – DCT

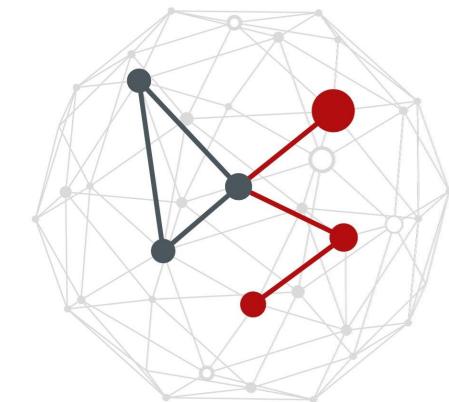
- Discrete Cosine Transform (DCT)
 - There are two main reasons for applying it
 - First: filterbanks are quite overlapping (meaning that filterbank energies are quite correlated with each other)
 - DCT decorrelates filterbank energies
 - This means that diagonal covariance matrices can be used to model the features with GMM and/or GMM/HMM systems (very desirable)
 - Also (energy compaction property of DCT) we obtain a compact representation in the first DCT coefficients
 - Second: discard some information for improved performance
 - Only 12 of the 26 DCT coefficients are kept (energy compaction of DCT)
 - Higher DCT coefficients represent fast changes in the filterbank energies
 - These fast changes degrade ASR performance
 - A (small) improvement is achieved by dropping them

The Mel scale - M(f)

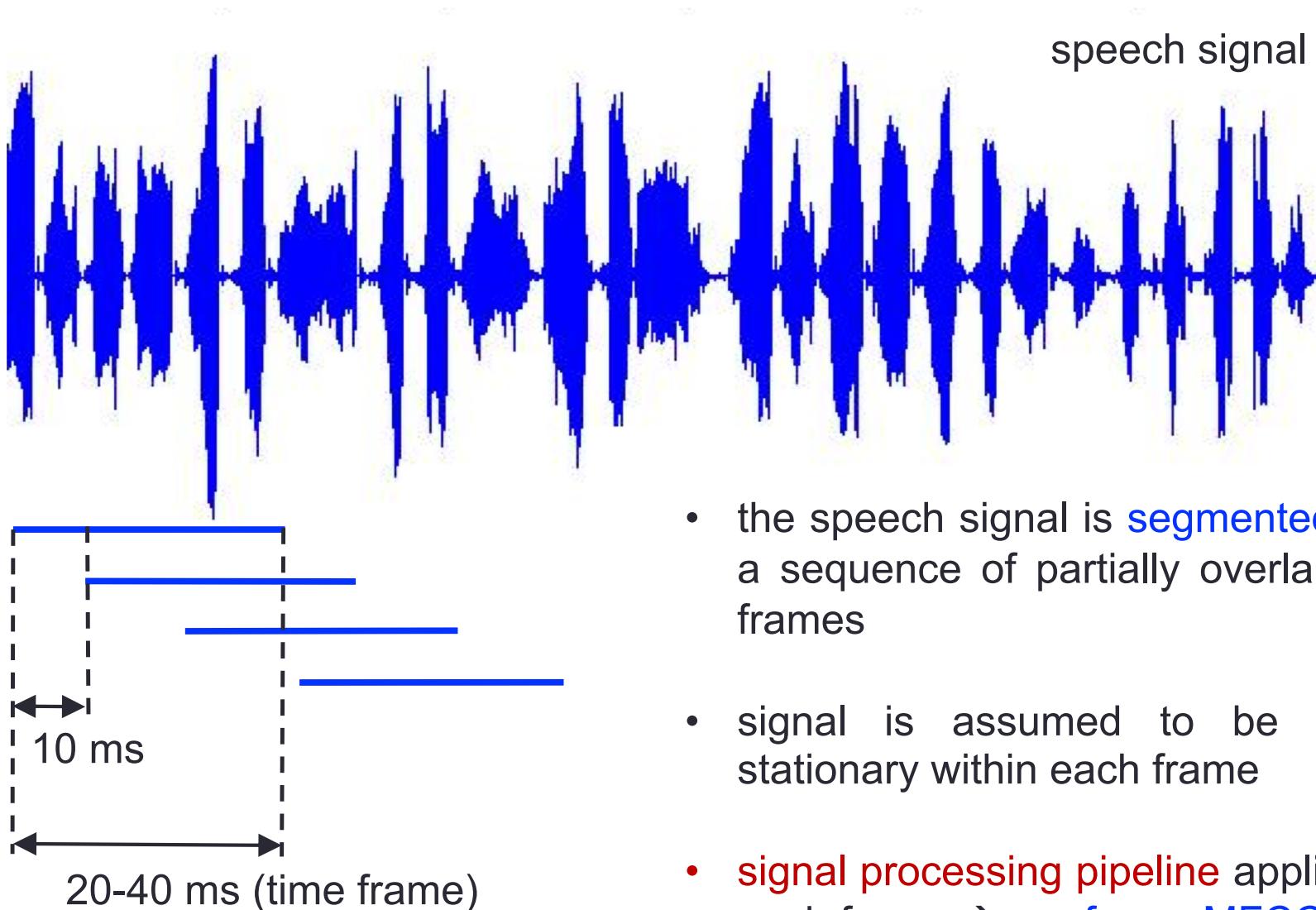
- Relates the *human perceived frequency* (or pitch) of a pure tone to its *actual measured (perceived) frequency*
- Humans are much better at discerning small changes in pitch at low frequencies than they are at high frequencies
- Incorporating it makes ASR match more closely what humans hear



STEPS 1-6 IN MORE DETAIL



Step 1 – framing (1/2)

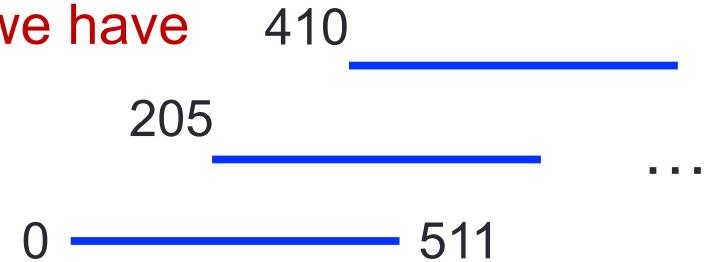


Step 1 – framing (2/2)

- Frame the audio signal into [20,40] ms frames (standard value is **25 ms**)

- Example: **20.480 kHz signal, with 25 ms we have**

- sampling rate: 20,480 samples/second
 - $0.025 * 20,480 = 512$ samples/frame



- Frame step is usually 10 ms (205 samples)

- This allows for some overlap between subsequent frames
 - First frame (of 512 samples) starts at sample 0
 - Second frame (still of 512 samples) start at sample 205, etc.
 - Keep framing until the end of the speech signal
 - If the signal does not divide into an even number of frames, pad it with zeros so it does

- Notation

- $s(n)$ is the “raw” audio signal (i.e., time domain signal)
 - **$s_i(n)$ is frame i with n ranging from 0 to 511** in our example

Step 2 – power spectrum (1/4)

- For each frame $s_i(n)$, $n = 0, 1, \dots, N - 1$ ($N = 512$)
 - Calculate the complex DFT (Discrete Fourier Transform)

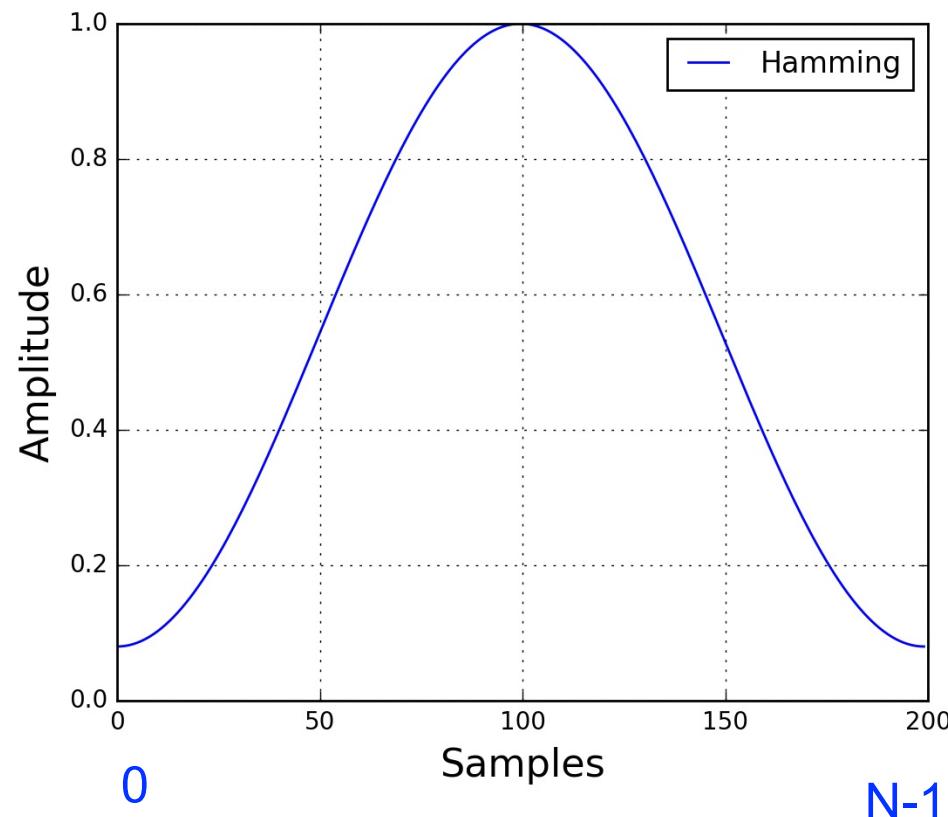
$$S_i(k) = \sum_{n=0}^{N-1} s_i(n)w(n)e^{-j2\pi kn/N}, \quad k = 0, 1, \dots, N - 1$$

- **DFT**
 - Converts a time sequence of N equally spaced samples into an N -long complex sequence, which is a **complex-valued function of frequency**
 - The DFT (with $w(n)=1$) completely describes the discrete Fourier transform of an **N -periodic time sequence**
 - When applying DFT, **we are implicitly applying it to an infinitely repeating (periodic) signal**. However, if this is not the case, e.g., first and last samples of $s_i(n)$ do not match, this is *interpreted as a discontinuity in the signal and generates a lot of high frequency response in the transformed signal*, that we do not want → use of “windowing” **w(n)**

Step 2 – power spectrum (2/4)

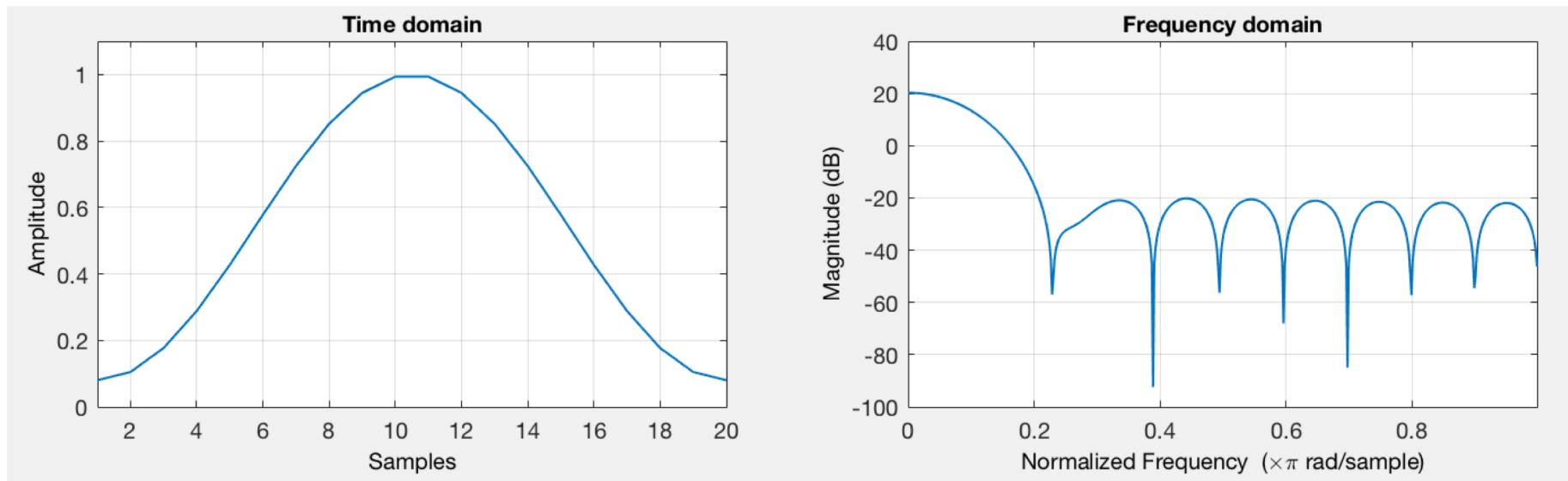
- Often used: **Hamming window** - $w(n)$

$$w(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right), \quad n = 0, \dots, N-1$$



Step 2 – power spectrum (3/4)

- Hamming window – transformation (with N=20)



- The center of the main lobe of a smoothing window occurs at each frequency component of the input signal (frequency plot is symmetric around zero)
- Center lobe width (across negative & positive freqs.) is $4\Omega_N = 4(2\pi/N)$
- Ideally, only the main lobe should be emphasized while the side lobes should be zeroed (this prevents energy leakage across subsequent frequencies)

Step 2 – power spectrum (4/4)

- Note
 - Among the $N=512$ (in our example) complex DFT values
 - Only the first $1+N/2$ values are significant, the remaining ones are complex conjugates of the first $1+N/2$ values (this is how DFT works)
 - Hence, the first $1+N/2=257$ elements *are kept*, while the remaining ones ($N/2-1$ values) *are discarded*
- Compute the periodogram estimate of the power spectrum:

$$P_i(k) = \frac{|S_i(k)|^2}{N}, k = 0, \dots, N/2$$

Remember that: $k=0$ is the DC component (frequency $f=0$). Then, with DFT the step size is F_s/N (F_s is the *sampling frequency*). Hence, $k=1$ corresponds to frequency $f_1=F_s/N$ and element $k=N/2$ corresponds to frequency $f_{N/2}=F_s/2$ (this is the **Nyquist critical frequency**, i.e., the highest signal frequency that can be represented by sampling at frequency F_s)

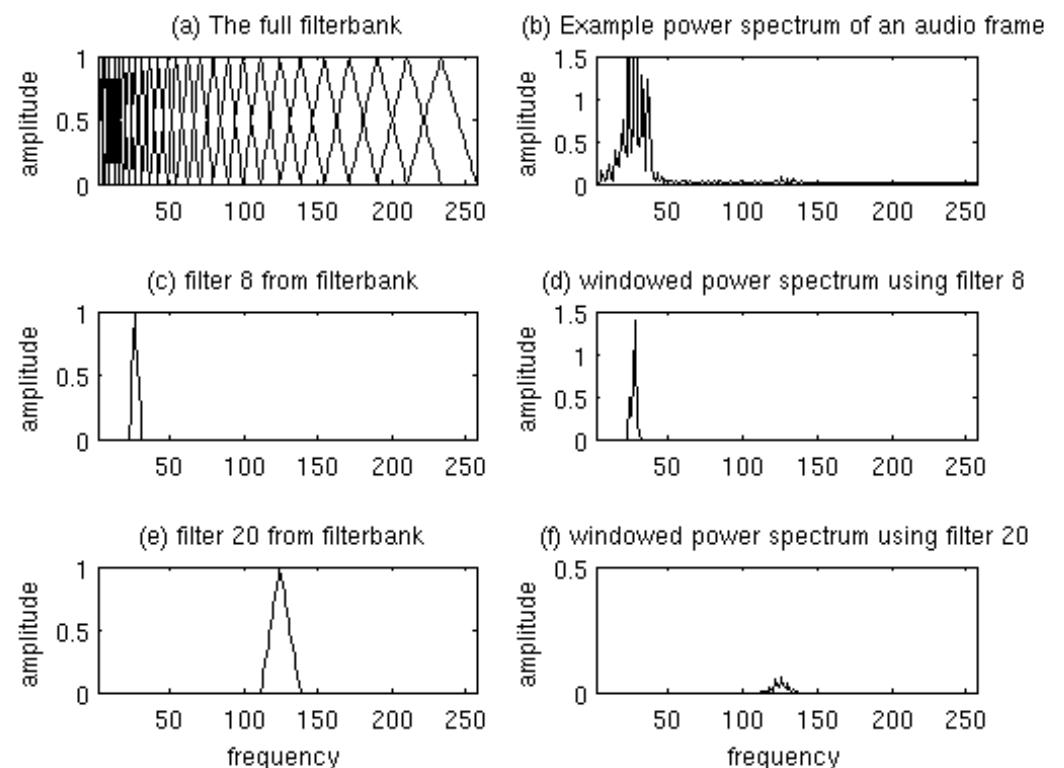
Step 3 – filterbanks (1/3)

- Compute Mel-spaced filterbanks (through Mel's equation)
 - They amounts to 20-40 triangular filters
 - Each filter (in our case) has 257 values ($1+N/2$, to match the output of step 2) and it is multiplied by the entire power spectrum from step 2

full filterbank (left)
full power spectrum (right)

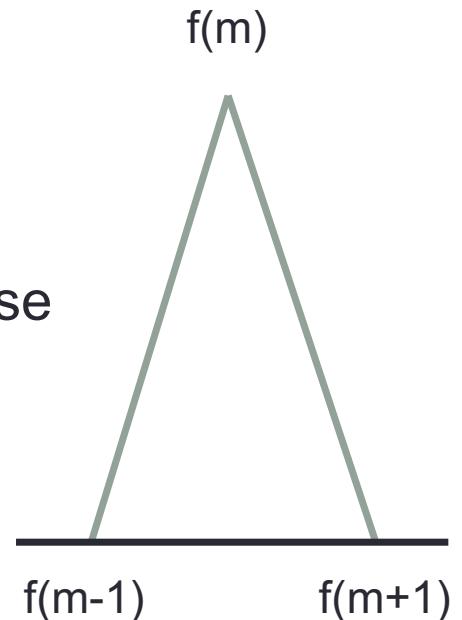
filter 8 from filterbank (left)
output when this filter is applied (right)

filter 20 from filterbank (left)
output when this filter is applied (right)



Step 3 – filterbanks (2/3)

- Obtain N_{fb} Mel-spaced triangular filters
 - Pick the number of filters (usually $N_{fb}=26$)
 - Set the maximum frequency as $f_{max}=F_s/2$
 - Set the minimum frequency as, e.g., $f_{min}=300$ Hz (user defined)
 - Compute N_{fb} center frequencies $f(1), f(2), \dots, f(N_{fb})$
 - Linearly spaced in Mel's domain
 - Non-linearly spaced in frequency domain [Hz]
- Each filter m
 - Is centered at frequency $f(m)$
 - Is equal to 1 for $f(m)$ and decreases linearly otherwise
 - Is zero at $f(m-1), f(m+1)$ and outside $[f(m-1), f(m+1)]$
 - N_{fb} filters require $N_{fb}+2$ thresholds



Step 3 – filterbanks (3/3)

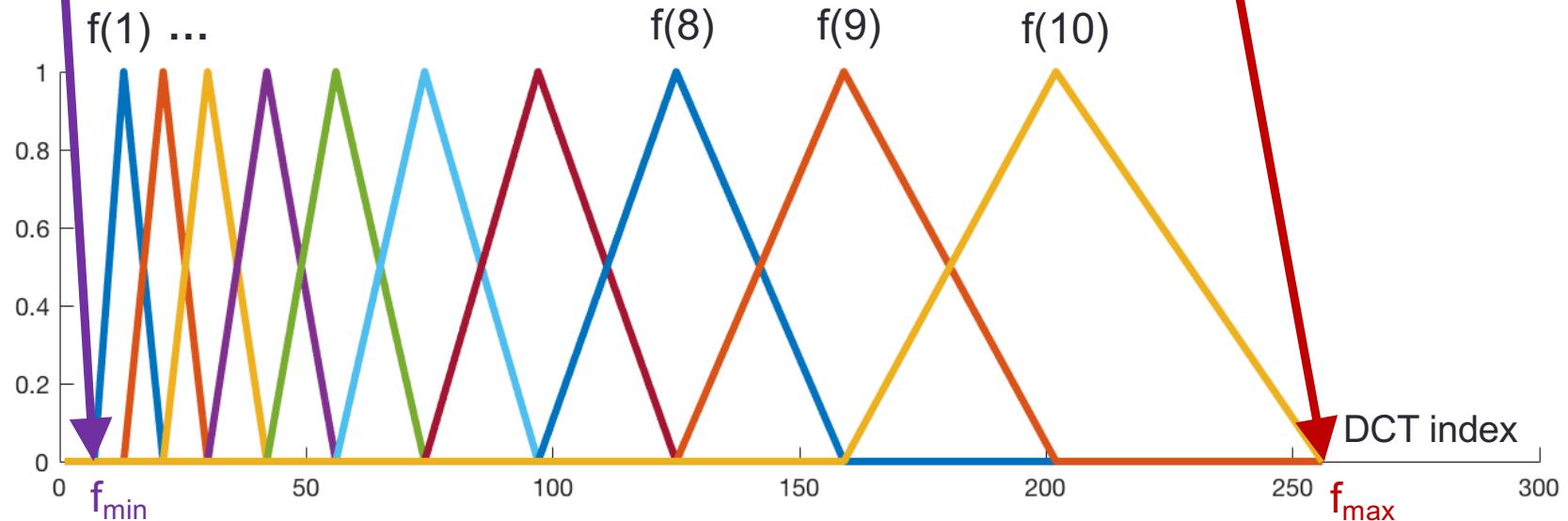
- Example
 - $F_s = 20.480 \text{ kHz}$, $N=512$, $N_{fb}=10$, $f_{\max}=F_s/2=10,240 \text{ Hz}$, $f_{\min}=300 \text{ Hz}$

- Frequencies [Hz]

300= f_{\min} 543= $f(1)$ 845= $f(2)$ 1,220= $f(3)$ 1,687= $f(4)$ 2,267= $f(5)$ 2,988= $f(6)$
3,883= $f(7)$ 4,997= $f(8)$ 6,381= $f(9)$ 8,102= $f(10)$ 10,240= f_{\max}

- Corresponding DFT indices

[7 13 21 30 42 56 74 97 125 159 202 256]



S.

```
9 FminMel=1125*log(1+(Fmin/700)); % min Mel's frequency
10 FmaxMel=1125*log(1+(Fmax/700)); % Max Mel's frequency
11
12 Nthresh=Nfb+2;      % number of required threshold values
13 step=(FmaxMel-FminMel)/(Nthresh-1); % step size (linearly partition Mel's space)
```

- Example

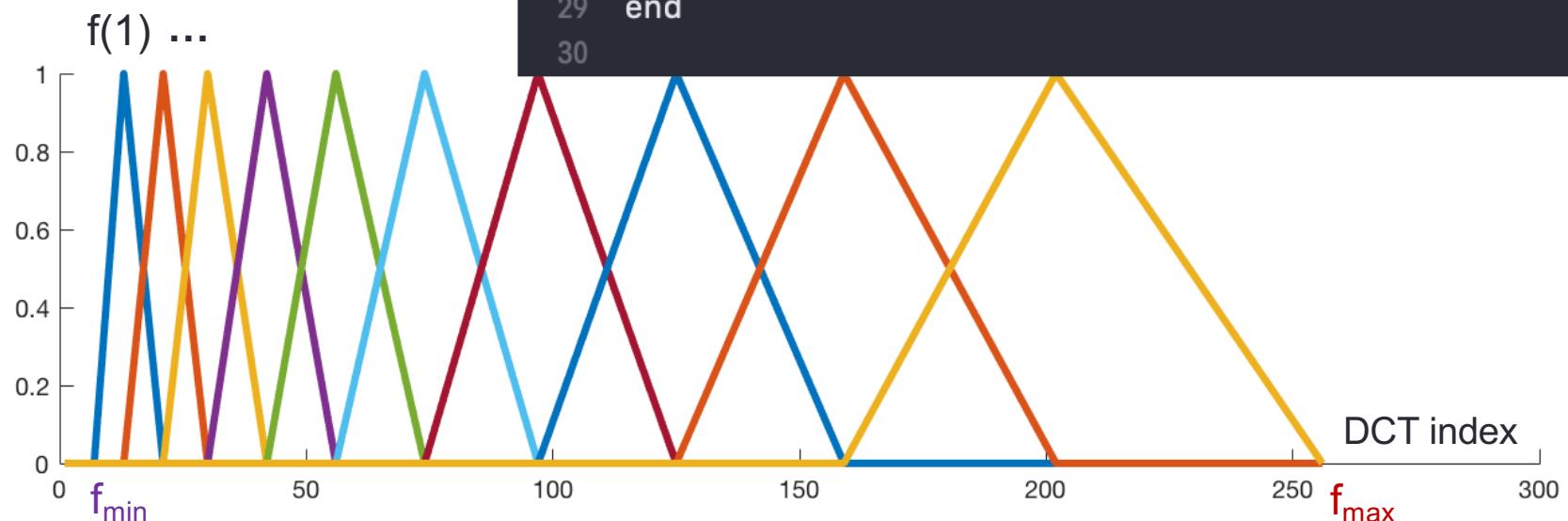
- $F_s = 20.480 \text{ kHz}$, $N=512$, $N_{fb}=8$

- Frequencies [Hz]

$$\begin{array}{lll} 300=f_{\min} & 543=f(1) & 845=f(2) \\ 3,883=f(7) & 4,997=f(8) & 6,320=f(9) \end{array}$$

- Corresponding DFT indices

$$[7 \quad 13 \quad 21 \quad 30 \quad 42 \quad 56]$$



```
18
19 % for each threshold do
20 for i = 0:(Nthresh-1)
21     % Mel frequencies, evenly distributed
22     melvec(i+1) = FminMel+i*step;
23
24     % Hz frequencies, obtained inverting Mel relation
25     freqvec(i+1) = 700*(exp(melvec(i+1)/1125)-1);
26
27     % DFT indices, obtained from Hz frequencies
28     f(i+1) = floor((nDFT+1)*freqvec(i+1)/Fs);
29 end
30
```

Step 4 – logarithms

- We now have N_{fb} Mel-spaced triangular filters
- For each filter $m=1,2,\dots, N_{fb}$
 1. Multiply filter m by the full power spectrum from step 2 (the resulting energy trace will be mostly 0 (besides around $f(m)$, the filter acts as a mask, centered at $f(m)$)
 2. Add up the non-zero coefficients in this trace → obtain E_m
 3. Take the logarithm: $\log(E_m)$
- This gives us N_{fb} real values:
- $\log(E_1), \log(E_2), \dots, \log(E_M)$, with $M=N_{fb}$
 - One for each Mel's frequency band
 - They tell how much energy there is within each band

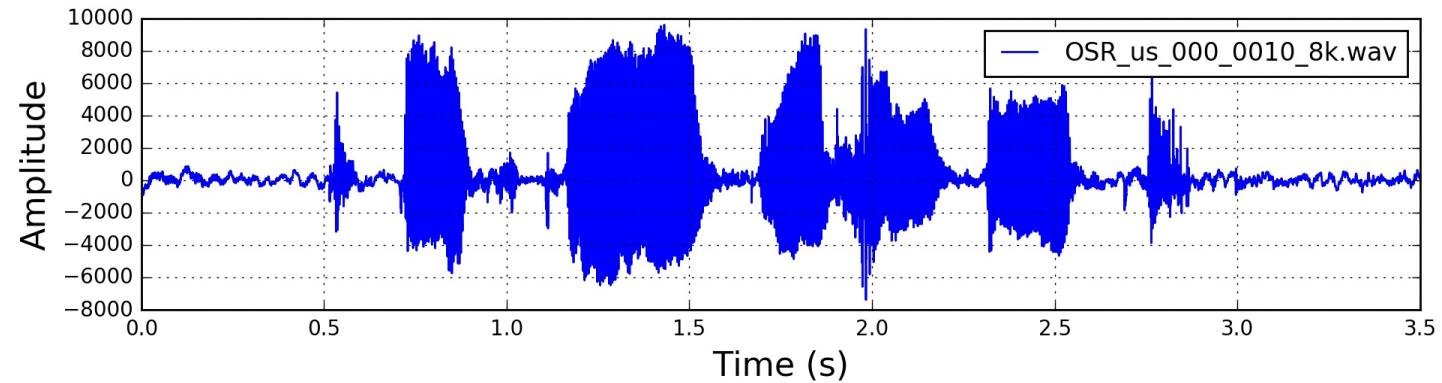
Steps 5 and 6

- Step 5 – for each frame
 - Take the DCT (Discrete Cosine Transform) of the vector containing the logarithms of the energy within each band
 - The result of the DCT is a vector of N_{fb} cepstral coefficients
- Step 6 – for each frame
 - For ASR, the cepstral coefficients no. 2,3, ...,13 are kept
 - The remaining ones are discarded
 - Note that: the first DCT coefficient is the sum of all the log-energies computed at the previous step (by the very def. of DCT) – thus, it is an overall measure of signal **loudness** and is not very informative - it is often discarded for *speech recognition* or *speaker id* applications where the system has to be robust to loudness variations
- Final result of this processing is
 - 12 cepstral coefficients for each frame. For frame i :

$$\mathbf{c}(i) = [c_1(i), c_2(i), \dots, c_{12}(i)]^T$$

Some visual insight (1/2)

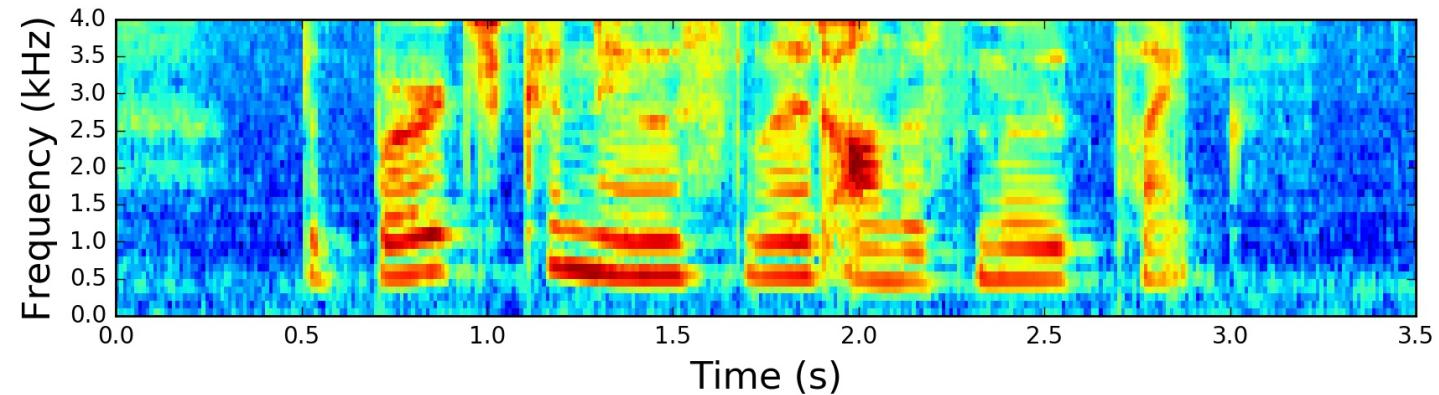
$F_s = 8$ kHz time signal



Signal in the Time Domain

After applying the filterbank to the power spectrum (periodogram):

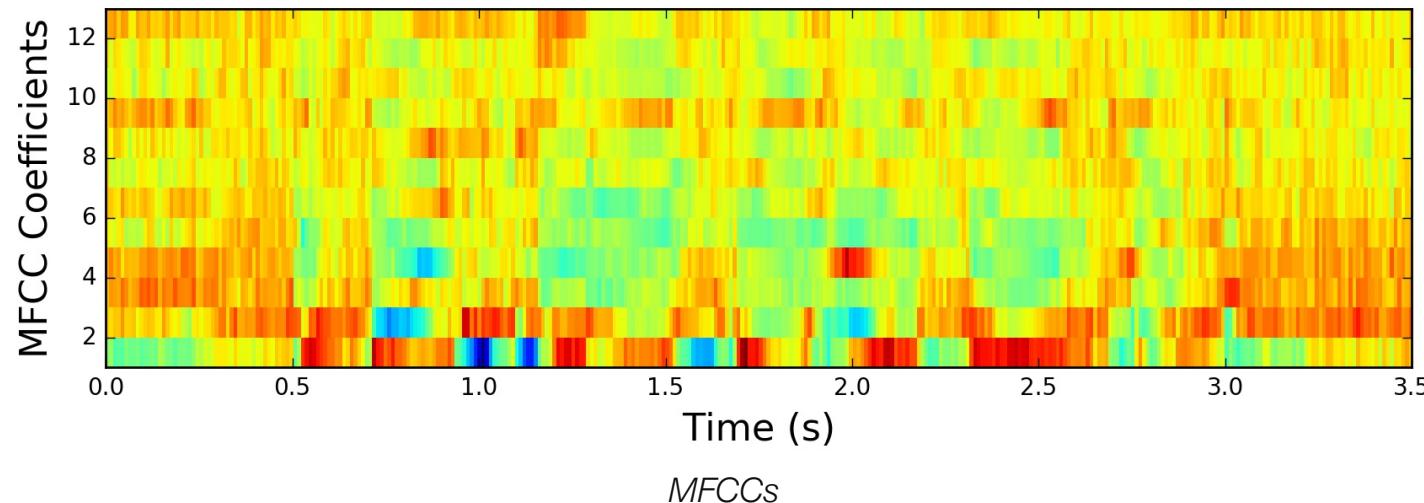
- colors represent log-energies



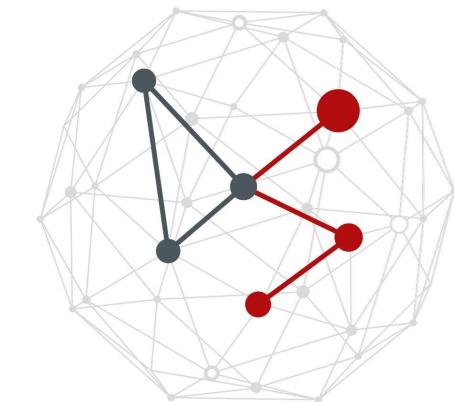
Spectrogram of the Signal

Some visual insight (2/2)

Mel's Frequency Cepstral Coefficients (MFCCs):



ADDITIONAL FEATURES



Additional features

- Up to now, we got 12 MFCC coefficients, for each frame i :

$$\mathbf{c}(i) = [c_1(i), c_2(i), \dots, c_{12}(i)]^T$$

- Additional features are usually tracked:
 1. Energy (1 additional coefficient)
 2. Δ coefficients (12 additional coefficients)
 3. $\Delta-\Delta$ coefficients (12 additional coefficients)
- Adding these leads to much better ASR performance

Energy

- Let the raw signal (time domain) be represented by $s(n)$
- Where n is the discrete sampling time
- The energy of the generic frame i :

$$E(s_i) = \sum_{n=n_1(i)}^{n_2(i)} s(n)^2$$

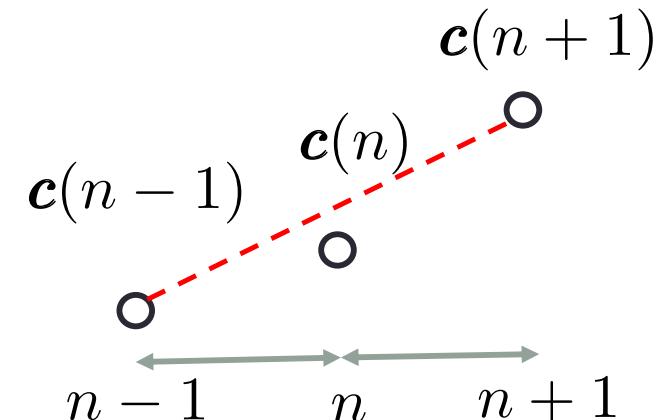
- $n_1(i)$ and $n_2(i)$ are **first** and **last** audio sample in frame i
- Sometimes, logarithms are used to limit dynamics

$$\log_{10} E(s_i)$$

Delta coefficients (1/2)

- MFCC feature vector
 - Only describes the power spectrum of a single frame
 - However, speech also has information in the dynamics, i.e.,
 - trajectories of MFCC coefficients over time
- Delta MFCC coefficients
 - Represent local derivatives of MFCC coefficients (“velocity” of $s(n)$)
 - Append those coefficients to the MFCC feature vector
 - A delta is computed for each element of the MFCC feature vector
 - 12 MFCC elements \rightarrow 12 Δ coefficients
 - At sampling time n we have:

$$\Delta(n) = \frac{\mathbf{c}(n+1) - \mathbf{c}(n-1)}{2}$$



Delta coefficients (2/2)

- Common regression formula (e.g., used by HTK)

$$\Delta(n) = \frac{\sum_{m=1}^M m(\mathbf{c}(n+m) - \mathbf{c}(n-m))}{2 \sum_{m=1}^M m^2}, \text{ usually } M = 2$$

Delta-Delta coefficients

- Trajectory in the Delta coefficients (“acceleration” of $s(n)$)
- So the new estimation formula is:

$$\Delta^2(n) = \frac{\Delta(n+1) - \Delta(n-1)}{2}$$

- In practice, the following is implemented:

$$\Delta^2(n) = \frac{\sum_{m=1}^M m(\Delta(n+m) - \Delta(n-m))}{2 \sum_{m=1}^M m^2}, \text{ usually } M = 2$$

Full feature vector

- For each frame:
 - 12 MFCC coefficients
 - 12 Delta coefficients
 - 12 Delta-Delta coefficients
 - 1 signal energy coefficient
 - 1 Delta energy coefficient
 - 1 Delta-Delta energy coefficient
-

- Total: **39 coefficients**
 - Used in most speech recognition applications

$$\begin{bmatrix} c_1 \\ \vdots \\ c_{12} \\ \hline \Delta_1 \\ \vdots \\ \Delta_{12} \\ \hline \Delta_1^2 \\ \vdots \\ \Delta_{12}^2 \\ \hline E(s) \\ E(\Delta) \\ E(\Delta^2) \end{bmatrix}$$

Implementations

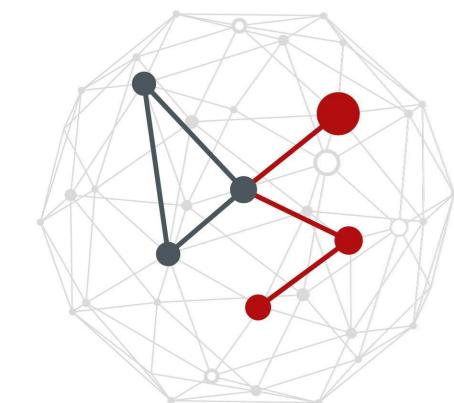
- HTK Speech recognition toolkit

<http://htk.eng.cam.ac.uk/>

- Phyton library to extract MFCCs

https://github.com/jameslyons/python_speech_features

AUTOMATIC SPEECH RECOGNITION SYSTEMS

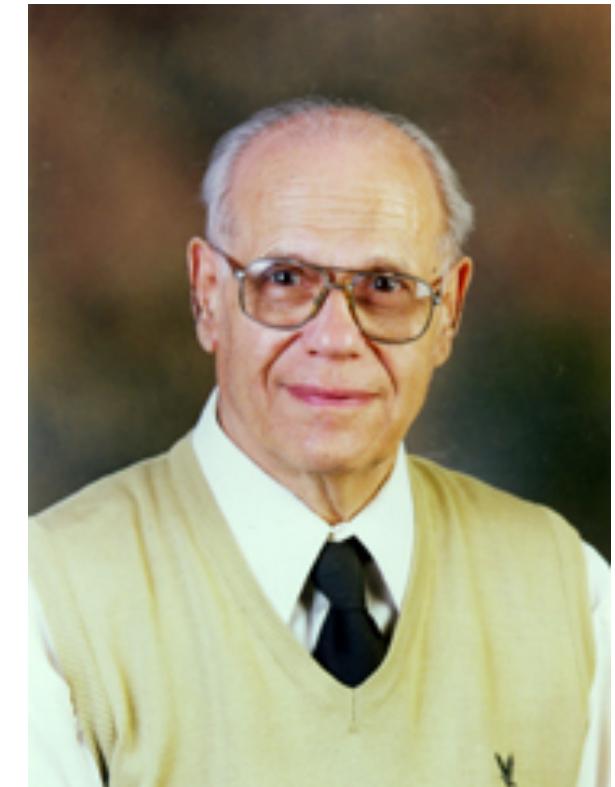


Automatic Speech Recognition Systems (ASRS): some history

- Initial work at Carnegie Mellon & IBM in the 1970's
 - Discrete density HMM (to model observations)
- Subsequent work @ Bell Labs
 - Continuous density HMM (GMM)
- Initially
 - Speaker *dependent* large vocabularies
 - Speaker *independent* small vocabularies
- In the mid-1990's
 - Large speaker independent vocabularies
 - DARPA funded several research programs (since 1971), e.g., the "1,000 words vocabulary challenge"
 - Worthy of note: the HTK Software Toolkit (Cambridge University)

The statistical HMM approach

- Main tool is the HMM
 - Learning is based on *forward-backward algo*
 - Also called the “**Baum-Welch algorithm**”
 - Named after **Leonard Esau-Baum**, a mathematician that developed HMM algorithms at the Institute for Defense Analyses (IDA), Princeton University, in the late 1960’s



Leonard E. Baum
(August 23, 1931 – August 14, 2017)

1980 – 1990: HMM become ubiquitous

HMMs become extensively used

- Lawrence R. Rabiner
 - writes a wonderful tutorial
 - pushes on ASR research using HMMs

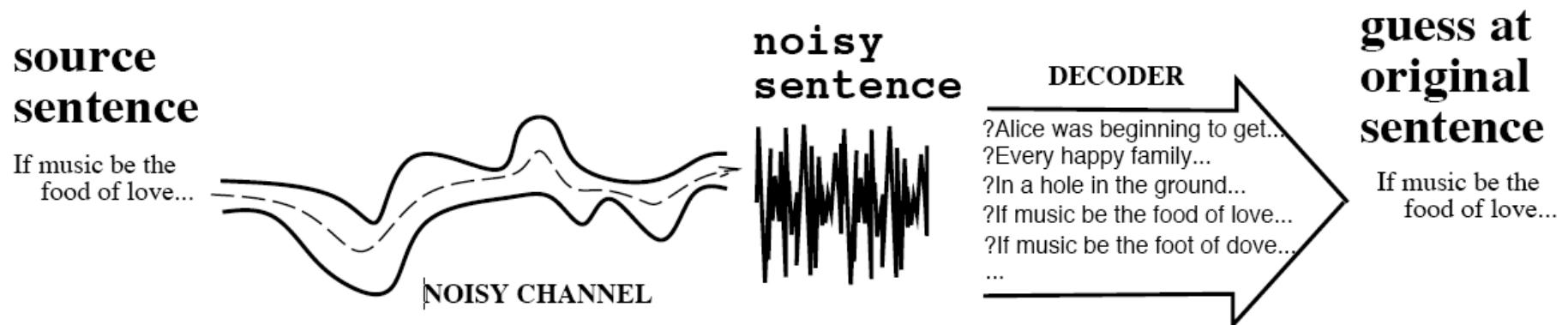
R. V. Cox, C. A. Kamm, L. R. Rabiner, J. Schroeter, and J. G. Wilpon, [Speech and Language Processing for Next-Millennium Communications Services](#), Proceedings of the IEEE, Vol. 88, No. 8, August 2000.

L. R. Rabiner, [A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition](#), Proceeding of the IEEE, Vol. 77, No. 2, February 1989.



Lawrence R. Rabiner
(born 28 September 1943)

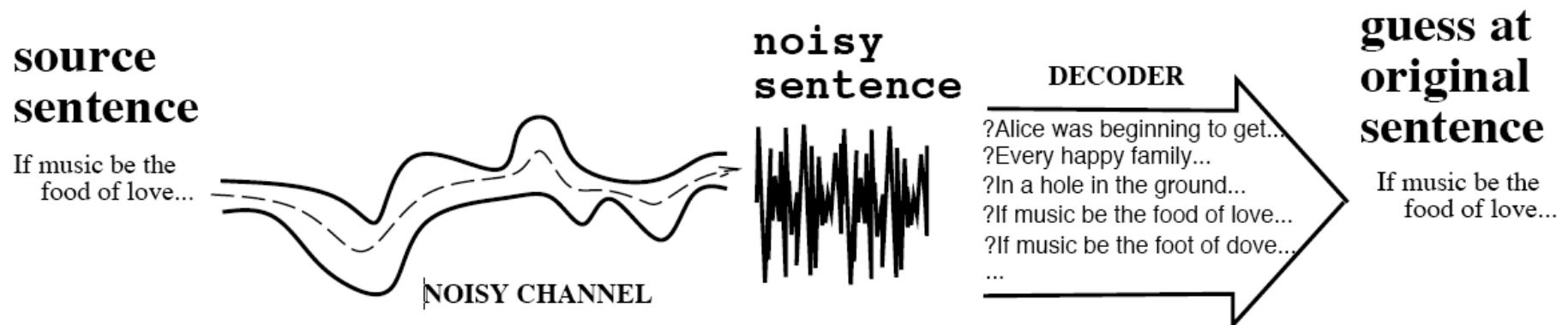
The noisy channel model



- **Idea:** treat the acoustic waveform as a “noisy” version of the string of words that was spoken, i.e., a string that has been passed through a *noisy communication channel*
- **Goal:** is to build a model of the noisy channel so that we can figure out how it modifies the true sentence and, in turn, retrieve it
- **Insight:** if we know how the channel distorts the source, we could find the true source sentence by:

(i) taking every possible sentence in the language

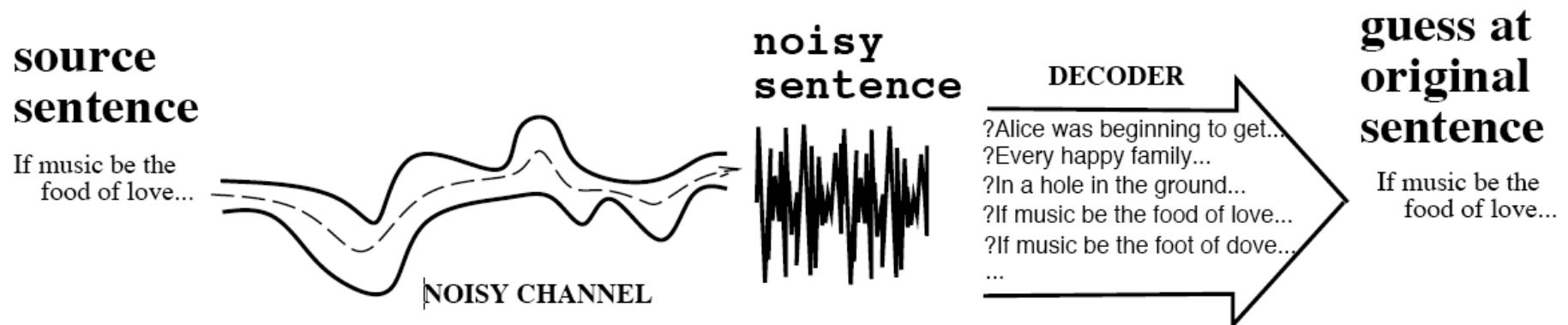
The noisy channel model



- **Idea:** treat the acoustic waveform as a “noisy” version of the string of words that was spoken, i.e., a string that has been passed through a *noisy communication channel*
- **Goal:** is to build a model of the noisy channel so that we can figure out how it modifies the true sentence and, in turn, retrieve it
- **Insight:** if we know how the channel distorts the source, we could find the true source sentence by:

(ii) running each sentence through our noisy channel model

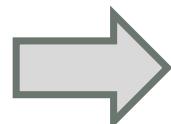
The noisy channel model



- **Idea:** treat the acoustic waveform as a “noisy” version of the string of words that was spoken, i.e., a string that has been passed through a *noisy communication channel*
- **Goal:** is to build a model of the noisy channel so that we can figure out how it modifies the true sentence and, in turn, retrieve it
- **Insight:** if we know how the channel distorts the source, we could find the true source sentence by:
 - (iii) seeing whether it matches the output: we then select the best matching source sentence as our desired source sentence

Decoding

- We need a complete metric for a “best match”
- **Problem:** speech is very variable
 - No acoustic model will provide an exact match for a given sentence
- **Solution:** use probabilistic models
 - Speech recognition as a special case of **Bayesian inference**
- **Goal:** to combine various probabilistic models to get a complete estimate for the probability of a noisy acoustic observation-sequence *given a candidate source sentence*. We can then **search** through the **space of all sentences**, and choose the one with the **highest probability**



DECONDING and SEARCH

ARS: the challenge

- 1) Record a speech waveform representing a sentence
- 2) Feature extraction: *input speech waveform* is converted into a number of *feature vectors* (e.g., 25ms each, spaced 10 ms apart)

$$Y_{1:T} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_T\}$$

feature vectors

What is the most likely sentence out of all sentences in the language \mathcal{L} given some acoustic input $Y_{1:T}$?

ARS (1/3)

- 3) Decoder: attempts to find the sequence of words

$$\boldsymbol{w}_{1:L} = \{w_1, w_2, \dots, w_L\}$$

which is most likely to have generated $\boldsymbol{Y}_{1:T}$
(subscripts dropped for the sake of notation's compactness)

$$\tilde{\boldsymbol{w}} = \operatorname{argmax}_{\boldsymbol{w} \in \mathcal{L}} P(\boldsymbol{w} | \boldsymbol{Y})$$

ARS (2/3)

$$\tilde{\mathbf{w}} = \operatorname{argmax}_{\mathbf{w} \in \mathcal{L}} P(\mathbf{w} | \mathbf{Y})$$

- This equation returns the **optimal sentence estimate**
- But it is **very difficult** to use, we need to make it operational:

$$\tilde{\mathbf{w}} = \operatorname{argmax}_{\mathbf{w} \in \mathcal{L}} P(\mathbf{w} | \mathbf{Y}) = \operatorname{argmax}_{\mathbf{w} \in \mathcal{L}} \frac{P(\mathbf{Y} | \mathbf{w}) P(\mathbf{w})}{P(\mathbf{Y})}$$

- **P(w)**: prob. of the word string, estimated by an **N-gram language model**
- **P(Y|w)**: can be easily estimated as well (**HMM**)
- **P(Y)**: very difficult to estimate, but we do not need it. As it is a constant term that appears for each w.

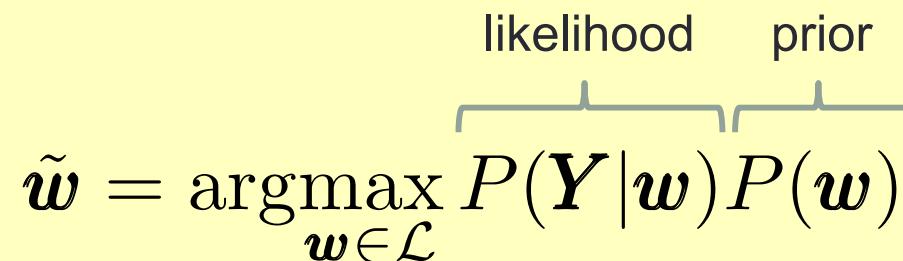


The search for a maximizer does not depend on it!

ARS (3/3)

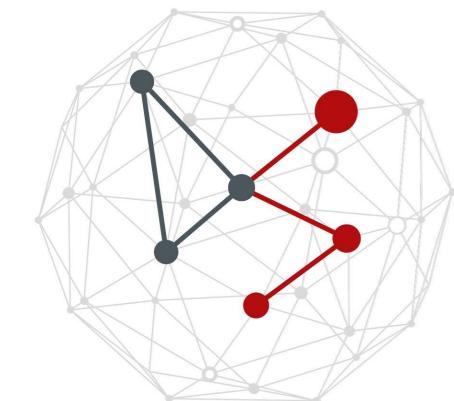
$$\tilde{\mathbf{w}} = \operatorname{argmax}_{\mathbf{w} \in \mathcal{L}} P(\mathbf{Y}|\mathbf{w})P(\mathbf{w})$$

likelihood prior



- The **prior $P(\mathbf{w})$** is computed through the **language model**
- The **likelihood $P(\mathbf{Y}|\mathbf{w})$** is computed by the **acoustic model (HMM)**

MODELING LANGUAGE AND LEXICON (PRONUNCIATION)



Language modeling

- Language Models (LMs)
 - capture *regularities* in spoken language
- Context-free grammars
 - Contain grammatical constraints that are hand-crafted
 - Are suitable for small to medium size dictionaries
- Large Vocabulary Continuous Speech Recognition (LVCSR)
 - Are all based on data driven approaches
 - Dependencies among words are automatically discovered
- N-gram model
 - The most popular approach

N-gram model (1/3)

$$P(\mathbf{w}_{1:L}) = \prod_{\ell=1}^L P(w_\ell | w_{\ell-1}, w_{\ell-2}, \dots, w_{\ell-N+1})$$

- Probability of word sequence $\mathbf{w}_{1:L}$
 - **Markovian assumption**
 - occurrence of each word depends on previous N-1 words
 - Word history (memory) reduced to previous N-1 words
 - Usually N=3 (tradeoff between *complexity & performance*)

N-gram model (2/3)

- Obtaining the model
 - Very large text corpus
 - To compute *N-gram transition probabilities* (usually N=3, “trigrams”)
 - Most steps are standards: automatic counting of words occurrences
 - Important choices
 - Choice of the vocabulary: minimizing *out-of-vocabulary* occurrences
 - Definition & treatment of words (e.g., compound words, acronyms)
 - Normalization
 - Reduce lexical variability (increases the coverage of fixed size vocabulary)
 - E.g., numerical expression are expanded to approximate spoken form
 - 150\$ -> *one hundred fifty dollars*
 - 2018 -> *twenty eighteen* or *two thousand and eighteen*
 - hundred <nb> -> replaced by *hundred and <nb>* 50% of the time
 - Semi-automatic processing to correct common text problems (misspelling)

N-gram model (3/3)

- Estimation of probabilities
 - Uses the Maximum Likelihood (ML) criterion
 - Estimated from the frequencies of text sequences of length N in the training corpus (texts or speech transcriptions)
- $P(w_i|w_{i-1}, w_{i-2}) \approx \frac{C(w_i, w_{i-1}, w_{i-2})}{C(w_{i-1}, w_{i-2})}$
 - C: counts how many times the tri(bi)grams appear in the training data
- Many additional implementation details are needed
 - Back-off: estimate probabilities for rare N-grams (not present in the training). Probs. of N-grams estimated from probs. on (N-1)-grams
 - Model interpolation: often needed; to merge language models obtained from heterogeneous (in size, context, format) training corpora

Pronunciation modeling

- Pronunciation dictionary
 - Links acoustic level representation and lexical items output by the speech decoder
 - Associated with each lexical entry (“word”)
 - One or more pronunciations
 - Described using elementary units (“phones” or “phonemes”)
 - For example word “bat” has three phones \b\ \ae\ \t\
 - Standard phones
 - 45 for English
 - 50 for Italian and German
 - 35 for French and Mandarin
 - 25 for Spanish
- For any given word w
 - The corresponding acoustic model is synthesized by concatenating phone models, as defined by a pronunciation dictionary

Open source pronunciation dictionary

- The MCU pronunciation dictionary
 - <http://www.speech.cs.cmu.edu/cgi-bin/cmudict>

Input: “POLLY WANTS A CRACKER”

(Kurt Cobain, 1991)

Output: P AA L IY . W AA N T S . AH . K R AE K ER

(input sentence broken down into phones)

English phones

- The 45 phones for English
 - Illustrative word examples are shown for each phone
 - Portion corresponding to the phone sound is underlined

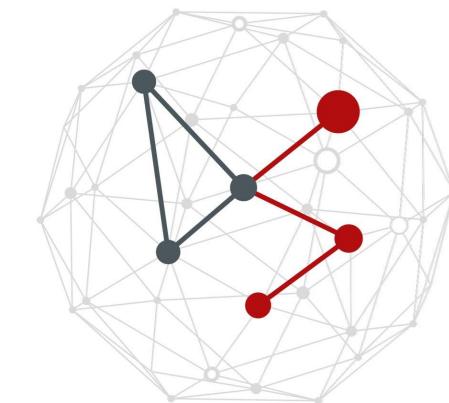
Phone	Example	Phone	Example
Vowels		Fricatives	
i	be <u>e</u> t	s	<u>s</u> ue
I	bi <u>t</u>	z	<u>z</u> oo
e	ba <u>it</u>	ʃ	<u>sh</u> oe
ɛ	b <u>E</u> t	ʒ	<u>meas</u> ure
æ	ba <u>t</u>	f	<u>f</u> an
ʌ	bu <u>t</u>	v	<u>v</u> an
ɑ	bo <u>t</u> t	θ	<u>th</u> in
ɒ	bo <u>at</u>	Plosives	
ʊ	bo <u>o</u> t	b	<u>be</u> t
ɔ:	bo <u>o</u> k	d	<u>de</u> bт
ɜ:	bi <u>rd</u>	g	<u>ge</u> t
Diphthongs		p	<u>pe</u> t
ɑ: ^j	bi <u>te</u>	t	<u>ta</u> t
ɔ: ^j	bo <u>y</u>	k	<u>ca</u> t
ɑ: ^w	bo <u>u</u> t	Affricates	
Reduced Vowels		tʃ	<u>chea</u> p
ə	x <u>bout</u>	dʒ	<u>jeep</u>
ɪ	da <u>ted</u>	Nasals	
ɔ:	bu <u>tter</u>	m	<u>met</u>
Semivowels		n	<u>net</u>
l	le <u>d</u>	ŋ	<u>thing</u>
r	re <u>d</u>	Syllabics	
w	we <u>d</u>	m	bott <u>om</u>
y	ye <u>t</u>	n	button <u>u</u> n
h	ha <u>t</u>	l	bottle <u>le</u>

Pronunciation (lexicon)

COUPON	kupən (0.63) kyupən (0.37)
ORGANIZATION	ɔrgənɪzeʃən (0.93) ɔrgəna ^j zeʃən (0.07)
HUNDRED	hʌndr̩d (0.44) hʌndr̩d (0.34)
	hʌn ³ d (0.18) hʌnr̩d (0.04)
MODERATE	mədət̩ (0.82) mədət̩ (0.18)
TO	tə (0.66) tu (0.34)
I_DON'T_KNOW	a ^j donno (0.57) a ^j dontno (0.05)
	a ^j dʌno (0.28) a ^j dno (0.10)
DON'T_KNOW	donno (0.73) dontno (0.18)
	dʌno (0.09)
DID_YOU	dɪdyu (0.65)
	dɪdʒə (0.30) dɪdyə (0.05)
GOING_TO	goɪŋt̩ə (0.13) goɪŋtu (0.09)
	gʌnə (0.70) gcnə (0.08)

Some examples of [lexical entries and their pronunciations](#) along with [estimated probabilities](#). For compound words, the original concatenated pronunciation is given in the 1st line, the reduced forms are given in the second line

THE ACOUSTIC MODEL



The HMM acoustic model (1/3)

- Each word w
 - Is decomposed into a sequence of K_w basic sounds called *phones*
 - This sequence is called pronunciation:
- To allow for the possibility of multiple pronunciations

$$P(Y|\mathbf{w}) = \sum_{\mathbf{Q}} P(Y|\mathbf{Q}) P(\mathbf{Q}|\mathbf{w}) \quad \text{where: } \mathbf{w} = \mathbf{w}_{1:L}$$
$$\mathbf{w}_{1:L} = \{w_1, w_2, \dots, w_L\}$$

- where \mathbf{Q} is a particular sequence of pronunciations
- formed by concatenating the constituent base phones

$$\mathbf{Q} = \{\mathbf{q}^{(w_1)}, \mathbf{q}^{(w_2)}, \dots, \mathbf{q}^{(w_L)}\}$$

The HMM acoustic model (1/3)

- Each word w
 - Is decomposed into a sequence of K_w basic sounds called *phones*
 - This sequence is called **pronunciation**: $\mathbf{q}_{1:K_w}^{(w)} = q_1 \ q_2 \ \dots \ q_{K_w}$
- To allow for the possibility of multiple pronunciations

$$P(\mathbf{Y}|\mathbf{w}) = \sum_{\mathbf{Q}} P(\mathbf{Y}|\mathbf{Q})P(\mathbf{Q}|\mathbf{w}) \text{ where: } \mathbf{w} = \mathbf{w}_{1:L}$$

- For $P(\mathbf{Q} | \mathbf{w})$ we have:

$$P(\mathbf{Q}|\mathbf{w}) = \prod_{\ell=1}^L P(\mathbf{q}^{w_\ell} | w_\ell)$$

The HMM acoustic model (2/3)

- Given the composite HMM \mathbf{Q}
 - formed by concatenating the constituent base phones

$$\mathbf{Q} = \{\mathbf{q}^{(w_1)}, \mathbf{q}^{(w_2)}, \dots, \mathbf{q}^{(w_L)}\}$$

- then, the acoustic likelihood is given by:

$$P(\mathbf{Y}|\mathbf{Q}) = \sum_{\boldsymbol{\theta}} P(\boldsymbol{\theta}, \mathbf{Y}|\mathbf{Q})$$

- where $\boldsymbol{\theta}$ is a sequence of HMM (hidden) states (represent sub-phones)

$$P(\boldsymbol{\theta}, \mathbf{Y}|\mathbf{Q}) = a_{\theta_0, \theta_1} \prod_{t=1}^T b_{\theta_t}(\mathbf{y}_t) a_{\theta_t, \theta_{t+1}}$$

- θ_0 and θ_{T+1} are the non-emitting entry and exit states

The HMM acoustic model (2/3)

- Given the composite HMM \mathbf{Q}
 - formed by concatenating the constituent base phones

$$\mathbf{Q} = \{\mathbf{q}^{(w_1)}, \mathbf{q}^{(w_2)}, \dots, \mathbf{q}^{(w_L)}\}$$

$T \rightarrow$ number of phones in current sentence, $t \in \{1, \dots, T\}$

$\theta_t \rightarrow$ hidden HMM states

$\mathbf{y}_t \rightarrow$ speech observation associated with phone t

$b_{\theta_t}(\mathbf{y}_t) \rightarrow$ emission probability, to generate \mathbf{y}_t if HMM state is θ_t

$$P(\boldsymbol{\theta}, \mathbf{Y} | \mathbf{Q}) = a_{\theta_0, \theta_1} \prod_{t=1}^T b_{\theta_t}(\mathbf{y}_t) a_{\theta_t, \theta_{t+1}}$$

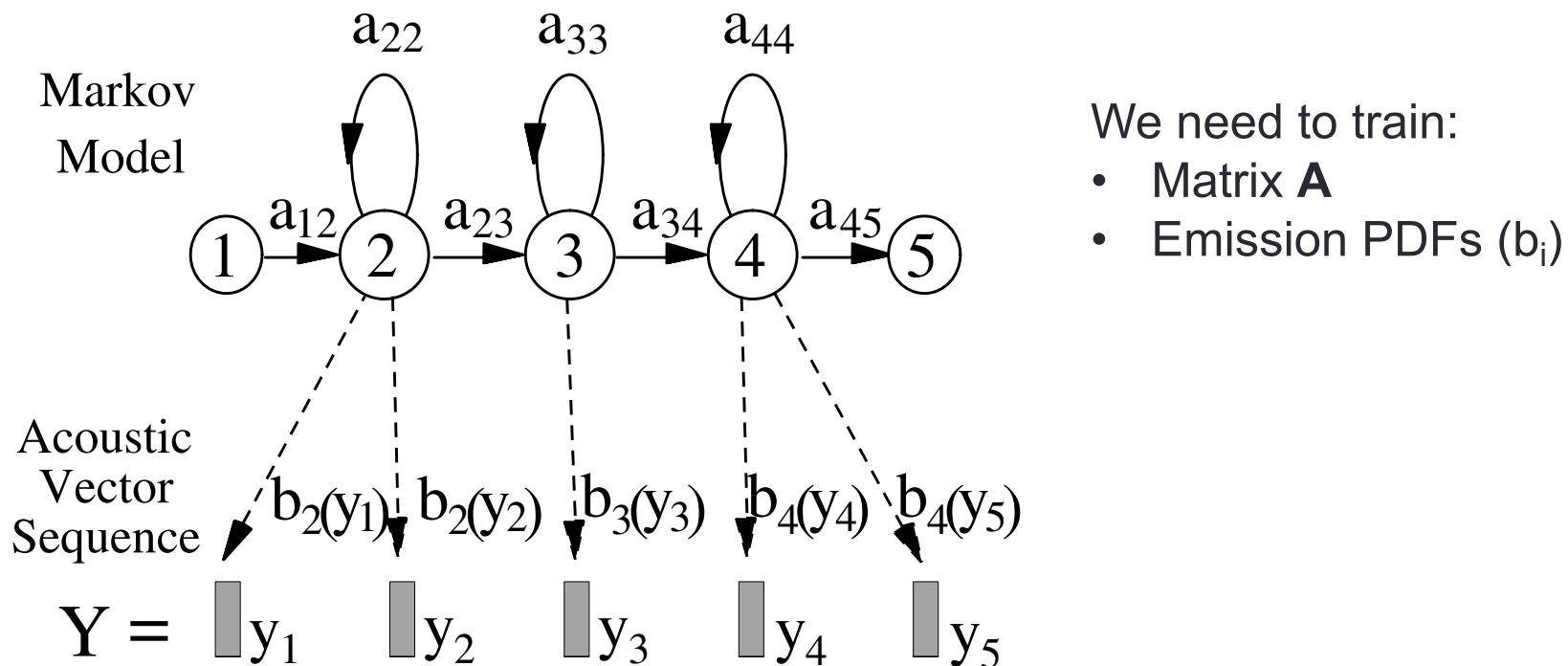
- θ_0 and θ_{T+1} are the non-emitting entry and exit states

The HMM acoustic model (3/3)

- The previous equations formally define the max. problem
- Amounts to finding the seq. of words (phones & sub-phones)
 - That maximizes the overall probability of the observations \mathbf{Y}
- **Question:** how do we implement this search efficiently?
 - Defining an HMM of concatenated phones
 - Applying Viterbi decoding
 - Viterbi returns the most likely path (across phones) given \mathbf{Y}
- In the next slides we see this through an example
 - Digit recognition

HMM-based phone model

- For each phone a continuous density HMM is trained
 - Three sub-phones (HMM states) for each phone model
 - $b_j(y_j)$: are the emission PDFs (Gaussian mixture model)



Components of the HMM acoustic model

$$\theta = \{\theta_1, \theta_2, \dots, \theta_N\}$$

A set of **HMM states** corresponding to **sub-phones**

$$A = [a_{ij}]$$

Transition probability matrix from previous state i to current state j. θ and A implement a **pronunciation lexicon**, a state graph structure for each word that the system is capable of recognizing

$$B = [b_i(y_n)] = [p(y_n | \theta_n = i)]$$

A set of **observation likelihoods B**: also called **emission probabilities**, each expressing the probability of a cepstral feature vector (observation y_n) being generated given that **sub-phone state (HMM state) is i**

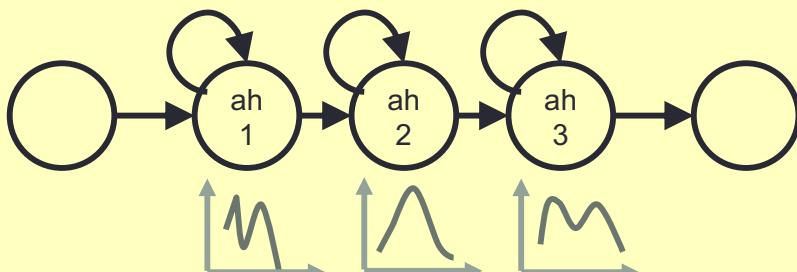
Example: digit recognition problem (1/5)

- 11 words: digits 0 to 9 plus “oh”
- Assume we have a pre-trained HMM for each phone
- Pronunciation dictionary (lexicon): maps words onto phones

Lexicon (11 words, 0-9 + “oh”)

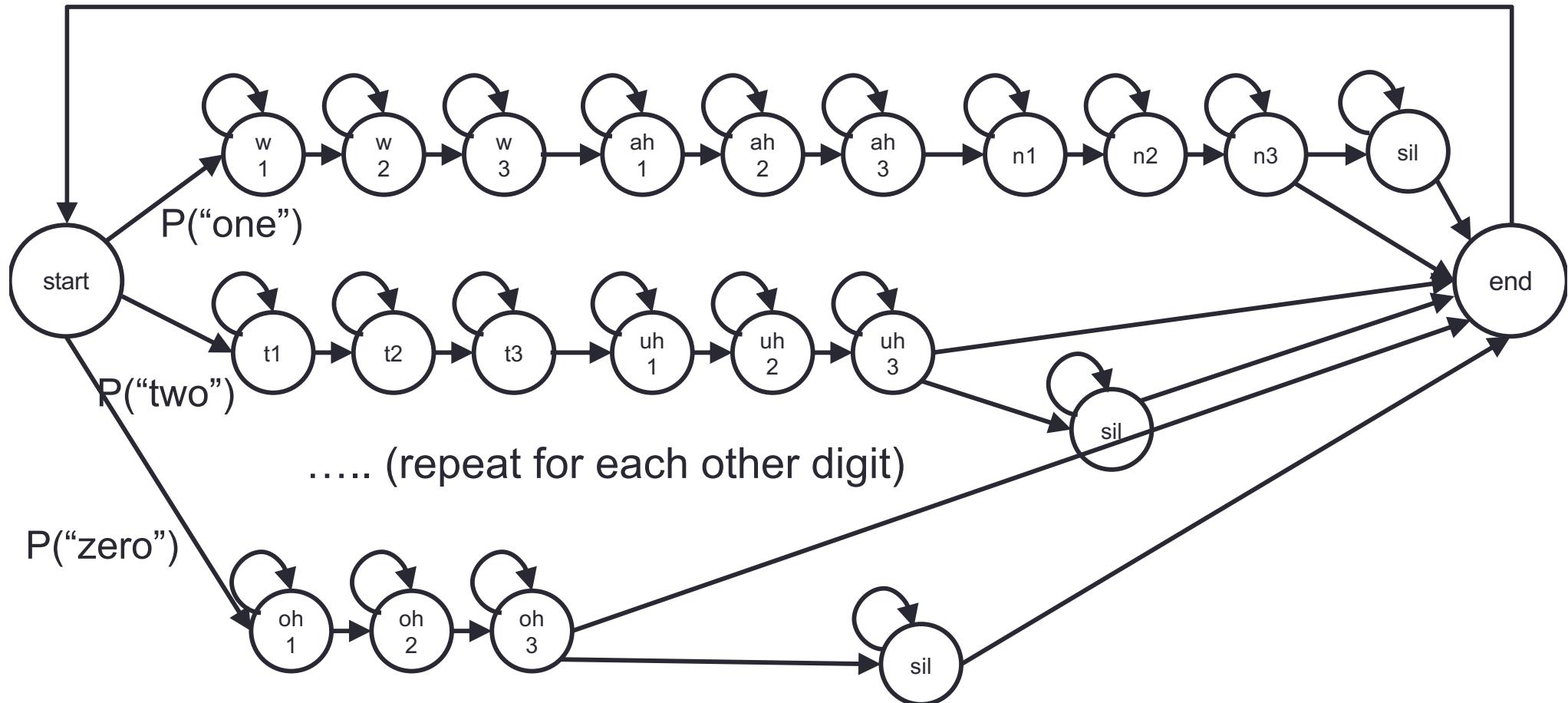
one	w ah n
two	t uw
three	th r iy
four	f ao r
five	f ay v
six	s ih k s
seven	s eh v ax n
eight	ey t
nine	n ay n
zero	z iy r ow
oh	ow

pre-trained phone model for “ah”



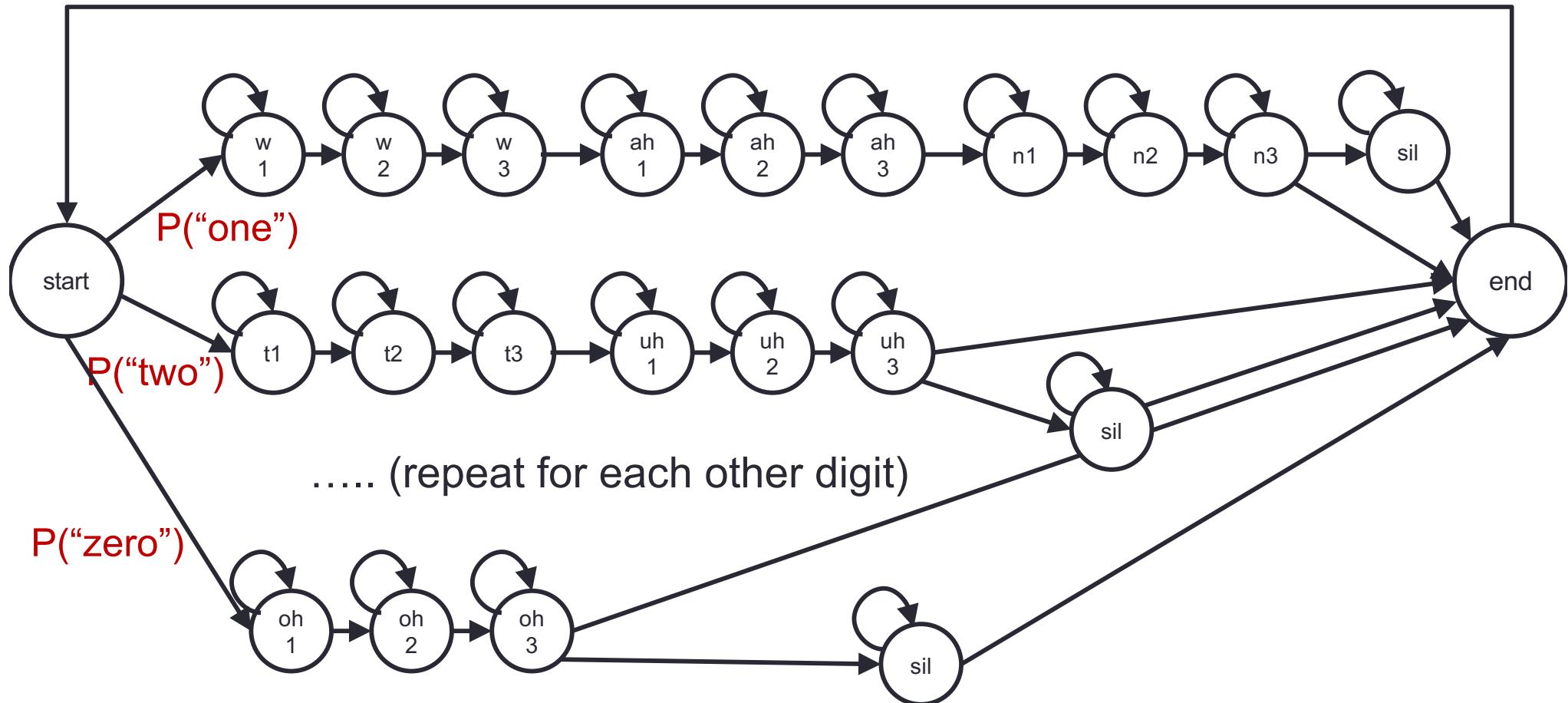
Matrix **A** (transitions) and matrix **B** (emissions) are pre-trained from labeled data using **Baum-Welch** algo

Digit recognition problem (2/5)



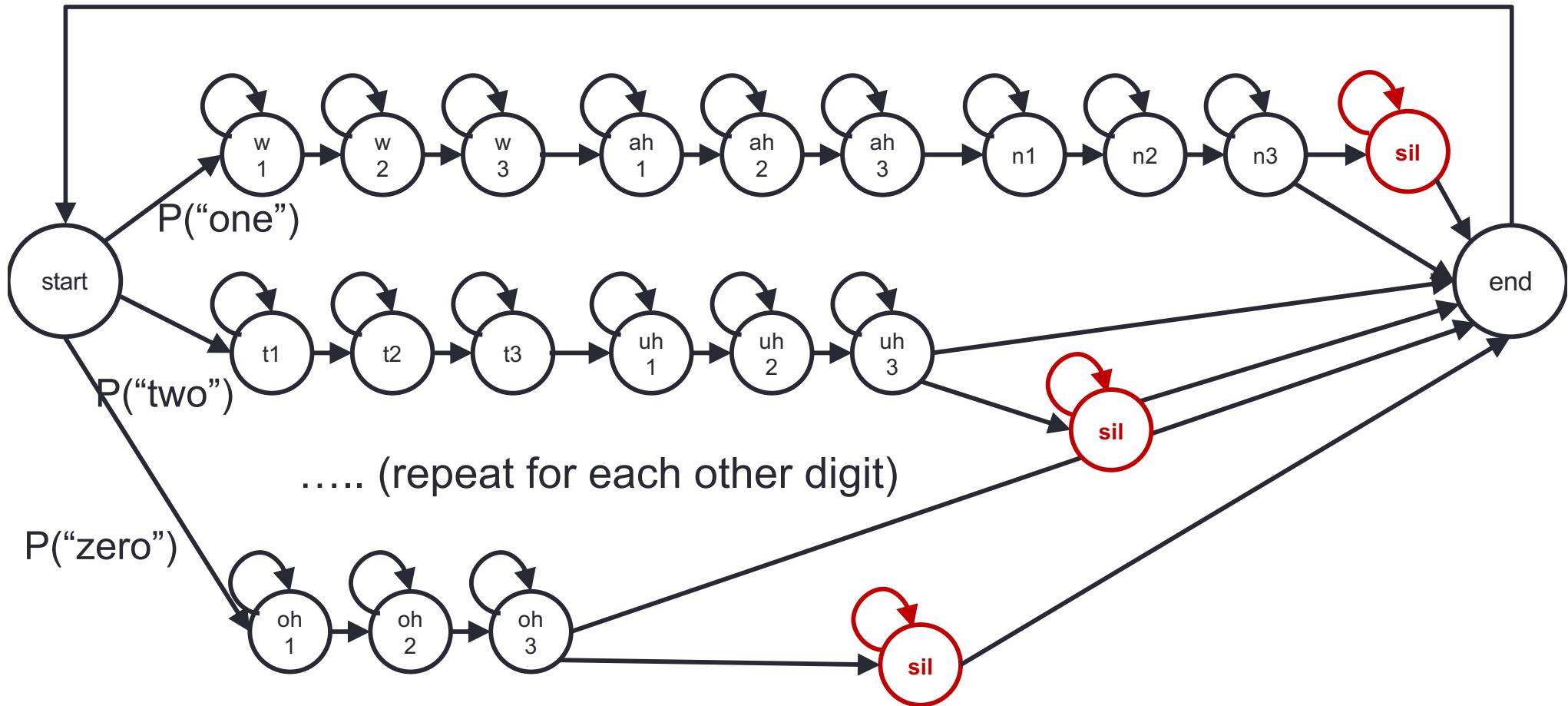
An HMM for the digit recognition task. A lexicon specifies the phone sequence (e.g., "w" "ah" "n"), each phone is composed of **three subphones**, each with GMM emission. Combining these and adding an optional silence at the end of each word, results in a single HMM for the whole task. The transition from the End state to the Start state to allow digit sequences of any length.

Digit recognition problem (3/5)



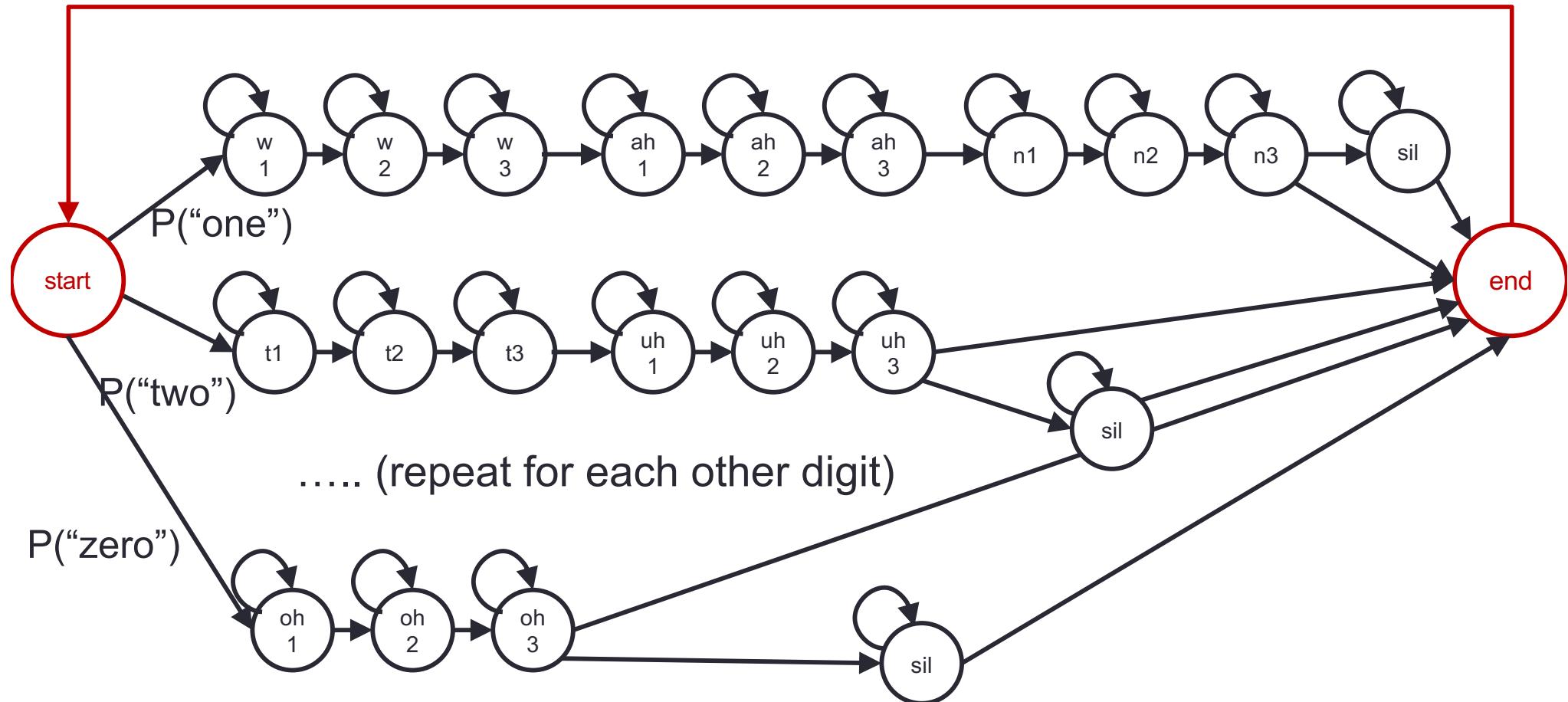
Trained **A** matrix and **B** matrices (GMMs) are used to model each phone, whereas connection among words are modeled using a **language model (prior)** → probabilities that the next digit is a “**one**”, “**two**”, etc. (these are conditional probabilities). This means that the transition matrices are concatenated and extended using the language model

Digit recognition problem (4/5)



The model is pre-trained → optimal decoding is achieved applying the **Viterbi decoder** on the trellis that is implied by this diagram, moving from the first to the last audio frame. Note that word segmentation / silence detection is automatic (it is part of the decoding as silence states "sil" are accounted for)

Digit recognition problem (5/5)

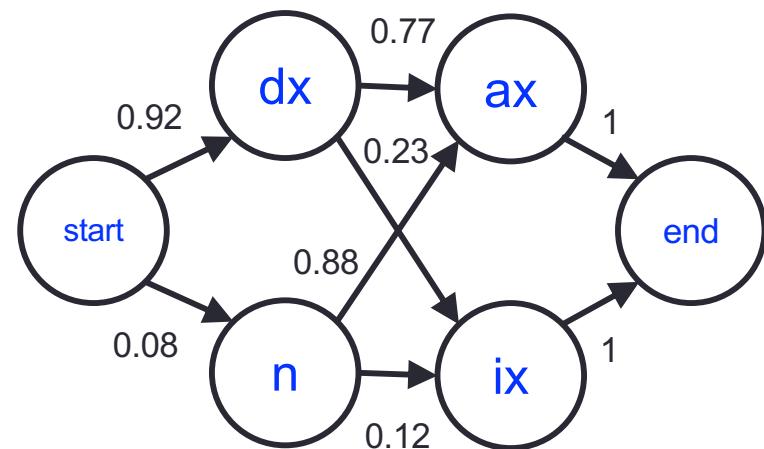


Finally, since we connect END to START

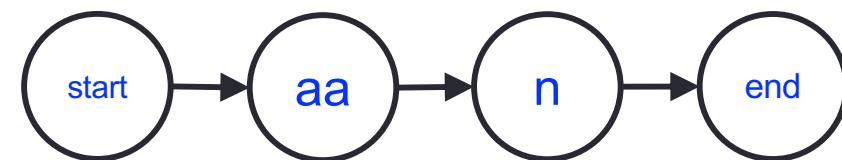
- we can decode sequences of digits of arbitrary length

Accounting for pronunciation

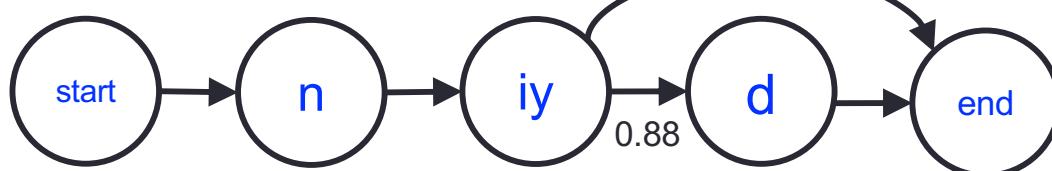
- The simplest way of representing words is concatenating phones
- But, word models can be complicated **without changing the approach**
- When a word can be pronounced in several manners, the model can be extended, **through a slightly more complex Markov chain**, e.g.:



Word model for “the”



Word model for “on”

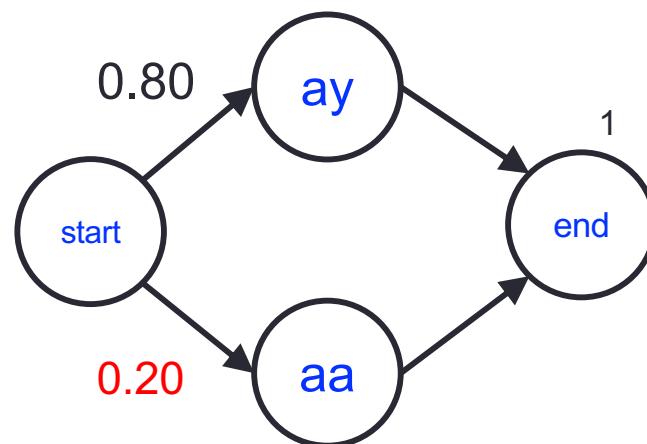


Word model for “need”

Thom Yorke - “You’re all I need”
0.12

Accounting for pronunciation

- The simplest way of representing words is concatenating phones
- But, word models can be complicated without changing the approach
- When a word can be pronounced in several manners, the model can be extended, **through a slightly more complex Markov chain**, e.g.:



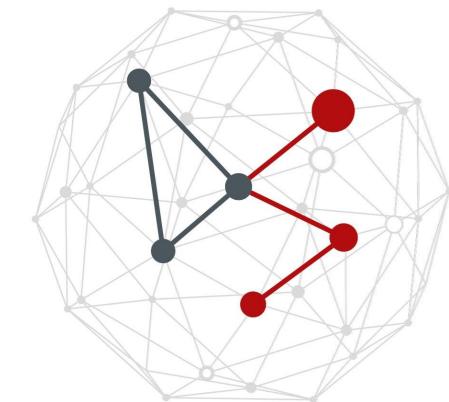
Word model for “I”

Breed – Nirvana
| don't care X5
Care if it's old
| don't mind X4
Mind, don't have a mind
....

Decoding general sentences

- It is conceptually equivalent to the digit recognition problem
- Differences are:
 - Many more words
 - a more complex language model
 - each word encodes multiple pronunciations
 - more conditional probabilities for words
(prob. of word that follow given previous word)
- This leads to a much larger HMM decoding diagram
 - the trellis to be used for decoding is much larger too

DECODING EXAMPLE



Viterbi trellis example

- **Example:** consider the word “five”
 - 3 phones: “f” “ay” “v”
 - We exemplify Viterbi decoding over a sequence of 5 feature vectors
$$Y_{1:5} = \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4, \mathbf{y}_5\}$$
- **For simplicity:**
 - a single state is considered for each phone (instead of 3 sub-phones)
 - \mathbf{y}_n represents the feature vector at time n (one per audio frame)
 - Given an observation \mathbf{y}_n , the probability that it is measured (observed), given that the HMM is in the hidden (phone) state i is:

$$b_i(\mathbf{y}_n) = p(\mathbf{y}_n | \theta_n = i)$$

- These posterior PDFs are trained through EM (see later)

Viterbi trellis example

Key variables

$$\nu_{n-1}(j)$$

Viterbi path probability for state j from previous time step n-1
(remember: implementations use logarithms)

$$\nu_n(j) = \max_{i=1, \dots, K} [\nu_{n-1}(i) a_{ij} b_j(\mathbf{y}_n)]$$

$$A = [a_{ij}]$$

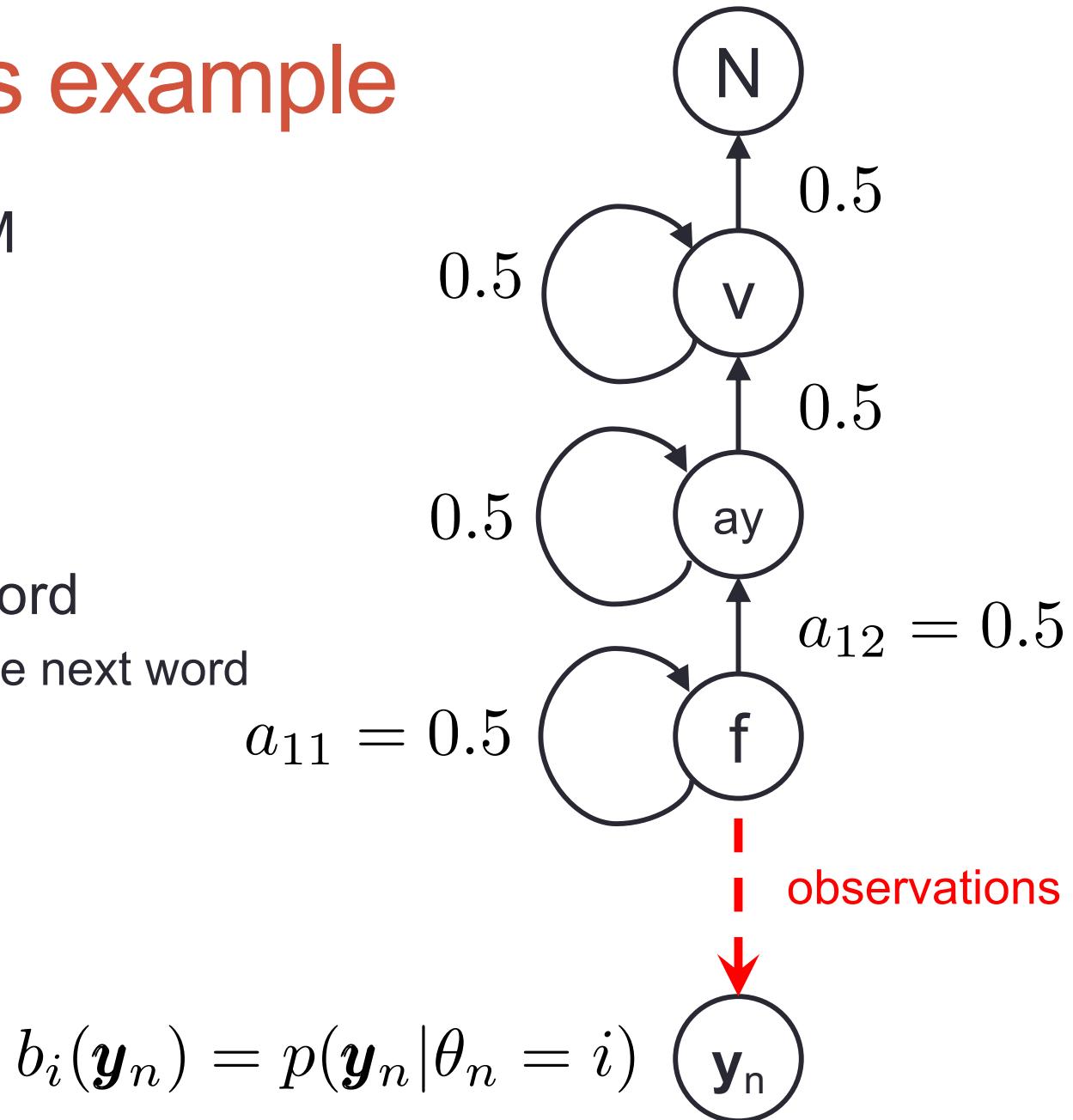
Transition probability matrix from previous state i to current state j

$$b_i(\mathbf{y}_n) = p(\mathbf{y}_n | \theta_n = i)$$

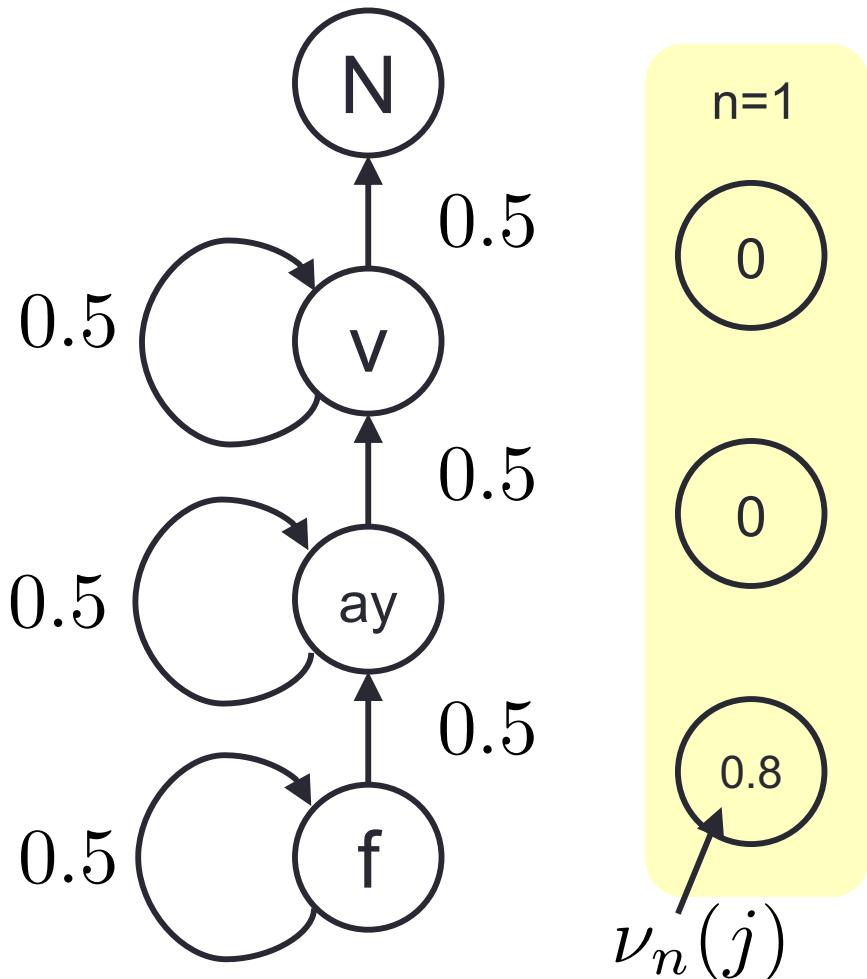
Emission probabilities (\mathbf{y}_n is feature vector at time n, j is hidden sub-phone state)

Viterbi trellis example

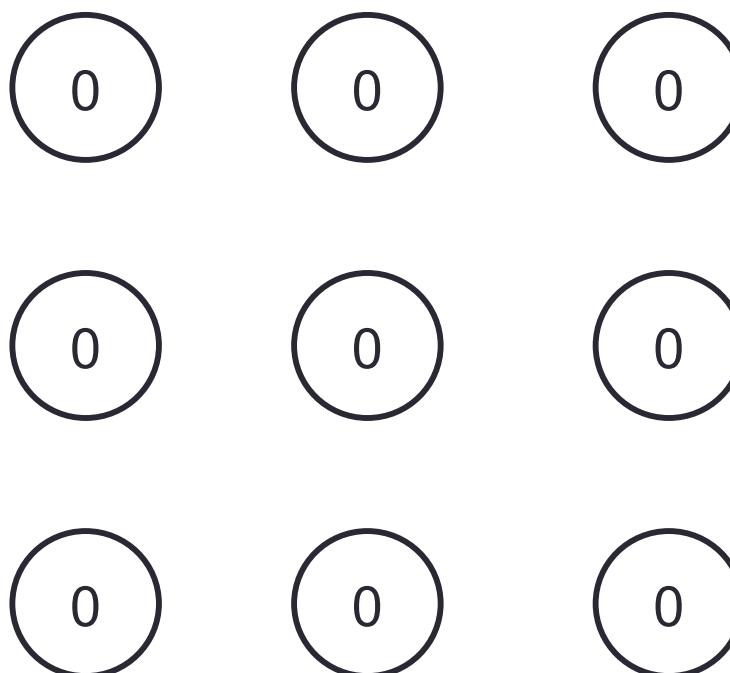
- The trained HMM
 - State 1 = “f”
 - State 2 = “ay”
 - State 3 = “v”
- **State N** = next word
 - Proceed to decode next word



Viterbi trellis decoding



$$\boldsymbol{\pi} = [\pi_f \ \pi_{ay} \ \pi_v]^T = [1 \ 0 \ 0]^T$$

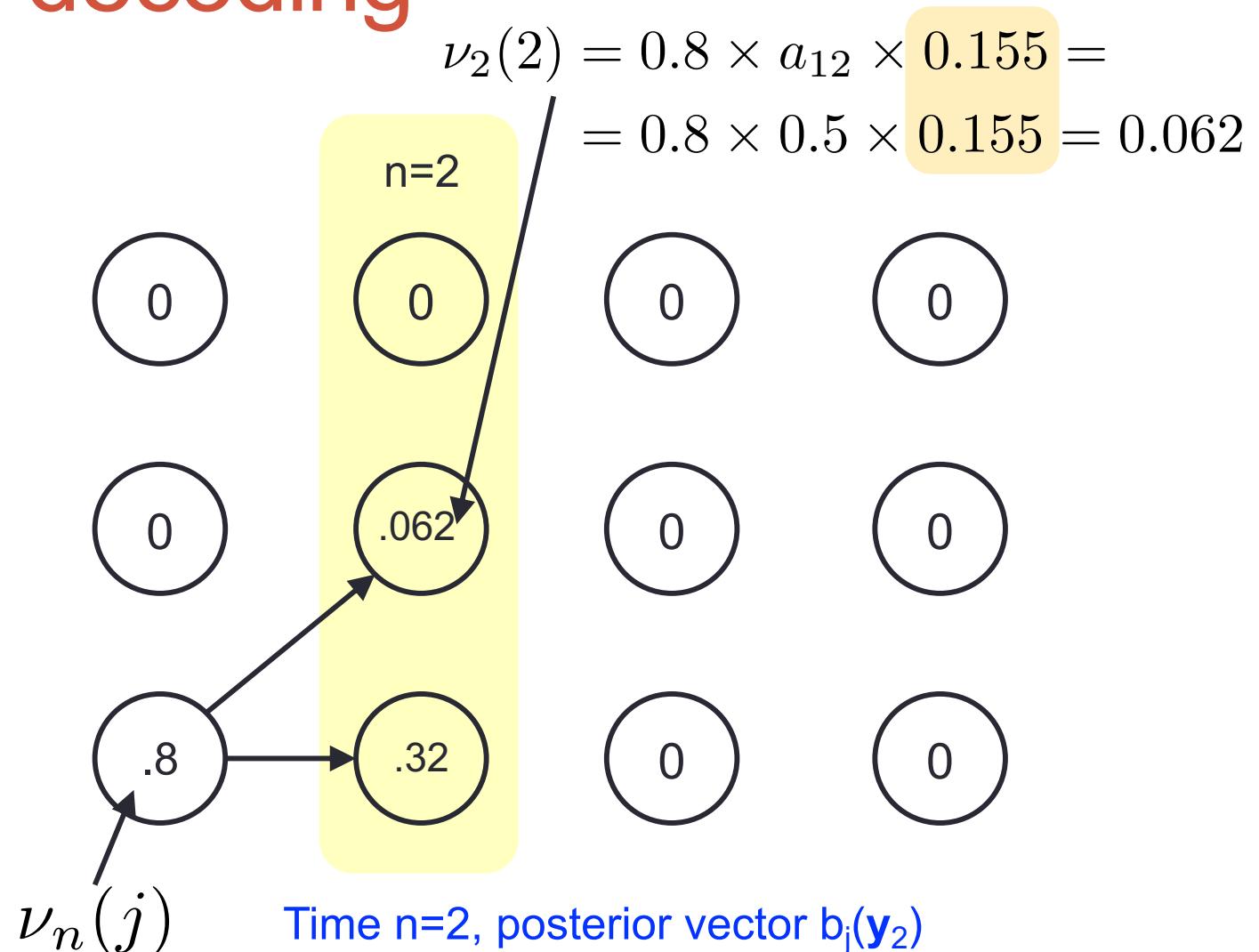
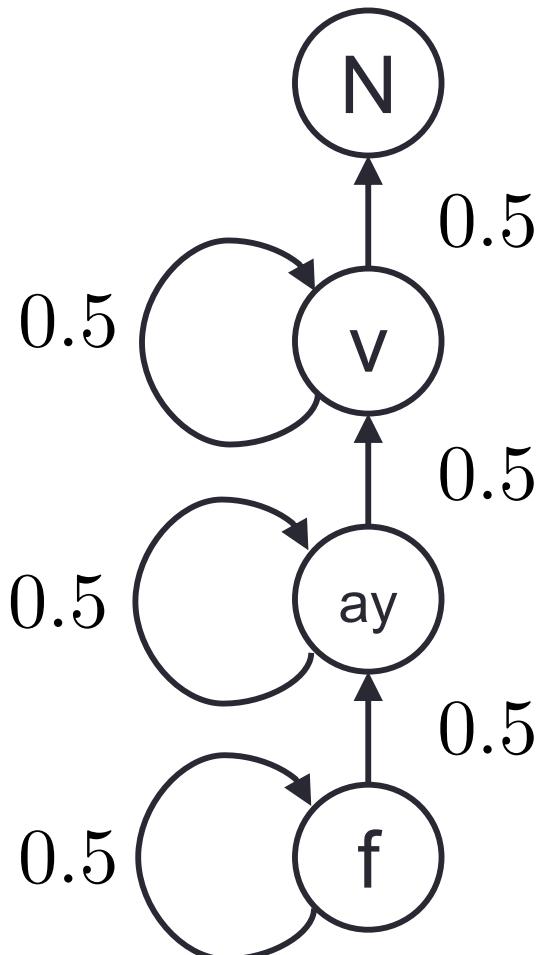


Time $n=1$, posterior vector $b_j(\mathbf{y}_1)$

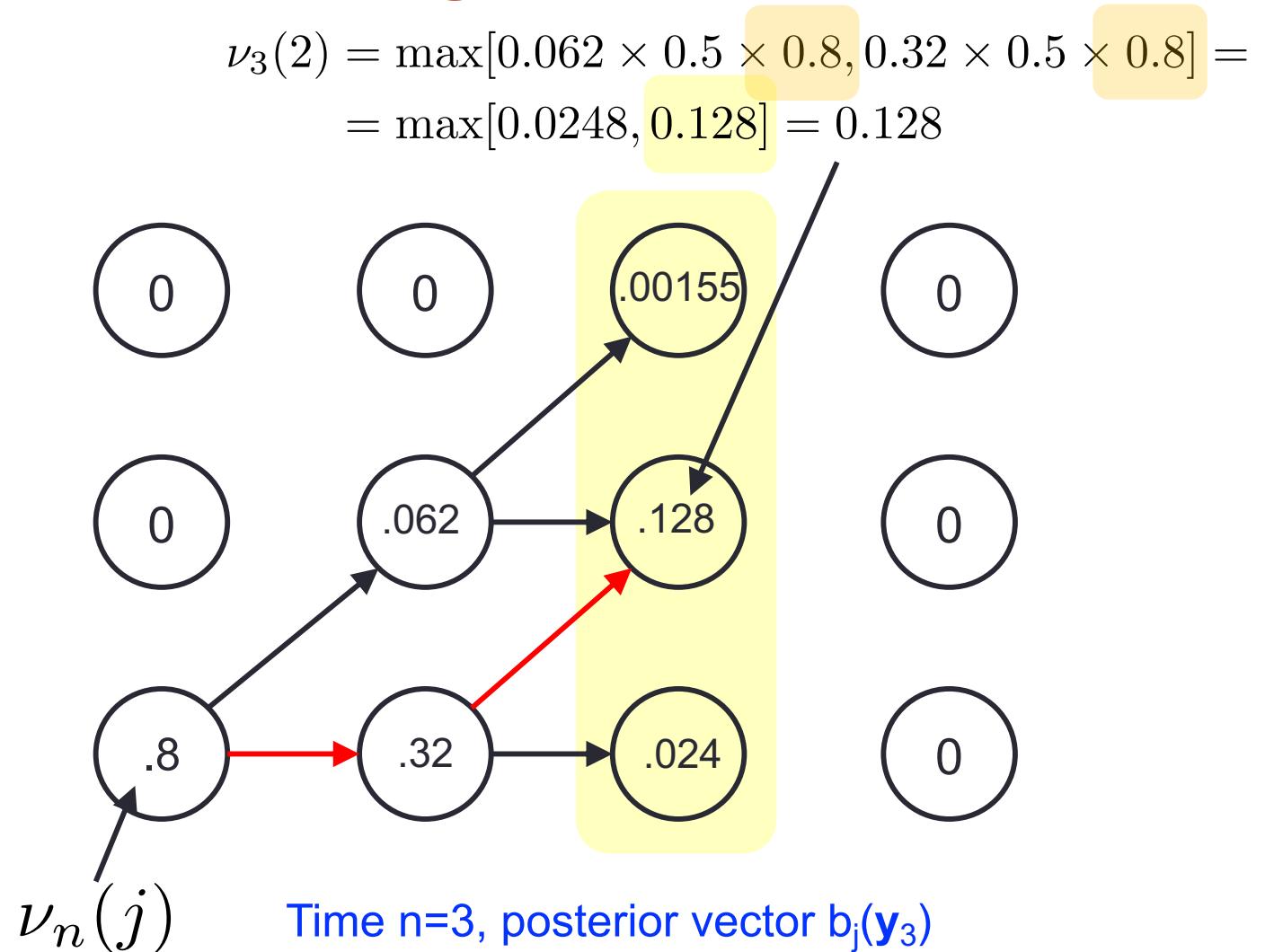
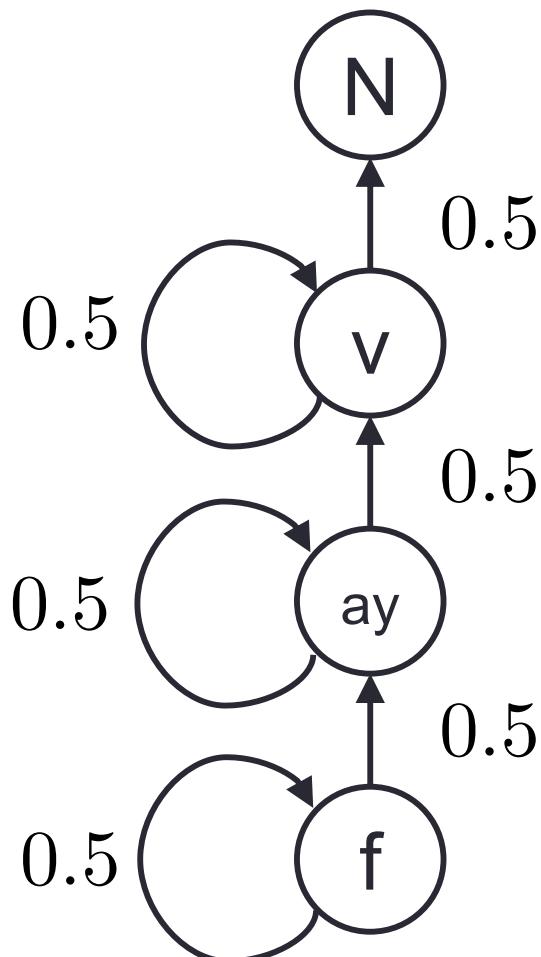
$$\mathbf{b} = [0.8 \ 0.155 \ 0.045]^T$$

Prob. feature vector is \mathbf{y}_1 given that state is 1 (means "f")

Viterbi trellis decoding

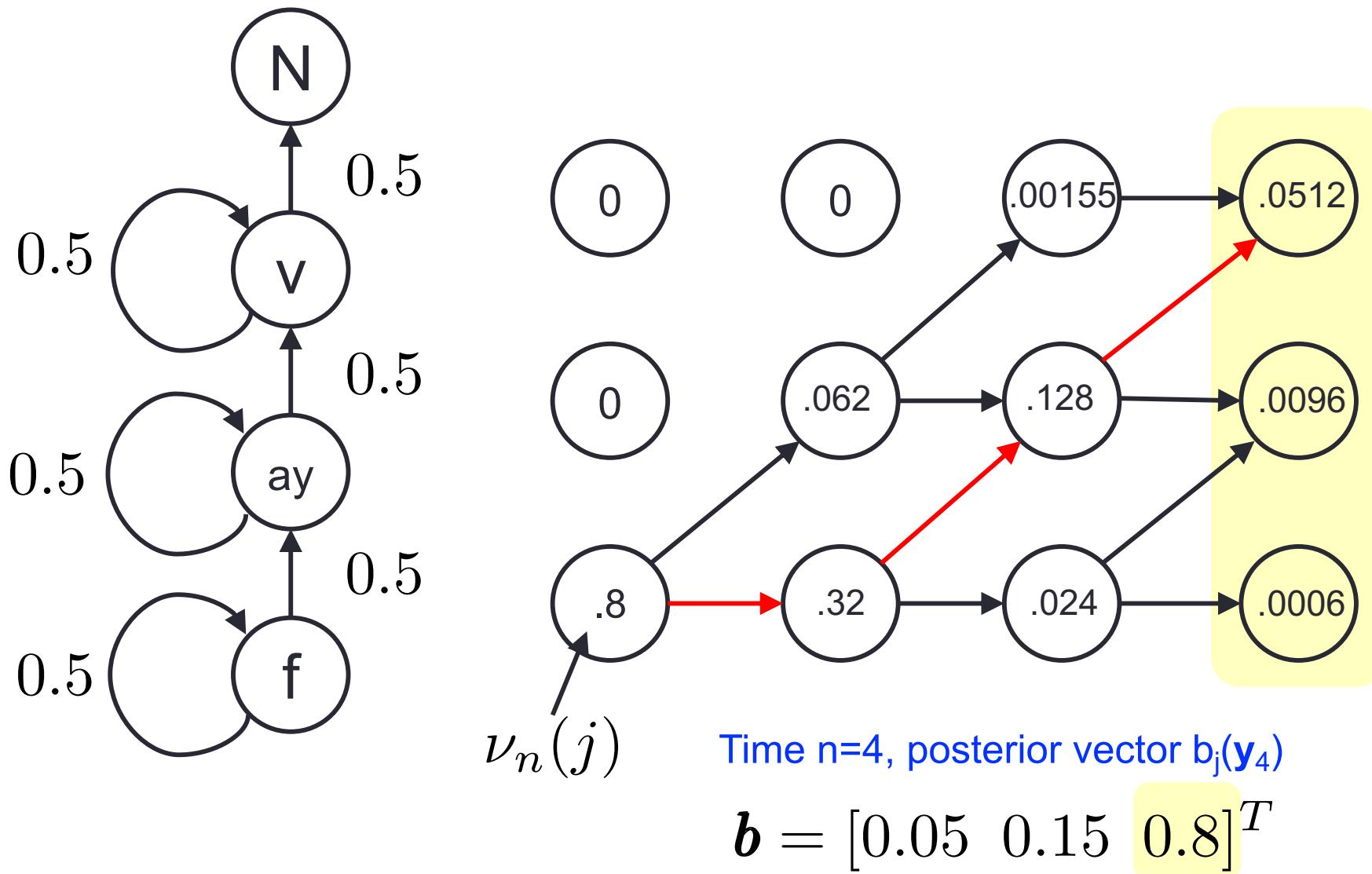


Viterbi trellis decoding



$$\mathbf{b} = [0.15 \quad 0.8 \quad 0.05]^T$$

Viterbi trellis decoding



Finally, assume next words can be “one” or “two”
 We concatenate to next possible words:

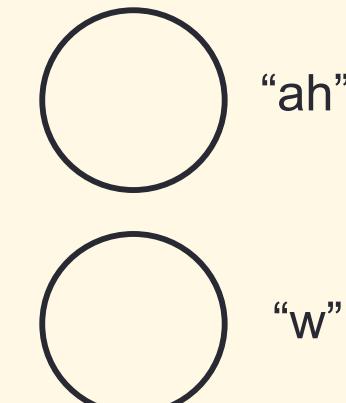
$$P(\text{"one"}|\text{"five"}) = P_{51} = 0.6, P(\text{"two"}|\text{"five"}) = P_{52} = 0.4$$

$$\mathbf{b} = [0.01 \ 0.05 \ 0.15 \ 0.2 \ 0.1 \ 0.6 \ 0.09]^T$$

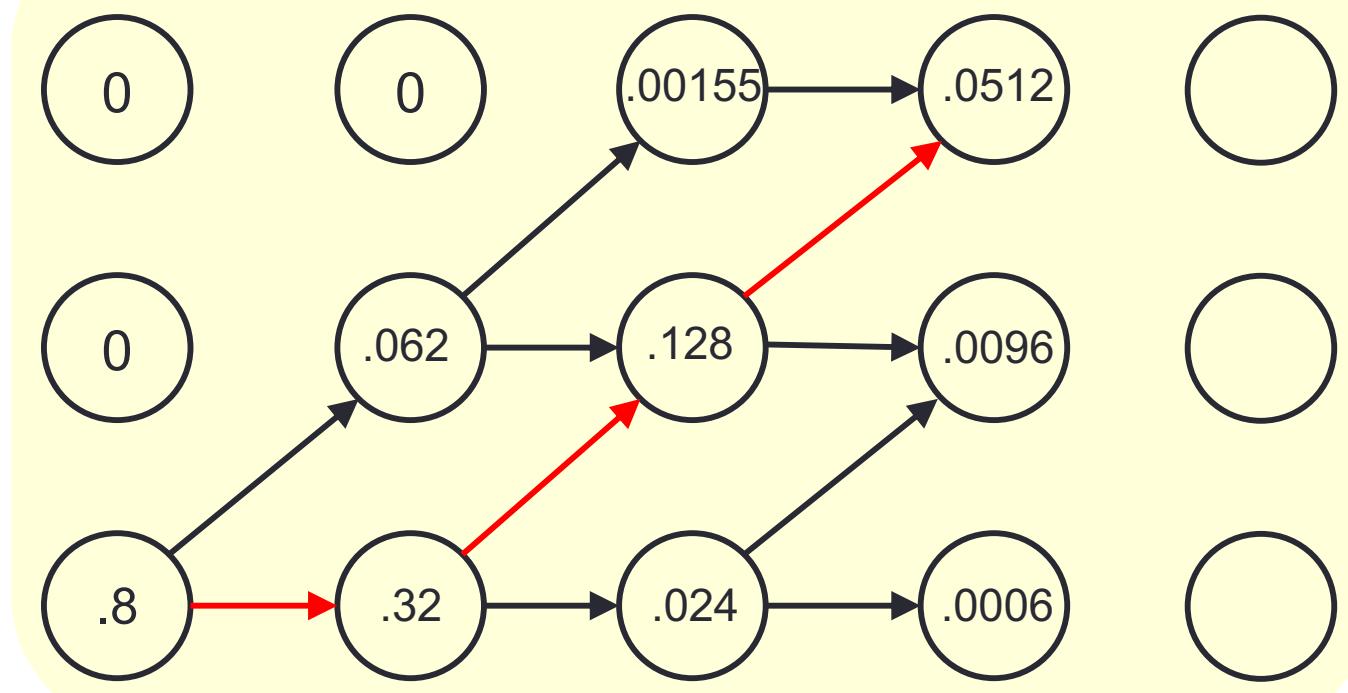
“f” “ay” “v” “w” “ah” “t” “uw”

Trellis for one = “w” “ah” “n”

.....



Trellis for word five



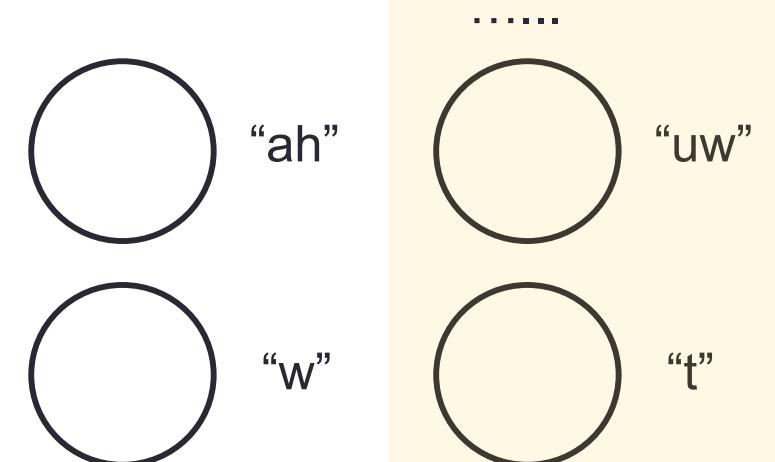
Finally, assume next words can be “one” or “two”
 We concatenate to next possible words:

$$P(\text{"one"} | \text{"five"}) = P_{51} = 0.6, P(\text{"two"} | \text{"five"}) = P_{52} = 0.4$$

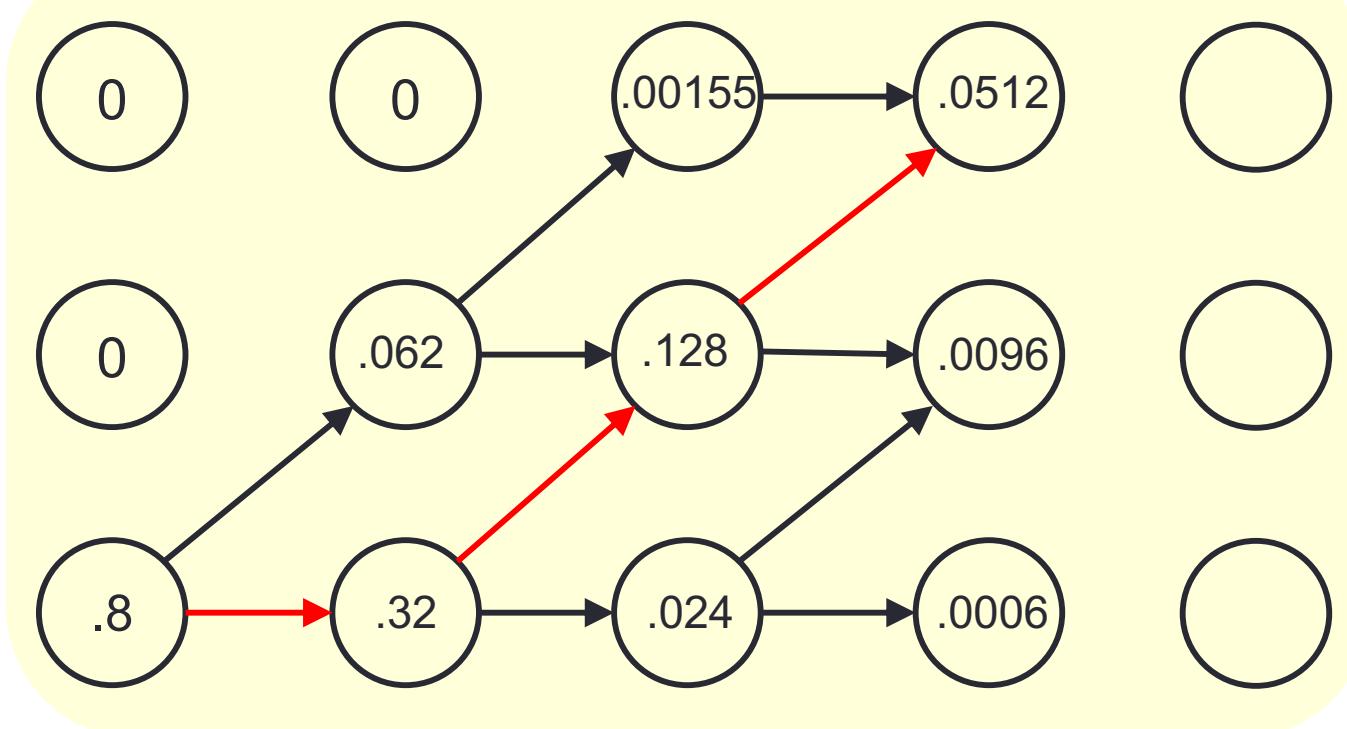
$$\mathbf{b} = [0.01 \ 0.05 \ 0.15 \ 0.2 \ 0.1 \ 0.6 \ 0.09]^T$$

“f” “ay” “v” “w” “ah” “t” “uw”

Trellis for two = “t” “uw”



Trellis for word five



Finally, assume next words can be “one” or “two”
 We concatenate to next possible words:

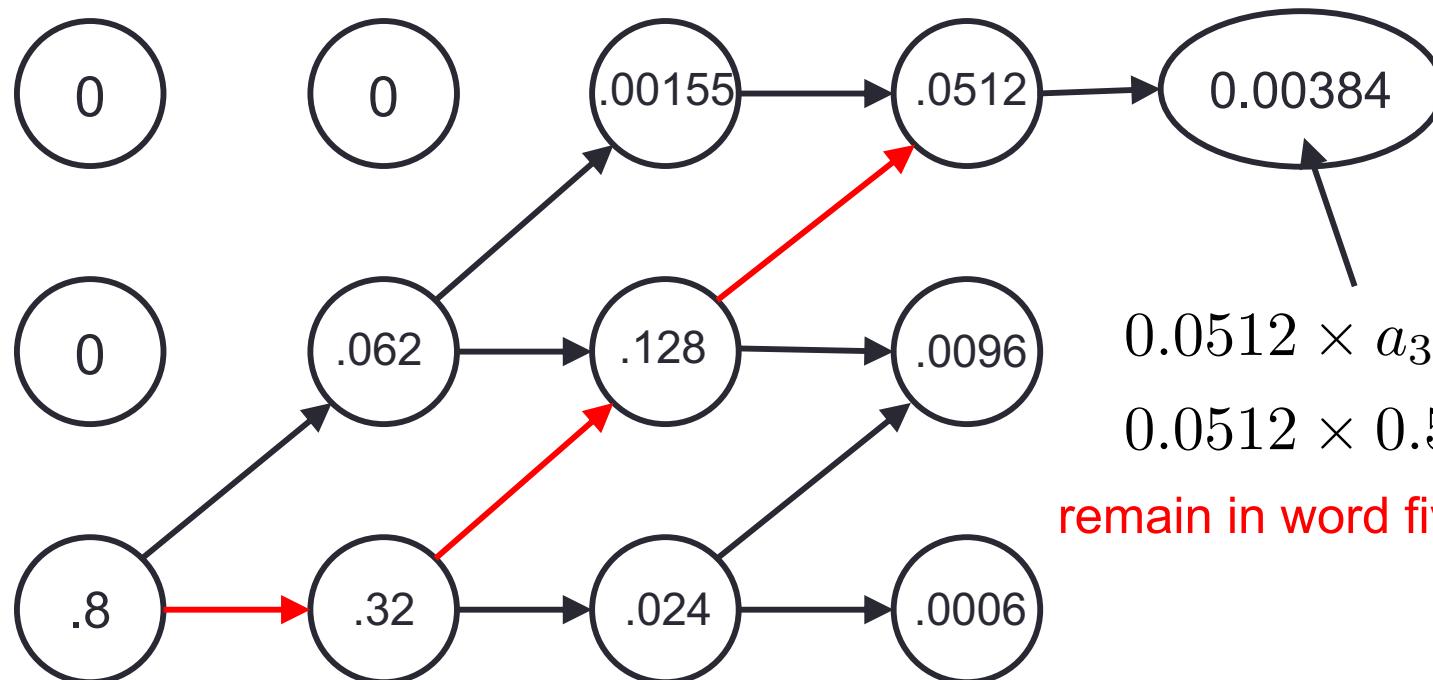
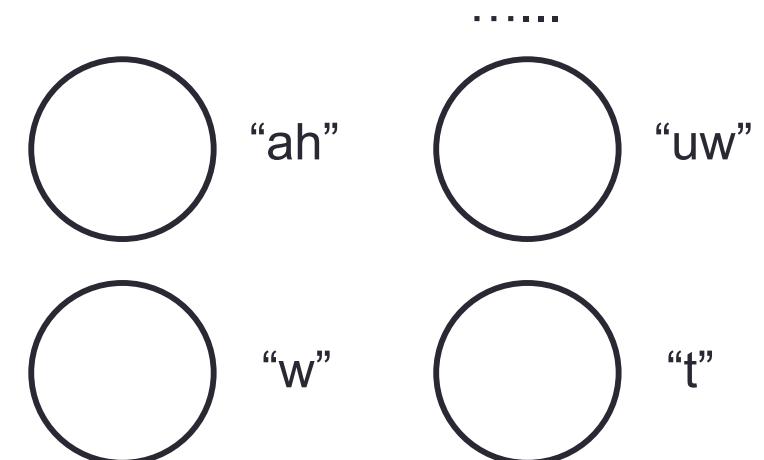
Trellis for two = “t” “uw”

$$P(\text{"one"} | \text{"five"}) = P_{51} = 0.6, P(\text{"two"} | \text{"five"}) = P_{52} = 0.4$$

$$\mathbf{b} = [0.01 \ 0.05 \ 0.15 \ 0.2 \ 0.1 \ 0.6 \ 0.09]^T$$

b_3
0.15
“v”

“f”	“ay”	“v”	“w”	“ah”	“t”	“uw”
-----	------	-----	-----	------	-----	------



$$0.0512 \times a_{33} \times b_3 = \\ 0.0512 \times 0.5 \times 0.15 = 0.00384$$

remain in word five (a_{33}) and emit y_5 (b_3)

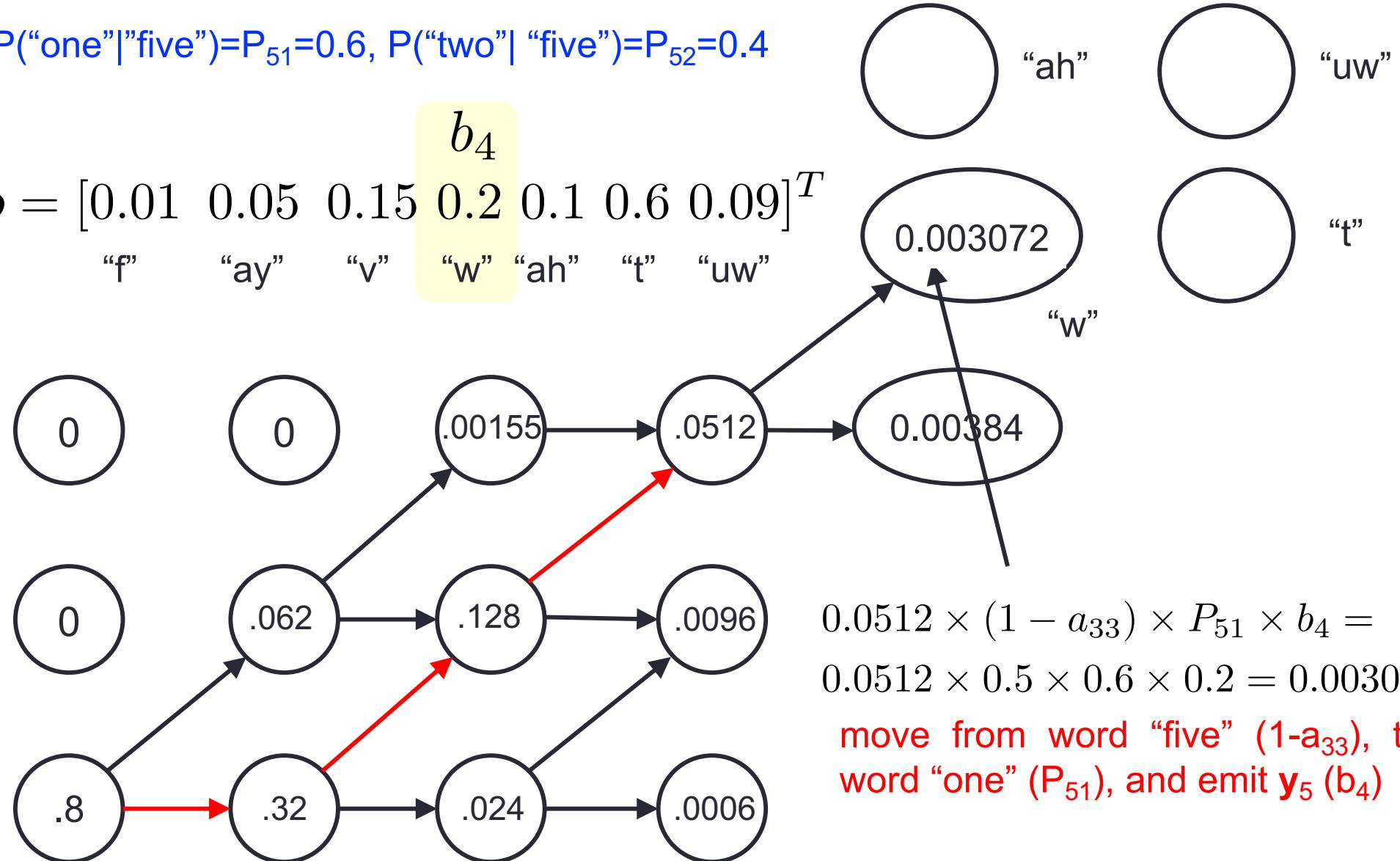
Finally, assume next words can be “one” or “two”
 We concatenate to next possible words:

Trellis for two = “t” “uw”

$$P(\text{"one"} | \text{"five"}) = P_{51} = 0.6, P(\text{"two"} | \text{"five"}) = P_{52} = 0.4$$

$$\mathbf{b} = [0.01 \ 0.05 \ 0.15 \ 0.2 \ 0.1 \ 0.6 \ 0.09]^T$$

b_4
 “f” “ay” “v” “w” “ah” “t” “uw”



$$\begin{aligned}
 &0.0512 \times (1 - a_{33}) \times P_{51} \times b_4 = \\
 &0.0512 \times 0.5 \times 0.6 \times 0.2 = 0.003072
 \end{aligned}$$

move from word “five” ($1-a_{33}$), to word “one” (P_{51}), and emit \mathbf{y}_5 (b_4)

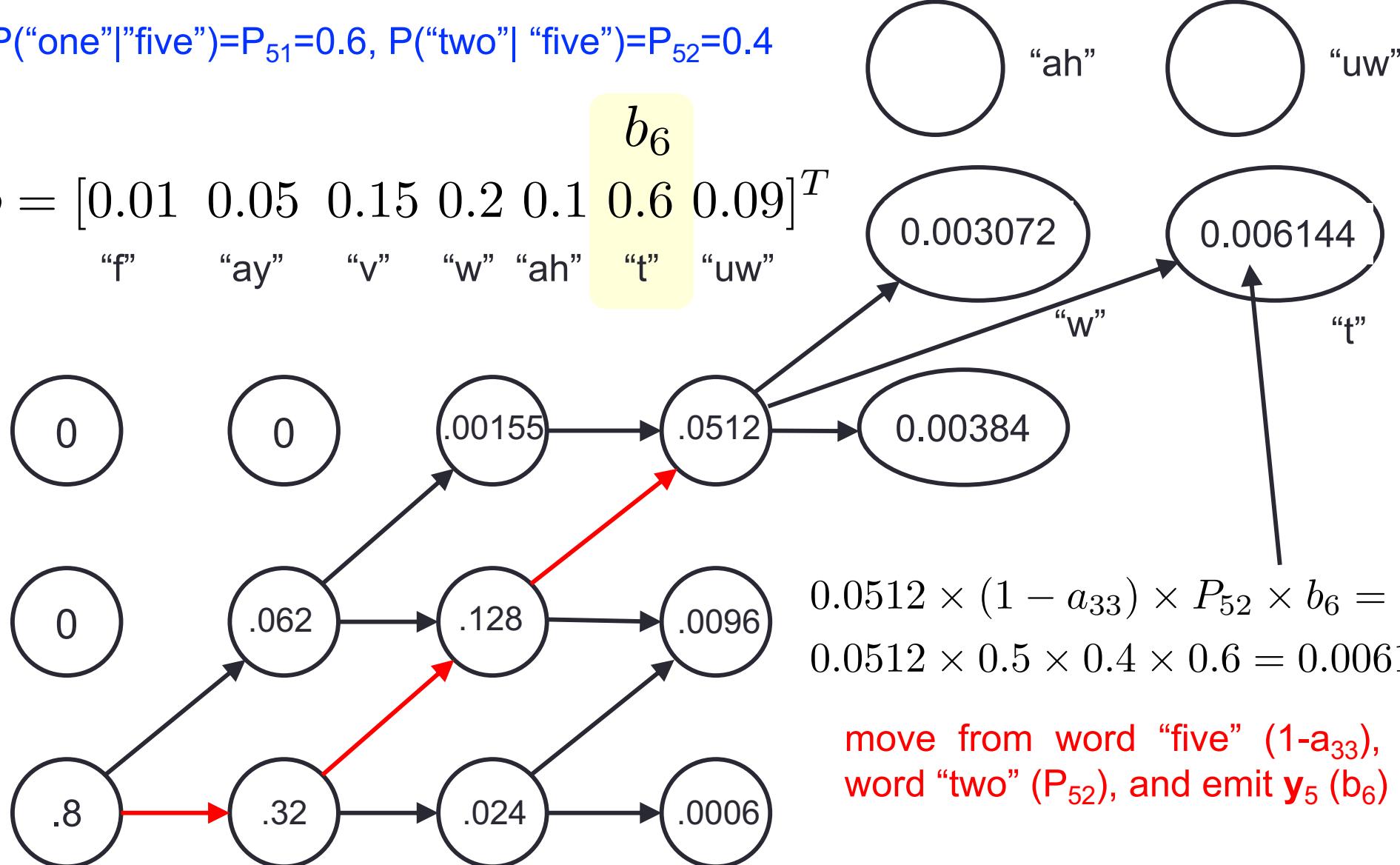
Finally, assume next words can be “one” or “two”
 We concatenate to next possible words:

Trellis for two = “t” “uw”

$$P(\text{"one"} | \text{"five"}) = P_{51} = 0.6, P(\text{"two"} | \text{"five"}) = P_{52} = 0.4$$

$$\mathbf{b} = [0.01 \ 0.05 \ 0.15 \ 0.2 \ 0.1 \ 0.6 \ 0.09]^T$$

“f” “ay” “v” “w” “ah” “t” “uw”



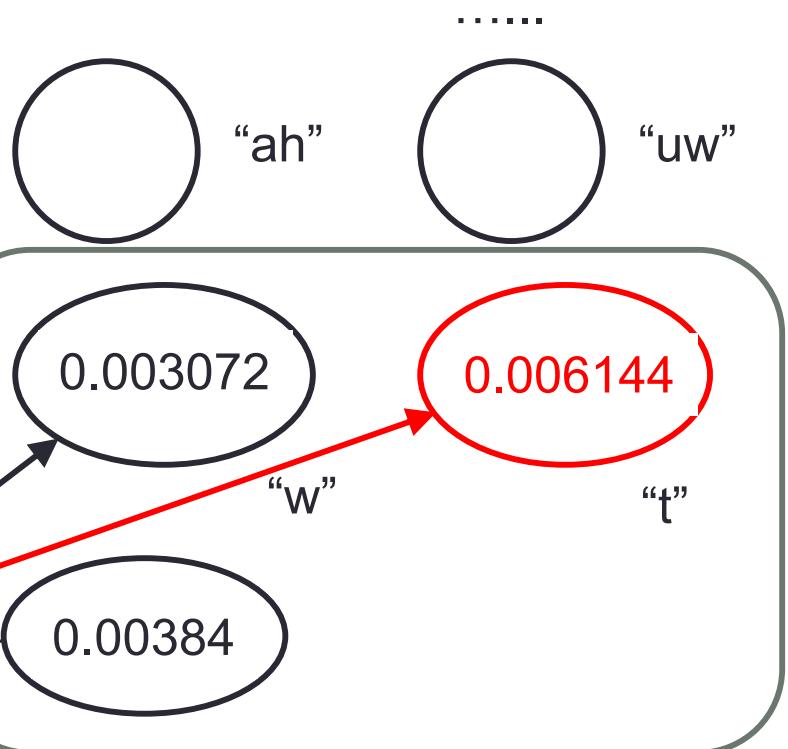
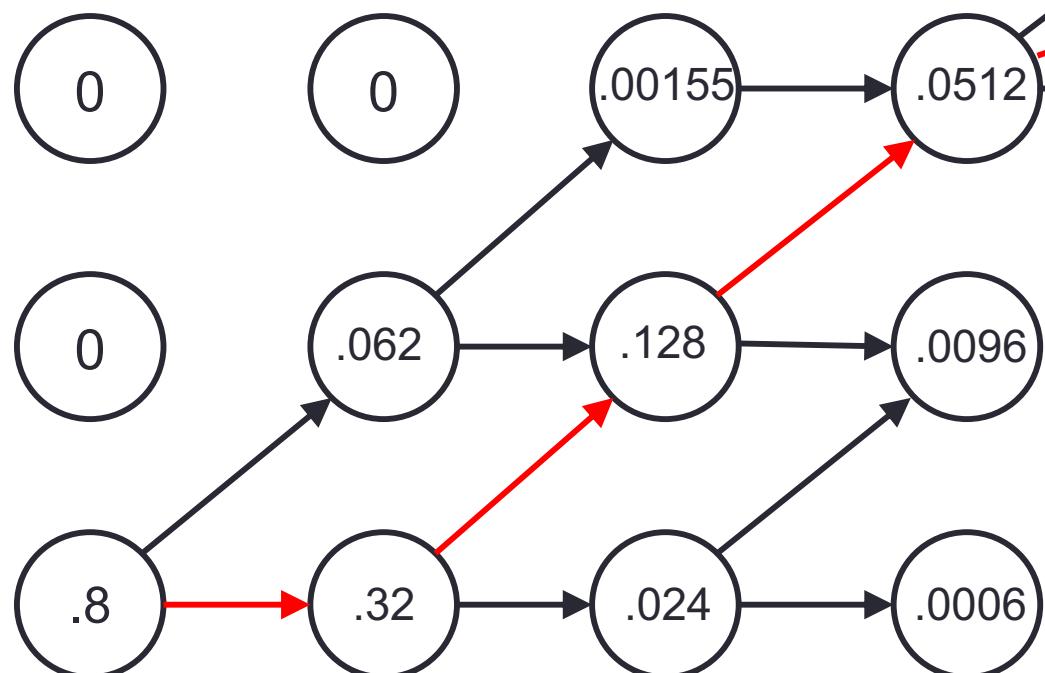
Finally, assume next words can be “one” or “two”
 We concatenate to next possible words:

Trellis for two = “t” “uw”

$$P(\text{"one"}|\text{"five"})=P_{51}=0.6, P(\text{"two"}|\text{"five"})=P_{52}=0.4$$

$$\mathbf{b} = [0.01 \ 0.05 \ 0.15 \ 0.2 \ 0.1 \ 0.6 \ 0.09]^T$$

“f” “ay” “v” “w” “ah” “t” “uw”



Next optimal state is “t”
 So next word will be “two”

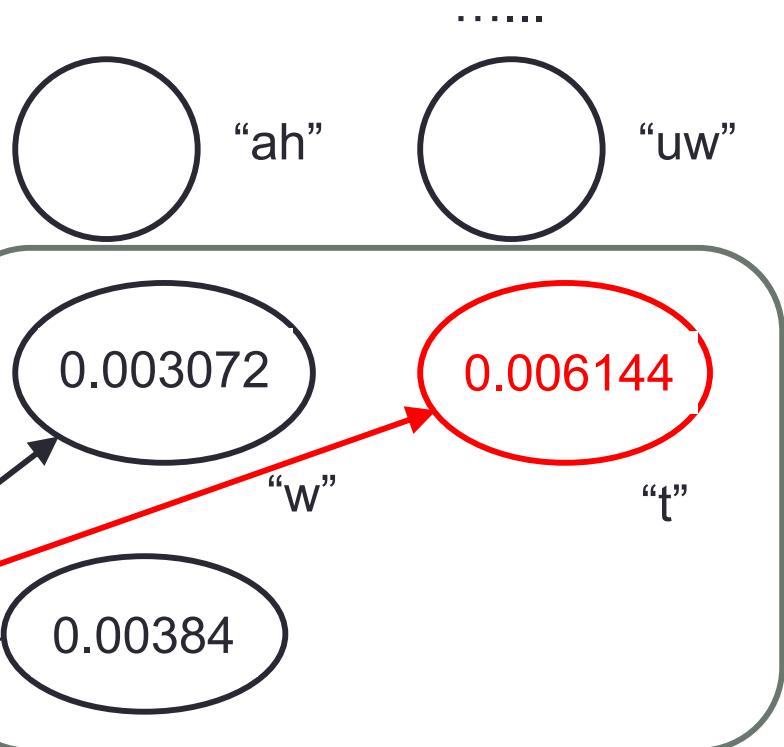
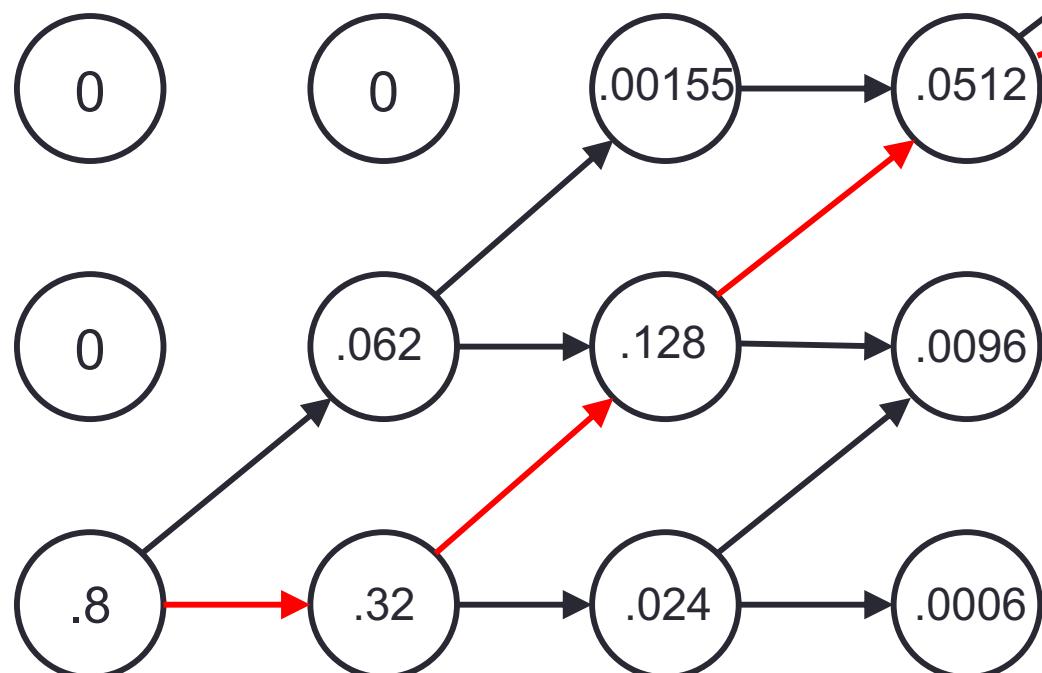
Finally, assume next words can be “one” or “two”
 We concatenate to next possible words:

Trellis for two = “t” “uw”

$$P(\text{"one"} | \text{"five"}) = P_{51} = 0.6, P(\text{"two"} | \text{"five"}) = P_{52} = 0.4$$

$$\mathbf{b} = [0.01 \ 0.05 \ 0.15 \ 0.2 \ 0.1 \ 0.6 \ 0.09]^T$$

b_4 b_6
 “f” “ay” “v” “w” “ah” “t” “uw”



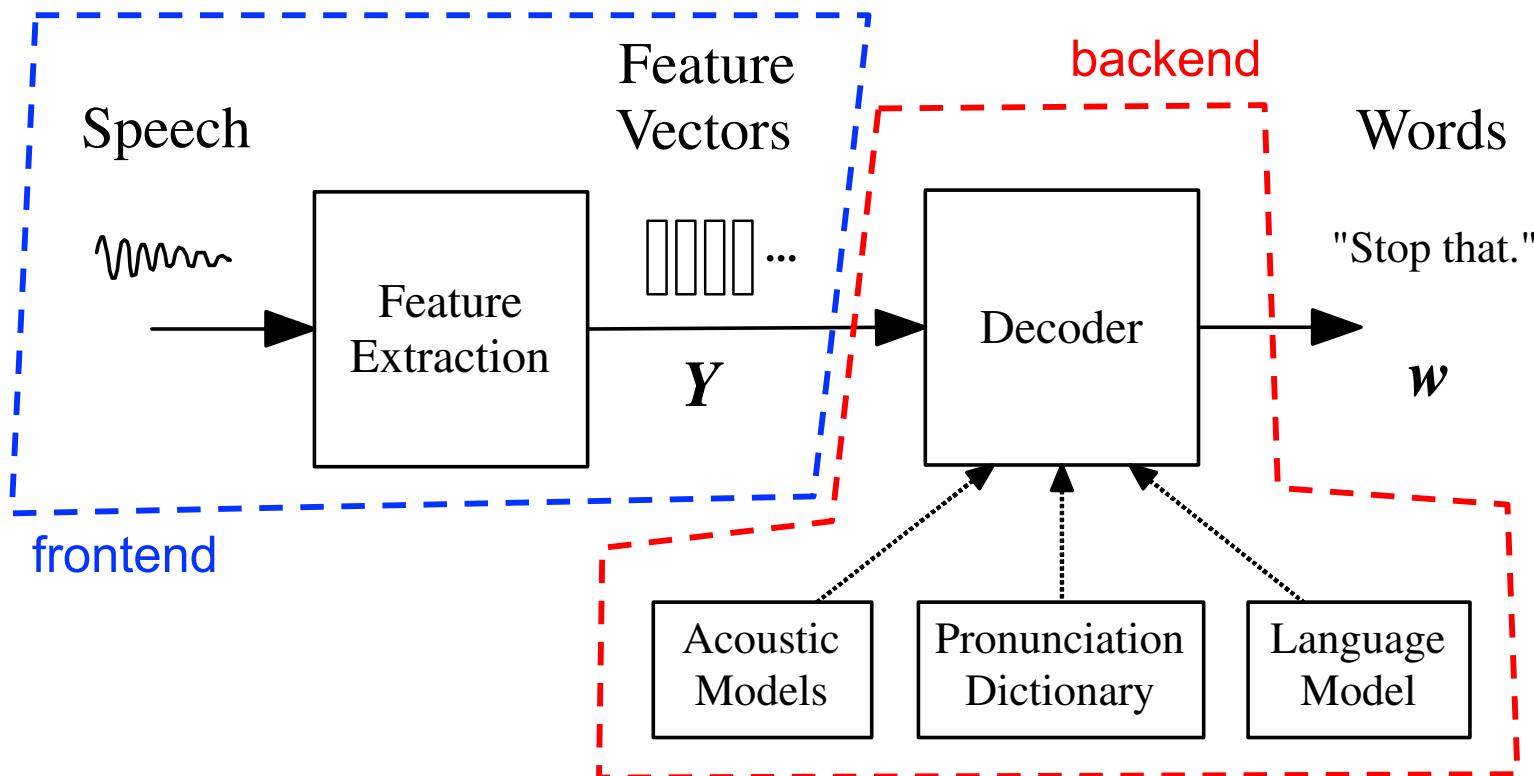
NICE TO OBSERVE

“two” is selected although
 $P_{52} < P_{51}$ (language model)
 as $b_6 >> b_4$ (acoustic model)

In a real HMM speech decoder

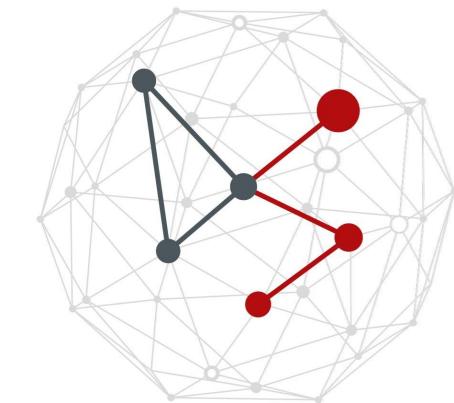
- Words structures can be more complex
 - to account for different pronunciations
- Phones
 - are expanded into sub-phones
- Language model
 - accounts for N-1 previous words in the Viterbi trellis (N-gram)
- Programming
 - Logarithms and other tricks for numerical stability, etc.
 - Not all possible words that follow are fully explored, low probability paths are pruned: “beam search”, i.e., prune paths to next word with prob. smaller than a threshold wrt best following path (“word”). With beam search 90-95% of the states are not considered
- Conceptually
 - nothing changes, the Viterbi algorithm will find the most probable path

ASR architecture



- **Feature extraction:** aka frontend processing (transforms waveform into features)
- **Acoustic model (HMM):** probability of observation *given* a word sequence
- **Pronunciation dictionary (lexicon):** mapping each word into a phone sequence
- **Language Model:** computes the prior probability of any word sequence

ON HMM MODEL TRAINING



HMM training - ingredients

$$\theta = \{\theta_1, \theta_2, \dots, \theta_N\}$$

A set of **states** corresponding to **sub-phones**

$$A = [a_{ij}]$$

Transition probability matrix from previous state i to current state j. θ and A implement a **pronunciation lexicon**, a state graph structure for each word that the system is capable of recognizing.

$$B = [b_i(y_n)] = [p(y_n | \theta_n = i)]$$

A set of **observation likelihoods B**: also called **emission probabilities**, each expressing the probability of a cepstral feature vector (observation y_n) being generated given that subphone (HMM) state is i

HMM training: considerations

- It is expensive to use humans to hand-label phonetic boundaries (sub-phones)
 - it can take up to 400 times real time (i.e., 400 labeling hours to label just 1 hour of speech)
 - humans do not do phonetic labeling very well for units smaller than the phone
 - humans are bad at consistently finding the boundaries of sub-phones
- We then use **embedded training**
 - We provide the system with sentences
 - Words within each sentence were labeled
 - Sub-phone boundaries are automatically found by the training algorithm

Training: full picture

Transcribed sentence:

LUCY IN THE SKY WITH DIAMONDS



PRONUNCIATION DICTIONARY



L UW S IY IH N DH AH SKAY W IH DH
DAY MAH N D Z



word1

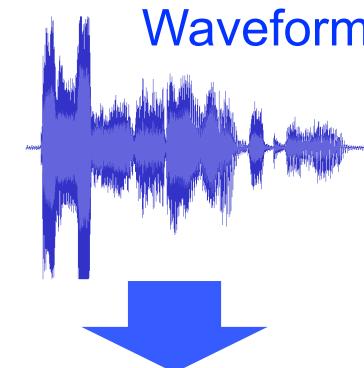
word2

...

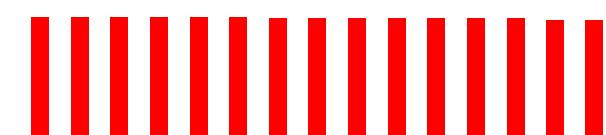
wordN



words in the sentence: HMM for each from phone model, concatenated into a raw HMM



FEATURE EXTRACTION

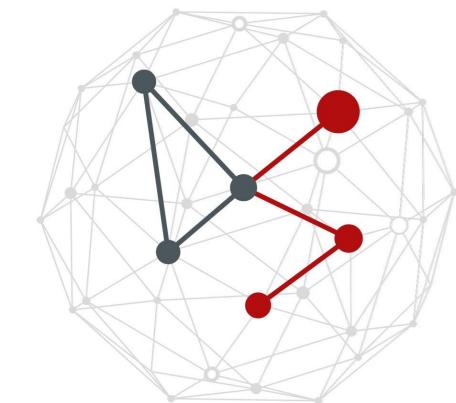


Baum-Welch
training

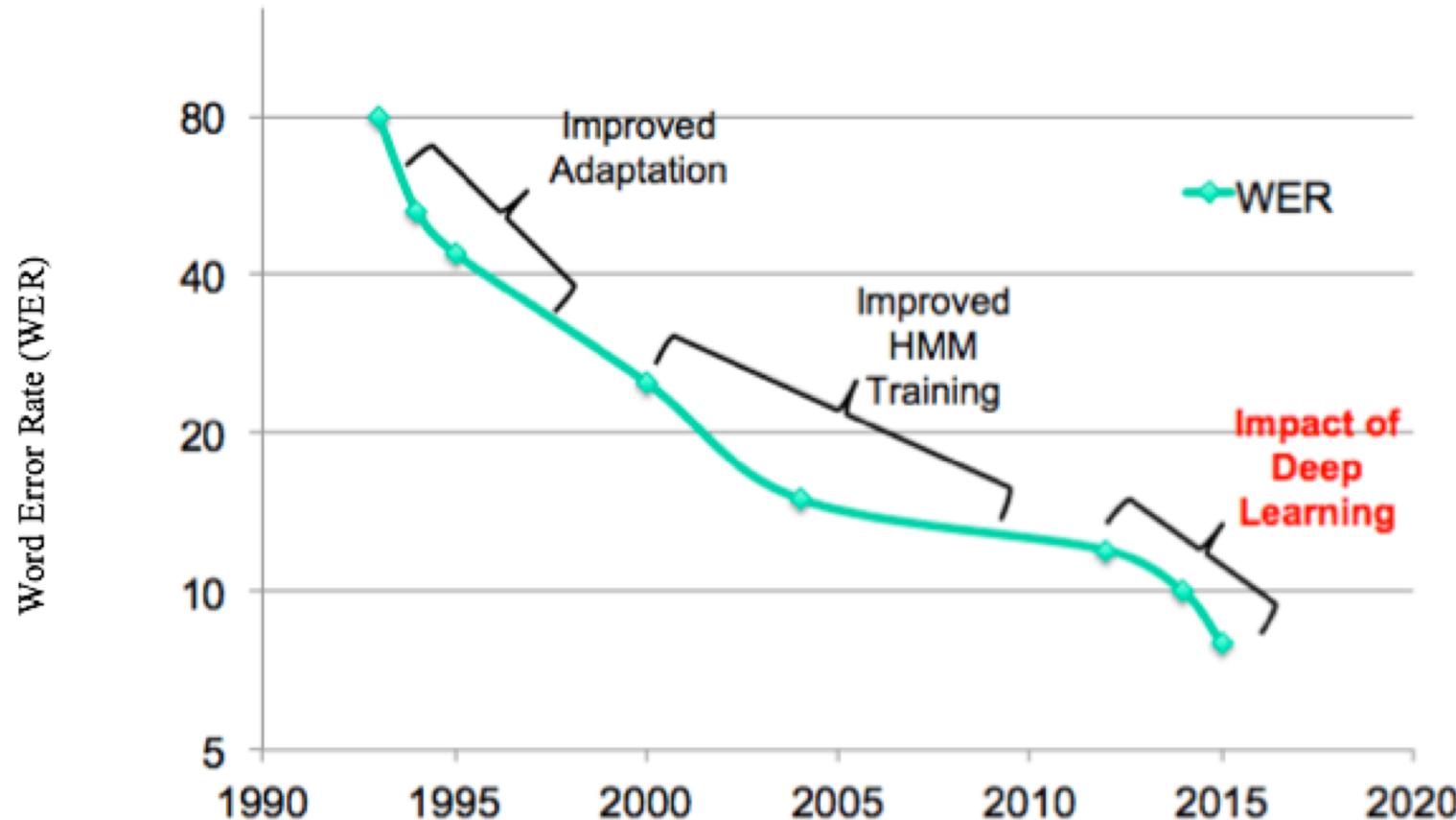
Training: some further discussion

- **Beauty of Baum-Welch training**
 - We do not need phonetically transcribed data
 - We do not even need to know *where each word starts and ends*
 - The Baum-Welch algorithm *will sum over all possible segmentations of words and phones*
 - Matrices **A** and **B** will be obtained to maximize the likelihood of observing the provided (input) feature vectors
- **HMM Initialization for training: FLAT START**
 - Set to zero all transition probabilities that we want to be structurally zero (phones that are not in the sentence and/or that are not consecutive given the input word sequence)
 - If state i has trans. probs. a_{ij} and K possible following states $\rightarrow a_{ij}=1/K$
 - Mean vector and covariance matrix of state i are taken equal to mean and covariance matrix of the entire sentence

PERFORMANCE AND FINAL CONSIDERATIONS



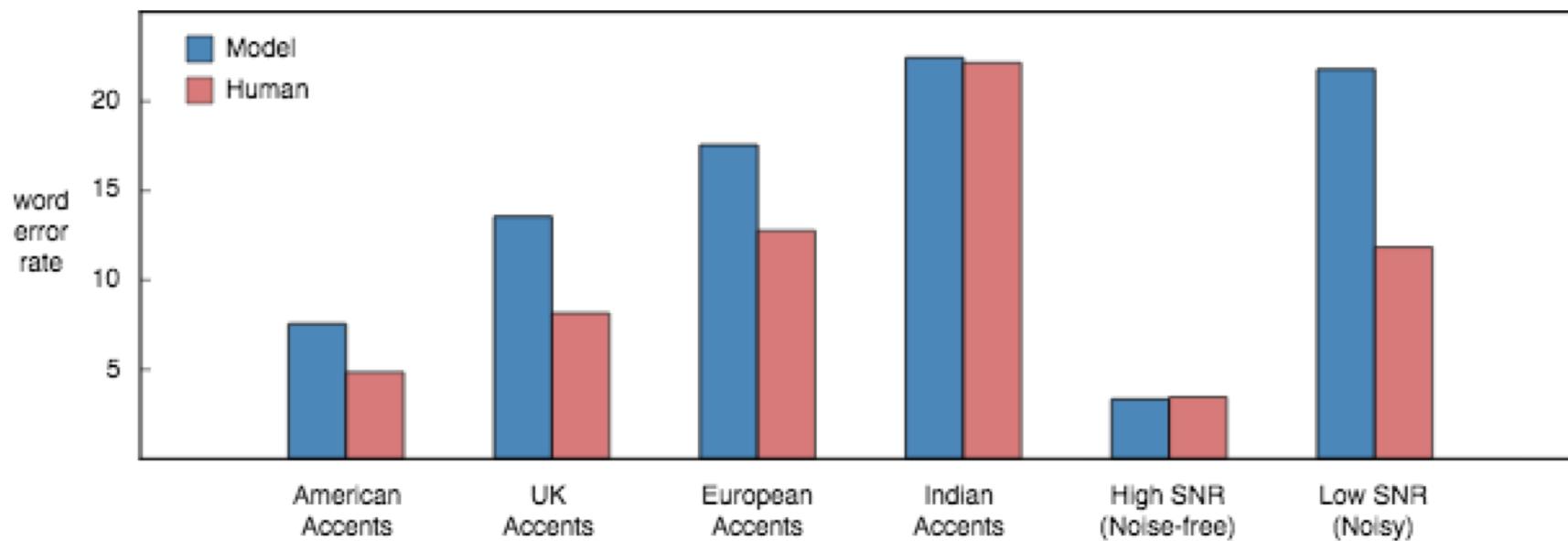
Performance



ASR word error rate (WER) over the last two decades on a public benchmark test for English conversational speech. Improved adaptation and training of HMM resulted in significant progress until 2005. The performance reached a plateau between 2005 and 2012. The introduction of deep learning around 2012 is leading to a future generation of speech recognition systems

Open issues

- Background noise
 - SNR can be as low as -5dB in a moving car
 - People do not have much problem but ASR systems struggle
- Accents
 - Non-US accents are more difficult to comprehend



Open issues

- Semantic errors
 - WER alone might not be the right metric
 - Example1: “let’s meet up today” → “let’s meet Tuesday”
 - Example2: “uh” (often translated into) → “uh huh”
 - “uh” is a filler (no meaning)
 - “uh huh” is a backchannel acknowledgement
- Multiple users single channel
 - Source separation (diarisation) in arbitrary conditions
- Domain variations ASR should be robust to
 - Reverberation from varying the acoustic environment
 - Artefacts from the hardware
 - The codec used for the audio and compression artefacts
 - The sample rate
 - The age of the speaker

Next few years?

- Broadening the capabilities to:
 - new domains, accents and low SNR speech
- Incorporating more *context information*
 - Context example: people talking about being hungry
 - Spoken: I think I'm gonna have lunch early today
 - Translation: I think I'm gonna have lounge early today
- Diarisation
 - partitioning an audio stream according to the speaker's identity
- Semantic error rates and innovative evaluation methods
- Computational advancements
 - Super low-latency and efficient inference

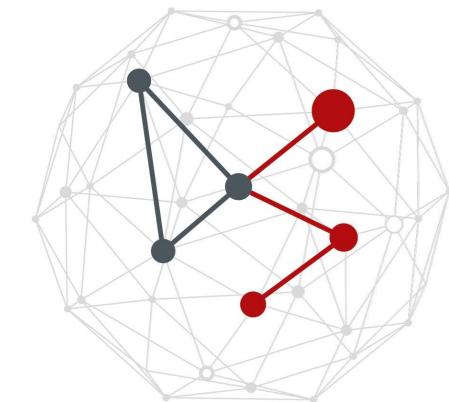
References

- [Jurafsky2008]:
 - Chapter 2 “Speech”
 - Section 2.9 “Automatic Speech Recognition”
- A place to look at:
 - <https://research.google.com/pubs/SpeechProcessing.html>

[Jurafsky2008] Daniel Jurafsky, James H. Martin, “Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition,” Prentice Hall Series in Artificial Intelligence, 2008.

APPENDIX A

Matlab code to compute filterbanks



Filterbanks Matlab code (1/2)

```
Fs=20480; % sampling rate
nDFT=512; % number of DFT samples
Fnyq=Fs/2; % Nyquist frequency

Nfb=10; % number of filterbanks to generate
Fmin=300; % min frequency of filterbanks [Hz]
Fmax=Fnyq; % max frequency of filterbanks [Hz]

FminMel=1125*log(1+(Fmin/700)); % min Mel's frequency
FmaxMel=1125*log(1+(Fmax/700)); % Max Mel's frequency

Nthresh=Nfb+2; % number of required frequency thresholds

step=(FmaxMel-FminMel)/(Nthresh-1); % step size (linearly partition Mel's space)

% allocate memory for frequency thresholds
melvec=zeros(1,Nthresh);
freqvec=zeros(1,Nthresh);
f=zeros(1,Nthresh);

% for each frequency threshold do
for i = 0:(Nthresh-1)
    melvec(i+1) = FminMel+i*step; % assign Mel thresholds (linearly)
    freqvec(i+1) = 700*(exp(melvec(i+1)/1125)-1); % compute corresponding frequencies (Mel's transformation)
    f(i+1) = floor((nDFT+1)*freqvec(i+1)/Fs); % express frequency thresholds in terms of DFT indeces
end
```

Filterbanks Matlab code (2/2)

```
H=zeros(Nfb,Nthresh); % filterbank matrix, here we store the Nfb filters
```

```
% for each filter m do (m is the filterbank index)
```

```
for m = 2:(Nfb+1)
    for k=1:f(Nthresh)
        if      ((k>=f(m-1)) && (k<=f(m))) H(m-1,k) = (k-f(m-1))/(f(m)-f(m-1)); % increase
        elseif ((k>=f(m)) && (k<=f(m+1))) H(m-1,k) = (f(m+1)-k)/(f(m+1)-f(m)); % decrease
        else   H(m-1,k) = 0;
        end
    end
end
```

```
clf
figure(1);
hold on
for m = 1:Nfb
    % read Filterbank matrix by row, each row contains a filter
    plot(H(m,:),'LineWidth',3)
end
```

SPEECH MODELS AND AUTOMATIC SPEECH RECOGNITION (ASR) SYSTEMS

Michele Rossi
rossi@dei.unipd.it

Dept. of Information Engineering
University of Padova, IT

