

# Automatic Speech Recognition: Introduction

Steve Renals & Hiroshi Shimodaira

Automatic Speech Recognition— ASR Lecture 1  
15 January 2018

# Automatic Speech Recognition — ASR

## Course details

- **Lectures:** About 18 lectures, plus a couple of extra lectures on basic introduction to neural networks
- **Labs:** Weekly lab sessions – using Kaldi ([kaldi-asr.org](http://kaldi-asr.org)) to build speech recognition systems.
  - Lab sessions in AT-4.12: Tuesdays 10:00, Wednesdays 10:00, Wednesdays 15:10, start week 2 (23/24 January)
  - Select one lab session at  
<https://doodle.com/poll/gxmh9kwp3a8espxx>
- **Assessment:**
  - Exam in April or May (worth 70%)
  - Coursework (worth 30%, building on the lab sessions): out on Monday 12 February; in by Wednesday 14 March
- **People:**
  - Lecturers: Steve Renals and Hiroshi Shimodaira
  - TAs: Joachim Fainberg and Ondrej Klejch



# Your background

If you have taken:

- Speech Processing *and* either of (MLPR or MLP)
  - Perfect!
- either of (MLPR or MLP) *but not* Speech Processing
  - You'll require some speech background:
    - A couple of the lectures will cover material that was in Speech Processing
    - Some additional background study (including material from Speech Processing)
- Speech Processing *but neither of* (MLPR or MLP)
  - You'll require some machine learning background (especially neural networks)
    - A couple of introductory lectures on neural networks
    - Some additional background study

# Labs

- Series of weekly labs using Kaldi.
- Labs start week 2 (next week)
- **Note:** Training speech recognisers can take time
  - ASR training in some labs will not finish in an hour...
  - Give yourself plenty of time to complete the coursework, don't leave it until the last couple of days

# What is speech recognition?

## Speech-to-text transcription

- Transform recorded audio into a sequence of words
- Just the words, no meaning.... But do need to deal with acoustic ambiguity: “Recognise speech?” or “Wreck a nice beach?”
- Speaker diarization: Who spoke when?
- Speech recognition: what did they say?
- Paralinguistic aspects: how did they say it? (timing, intonation, voice quality)
- Speech understanding: what does it mean?

# Why is speech recognition difficult?

# Variability in speech recognition

Several sources of variation

Size Number of word types in vocabulary, perplexity

# Variability in speech recognition

Several sources of variation

Size Number of word types in vocabulary, perplexity

Speaker Tuned for a particular speaker, or  
speaker-independent? Adaptation to speaker  
characteristics and accent

# Variability in speech recognition

Several sources of variation

Size Number of word types in vocabulary, perplexity

Speaker Tuned for a particular speaker, or  
speaker-independent? Adaptation to speaker  
characteristics and accent

Acoustic environment Noise, competing speakers, channel  
conditions (microphone, phone line, room acoustics)

# Variability in speech recognition

Several sources of variation

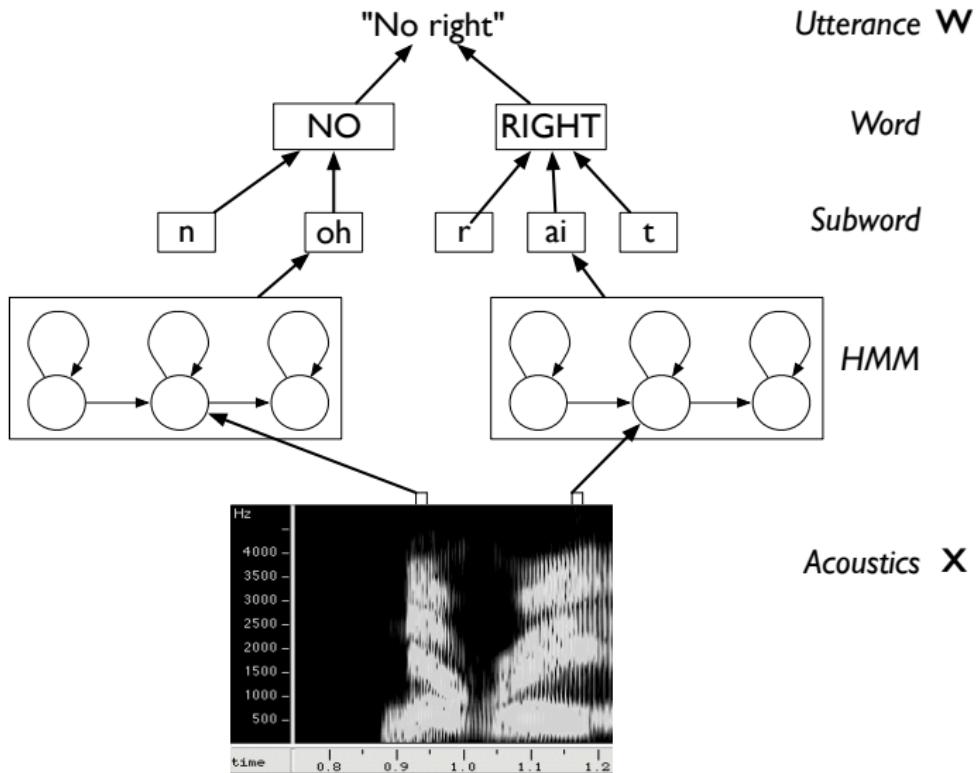
Size Number of word types in vocabulary, perplexity

Speaker Tuned for a particular speaker, or  
speaker-independent? Adaptation to speaker  
characteristics and accent

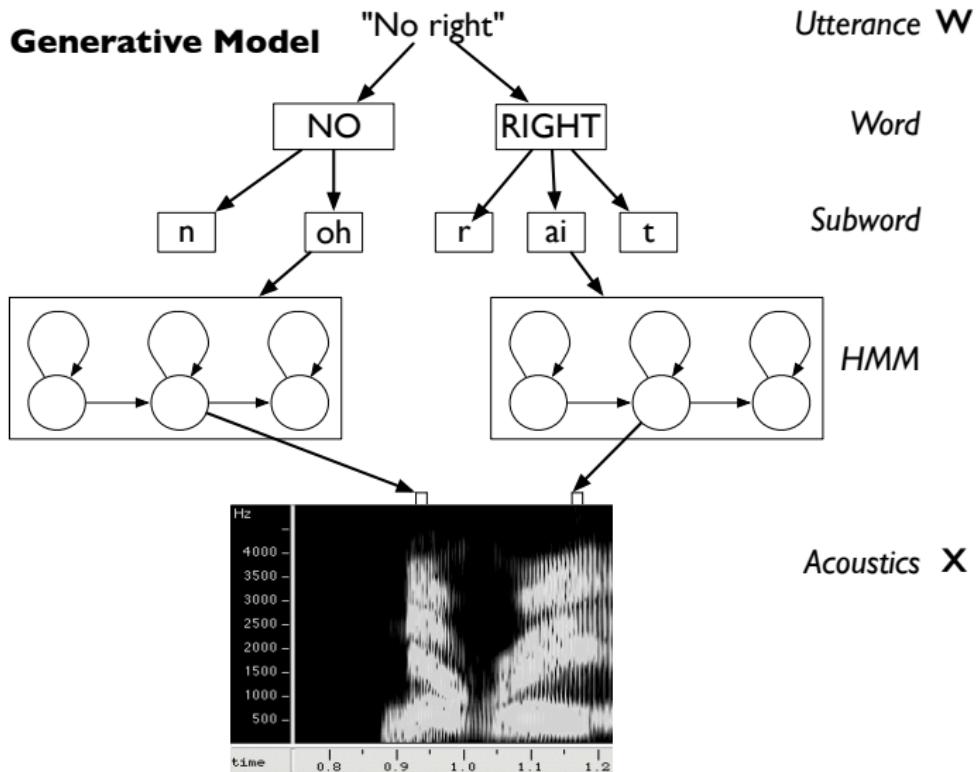
Acoustic environment Noise, competing speakers, channel  
conditions (microphone, phone line, room acoustics)

Style Continuously spoken or isolated? Planned monologue  
or spontaneous conversation?

# Hierarchical modelling of speech



# Hierarchical modelling of speech



# “Fundamental Equation of Statistical Speech Recognition”

If  $\mathbf{X}$  is the sequence of acoustic feature vectors (observations) and  $\mathbf{W}$  denotes a word sequence, the most likely word sequence  $\mathbf{W}^*$  is given by

$$\mathbf{W}^* = \arg \max_{\mathbf{W}} P(\mathbf{W} \mid \mathbf{X})$$

# “Fundamental Equation of Statistical Speech Recognition”

If  $\mathbf{X}$  is the sequence of acoustic feature vectors (observations) and  $\mathbf{W}$  denotes a word sequence, the most likely word sequence  $\mathbf{W}^*$  is given by

$$\mathbf{W}^* = \arg \max_{\mathbf{W}} P(\mathbf{W} | \mathbf{X})$$

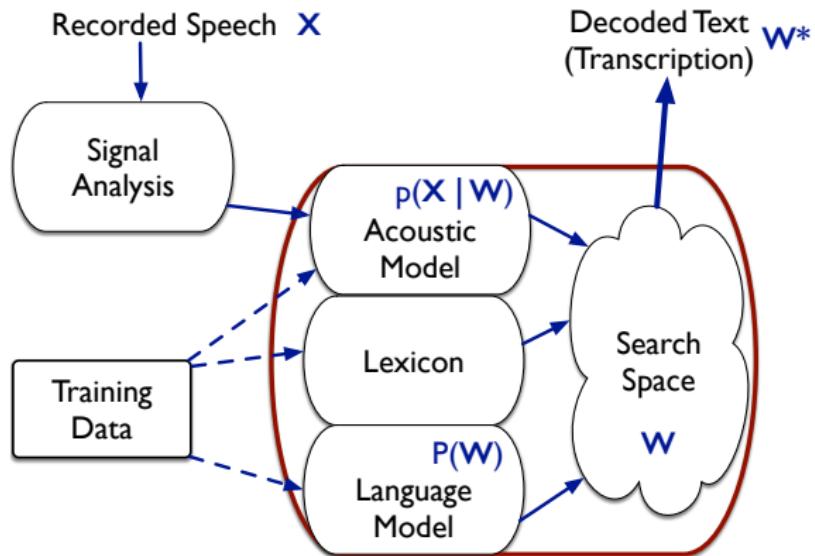
Applying Bayes' Theorem:

$$\begin{aligned} P(\mathbf{W} | \mathbf{X}) &= \frac{p(\mathbf{X} | \mathbf{W})P(\mathbf{W})}{p(\mathbf{X})} \\ &\propto p(\mathbf{X} | \mathbf{W})P(\mathbf{W}) \end{aligned}$$
$$\mathbf{W}^* = \arg \max_{\mathbf{W}} \underbrace{p(\mathbf{X} | \mathbf{W})}_{\text{Acoustic model}} \underbrace{P(\mathbf{W})}_{\text{Language model}}$$

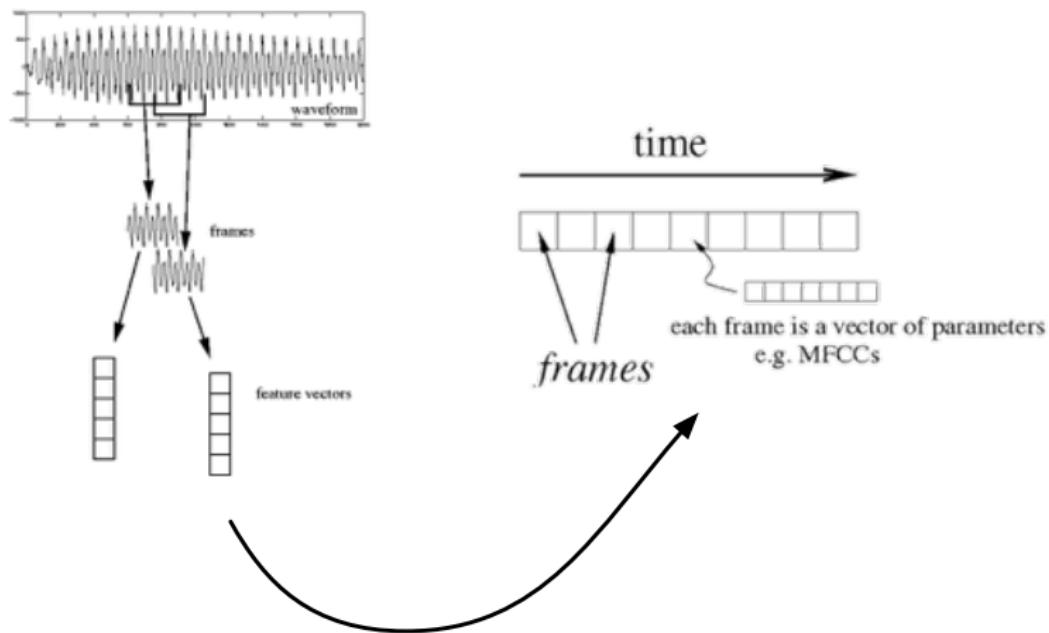
# Speech Recognition Components

$$\mathbf{W}^* = \arg \max_{\mathbf{W}} p(\mathbf{X} | \mathbf{W})P(\mathbf{W})$$

Use an acoustic model, language model, and lexicon to obtain the most probable word sequence  $\mathbf{W}^*$  given the observed acoustics  $\mathbf{X}$



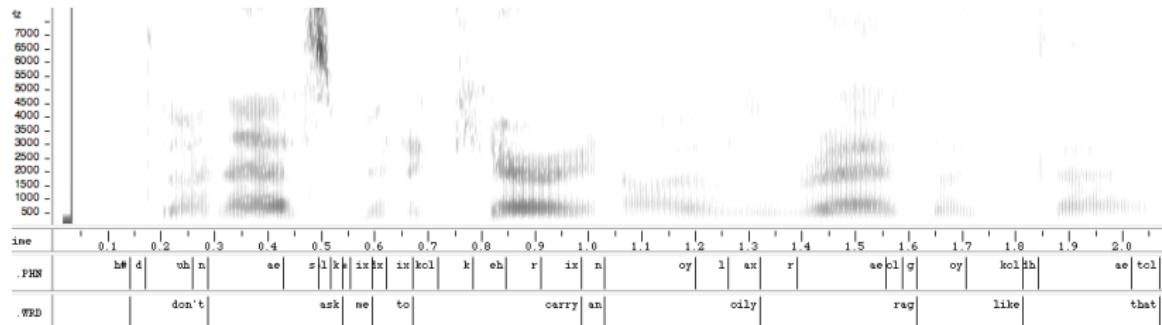
# Representing recorded speech (X)



Represent a recorded utterance as a sequence of *feature vectors*

Reading: Jurafsky & Martin section 9.3

# Labelling speech (W)



Labels may be at different levels: words, phones, etc.

Labels may be *time-aligned* – i.e. the start and end times of an acoustic segment corresponding to a label are known

Reading: Jurafsky & Martin chapter 7 (especially sections 7.4, 7.5)

# Phones and Phonemes

- **Phonemes**

- abstract unit defined by linguists based on contrastive role in word meanings (eg “cat” vs “bat”)
- 40–50 phonemes in English

- **Phones**

- speech sounds defined by the acoustics
- many *allophones* of the same phoneme (eg /p/ in “pit” and “spit”)
- limitless in number
- Phones are usually used in speech recognition – but no conclusive evidence that they are the basic units in speech recognition
- Possible alternatives: syllables, automatically derived units, ...

(Slide taken from Martin Cooke from long ago)

## Example: TIMIT Corpus

- TIMIT corpus (1986)—first widely used corpus, still in use
  - Utterances from 630 North American speakers
  - Phonetically transcribed, time-aligned
  - Standard training and test sets, agreed evaluation metric (phone error rate)
- TIMIT phone recognition - label the audio of a recorded utterance using a sequence of phone symbols
  - Frame classification – attach a phone label to each frame data
  - Phone classification – given a segmentation of the audio, attach a phone label to each (multi-frame) segment
  - Phone recognition – supply the sequence of labels corresponding to the recorded utterance

# Basic speech recognition on TIMIT

- Train a classifier of some sort to associate each feature vector with its corresponding label. Classifier could be
  - Neural network
  - Gaussian mixture model
  - ...

The at test time, a label is assigned to each frame

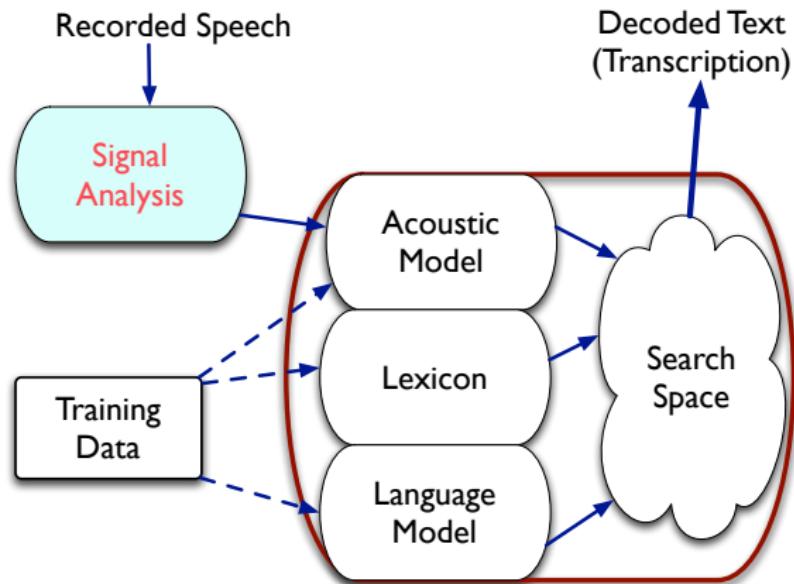
- Questions
  - What's good about this approach?
  - What the limitations? How might we address them?

- How accurate is a speech recognizer?
- String edit distance
  - Use dynamic programming to align the ASR output with a reference transcription
  - Three type of error: insertion, deletion, substitutions
- Word error rate (WER) sums the three types of error. If there are  $N$  words in the reference transcript, and the ASR output has  $S$  substitutions,  $D$  deletions and  $I$  insertions, then:

$$\text{WER} = 100 \cdot \frac{S + D + I}{N} \% \quad \text{Accuracy} = 100 - \text{WER}\%$$

- For TIMIT, define phone error error rate analogously to word error rate
- Speech recognition evaluations: common training and development data, release of new test sets on which different systems may be evaluated using word error rate

# Next Lecture



# Reading

- Jurafsky and Martin (2008). *Speech and Language Processing* (2nd ed.): Chapter 7 (esp 7.4, 7.5) and Section 9.3.
- General interest: *The Economist Technology Quarterly*, “Language: Finding a Voice”, Jan 2017.  
<http://www.economist.com/technology-quarterly/2017-05-01/language>

# Speech Signal Analysis

Hiroshi Shimodaira and Steve Renals

Automatic Speech Recognition— ASR Lectures 2&3  
18,22 January 2018

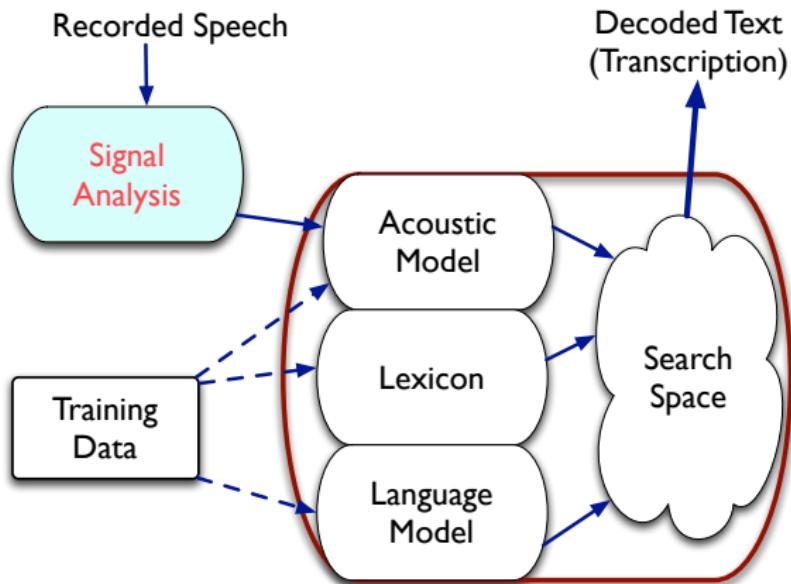
## Speech Signal Analysis for ASR

- Features for ASR
- Spectral analysis
- Cepstral analysis
- Standard features for ASR: FBANK, MFCCs and PLP analysis
- Dynamic features

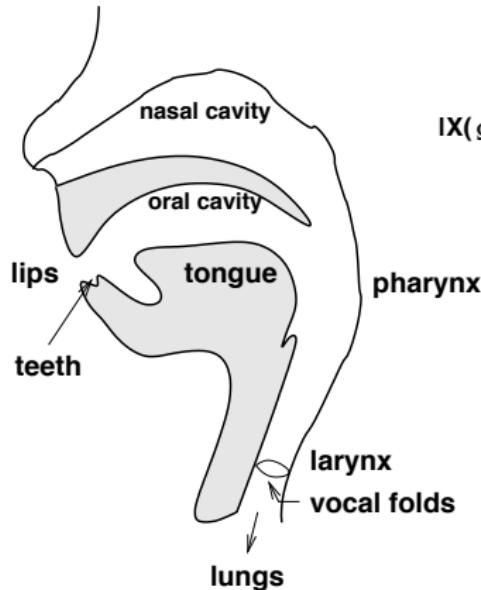
Reading:

- Jurafsky & Martin, sec 9.3
- P Taylor, *Text-to-Speech Synthesis*, chapter 12, signal processing background chapter 10

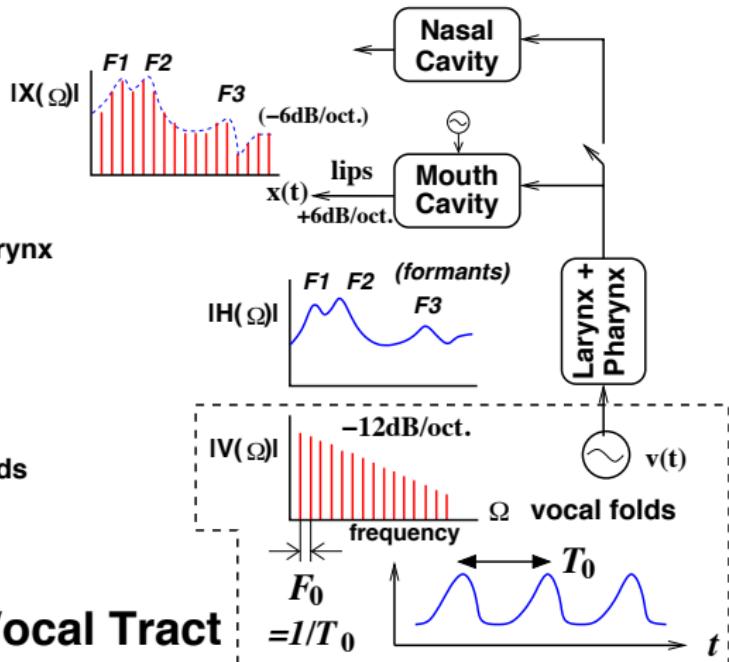
# Speech signal analysis for ASR



# Speech production model



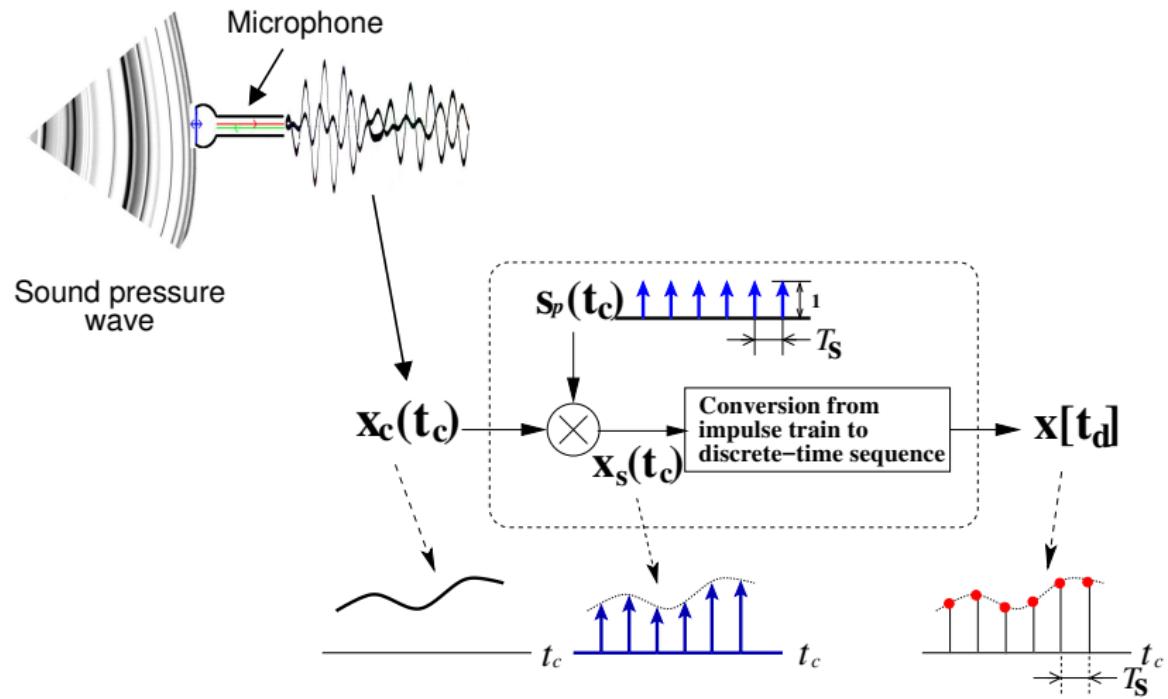
## Vocal Organs & Vocal Tract



( $F_0$  : fundamental frequency)

# A/D conversion — Sampling

Convert analogue signals in digital form



# A/D conversion — Sampling (cont.)

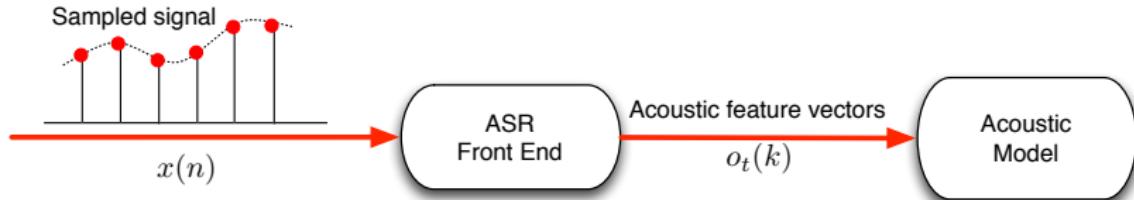
Things to know:

- Sampling Frequency ( $F_s = 1/T_s$ )

Speech	Sufficient $F_s$
Microphone voice (< 10kHz)	20 kHz
Telephone voice (< 4kHz)	8 kHz

- Analogue low-pass filtering to avoid 'aliasing'  
NB: the cut-off frequency should be less than the  
**Nyquist frequency** ( $= F_s/2$ )

# Acoustic Features for ASR



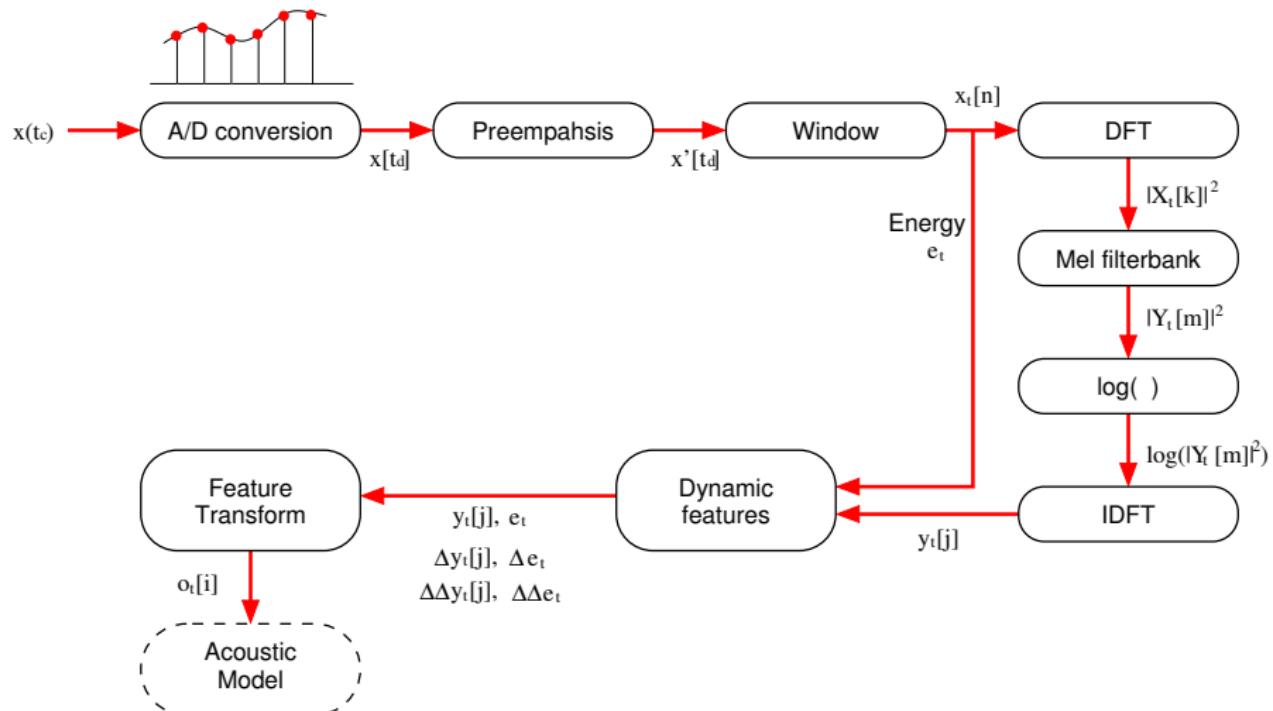
Speech signal analysis to produce a sequence of acoustic feature vectors

# Acoustic Features for ASR

Desirable characteristics of acoustic features used for ASR:

- Features should contain sufficient information to distinguish between phones
  - good time resolution (10ms)
  - good frequency resolution (20 ~ 40 channels)
- Be separated from  $F_0$  and its harmonics
- Be robust against speaker variation
- Be robust against noise or channel distortions
- Have good “pattern recognition characteristics”
  - low feature dimension
  - features are independent of each other (NB: this applies to GMMs, but not required for NN-based systems)

# MFCC-based front end for ASR

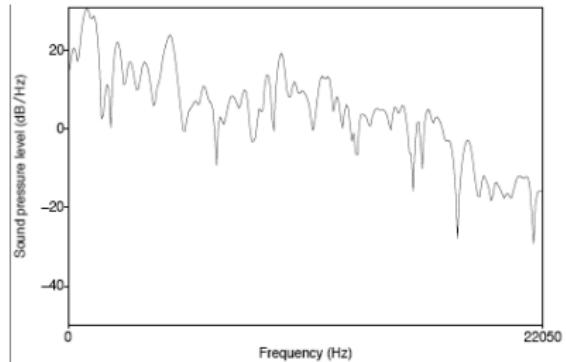
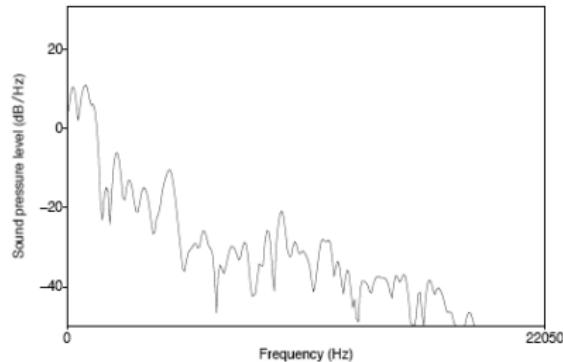


# Pre-emphasis and spectral tilt

- Pre-emphasis increases the magnitude of higher frequencies in the speech signal compared with lower frequencies
- *Spectral Tilt*
  - The speech signal has more energy at low frequencies (for voiced speech)
  - This is due to the glottal source (see the figure)
- Pre-emphasis (first-order) filter boosts higher frequencies:

$$x'[t_d] = x[t_d] - \alpha x[t_d - 1] \quad 0.95 < \alpha < 0.99$$

# Pre-emphasis: example



Vowel /aa/ - time slice of the spectrum

(Jurafsky & Martin, fig. 9.9)

# Windowing

- The speech signal is constantly changing (non-stationary)
- Signal processing algorithms usually assume that the signal is stationary
- Piecewise stationarity: model speech signal as a sequence of **frames** (each assumed to be stationary)
- **Windowing**: multiply the full waveform  $s[n]$  by a window  $w[n]$  (in time domain):

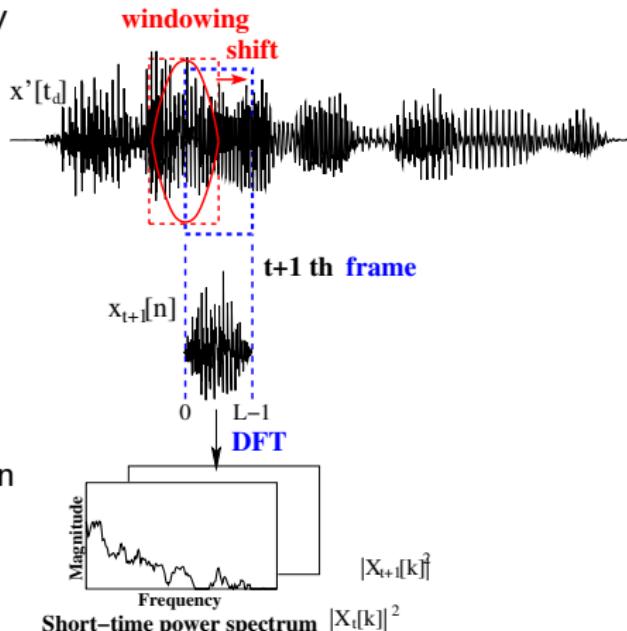
$$x[n] = w[n] s[n] \quad (x_t[n] = w[n] x'[t_d+n])$$

- Simply cutting out a short segment (frame) from  $s[n]$  is a rectangular window — causes discontinuities at the edges of the segment
- Instead, a tapered window is usually used  
e.g. *Hamming* ( $\alpha = 0.46164$ ) or *Hanning* ( $\alpha = 0.5$ ) window

$$w[n] = (1 - \alpha) - \alpha \cos\left(\frac{2\pi n}{L-1}\right) \quad L : \text{window width}$$

# Windowing and spectral analysis

- Window the signal  $x'[t_d]$  into frames  $x_t[n]$  and apply Fourier Transform to each segment.
  - Short frame width:  
*wide-band*,  
high time resolution,  
low frequency resolution
  - Long frame width:  
*narrow-band*,  
low time resolution,  
high frequency resolution

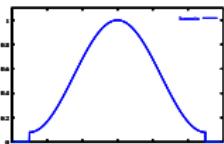


- For ASR:
  - frame width  $\sim 25ms$
  - frame shift  $\sim 10ms$

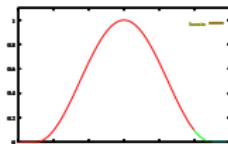
# Effect of windowing — time domain



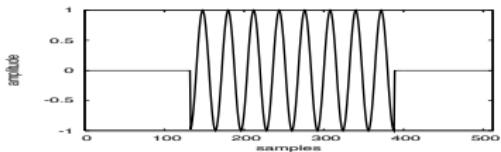
Rectangular



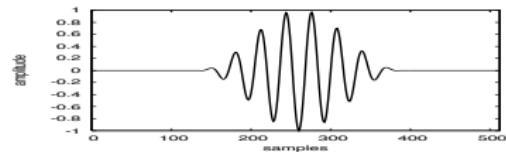
Hamming



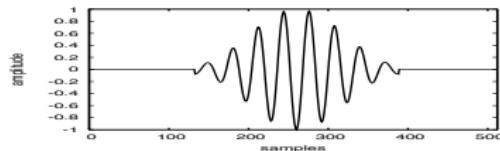
Hanning



(a) Rectangular window



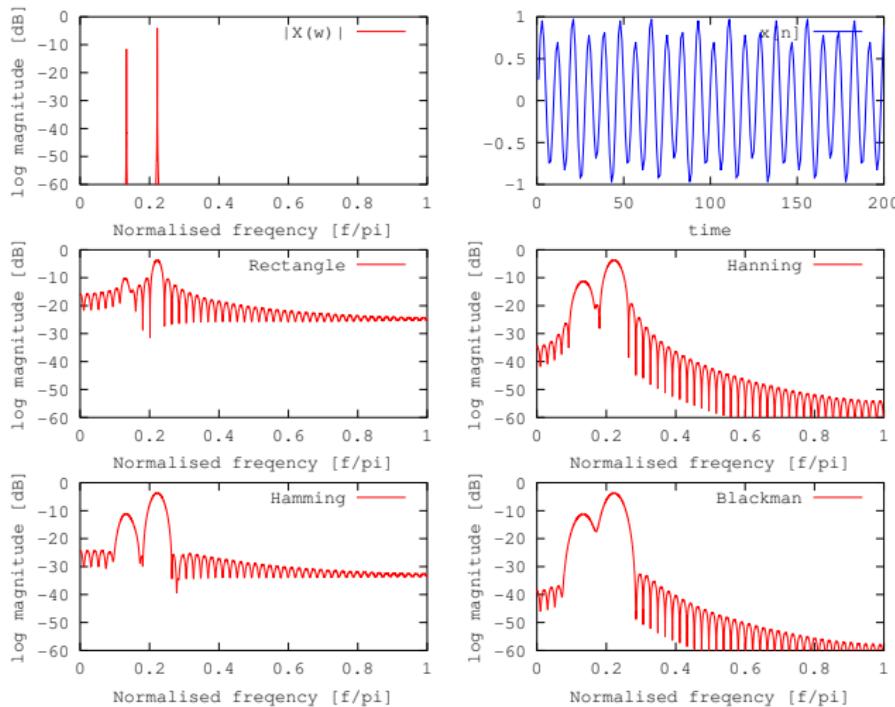
(b) Hanning window



(c) Hamming window

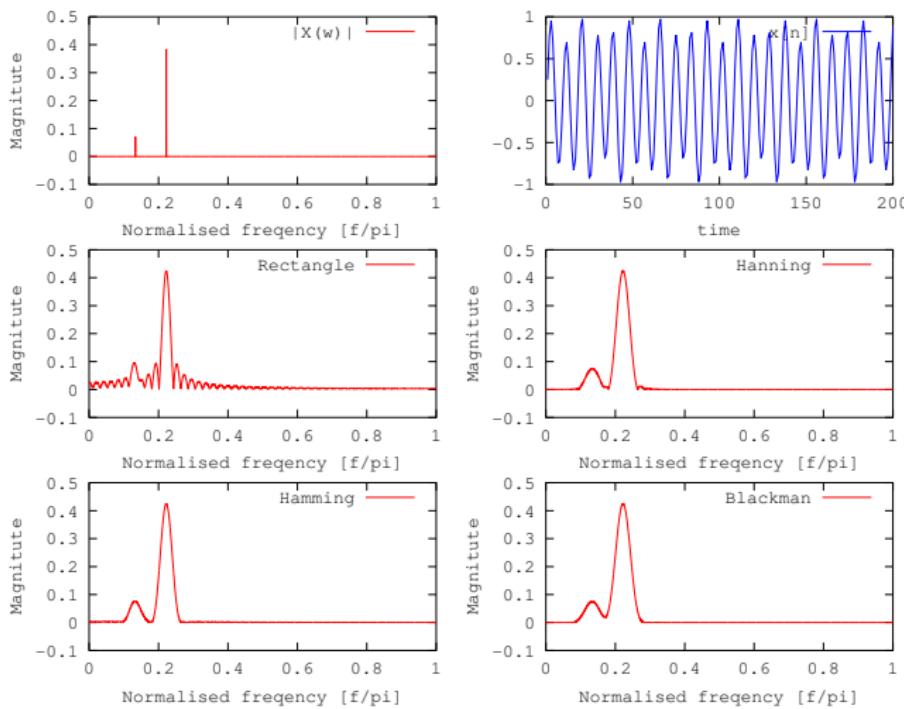
(Taylor, fig 12.1)

# Effect of windowing — frequency domain



$$x(t) = 0.15 \sin(2\pi f_1 t) + 0.85 \sin(2\pi f_2 t + 0.3)$$
$$f_1 = 0.13, f_2 = 0.22$$

# Effect of windowing — frequency domain



$$x(t) = 0.15 \sin(2\pi f_1 t) + 0.85 \sin(2\pi f_2 t + 0.3)$$
$$f_1 = 0.13, f_2 = 0.22$$

# Discrete Fourier Transform (DFT)

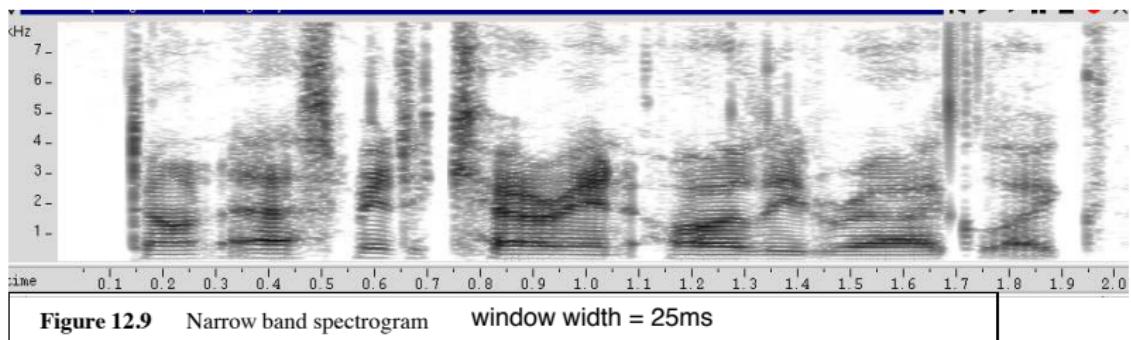
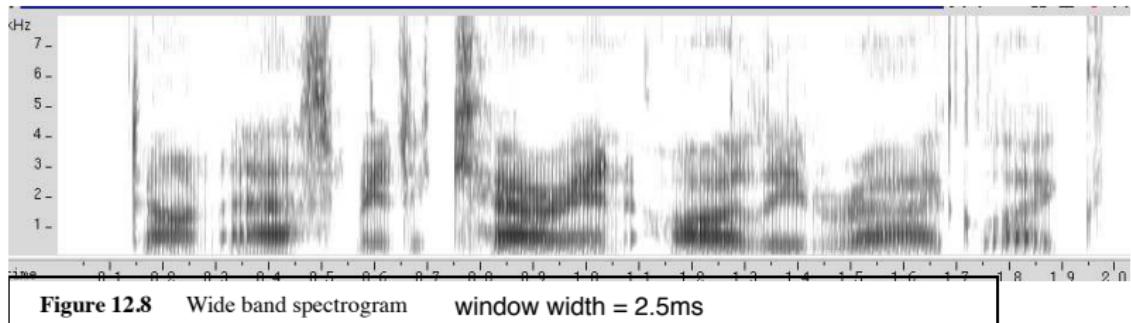
- Purpose: extracts spectral information from a windowed signal (i.e. how much energy at each frequency band)
- Input: windowed signal  $x[0], \dots, x[L-1]$  (time domain)
- Output: a complex number  $X[k]$  for each of  $N$  frequency bands representing magnitude and phase for the  $k$ th frequency component (frequency domain)
- Discrete Fourier Transform (DFT):

$$X[k] = \sum_{n=0}^{N-1} x[n] \exp\left(-j \frac{2\pi}{N} kn\right)$$

$$\text{NB: } \exp(j\theta) = e^{j\theta} = \cos(\theta) + j \sin(\theta)$$

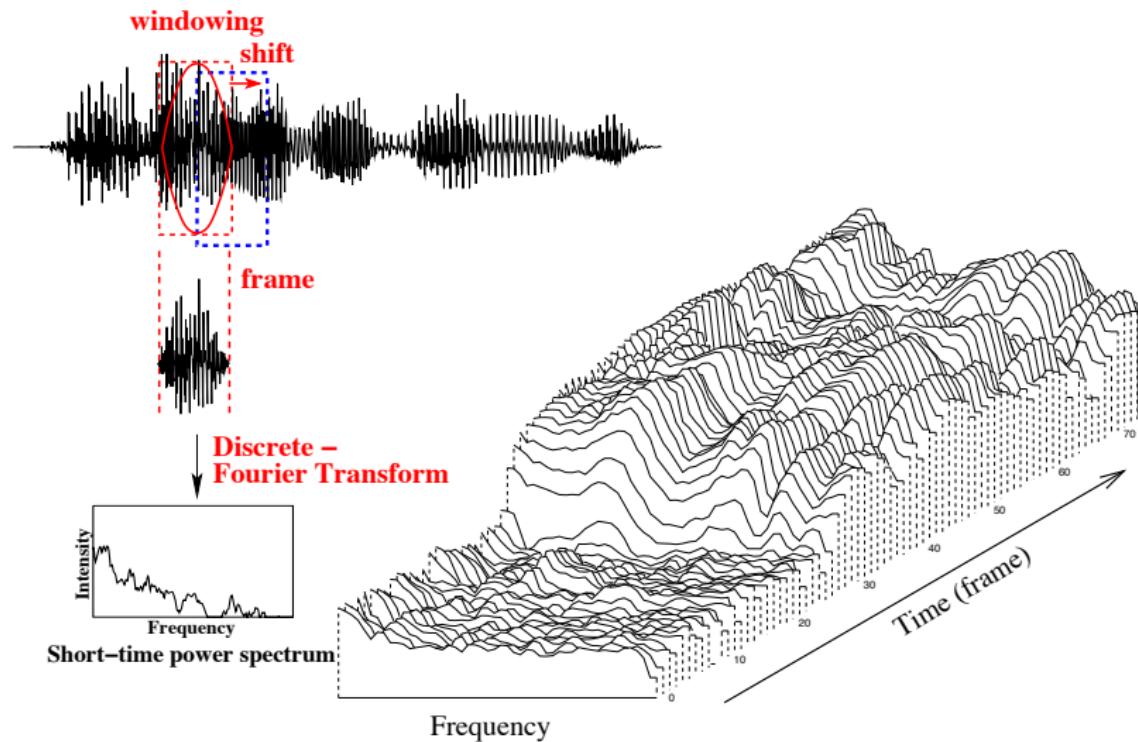
- Fast Fourier Transform (FFT) — efficient algorithm for computing DFT when  $N$  is a power of 2, and  $N \geq L$ .

# Wide-band and narrow-band spectrograms

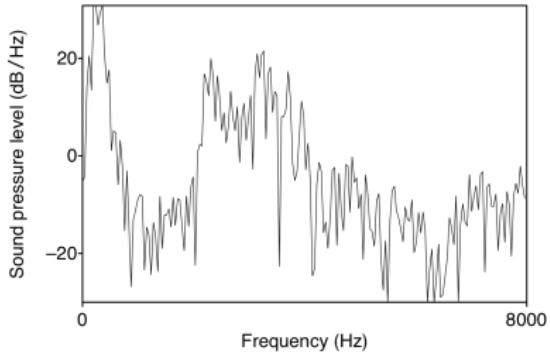
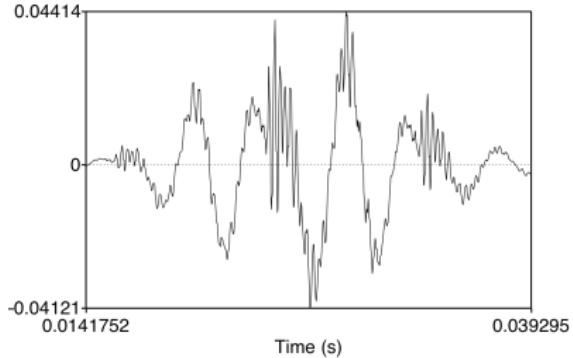


(Taylor, figs 12.8, 12.9)

# Short-time spectral analysis



# DFT Spectrum

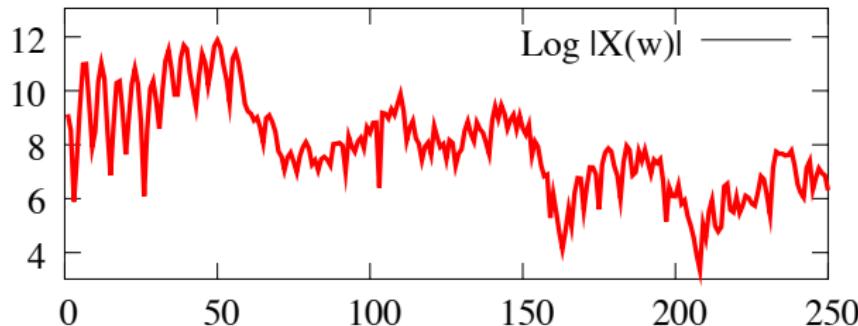


25ms Hamming window of vowel /iy/ and its spectrum computed by DFT

(Jurafsky and Martin, fig 9.12)

# DFT Spectrum Features for ASR

- Equally-spaced frequency bands — but human hearing less sensitive at higher frequencies (above  $\sim 1000\text{Hz}$ )
- The estimated power spectrum contains harmonics of F0, which makes it difficult to estimate the envelope of the spectrum



- Frequency bins of STFT are highly correlated each other, i.e. power spectrum representation is highly redundant

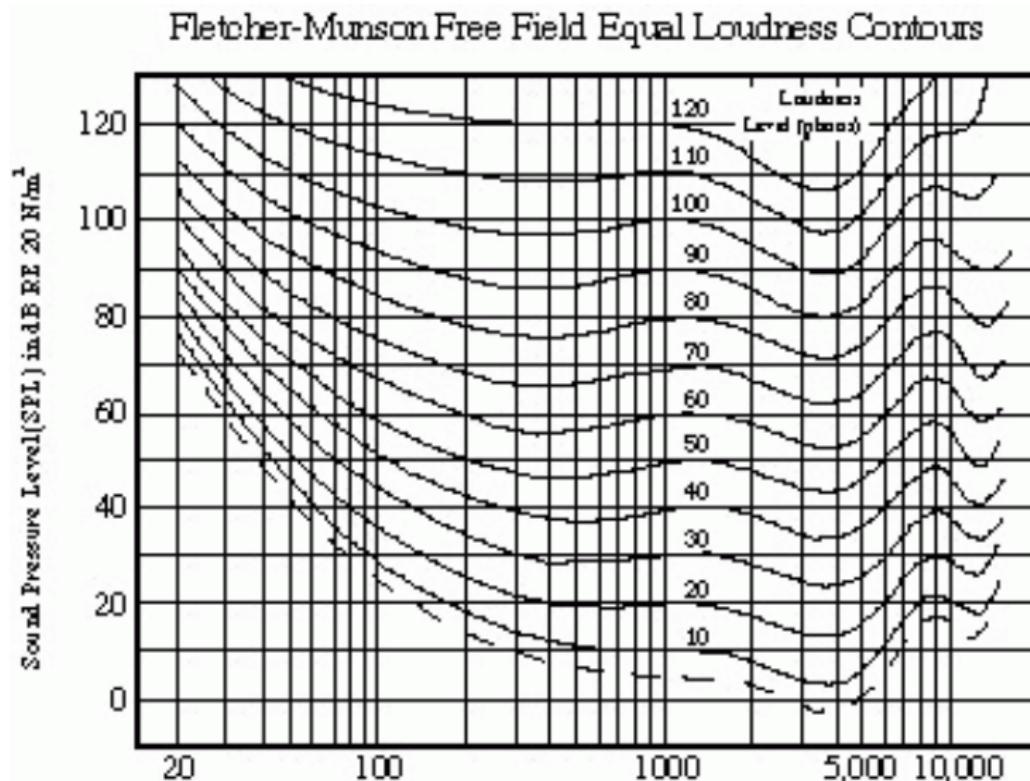
# Human hearing

Physical quality	Perceptual quality
Intensity	Loudness
Fundamental frequency	Pitch
Spectral shape	Timbre
Onset/offset time	Timing
Phase difference in binaural hearing	Location

## Technical terms

- equal-loudness contours
- masking
- auditory filters (critical-band filters)
- critical bandwidth

# Equal loudness contour

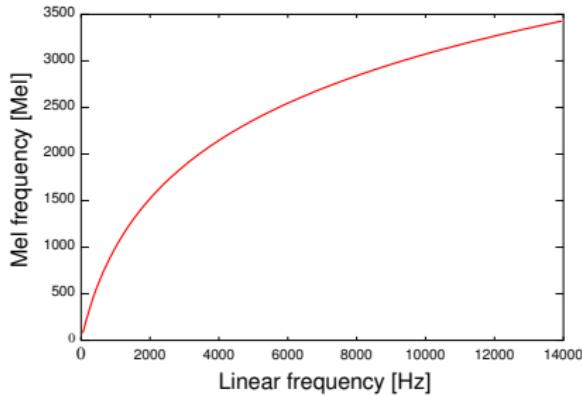


# Nonlinear frequency scaling

Human hearing is less sensitive to higher frequencies — thus human perception of frequency is nonlinear

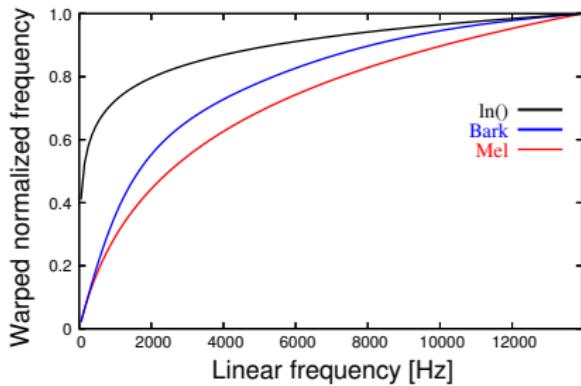
## Mel scale

$$M(f) = 1127 \ln(1 + f/700)$$



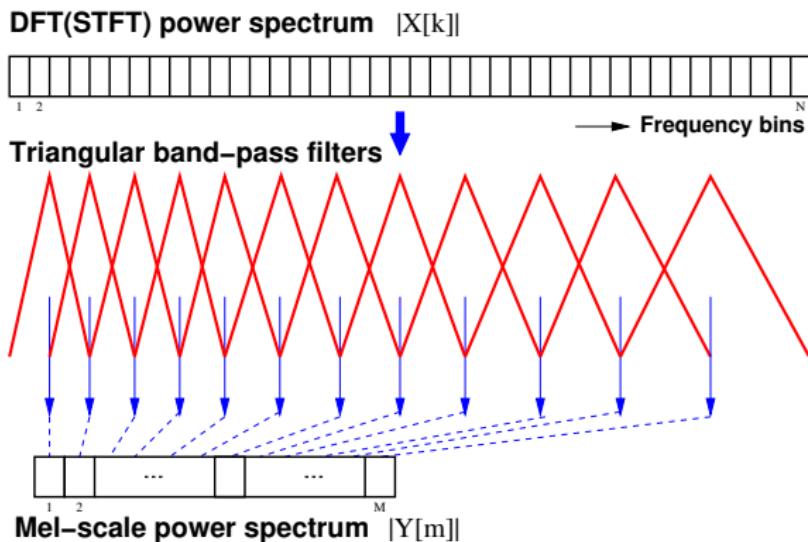
## Bark scale

$$b(f) = 13 \arctan(0.00076f) + 3.5 \arctan((f/7500)^2)$$



# Mel-Filter Bank

- Apply a mel-scale filter bank to DFT power spectrum to obtain mel-scale power spectrum
- Each filter collects energy from a number of frequency bands in the DFT
- Linearly spaced  $< 1000$  Hz, logarithmically spaced  $> 1000$  Hz



## Mel-Filter Bank *(cont.)*

$$|Y_t[m]| = \sum_{k=1}^N W_m[k] |X_t[k]|$$

where  $k$  : DFT bin number  $(1, \dots, N)$   
 $m$  : mel-filter bank number  $(1, \dots, M)$ .

- How many number of mel-filter channels?

$\approx 20$  for GMM-HMM based ASR

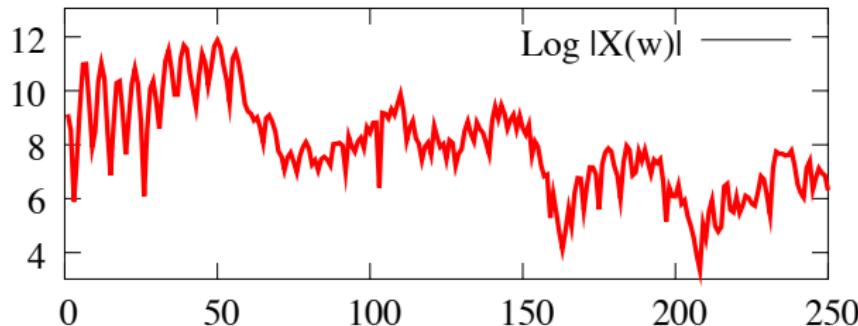
$20 \sim 40$  for DNN (+HMM) based ASR

# Log Mel Power Spectrum

- Compute the log magnitude squared of each mel-filter bank output:  $\log |Y[m]|^2$ 
  - Taking the log compresses the dynamic range
  - Human sensitivity to signal energy is logarithmic — i.e. humans are less sensitive to small changes in energy at high energy than small changes at low energy
  - Log makes features less variable to acoustic coupling variations
  - Removes phase information — not important for speech recognition (not everyone agrees with this)
- Aka “log mel-filter bank outputs” or “FBANK features”, which are widely used in recent DNN-HMM based ASR systems

# DFT Spectrum Features for ASR

- Equally-spaced frequency bands — but human hearing less sensitive at higher frequencies (above  $\sim 1000\text{Hz}$ )
- The estimated power spectrum contains harmonics of F0, which makes it difficult to estimate the envelope of the spectrum



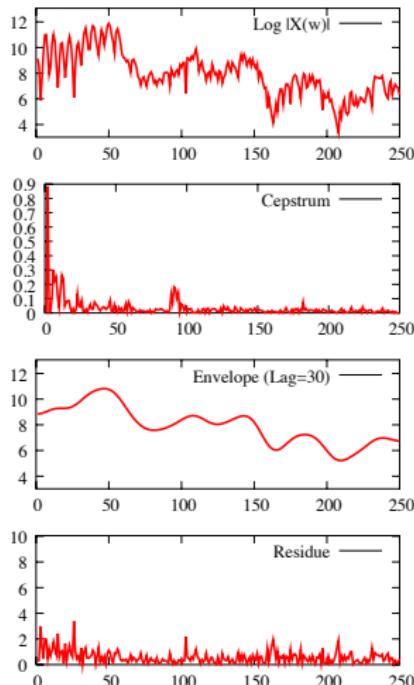
- Frequency bins of STFT are highly correlated each other, i.e. power spectrum representation is highly redundant

# Cepstral Analysis

- Source-Filter model of speech production
  - **Source:** Vocal cord vibrations create a glottal source waveform
  - **Filter:** Source waveform is passed through the vocal tract: position of tongue, jaw, etc. give it a particular shape and hence a particular filtering characteristic
- Source characteristics ( $F_0$ , dynamics of glottal pulse) do not help to discriminate between phones
- The filter specifies the position of the articulators
- ... and hence is directly related to phone discrimination
- Cepstral analysis enables us to separate source and filter

# Cepstral Analysis

Split power spectrum into spectral envelope and  $F_0$  harmonics.



Log spectrum (freq domain)

↓ Inverse Fourier Transform

Cepstrum (time domain) (quefrency)

↓ Liftering to get low/high part  
(lifter: filter used in cepstral domain)

↓ Fourier Transform

Smoothed log spectrum (freq domain)

[low-part of cepstrum]

+

Fine structure

[high-part of cepstrum]

# The Cepstrum

- Cepstrum obtained by applying inverse DFT to log magnitude spectrum (may be mel-scaled)
- Cepstrum is time-domain (we talk about quefrency)
- Inverse DFT:

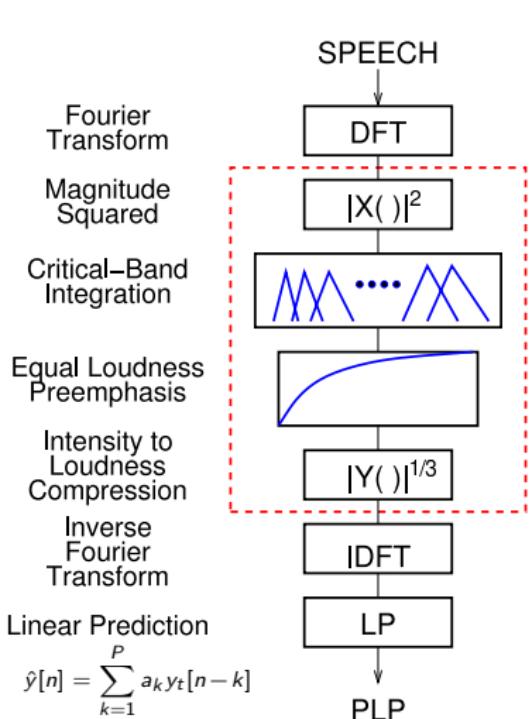
$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \exp\left(j \frac{2\pi}{N} nk\right)$$

- Since log power spectrum is real and symmetric the inverse DFT is equivalent to a discrete cosine transform (DCT)

$$y_t[n] = \sum_{m=0}^{M-1} \log(|Y_t[m]|) \cos\left(n(m+0.5)\frac{\pi}{M}\right), \quad n = 0, \dots, J$$

- Smoothed spectrum: transform to cepstral domain, truncate, transform back to spectral domain
- Mel-frequency cepstral coefficients (MFCCs): use the cepstral coefficients directly
  - Widely used as acoustic features in HMM-based ASR
  - First 12 MFCCs are often used as the feature vector (removes F0 information)
  - Less correlated than spectral features — easier to model than spectral features
  - Very compact representation — 12 features describe a 20ms frame of data
  - For standard HMM-based systems, MFCCs result in better ASR performance than filter bank or spectrogram features
  - MFCCs are not robust against noise

# PLP — Perceptual Linear Prediction



- PLP (Hermansky, JASA 1990)
  - Uses equal loudness pre-emphasis and cube-root compression (motivated by perceptual results) rather than log compression
  - Uses linear predictive auto-regressive modelling to obtain cepstral coefficients
  - PLP has been shown to lead to
    - slightly better ASR accuracy
    - slightly better noise robustness
- compared with MFCCs

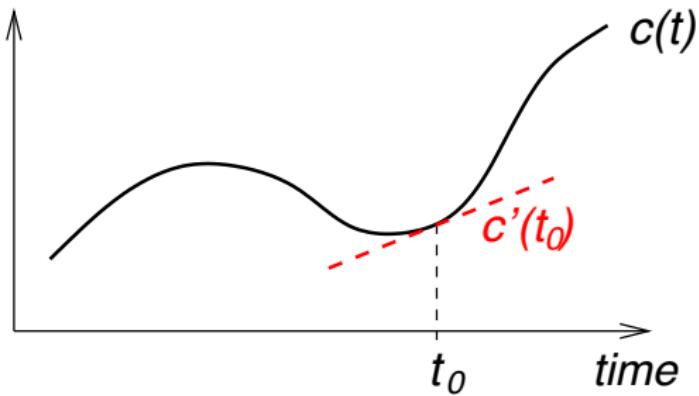
## Dynamic features

- Speech is not constant frame-to-frame, so we can add features to do with how the cepstral coefficients change over time
- $\Delta*$ ,  $\Delta^2*$  are delta features (dynamic features / time derivatives)
- Simple calculation of delta features  $d(t)$  at time  $t$  for cepstral feature  $c(t)$  (e.g.  $y_t[j]$ ):

$$d(t) = \frac{c(t+1) - c(t-1)}{2}$$

- More sophisticated approach estimates the temporal derivative by using regression to estimate the slope (typically using 4 frames each side)
- “Standard” ASR features (for GMM-based systems) are 39 dimensions:
  - 12 MFCCs, and energy
  - 12  $\Delta$ MFCCs,  $\Delta$ energy
  - 12  $\Delta^2$ MFCCs,  $\Delta^2$ energy

# Estimating dynamic features



# Feature Transforms

- Orthogonal transformation (orthogonal bases)
  - **DCT** (discrete cosine transform)
  - **PCA** (principal component analysis)
- Transformation based on the bases that maximises the separability between classes.
  - **LDA** (linear discriminant analysis) / Fisher's linear discriminant
  - **HLDA** (heteroscedastic linear discriminant analysis)

# Feature Normalisation

- **Basic Idea:** Transform the features to reduce mismatch between training and test
- *Cepstral Mean Normalisation* (CMN): subtract the average feature value from each feature, so each feature has a mean value of 0. makes features robust to some linear filtering of the signal (channel variation)
- *Cepstral Variance Normalisation* (CVN): Divide feature vector by standard deviation of feature vectors, so each feature vector element has a variance of 1
- Cepstral mean and variance normalisation, CMN/CVN:

$$\hat{y}_t[j] = \frac{y_t[j] - \mu(y[j])}{\sigma(y[j])}$$

- Compute mean and variance statistics over longest available segments with the same speaker/channel
- Real time normalisation: compute a moving average

# Acoustic features in state-of-the-art ASR systems

See Tables 1, 2, and 3 in

*Jinyu Li, Dong Yu, Jui-Ting Huang, and Yifan Gong,*  
"Improving Wideband Speech Recognition Using Mixed-Bandwidth  
Training Data In CD-DNN-HMM",  
2012 IEEE Workshop in Spoken Language Technology (SLT2012).  
<http://research-srv.microsoft.com/pubs/179159/li.pdf>

# Summary: Speech Signal Analysis for ASR

- Good characteristics of ASR features
- FBANK features
  - Short-time DFT analysis
  - Mel-filter bank
  - Log magnitude squared
  - Widely used for DNN ASR ( $M \approx 40$ )
- MFCCs - mel frequency cepstral coefficients
  - FBANK features
  - Inverse DFT (DCT)
  - Use first few (12) coefficients
  - Widely used for GMM-HMM ASR
- Delta features (dynamic features)
- 39-dimension feature vector (for GMM-HMM ASR):  
MFCC-12 + energy; + Deltas; + Delta-Deltas

# Hidden Markov Models and Gaussian Mixture Models

Hiroshi Shimodaira and Steve Renals

Automatic Speech Recognition— ASR Lectures 4&5  
25&29 January 2018

## HMMs and GMMs

- Key models and algorithms for HMM acoustic models
- Gaussians
- GMMs: Gaussian mixture models
- HMMs: Hidden Markov models
- HMM algorithms
  - Likelihood computation (forward algorithm)
  - Most probable state sequence (Viterbi algorithm)
  - Estimating the parameters (EM algorithm)

# Fundamental Equation of Statistical Speech Recognition

If  $\mathbf{X}$  is the sequence of acoustic feature vectors (observations) and  $\mathbf{W}$  denotes a word sequence, the most likely word sequence  $\mathbf{W}^*$  is given by

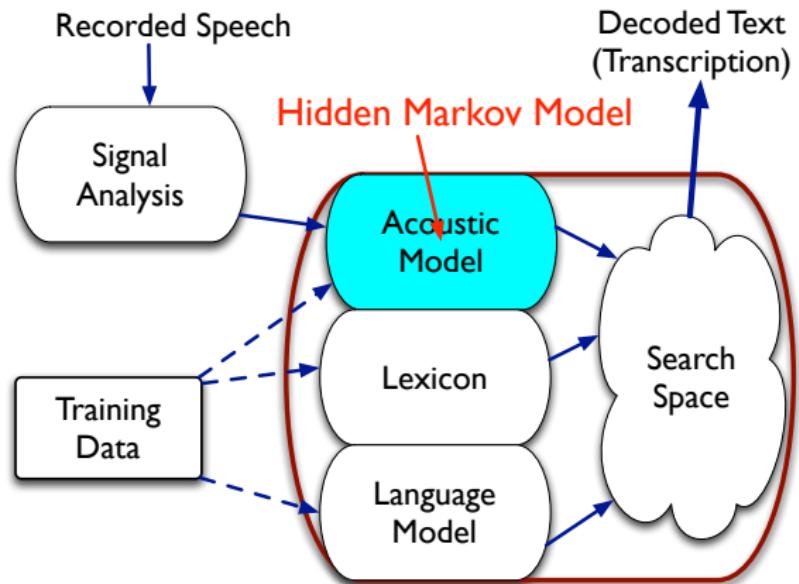
$$\mathbf{W}^* = \arg \max_{\mathbf{W}} P(\mathbf{W} | \mathbf{X})$$

Applying Bayes' Theorem:

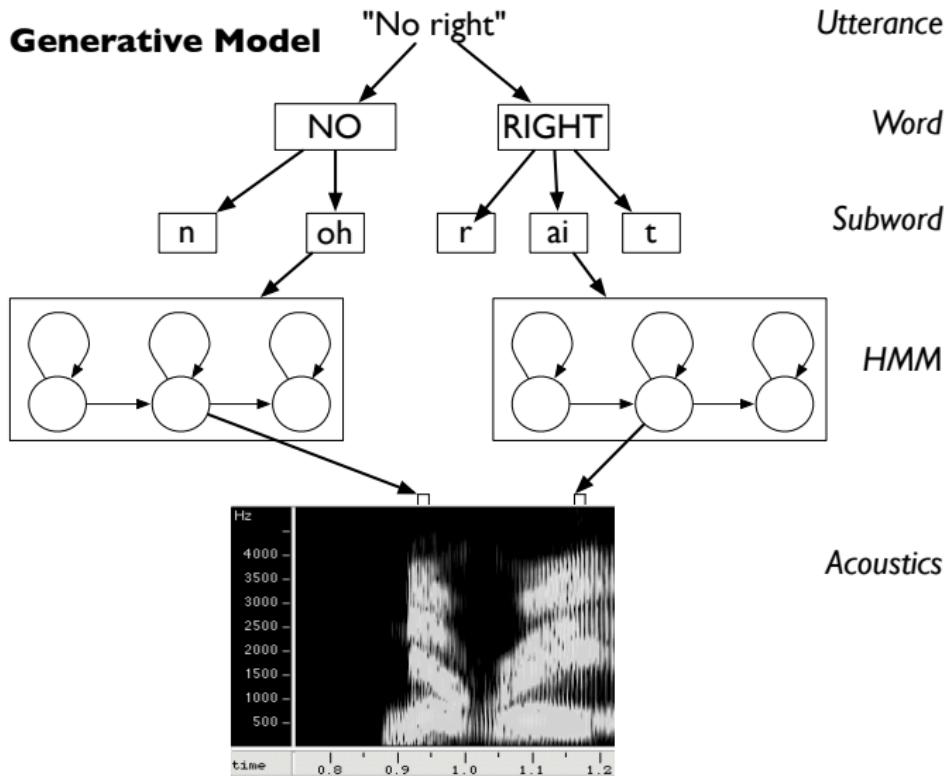
$$\begin{aligned} P(\mathbf{W} | \mathbf{X}) &= \frac{p(\mathbf{X} | \mathbf{W}) P(\mathbf{W})}{p(\mathbf{X})} \\ &\propto p(\mathbf{X} | \mathbf{W}) P(\mathbf{W}) \\ \mathbf{W}^* &= \arg \max_{\mathbf{W}} \underbrace{p(\mathbf{X} | \mathbf{W})}_{\text{Acoustic model}} \underbrace{P(\mathbf{W})}_{\text{Language model}} \end{aligned}$$

NB:  $\mathbf{X}$  is used hereafter to denote the output feature vectors from the signal analysis module rather than DFT spectrum.

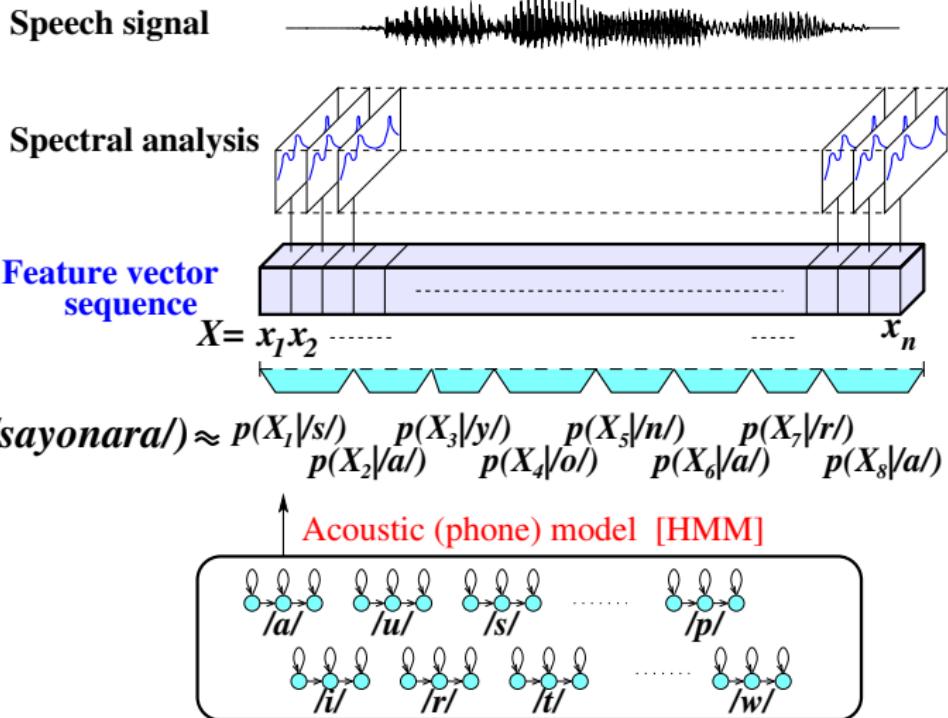
# Acoustic Modelling



# Hierarchical modelling of speech



# Calculation of $p(X|W)$



NB: some conditional independency is assumed here.

## How to calculate $p(X_1|s/)$ ?

Assume  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{T_1}$  corresponds to phoneme /s/,  
the *conditional probability* that we observe the sequence is

$$p(X_1|s/) = p(\mathbf{x}_1, \dots, \mathbf{x}_{T_1}|s/), \quad \mathbf{x}_i = (x_{1i}, \dots, x_{Di})^t \in \mathcal{R}^d$$

We know that HMM can be employed to calculate this. (*Viterbi* algorithm, *Forward* / *Backward* algorithm)

To grasp the idea of probability calculation, let's consider an **extremely simple case** where the length of input sequence is just one ( $T_1 = 1$ ), and the dimensionality of  $\mathbf{x}$  is one ( $d = 1$ ), so that we don't need HMM.

$$p(X_1|s/) \longrightarrow p(x_1|s/)$$

## How to calculate $p(X_1|s/)$ ? (cont.)

$p(x|s/)$  : conditional probability  
(conditional probability density function (*pdf*) of  $x$ )

- A *Gaussian / normal distribution* function could be employed for this:

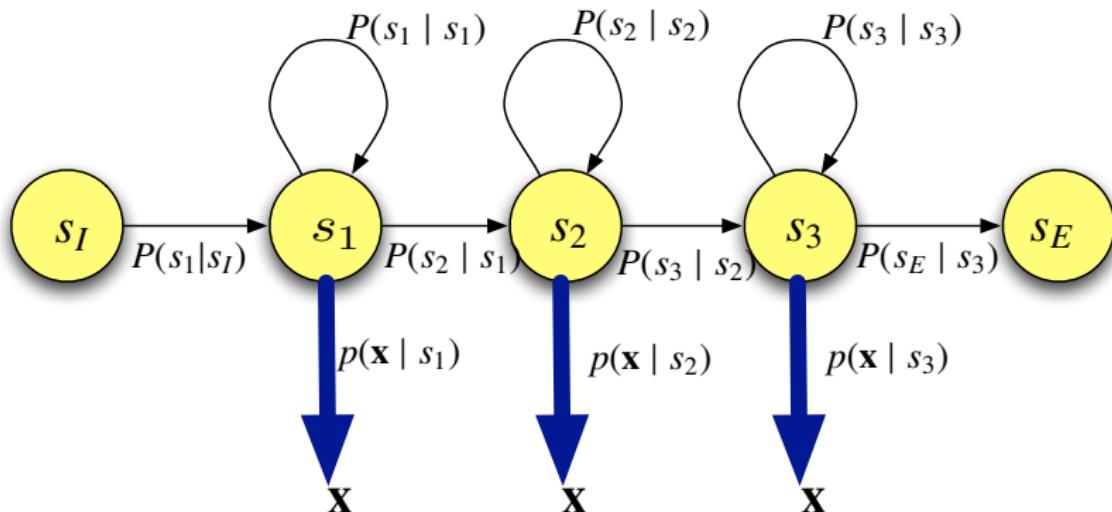
$$P(x|s/) = \frac{1}{\sqrt{2\pi\sigma_s^2}} e^{-\frac{(x-\mu_s)^2}{2\sigma_s^2}}$$

- The function has only two parameters,  $\mu_s$  and  $\sigma_s^2$
- Given a set of training samples  $\{x_1, \dots, x_N\}$ , we can estimate  $\mu_s$  and  $\sigma_s$

$$\hat{\mu}_s = \frac{1}{N} \sum_{i=1}^N x_i, \quad \hat{\sigma}_s^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{\mu}_s)^2$$

- For a general case where a phone lasts more than one frame, we need to employ HMM.

# Acoustic Model: Continuous Density HMM



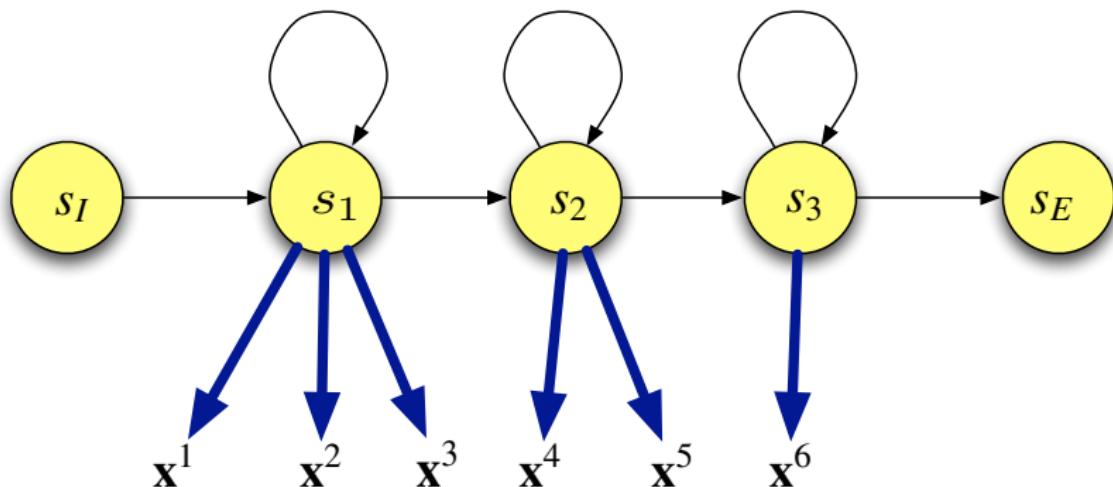
Probabilistic finite state automaton

Parameters  $\lambda$ :

- Transition probabilities:  $a_{kj} = P(S=j | S=k)$
- Output probability density function:  $b_j(\mathbf{x}) = p(\mathbf{x} | S=j)$

NB: Some textbooks use  $Q$  or  $q$  to denote the state variable  $S$ .  
 $x$  corresponds to  $\mathbf{o}_t$  in Lecture slides 02.

# Acoustic Model: Continuous Density HMM



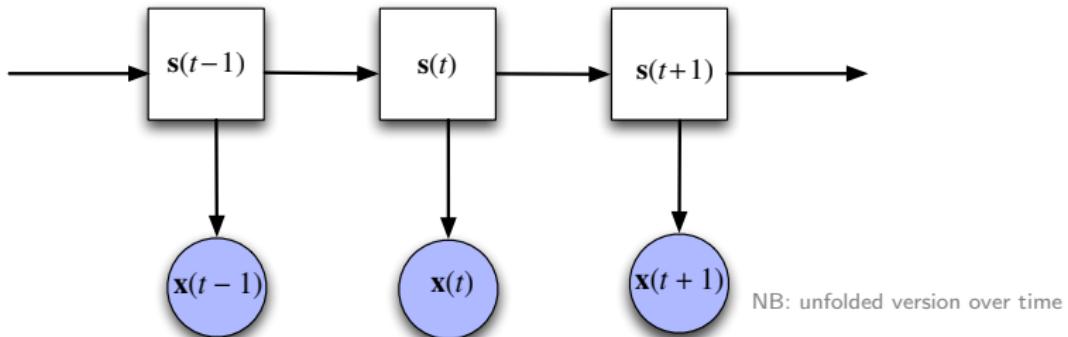
Probabilistic finite state automaton

Parameters  $\lambda$ :

- Transition probabilities:  $a_{kj} = P(S=j | S=k)$
- Output probability density function:  $b_j(\mathbf{x}) = p(\mathbf{x} | S=j)$

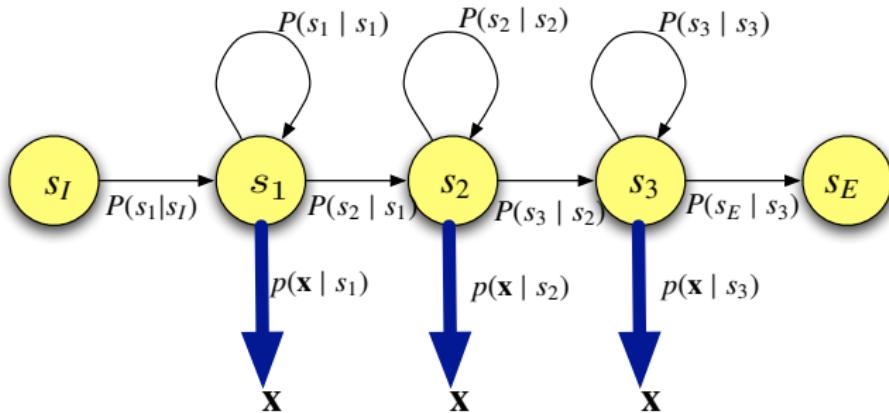
NB: Some textbooks use  $Q$  or  $q$  to denote the state variable  $S$ .  
 $x$  corresponds to  $\mathbf{o}_t$  in Lecture slides 02.

# HMM Assumptions



- ➊ **Markov process:** The probability of a state depends only on the previous state:  $P(S(t)|S(t-1), S(t-2), \dots, S(1)) = P(S(t)|S(t-1))$   
A state is conditionally independent of all other states given the previous state
- ➋ **Observation independence:** The output observation  $x(t)$  depends only on the state that produced the observation:  
 $p(x(t)|S(t), S(t-1), \dots, S(1), x(t-1), \dots, x(1)) = p(x(t)|S(t))$   
An acoustic observation  $x$  is conditionally independent of all other observations given the state that generated it

# Output distribution



- Single multivariate Gaussian with mean  $\mu_j$ , covariance matrix  $\Sigma_j$ :

$$b_j(\mathbf{x}) = p(\mathbf{x} | S=j) = \mathcal{N}(\mathbf{x}; \mu_j, \Sigma_j)$$

- $M$ -component Gaussian mixture model:

$$b_j(\mathbf{x}) = p(\mathbf{x} | S=j) = \sum_{m=1}^M c_{jm} \mathcal{N}(\mathbf{x}; \mu_{jm}, \Sigma_{jm})$$

- Neural network:

$$b_j(\mathbf{x}) \sim P(S=j | \mathbf{x}) / P(S=j) \quad \text{NB: NN outputs posterior probabilities}$$

## Background: cdf

Consider a real valued random variable  $X$

- Cumulative distribution function (cdf)  $F(x)$  for  $X$ :

$$F(x) = P(X \leq x)$$

- To obtain the probability of falling in an interval we can do the following:

$$\begin{aligned}P(a < X \leq b) &= P(X \leq b) - P(X \leq a) \\&= F(b) - F(a)\end{aligned}$$

## Background: pdf

- The rate of change of the cdf gives us the *probability density function* (pdf),  $p(x)$ :

$$p(x) = \frac{d}{dx} F(x) = F'(x)$$
$$F(x) = \int_{-\infty}^x p(x) dx$$

- $p(x)$  is **not** the probability that  $X$  has value  $x$ . But the pdf is proportional to the probability that  $X$  lies in a small interval centred on  $x$ .
- Notation:  $p$  for pdf,  $P$  for probability

# The Gaussian distribution (univariate)

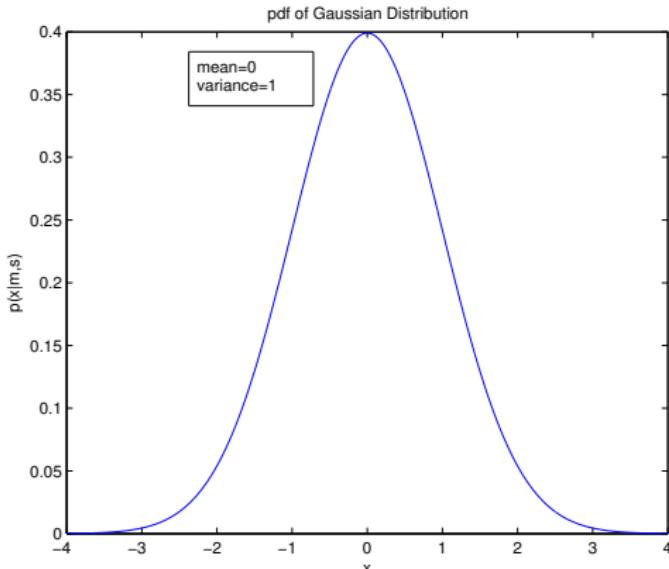
- The **Gaussian** (or **Normal**) distribution is the most common (and easily analysed) continuous distribution
- It is also a reasonable model in many situations (the famous “bell curve” )
- If a (scalar) variable has a Gaussian distribution, then it has a probability density function with this form:

$$p(x|\mu, \sigma^2) = \mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$

- The Gaussian is described by two parameters:
  - the mean  $\mu$  (location)
  - the variance  $\sigma^2$  (dispersion)

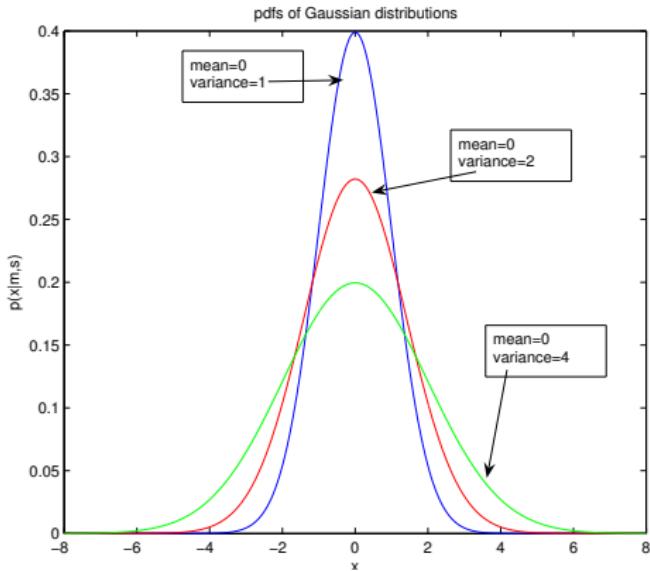
# Plot of Gaussian distribution

- Gaussians have the same shape, with the location controlled by the mean, and the spread controlled by the variance
- One-dimensional Gaussian with zero mean and unit variance ( $\mu = 0$ ,  $\sigma^2 = 1$ ):



# Properties of the Gaussian distribution

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$



## Parameter estimation

- Estimate mean and variance parameters of a Gaussian from data  $x_1, x_2, \dots, x_T$
- Use the following as the estimates:

$$\hat{\mu} = \frac{1}{T} \sum_{t=1}^T x_t \quad (\text{mean})$$

$$\hat{\sigma}^2 = \frac{1}{T} \sum_{t=1}^T (x_t - \hat{\mu})^2 \quad (\text{variance})$$

## Exercise — maximum likelihood estimation (MLE)

Consider the log likelihood of a set of  $T$  training data points  $\{x_1, \dots, x_T\}$  being generated by a Gaussian with mean  $\mu$  and variance  $\sigma^2$ :

$$\begin{aligned} L = \ln p(\{x_1, \dots, x_T\} | \mu, \sigma^2) &= -\frac{1}{2} \sum_{t=1}^T \left( \frac{(x_t - \mu)^2}{\sigma^2} - \ln \sigma^2 - \ln(2\pi) \right) \\ &= -\frac{1}{2\sigma^2} \sum_{t=1}^T (x_t - \mu)^2 - \frac{T}{2} \ln \sigma^2 - \frac{T}{2} \ln(2\pi) \end{aligned}$$

By maximising the the log likelihood function with respect to  $\mu$  show that the maximum likelihood estimate for the mean is indeed the sample mean:

$$\mu_{ML} = \frac{1}{T} \sum_{t=1}^T x_t.$$

# The multivariate Gaussian distribution

- The  $D$ -dimensional vector  $\mathbf{x} = (x_1, \dots, x_D)^T$  follows a multivariate Gaussian (or normal) distribution if it has a probability density function of the following form:

$$p(\mathbf{x} | \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$$

The pdf is parameterised by the mean vector  $\boldsymbol{\mu} = (\mu_1, \dots, \mu_D)^T$

and the covariance matrix  $\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{11} & \dots & \sigma_{1D} \\ \vdots & \ddots & \vdots \\ \sigma_{D1} & \dots & \sigma_{DD} \end{pmatrix}$ .

- The 1-dimensional Gaussian is a special case of this pdf
- The argument to the exponential  $0.5(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})$  is referred to as a *quadratic form*.

# Covariance matrix

- The mean vector  $\mu$  is the expectation of  $\mathbf{x}$ :

$$\mu = E[\mathbf{x}]$$

- The covariance matrix  $\Sigma$  is the expectation of the deviation of  $\mathbf{x}$  from the mean:

$$\Sigma = E[(\mathbf{x} - \mu)(\mathbf{x} - \mu)^T]$$

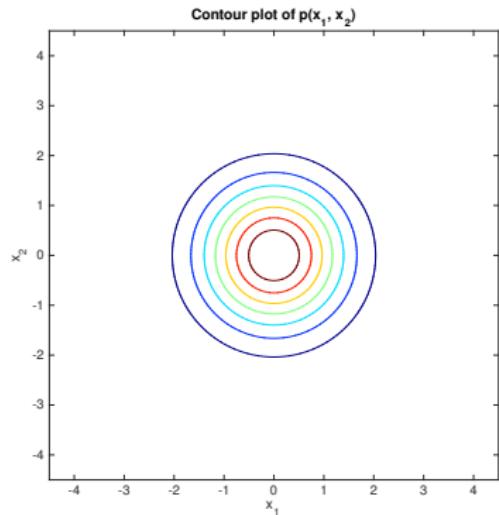
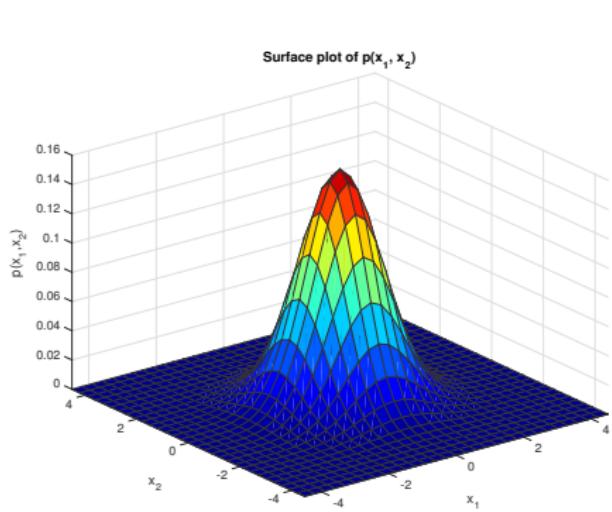
- $\Sigma$  is a  $D \times D$  symmetric matrix:

$$\sigma_{ij} = E[(x_i - \mu_i)(x_j - \mu_j)] = E[(x_j - \mu_j)(x_i - \mu_i)] = \sigma_{ji}$$

- The sign of the covariance helps to determine the relationship between two components:

- If  $x_j$  is large when  $x_i$  is large, then  $(x_i - \mu_i)(x_j - \mu_j)$  will tend to be positive;
- If  $x_j$  is small when  $x_i$  is large, then  $(x_i - \mu_i)(x_j - \mu_j)$  will tend to be negative.

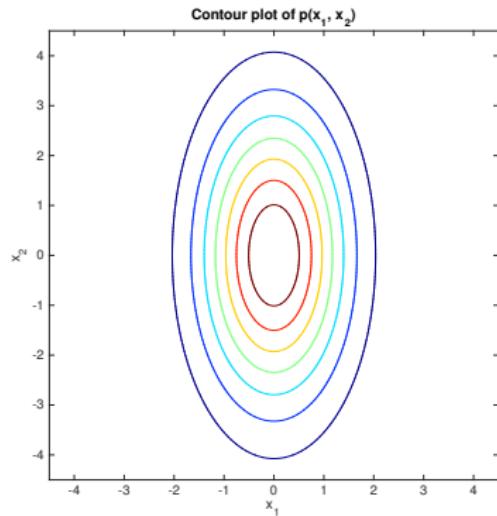
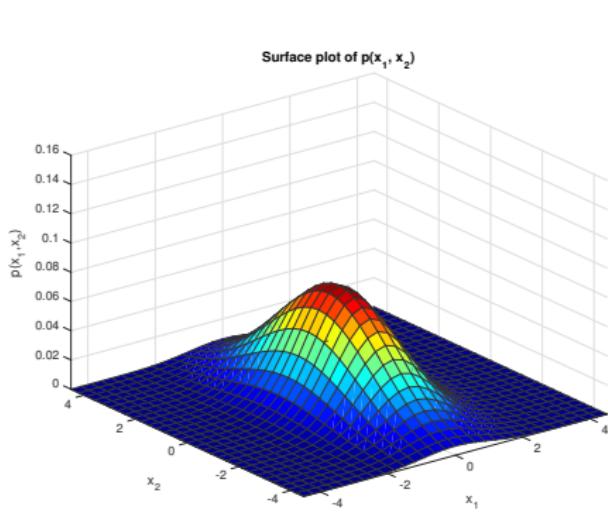
# Spherical Gaussian



$$\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \rho_{12} = 0$$

NB: Correlation coefficient  $\rho_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}} \quad (-1 \leq \rho_{ij} \leq 1)$

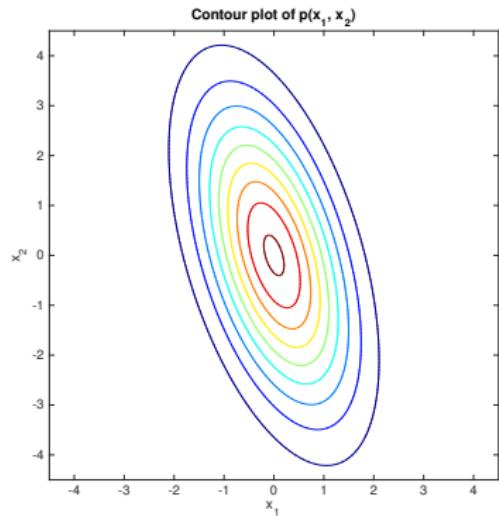
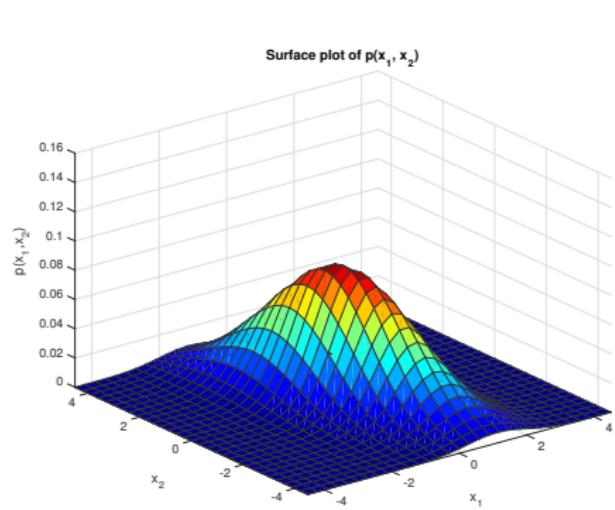
# Diagonal Covariance Gaussian



$$\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix} \quad \rho_{12} = 0$$

NB: Correlation coefficient  $\rho_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}} \quad (-1 \leq \rho_{ij} \leq 1)$

# Full covariance Gaussian



$$\boldsymbol{\mu} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & -1 \\ -1 & 4 \end{pmatrix} \quad \rho_{12} = -0.5$$

NB: Correlation coefficient  $\rho_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}} \quad (-1 \leq \rho_{ij} \leq 1)$

# Parameter estimation of a multivariate Gaussian distribution

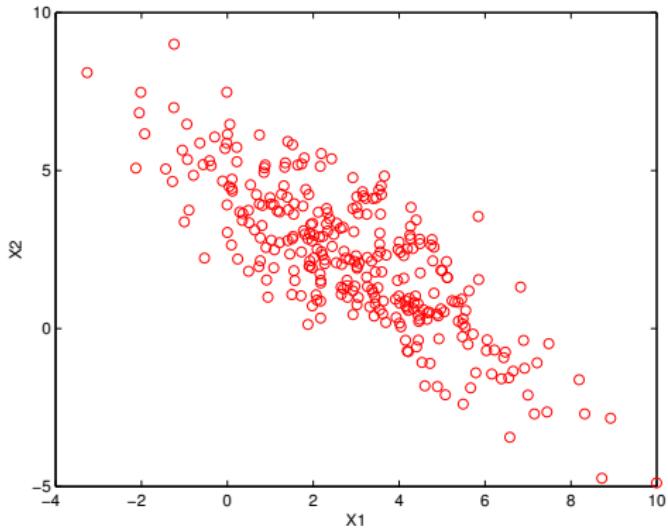
- It is possible to show that the mean vector  $\hat{\mu}$  and covariance matrix  $\hat{\Sigma}$  that maximise the likelihood of the training data are given by:

$$\hat{\mu} = \frac{1}{T} \sum_{t=1}^T \mathbf{x}_t$$
$$\hat{\Sigma} = \frac{1}{T} \sum_{t=1}^T (\mathbf{x}_t - \hat{\mu})(\mathbf{x}_t - \hat{\mu})^T$$

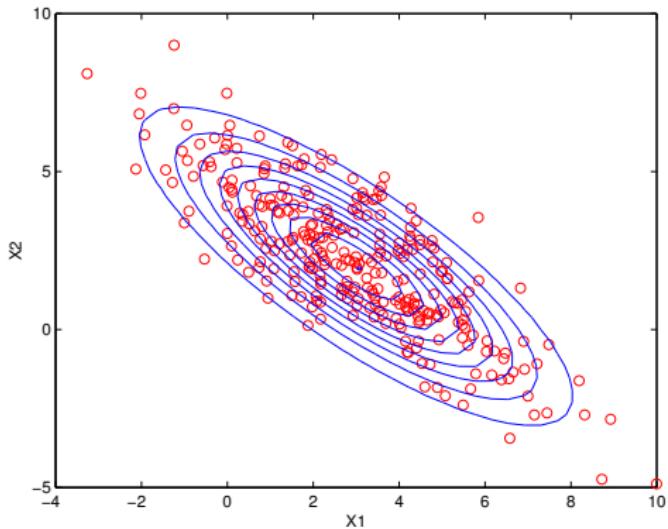
where  $\mathbf{x}_t = (x_{t1}, \dots, x_{tD})^T$ .

NB:  $T$  denotes either the number of samples or vector transpose depending on context.

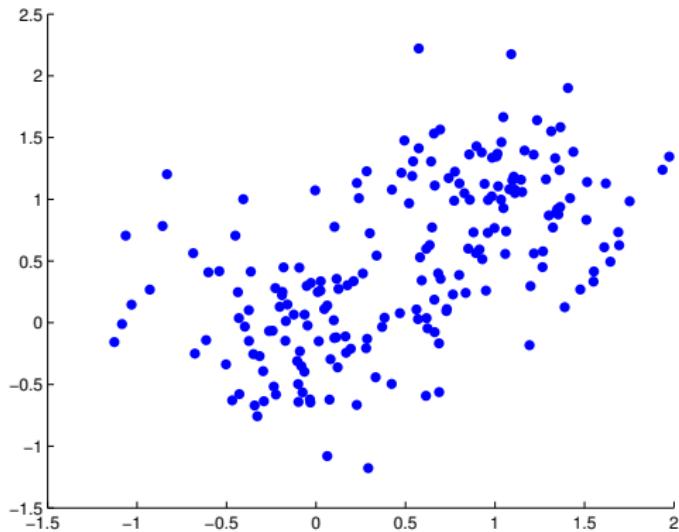
# Example data



# Maximum likelihood fit to a Gaussian

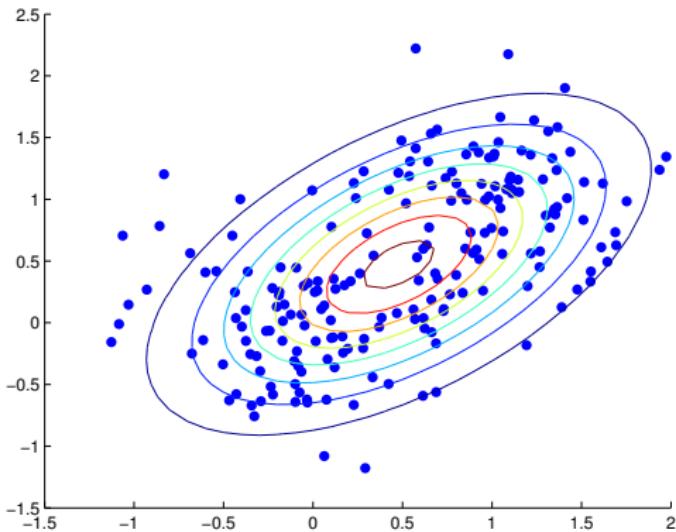


## Data in clusters (example 1)



$$\mu_1 = (0, 0)^T \quad \mu_2 = (1, 1)^T \quad \Sigma_1 = \Sigma_2 = 0.2 \mathbf{I}$$

## Example 1 fit by a Gaussian

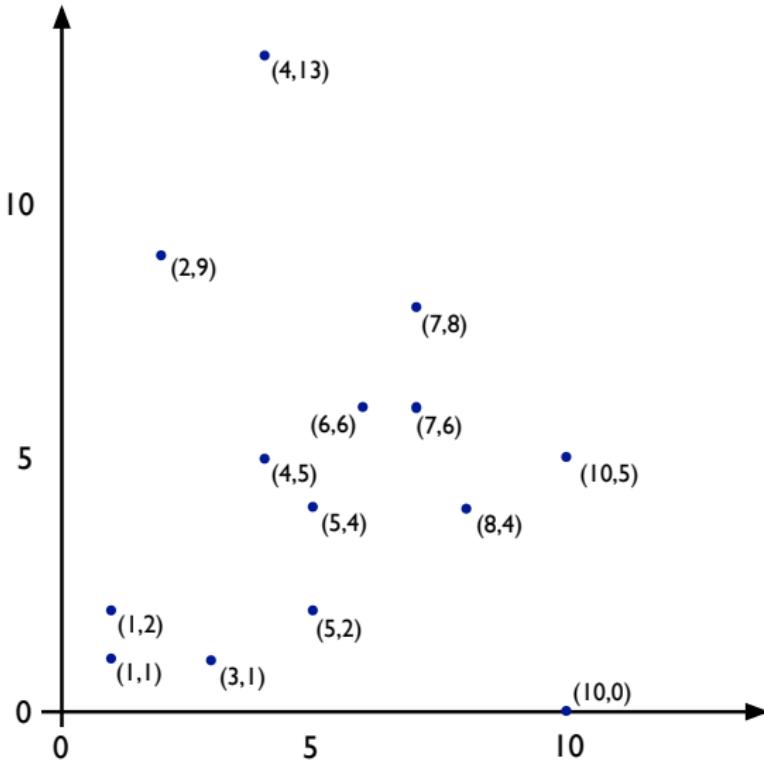


$$\mu_1 = (0, 0)^T \quad \mu_2 = (1, 1)^T \quad \Sigma_1 = \Sigma_2 = 0.2 \mathbf{I}$$

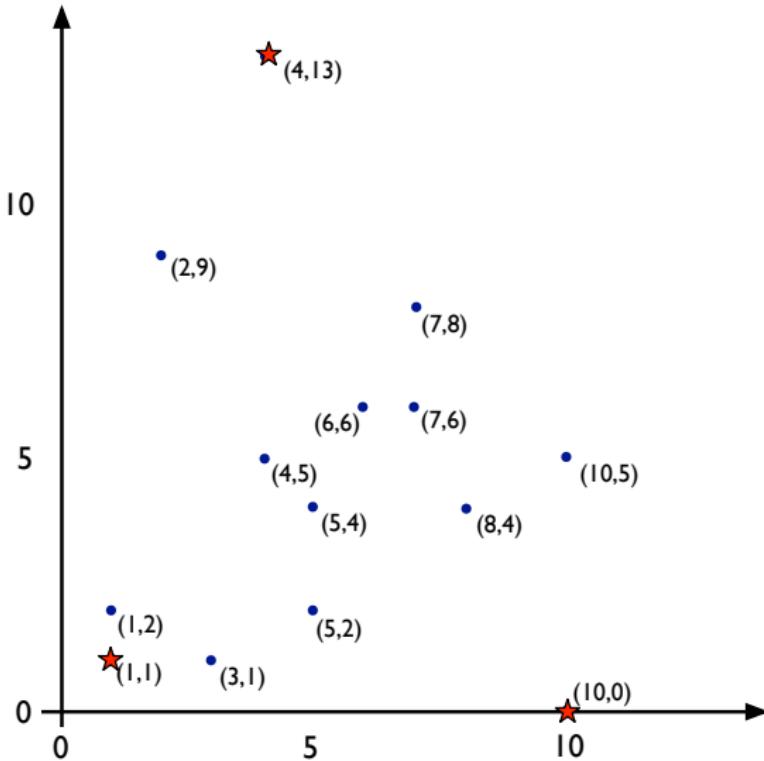
# k-means clustering

- k-means is an automatic procedure for clustering unlabelled data
- Requires a prespecified number of clusters
- Clustering algorithm chooses a set of clusters with the minimum within-cluster variance
- Guaranteed to converge (eventually)
- Clustering solution is dependent on the initialisation

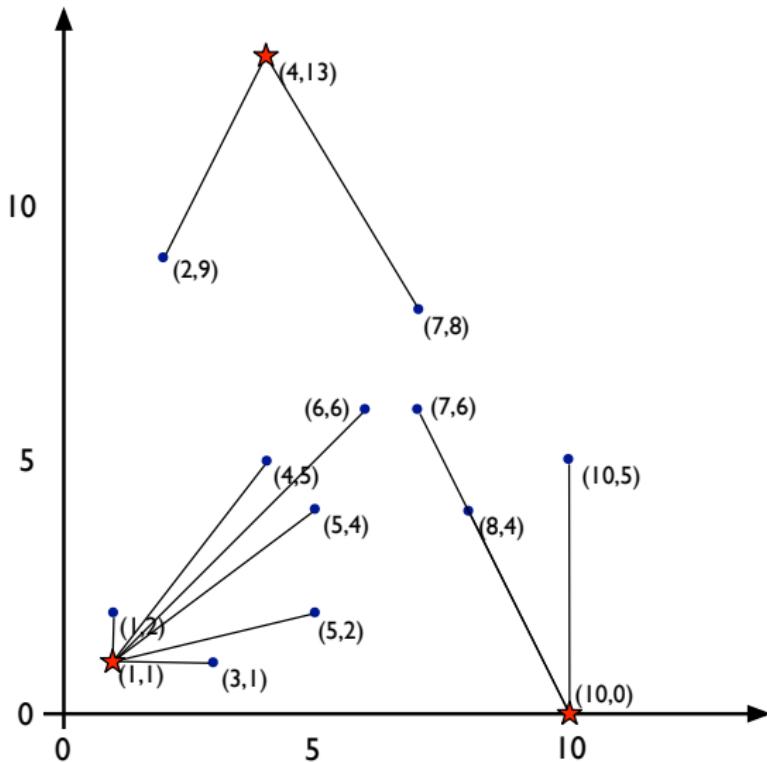
## k-means example: data set



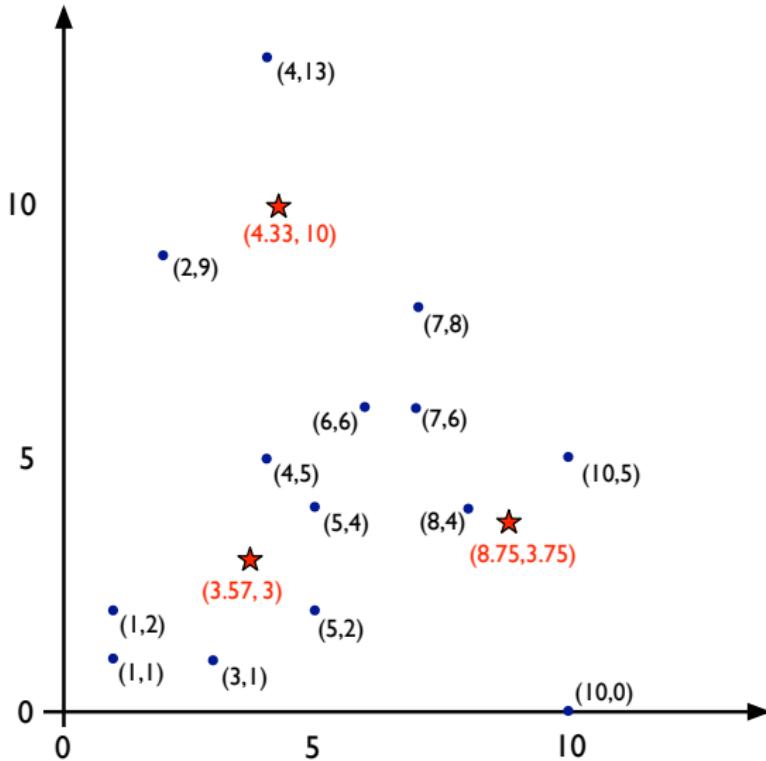
## k-means example: initialisation



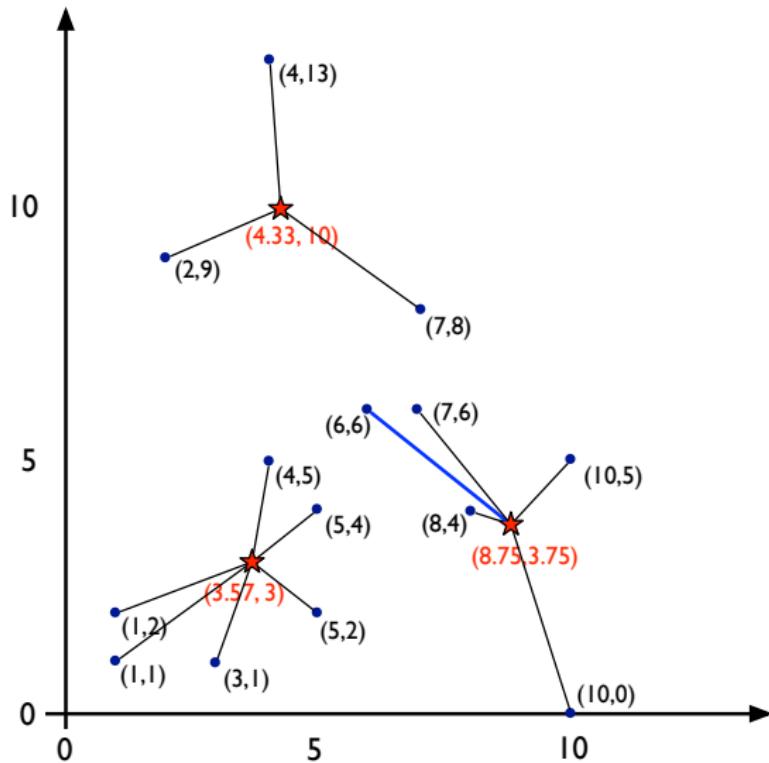
## k-means example: iteration 1 (assign points to clusters)



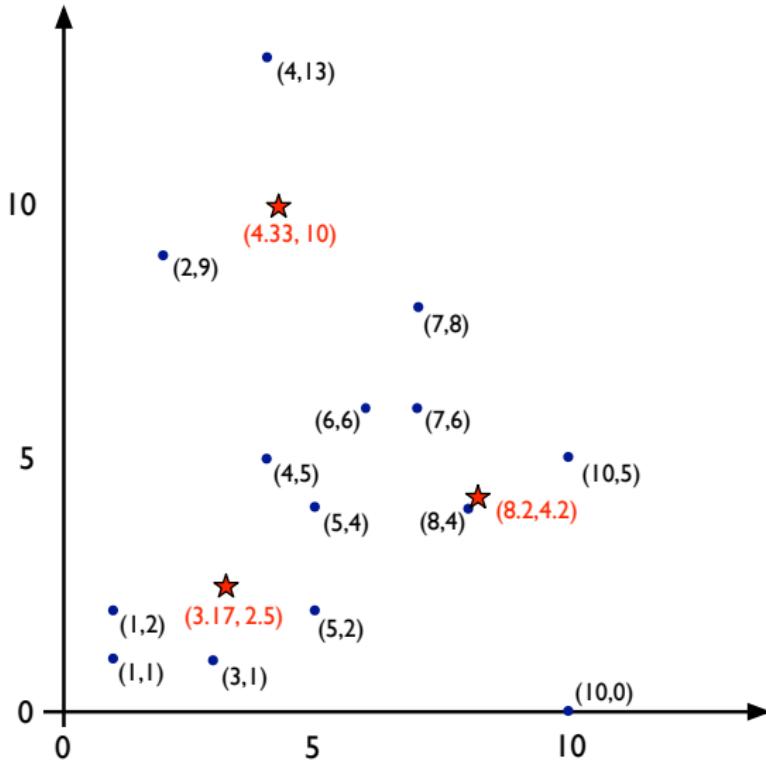
## k-means example: iteration 1 (recompute centres)



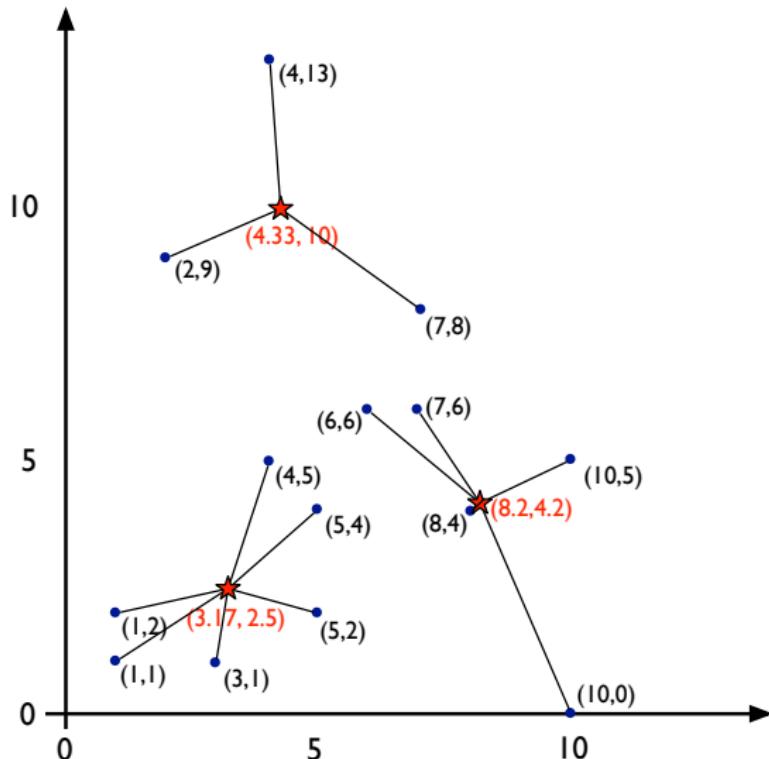
## k-means example: iteration 2 (assign points to clusters)



## k-means example: iteration 2 (recompute centres)



## k-means example: iteration 3 (assign points to clusters)



No changes, so converged

# Mixture model

- A more flexible form of density estimation is made up of a linear combination of component densities:

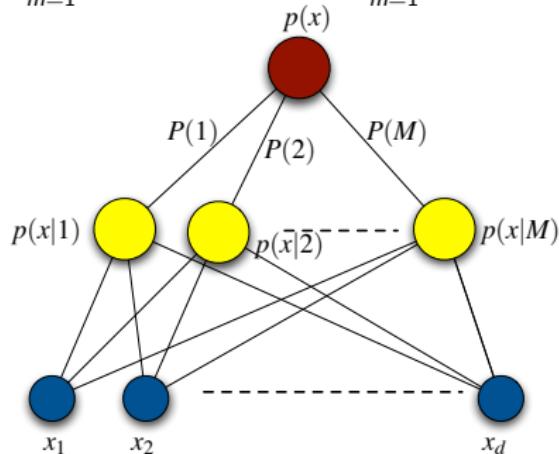
$$p(\mathbf{x}) = \sum_{m=1}^M P(m)p(\mathbf{x} | m)$$

- This is called a *mixture model* or a *mixture density*
- $p(\mathbf{x} | m)$  : component densities
- $P(m)$  : mixing parameters
- Generative model:
  - ① Choose a mixture component based on  $P(m)$
  - ② Generate a data point  $\mathbf{x}$  from the chosen component using  $p(\mathbf{x} | m)$

# Gaussian mixture model

- The most important mixture model is the *Gaussian Mixture Model* (GMM), where the component densities are Gaussians
- Consider a GMM, where each component Gaussian  $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$  has mean  $\boldsymbol{\mu}_m$  and a **spherical covariance**  $\boldsymbol{\Sigma}_m = \sigma_m^2 \mathbf{I}$

$$p(\mathbf{x}) = \sum_{m=1}^M P(m) p(\mathbf{x} | m) = \sum_{m=1}^M P(m) \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_m, \sigma_m^2 \mathbf{I})$$



## GMM Parameter estimation when we know which component generated the data

- Define the indicator variable  $z_{mt} = 1$  if component  $m$  generated data point  $\mathbf{x}_t$  (and 0 otherwise)
- If  $z_{mt}$  wasn't hidden then we could count the number of observed data points generated by  $m$ :

$$N_m = \sum_{t=1}^T z_{mt}$$

- And estimate the mean, variance and mixing parameters as:

$$\hat{\mu}_m = \frac{\sum_t z_{mt} \mathbf{x}_t}{N_m}$$

$$\hat{\sigma}_m^2 = \frac{\sum_t z_{mt} \|\mathbf{x}_t - \hat{\mu}_m\|^2}{N_m}$$

$$\hat{P}(m) = \frac{1}{T} \sum_t z_{mt} = \frac{N_m}{T}$$

## GMM Parameter estimation when we don't know which component generated the data

- **Problem:** we don't know  $z_{mt}$  - which mixture component a data point comes from...
- Idea: use the posterior probability  $P(m|\mathbf{x})$ , which gives the probability that component  $m$  was responsible for generating data point  $\mathbf{x}$ .

$$P(m|\mathbf{x}) = \frac{p(\mathbf{x}|m) P(m)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|m) P(m)}{\sum_{m'=1}^M p(\mathbf{x}|m') P(m')}$$

- The  $P(m|\mathbf{x})$ s are called the *component occupation probabilities* (or sometimes called the *responsibilities*)
- Since they are posterior probabilities:

$$\sum_{m=1}^M P(m|\mathbf{x}) = 1$$

# Soft assignment

- Estimate “*soft counts*” based on the component occupation probabilities  $P(m|\mathbf{x}_t)$ :

$$N_m^* = \sum_{t=1}^T P(m|\mathbf{x}_t)$$

- We can imagine assigning data points to component  $m$  weighted by the component occupation probability  $P(m|\mathbf{x}_t)$
- So we could imagine estimating the mean, variance and prior probabilities as:

$$\hat{\mu}_m = \frac{\sum_t P(m|\mathbf{x}_t) \mathbf{x}_t}{\sum_t P(m|\mathbf{x}_t)} = \frac{\sum_t P(m|\mathbf{x}_t) \mathbf{x}_t}{N_m^*}$$

$$\hat{\sigma}_m^2 = \frac{\sum_t P(m|\mathbf{x}_t) \|\mathbf{x}_t - \hat{\mu}_m\|^2}{\sum_t P(m|\mathbf{x}_t)} = \frac{\sum_t P(m|\mathbf{x}_t) \|\mathbf{x}_t - \hat{\mu}_m\|^2}{N_m^*}$$

$$\hat{P}(m) = \frac{1}{T} \sum_t P(m|\mathbf{x}_t) = \frac{N_m^*}{T}$$

# EM algorithm

- **Problem!** Recall that:

$$P(m|\mathbf{x}) = \frac{p(\mathbf{x}|m)P(m)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|m)P(m)}{\sum_{m'=1}^M p(\mathbf{x}|m')P(m')}$$

We need to know  $p(\mathbf{x}|m)$  and  $P(m)$  to estimate the parameters of  $P(m|\mathbf{x})$ , and to estimate  $P(m)$ ....

- Solution: an iterative algorithm where each iteration has two parts:
  - Compute the component occupation probabilities  $P(m|\mathbf{x})$  using the current estimates of the GMM parameters (means, variances, mixing parameters) (E-step)
  - Computer the GMM parameters using the current estimates of the component occupation probabilities (M-step)
- Starting from some initialisation (e.g. using k-means for the means) these steps are alternated until convergence
- This is called the **EM Algorithm** and can be shown to maximise the likelihood. (NB: local maximum rather than global)

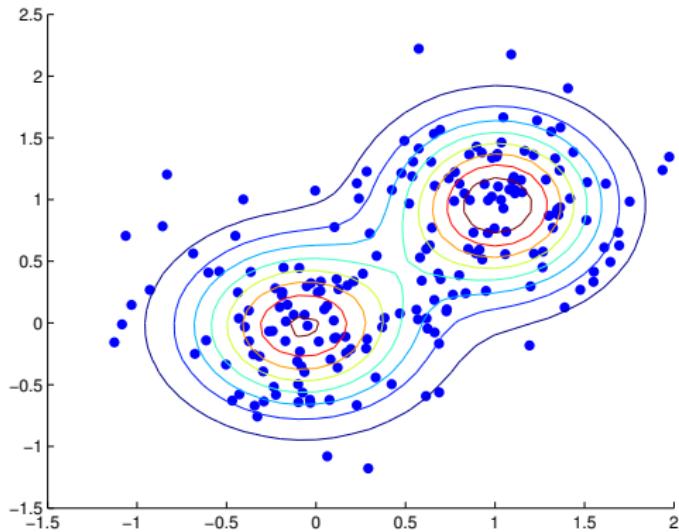
# Maximum likelihood parameter estimation

- The likelihood of a data set  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\}$  is given by:

$$\mathcal{L} = \prod_{t=1}^T p(\mathbf{x}_t) = \prod_{t=1}^T \sum_{m=1}^M p(\mathbf{x}_t | m) P(m)$$

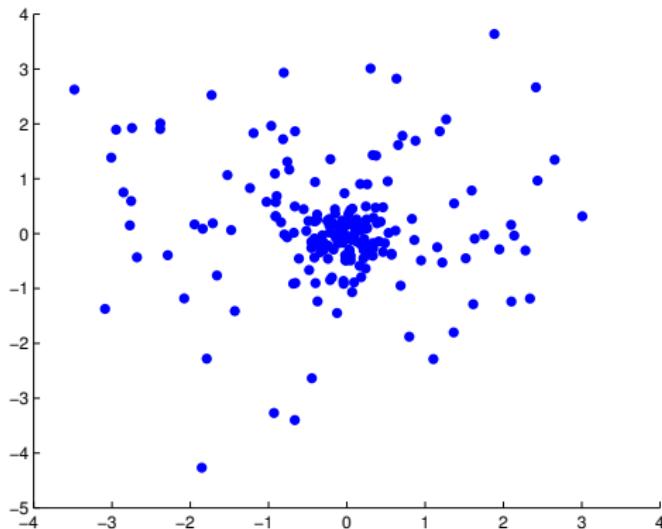
- We can regard the *negative log likelihood* as an error function:
- Considering the derivatives of  $E$  with respect to the parameters, gives expressions like the previous slide

## Example 1 fit using a GMM



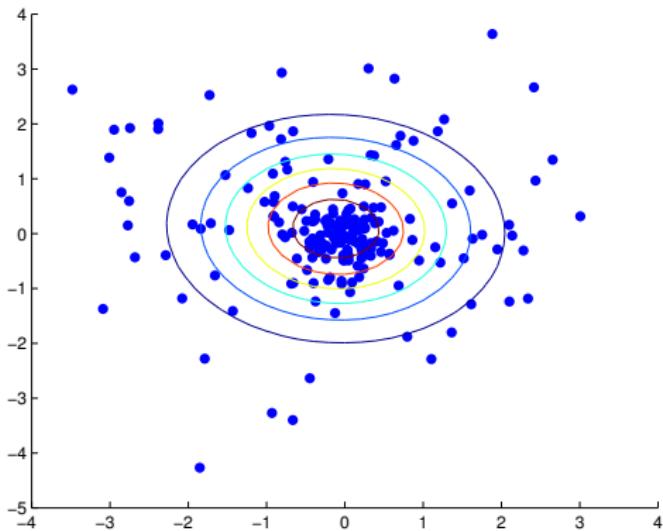
Fitted with a two component GMM using EM

## Peakily distributed data (Example 2)



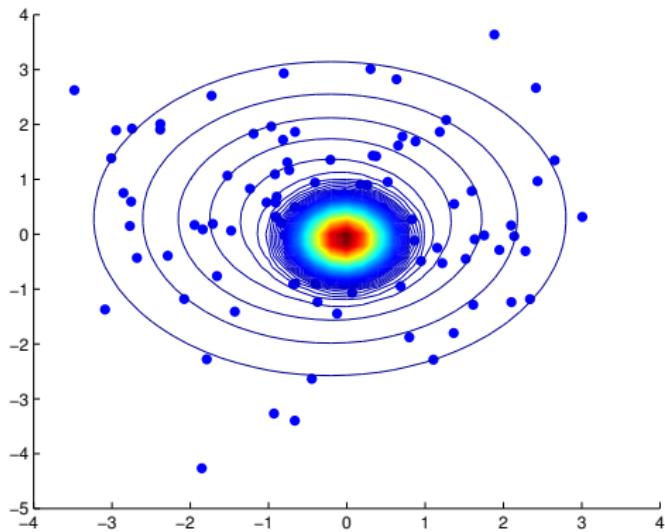
$$\mu_1 = \mu_2 = [0 \quad 0]^T \quad \Sigma_1 = 0.1\mathbf{I} \quad \Sigma_2 = 2\mathbf{I}$$

## Example 2 fit by a Gaussian



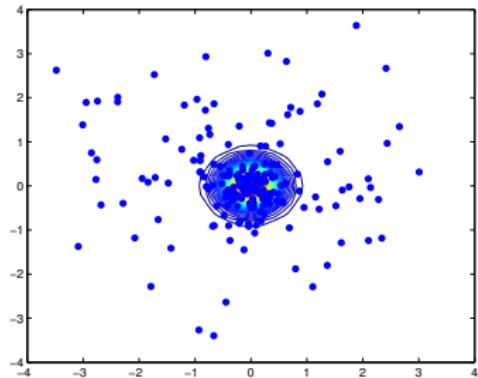
$$\mu_1 = \mu_2 = [0 \quad 0]^T \quad \Sigma_1 = 0.1\mathbf{I} \quad \Sigma_2 = 2\mathbf{I}$$

## Example 2 fit by a GMM

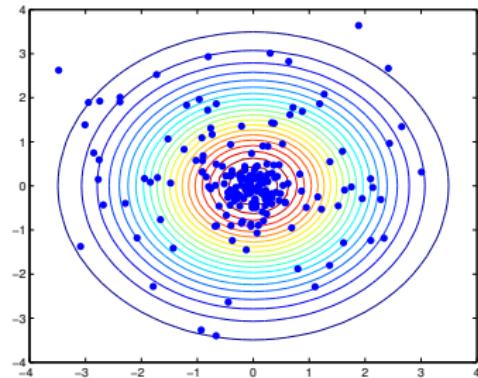


Fitted with a two component GMM using EM

## Example 2: component Gaussians



$$P(\mathbf{x} \mid m=1)$$

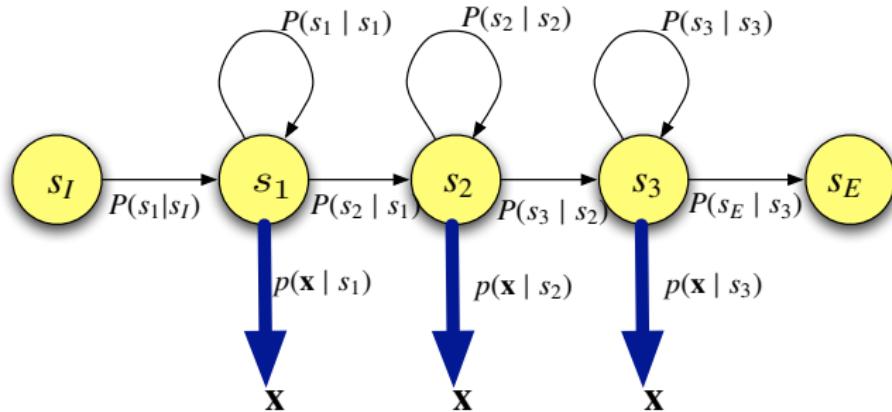


$$P(\mathbf{x} \mid m=2)$$

## Comments on GMMs

- GMMs trained using the EM algorithm are able to self organise to fit a data set
- Individual components take responsibility for parts of the data set (probabilistically)
- Soft assignment to components not hard assignment — “soft clustering”
- GMMs scale very well, e.g.: large speech recognition systems can have 30,000 GMMs, each with 32 components: sometimes 1 million Gaussian components!! And the parameters all estimated from (a lot of) data by EM

# Back to HMMs...



Output distribution:

- Single multivariate Gaussian with mean  $\mu_j$ , covariance matrix  $\Sigma_j$ :

$$b_j(\mathbf{x}) = p(\mathbf{x} | S=j) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

- $M$ -component Gaussian mixture model:

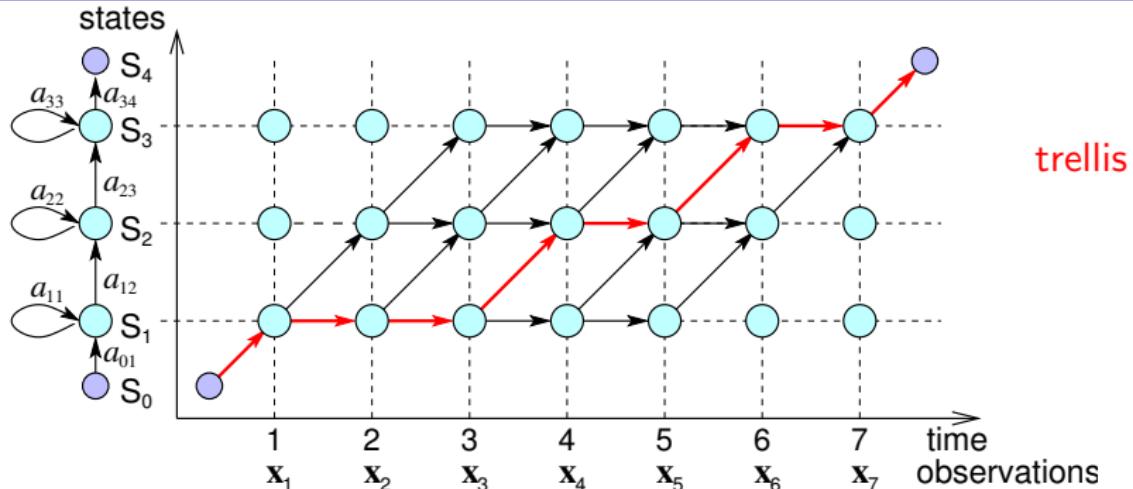
$$b_j(\mathbf{x}) = p(\mathbf{x} | S=j) = \sum_{m=1}^M c_{jm} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})$$

# The three problems of HMMs

Working with HMMs requires the solution of three problems:

- ① **Likelihood** Determine the overall likelihood of an observation sequence  $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_t, \dots, \mathbf{x}_T)$  being generated by an HMM.
- ② **Decoding** Given an observation sequence and an HMM, determine the most probable hidden state sequence
- ③ **Training** Given an observation sequence and an HMM, learn the best HMM parameters  $\lambda = \{\{a_{jk}\}, \{b_j(\cdot)\}\}$

# 1. Likelihood: how to calculate?



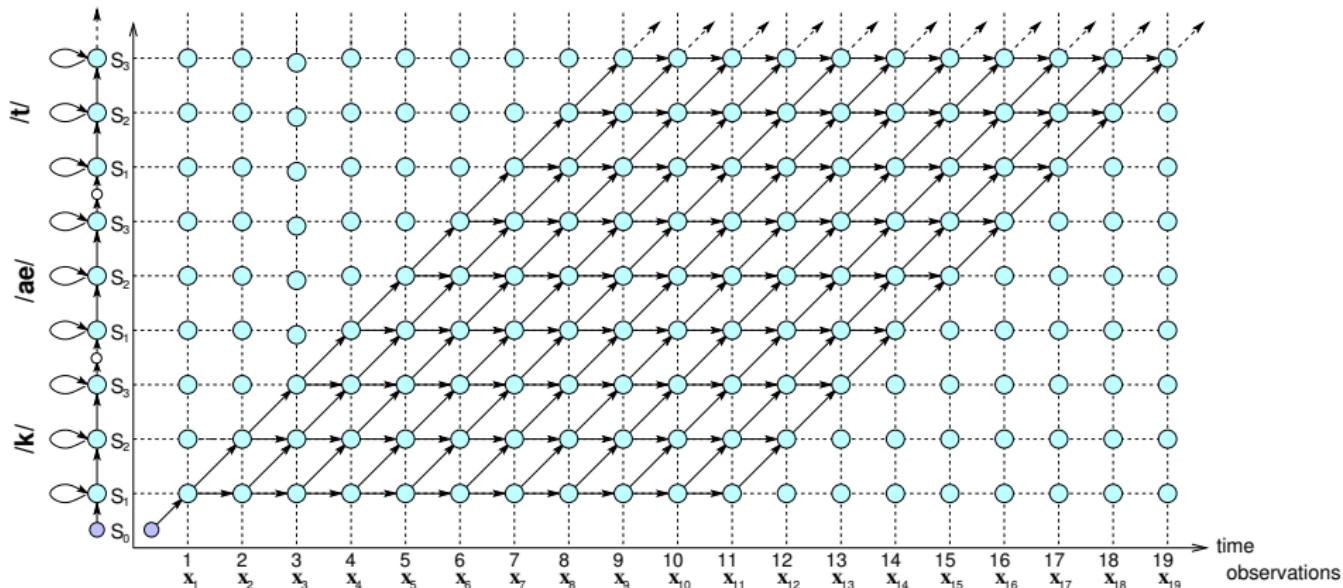
$$\begin{aligned} p(\mathbf{X}, \text{path}_\ell | \lambda) &= p(\mathbf{X} | \text{path}_\ell, \lambda) P(\text{path}_\ell | \lambda) \\ &= p(\mathbf{X} | s_0 s_1 s_1 s_1 s_2 s_2 s_3 s_3 s_4, \lambda) P(s_0 s_1 s_1 s_1 s_2 s_2 s_3 s_3 s_4 | \lambda) \\ &= b_1(x_1) b_1(x_2) b_1(x_3) b_2(x_4) b_2(x_5) b_3(x_6) b_3(x_7) a_{01} a_{11} a_{11} a_{12} a_{22} a_{23} a_{33} a_{34} \end{aligned}$$

$$p(\mathbf{X} | \lambda) = \sum_{\{\text{path}_\ell\}} p(\mathbf{X}, \text{path}_\ell | \lambda) \simeq \max_{\text{path}_\ell} p(\mathbf{X}, \text{path}_\ell | \lambda)$$

forward(backward) algorithm

Viterbi algorithm

# Trellis for /k ae t/



# 1. Likelihood: The Forward algorithm

- Goal: determine  $p(\mathbf{X} | \lambda)$
- Sum over all possible state sequences  $s_1 s_2 \dots s_T$  that could result in the observation sequence  $\mathbf{X}$
- Rather than enumerating each sequence, compute the probabilities recursively (exploiting the Markov assumption)
- How many paths calculations in  $p(\mathbf{X} | \lambda)$ ?

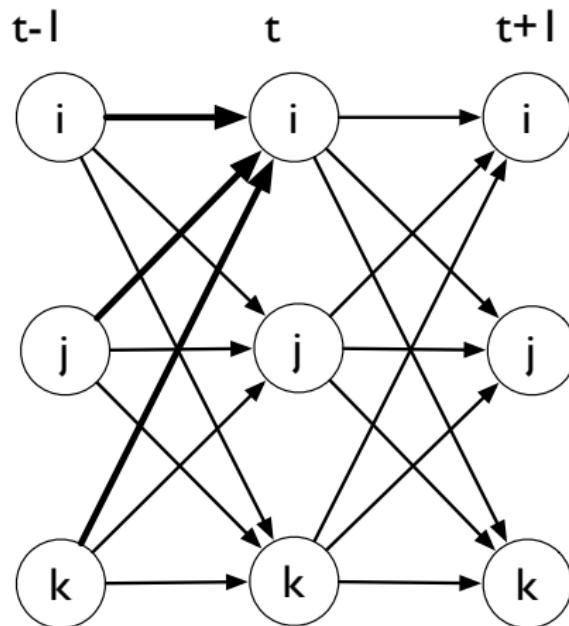
$$\sim \underbrace{N \times N \times \cdots N}_{T \text{ times}} = N^T \quad \begin{aligned} N &: \text{number of HMM states} \\ T &: \text{length of observation} \end{aligned}$$

e.g.  $N^T \approx 10^{10}$  for  $N=3, T=20$

- Computation complexity of multiplication:  $O(2T N^T)$
- The Forward algorithm reduces this to  $O(TN^2)$

# Recursive algorithms on HMMs

Visualise the problem as a *state-time trellis*



# 1. Likelihood: The Forward algorithm

- Goal: determine  $p(\mathbf{X} | \lambda)$
- Sum over all possible state sequences  $s_1 s_2 \dots s_T$  that could result in the observation sequence  $\mathbf{X}$
- Rather than enumerating each sequence, compute the probabilities recursively (exploiting the Markov assumption)
- *Forward probability*,  $\alpha_t(j)$ : the probability of observing the observation sequence  $\mathbf{x}_1 \dots \mathbf{x}_t$  and being in state  $j$  at time  $t$ :

$$\alpha_t(j) = p(\mathbf{x}_1, \dots, \mathbf{x}_t, S(t)=j | \lambda)$$

# 1. Likelihood: The Forward recursion

- Initialisation

$$\begin{aligned}\alpha_0(s_I) &= 1 \\ \alpha_0(j) &= 0 \quad \text{if } j \neq s_I\end{aligned}$$

- Recursion

$$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(\mathbf{x}_t) \quad 1 \leq j \leq N, 1 \leq t \leq T$$

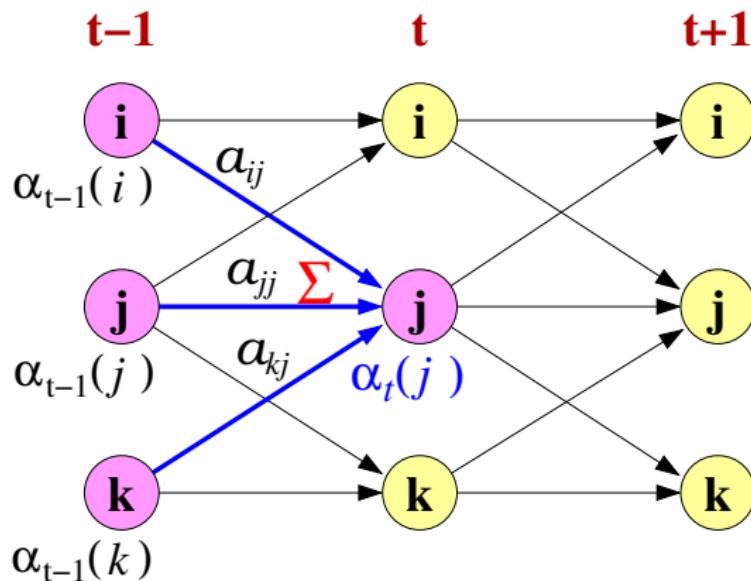
- Termination

$$p(\mathbf{X} | \boldsymbol{\lambda}) = \alpha_T(s_E) = \sum_{i=1}^N \alpha_T(i) a_{iE}$$

$s_I$ : initial state,  $s_E$ : final state

# 1. Likelihood: Forward Recursion

$$\alpha_t(j) = p(\mathbf{x}_1, \dots, \mathbf{x}_t, S(t)=j | \boldsymbol{\lambda}) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(\mathbf{x}_t)$$



## Viterbi approximation

- Instead of summing over all possible state sequences, just consider the most likely
- Achieve this by changing the summation to a maximisation in the recursion:

$$V_t(j) = \max_i V_{t-1}(i) a_{ij} b_j(\mathbf{x}_t)$$

- Changing the recursion in this way gives the likelihood of the most probable path
- We need to keep track of the states that make up this path by keeping a sequence of *backpointers* to enable a Viterbi *backtrace*: the backpointer for each state at each time indicates the previous state on the most probable path

# Viterbi Recursion

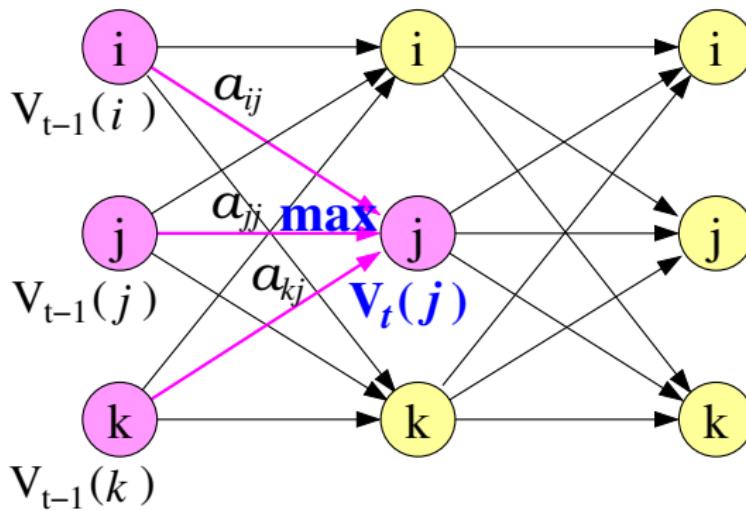
$$V_t(j) = \max_i V_{t-1}(i) a_{ij} b_j(\mathbf{x}_t)$$

Likelihood of the most probable path

t-1

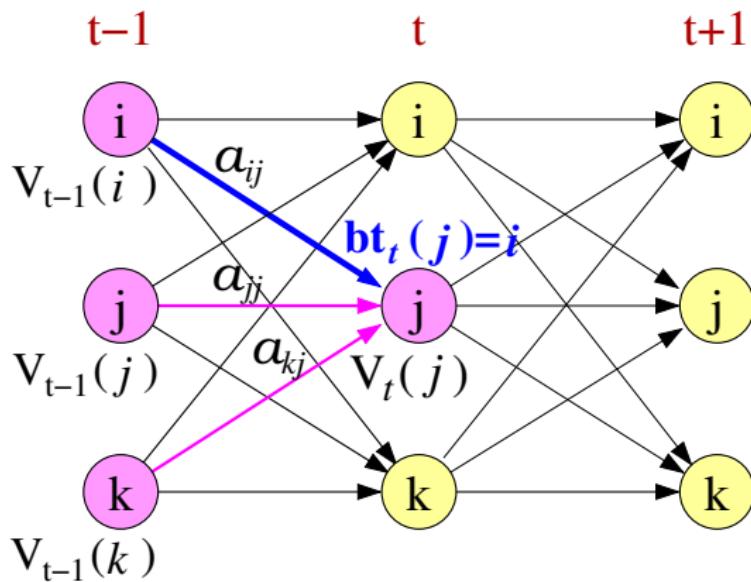
t

t+1



# Viterbi Recursion

Backpointers to the previous state on the most probable path



## 2. Decoding: The Viterbi algorithm

- Initialisation

$$V_0(i) = 1$$

$$V_0(j) = 0 \quad \text{if } j \neq i$$

$$\text{bt}_0(j) = 0$$

- Recursion

$$V_t(j) = \max_{i=1}^N V_{t-1}(i) a_{ij} b_j(\mathbf{x}_t)$$

$$\text{bt}_t(j) = \arg \max_{i=1}^N V_{t-1}(i) a_{ij} b_j(\mathbf{x}_t)$$

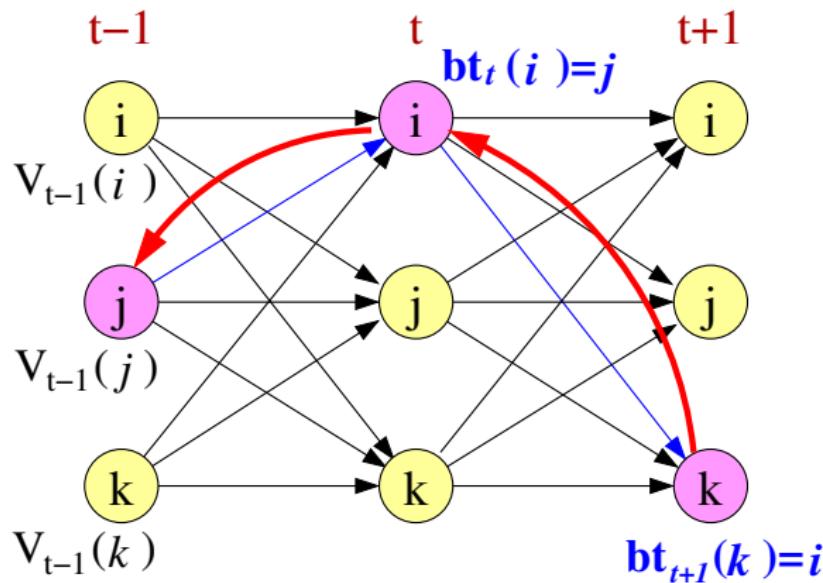
- Termination

$$P^* = V_T(s_E) = \max_{i=1}^N V_T(i) a_{iE}$$

$$s_T^* = \text{bt}_T(q_E) = \arg \max_{i=1}^N V_T(i) a_{iE}$$

# Viterbi Backtrace

Backtrace to find the state sequence of the most probable path



### 3. Training: Forward-Backward algorithm

- Goal: Efficiently estimate the parameters of an HMM  $\lambda$  from an observation sequence
- Assume single Gaussian output probability distribution

$$b_j(\mathbf{x}) = p(\mathbf{x} \mid j) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

- Parameters  $\lambda$ :
  - Transition probabilities  $a_{ij}$ :

$$\sum_j a_{ij} = 1$$

- Gaussian parameters for state  $j$ :  
mean vector  $\boldsymbol{\mu}_j$ ; covariance matrix  $\boldsymbol{\Sigma}_j$

# Viterbi Training

- If we knew the state-time alignment, then each observation feature vector could be assigned to a specific state
- A state-time alignment can be obtained using the most probable path obtained by Viterbi decoding
- Maximum likelihood estimate of  $a_{ij}$ , if  $C(i \rightarrow j)$  is the count of transitions from  $i$  to  $j$

$$\hat{a}_{ij} = \frac{C(i \rightarrow j)}{\sum_k C(i \rightarrow k)}$$

- Likewise if  $Z_j$  is the set of observed acoustic feature vectors assigned to state  $j$ , we can use the standard maximum likelihood estimates for the mean and the covariance:

$$\hat{\mu}_j = \frac{\sum_{\mathbf{x} \in Z_j} \mathbf{x}}{|Z_j|}$$

$$\hat{\Sigma}_j = \frac{\sum_{\mathbf{x} \in Z_j} (\mathbf{x} - \hat{\mu}_j)(\mathbf{x} - \hat{\mu}_j)^T}{|Z_j|}$$

# EM Algorithm

- Viterbi training is an approximation—we would like to consider *all* possible paths
- In this case rather than having a hard state-time alignment we estimate a probability
- *State occupation probability*: The probability  $\gamma_t(j)$  of occupying state  $j$  at time  $t$  given the sequence of observations.

Compare with component occupation probability in a GMM

- We can use this for an iterative algorithm for HMM training: the **EM algorithm** (whose adaption to HMM is called '[Baum-Welch algorithm](#)')
- Each iteration has two steps:

**E-step** estimate the state occupation probabilities  
(Expectation)

**M-step** re-estimate the HMM parameters based on the estimated state occupation probabilities  
(Maximisation)

# Backward probabilities

- To estimate the state occupation probabilities it is useful to define (recursively) another set of probabilities—the *Backward probabilities*

$$\beta_t(j) = p(\mathbf{x}_{t+1}, \dots, \mathbf{x}_T | S(t)=j, \lambda)$$

The probability of future observations given a the HMM is in state  $j$  at time  $t$

- These can be recursively computed (going backwards in time)
  - Initialisation

$$\beta_T(i) = a_{iE}$$

- Recursion

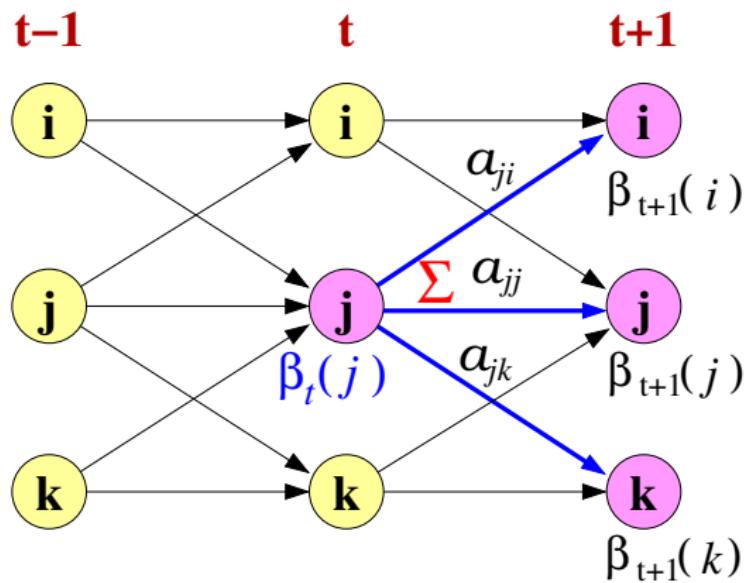
$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(\mathbf{x}_{t+1}) \beta_{t+1}(j) \quad \text{for } t = T-1, \dots, 1$$

- Termination

$$p(\mathbf{X} | \lambda) = \beta_0(I) = \sum_{j=1}^N a_{Ij} b_j(\mathbf{x}_1) \beta_1(j) = \alpha_T(s_E)$$

# Backward Recursion

$$\beta_t(j) = p(\mathbf{x}_{t+1}, \dots, \mathbf{x}_T | S(t)=j, \lambda) = \sum_{j=1}^N a_{ij} b_j(\mathbf{x}_{t+1}) \beta_{t+1}(j)$$



# State Occupation Probability

- The **state occupation probability**  $\gamma_t(j)$  is the probability of occupying state  $j$  at time  $t$  given the sequence of observations
- Express in terms of the forward and backward probabilities:

$$\gamma_t(j) = P(S(t)=j | \mathbf{X}, \boldsymbol{\lambda}) = \frac{1}{\alpha_T(s_E)} \alpha_t(j) \beta_t(j)$$

recalling that  $p(\mathbf{X} | \boldsymbol{\lambda}) = \alpha_T(s_E)$

- Since

$$\begin{aligned}\alpha_t(j) \beta_t(j) &= p(\mathbf{x}_1, \dots, \mathbf{x}_t, S(t)=j | \boldsymbol{\lambda}) \\ &\quad p(\mathbf{x}_{t+1}, \dots, \mathbf{x}_T | S(t)=j, \boldsymbol{\lambda}) \\ &= p(\mathbf{x}_1, \dots, \mathbf{x}_t, \mathbf{x}_{t+1}, \dots, \mathbf{x}_T, S(t)=j | \boldsymbol{\lambda}) \\ &= p(\mathbf{X}, S(t)=j | \boldsymbol{\lambda})\end{aligned}$$

$$P(S(t)=j | \mathbf{X}, \boldsymbol{\lambda}) = \frac{p(\mathbf{X}, S(t)=j | \boldsymbol{\lambda})}{p(\mathbf{X} | \boldsymbol{\lambda})}$$

## Re-estimation of Gaussian parameters

- The sum of state occupation probabilities through time for a state, may be regarded as a “soft” count
- We can use this “soft” alignment to re-estimate the HMM parameters:

$$\hat{\mu}_j = \frac{\sum_{t=1}^T \gamma_t(j) \mathbf{x}_t}{\sum_{t=1}^T \gamma_t(j)}$$

$$\hat{\Sigma}_j = \frac{\sum_{t=1}^T \gamma_t(j) (\mathbf{x}_t - \hat{\mu}_j)(\mathbf{x}_t - \hat{\mu}_j)^T}{\sum_{t=1}^T \gamma_t(j)}$$

## Re-estimation of transition probabilities

- Similarly to the state occupation probability, we can estimate  $\xi_t(i, j)$ , the probability of being in  $i$  at time  $t$  and  $j$  at  $t + 1$ , given the observations:

$$\begin{aligned}\xi_t(i, j) &= P(S(t)=i, S(t+1)=j | \mathbf{X}, \boldsymbol{\lambda}) \\ &= \frac{p(S(t)=i, S(t+1)=j, \mathbf{X} | \boldsymbol{\lambda})}{p(\mathbf{X} | \boldsymbol{\lambda})} \\ &= \frac{\alpha_t(i) a_{ij} b_j(\mathbf{x}_{t+1}) \beta_{t+1}(j)}{\alpha_T(s_E)}\end{aligned}$$

- We can use this to re-estimate the transition probabilities

$$\hat{a}_{ij} = \frac{\sum_{t=1}^T \xi_t(i, j)}{\sum_{k=1}^N \sum_{t=1}^T \xi_t(i, k)}$$

# Pulling it all together

- Iterative estimation of HMM parameters using the EM algorithm. At each iteration

E step For all time-state pairs

- ① Recursively compute the forward probabilities  $\alpha_t(j)$  and backward probabilities  $\beta_t(j)$
- ② Compute the state occupation probabilities  $\gamma_t(j)$  and  $\xi_t(i, j)$

M step Based on the estimated state occupation probabilities re-estimate the HMM parameters: mean vectors  $\mu_j$ , covariance matrices  $\Sigma_j$  and transition probabilities  $a_{ij}$

- The application of the EM algorithm to HMM training is sometimes called the Forward-Backward algorithm or Baum-Welch algorithm

## Extension to a corpus of utterances

- We usually train from a large corpus of  $R$  utterances
- If  $\mathbf{x}_t^r$  is the  $t$ th frame of the  $r$ th utterance  $\mathbf{X}^r$  then we can compute the probabilities  $\alpha_t^r(j)$ ,  $\beta_t^r(j)$ ,  $\gamma_t^r(j)$  and  $\xi_t^r(i, j)$  as before
- The re-estimates are as before, except we must sum over the  $R$  utterances, eg:

$$\hat{\mu}_j = \frac{\sum_{r=1}^R \sum_{t=1}^T \gamma_t^r(j) \mathbf{x}_t^r}{\sum_{r=1}^R \sum_{t=1}^T \gamma_t^r(j)}$$

- In addition, we usually employ “*embedded training*”, in which fine tuning of phone labelling with “*forced Viterbi alignment*” or forced alignment is involved. (For details see Section 9.7 in Jurafsky and Martin’s SLP)

## Extension to Gaussian mixture model (GMM)

- The assumption of a Gaussian distribution at each state is very strong; in practice the acoustic feature vectors associated with a state may be strongly non-Gaussian
- In this case an  $M$ -component Gaussian mixture model is an appropriate density function:

$$b_j(\mathbf{x}) = p(\mathbf{x} | S=j) = \sum_{m=1}^M c_{jm} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})$$

Given enough components, this family of functions can model any distribution.

- Train using the EM algorithm, in which the component estimation probabilities are estimated in the E-step

# EM training of HMM/GMM

- Rather than estimating the state-time alignment, we estimate the component/state-time alignment, and component-state occupation probabilities  $\gamma_t(j, m)$ : the probability of occupying mixture component  $m$  of state  $j$  at time  $t$ .  
( $\xi_{tm}(j)$  in Jurafsky and Martin's SLP)
- We can thus re-estimate the mean of mixture component  $m$  of state  $j$  as follows

$$\hat{\mu}_{jm} = \frac{\sum_{t=1}^T \gamma_t(j, m) \mathbf{x}_t}{\sum_{t=1}^T \gamma_t(j, m)}$$

And likewise for the covariance matrices (mixture models often use diagonal covariance matrices)

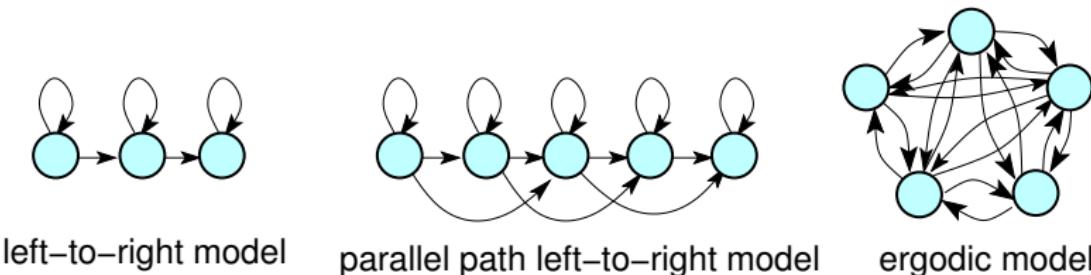
- The mixture coefficients are re-estimated in a similar way to transition probabilities:

$$\hat{c}_{jm} = \frac{\sum_{t=1}^T \gamma_t(j, m)}{\sum_{m'=1}^M \sum_{t=1}^T \gamma_t(j, m')}$$

## Doing the computation

- The forward, backward and Viterbi recursions result in a long sequence of probabilities being multiplied
- This can cause floating point *underflow* problems
- In practice computations are performed in the log domain (in which multiplies become adds)
- Working in the log domain also avoids needing to perform the exponentiation when computing Gaussians

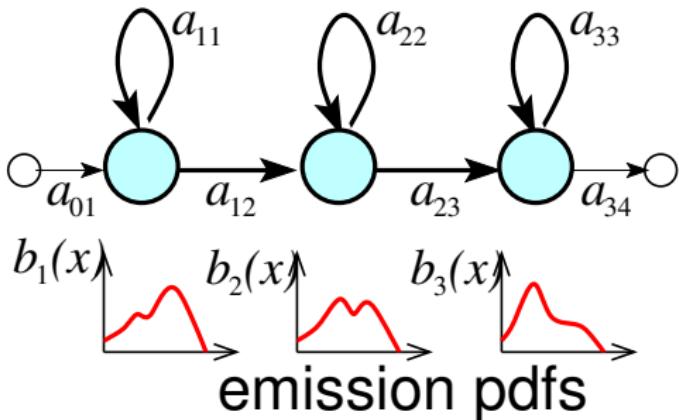
# A note on HMM topology



$$\begin{pmatrix} a_{11} & a_{12} & 0 \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{pmatrix} \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ 0 & a_{22} & a_{23} & a_{24} & 0 \\ 0 & 0 & a_{33} & a_{34} & a_{35} \\ 0 & 0 & 0 & a_{44} & a_{45} \\ 0 & 0 & 0 & 0 & a_{55} \end{pmatrix} \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix}$$

Speech recognition: left-to-right HMM with  $3 \sim 5$  states  
Speaker recognition: ergodic HMM

# A note on HMM emission probabilities



	Emission prob.
Continuous (density) HMM	continuous density
Discrete (probability) HMM	discrete probability
Semi-continuous HMM (tied-mixture HMM)	GMM, NN/DNN VQ tied mixture

# Summary: HMMs

- HMMs provide a generative model for statistical speech recognition
- Three key problems
  - ① Computing the overall likelihood: the Forward algorithm
  - ② Decoding the most likely state sequence: the Viterbi algorithm
  - ③ Estimating the most likely parameters: the EM (Forward-Backward) algorithm
- Solutions to these problems are tractable due to the two key HMM assumptions
  - ① Conditional independence of observations given the current state
  - ② Markov assumption on the states

## References: HMMs

- Gales and Young (2007). “The Application of Hidden Markov Models in Speech Recognition”, *Foundations and Trends in Signal Processing*, **1** (3), 195–304: section 2.2.
- Jurafsky and Martin (2008). *Speech and Language Processing* (2nd ed.): sections 6.1–6.5; 9.2; 9.4. (Errata at <http://www.cs.colorado.edu/~martin/SLP/Errata/SLP2-PIEV-Errata.html>)
- Rabiner and Juang (1989). “An introduction to hidden Markov models”, *IEEE ASSP Magazine*, **3** (1), 4–16.
- Renals and Hain (2010). “Speech Recognition”, *Computational Linguistics and Natural Language Processing Handbook*, Clark, Fox and Lappin (eds.), Blackwells.

# Context-dependent phone models

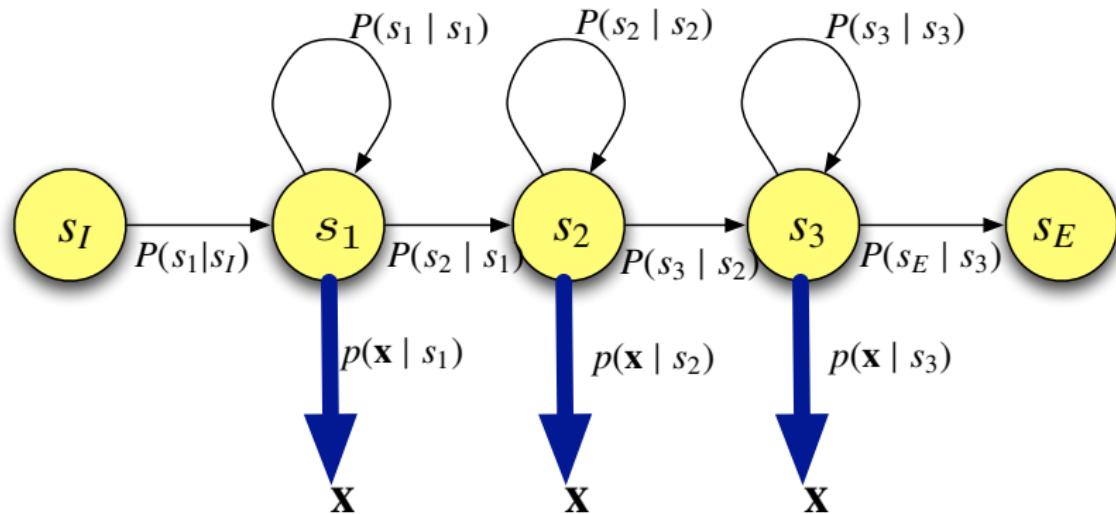
Steve Renals and Hiroshi Shimodaira

Automatic Speech Recognition  
ASR Lecture 6  
1 February 2018

## Phone models

- Modelling phones with HMMs
- The need to model phonetic context
- Triphone models
- Parameter sharing – sharing parameters across different contexts
- Choosing which states to share – phonetic decision trees

# Recap: Continuous Density HMM

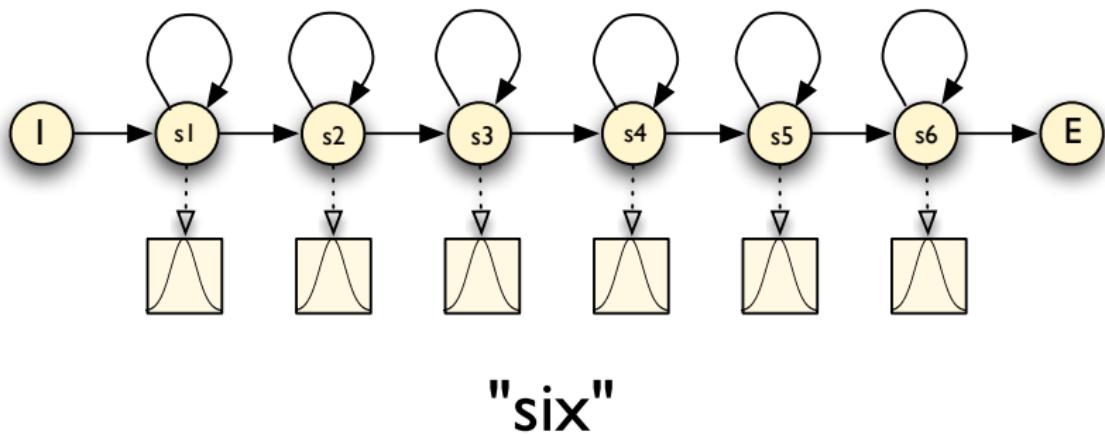


Probabilistic finite state automaton

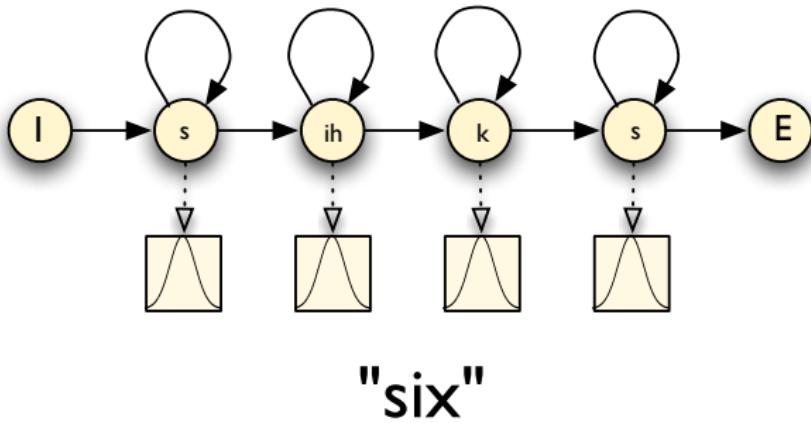
Parameters  $\lambda$ :

- Transition probabilities:  $a_{kj} = P(s_j | s_k)$
- Output probability density function:  $b_j(x) = p(x | s_j)$

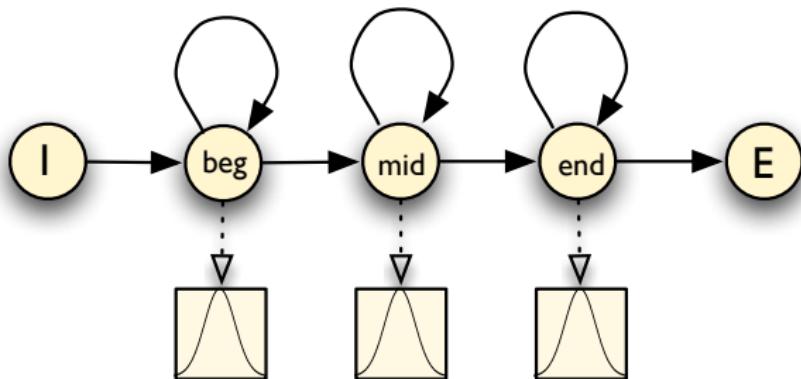
# Whole word models



# One state per phone models

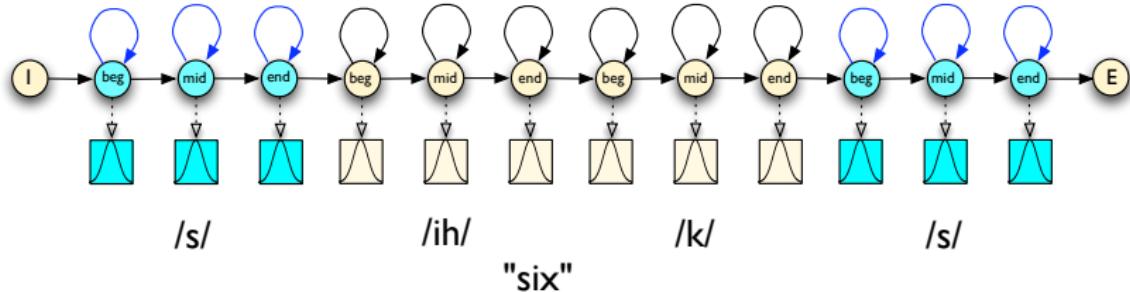


# Three-state phone models

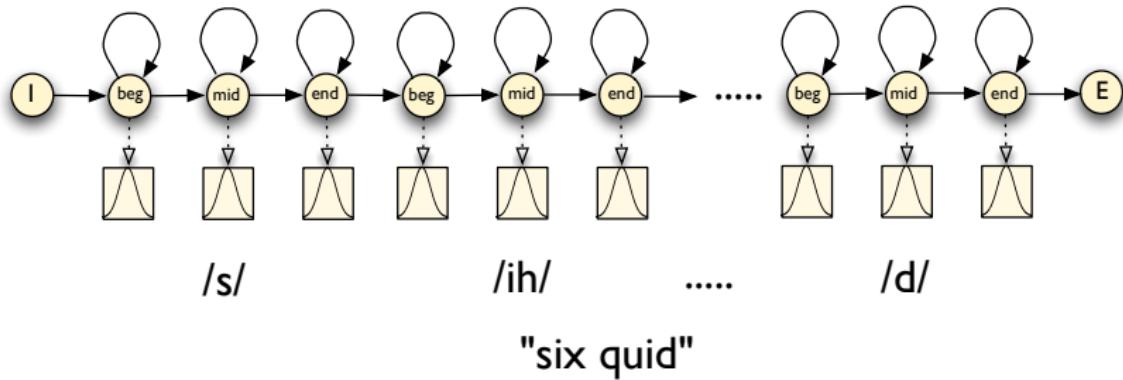


/ih/

# Word model made of phone models



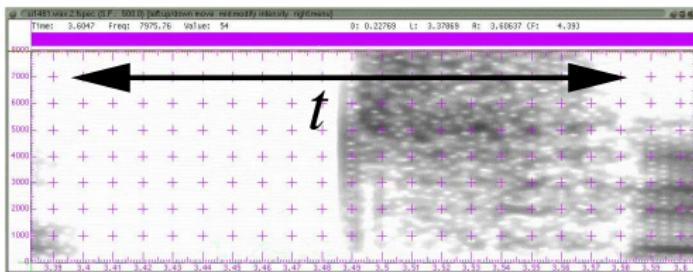
# Word sequence models



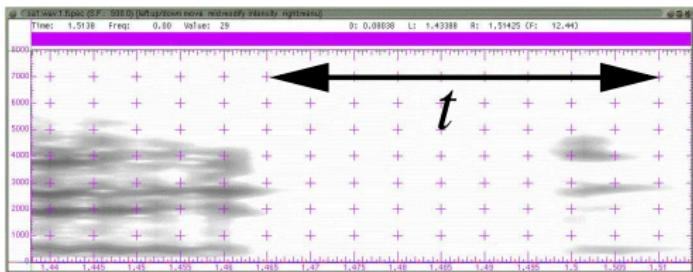
# Phonetic Context

- **Context** The acoustic phonetic context of a speech unit has an effect on its acoustic realization
- **Coarticulation** the place of articulation for one speech sound depends on a neighbouring speech sound.
- Consider /n/ in **ten** and **tenth**
  - dental in **ten**
  - alveolar in **tenth**

# Phonetic Context Example



"tube"



"suit"

# Modelling Context

- **Subword units** Individual phone units need to deal with a lot of variability
  - Use longer units that incorporate context, eg: diphones, demisyllables, syllables
  - Use multiple models for each: **context-dependent** phone models
  - Context-dependent phones are termed **allophones** of the parent phone
- **Pronunciations**
  - “did you” d ih jh y ah
  - “around this” ix r aw n ih s

## Divide and conquer

- Context-dependent models are more **specific** than context-independent models
- Increase the detail of modelling by extending the state space – but by defining multiple context dependent models, rather than more complex context independent models
- Divide and conquer: as more context-dependent models are defined, each one becomes responsible for a smaller region of the acoustic-phonetic space
- Let the data tell us how many contexts to model

# Context-dependent phone models

- **Triphones** Each phone has a unique model for each left and right context. Represent a phone  $x$  with left context  $l$  and right context  $r$  as  $l-x+r$
- **Word-internal triphones** Only take account of context within words, so “don’t ask” is represented by:  
sil d+oh d-oh+n oh-n+t n-t ah+s ah-s+k s-k sil  
Word internal triphones result in far fewer models than cross-word models, and enable the subword sequence for a word to be known independent of the neighbouring words.  
But: context is not well-modelled at word boundaries.
- **Cross-word triphones** “don’t ask” is represented by:  
sil sil-d+oh d-oh+n oh-n+t n-t+ah t-ah+s ah-s+k s-k+sil sil  
Note that triphone context extends across words (eg unit n-t+ah)

# Triphone models

- **How many triphones are there?** Consider a 40 phone system.  
 $40^3 = 64\,000$  possible triphones. In a cross-word system  
maybe 50 000 can occur
- Number of parameters:
  - 50 000 three-state HMMs, with 10 component Gaussian mixtures per state: 1.5M Gaussians
  - 39-dimension feature vectors (12 MFCCs + energy), deltas and accelerations
  - Assuming diagonal Gaussians: about 790 parameters/state
  - **Total** about 118 million parameters!
- We would need a very large amount of training data to train such a system
  - to enable robust estimation of all parameters
  - to ensure that all possible triphones are observed (more than once) in the training data

# Modelling infrequent triphones

The number of possible triphone types is much greater than the number of observed triphone tokens.

- Smoothing – combine less-specific and more-specific models
- **Parameter Sharing** – different contexts share models
  - Bottom-up – start with all possible contexts, then merge
  - Top-down – start with a single context, then split
- All approaches are data driven

NB: knowledge is used to make it work effectively

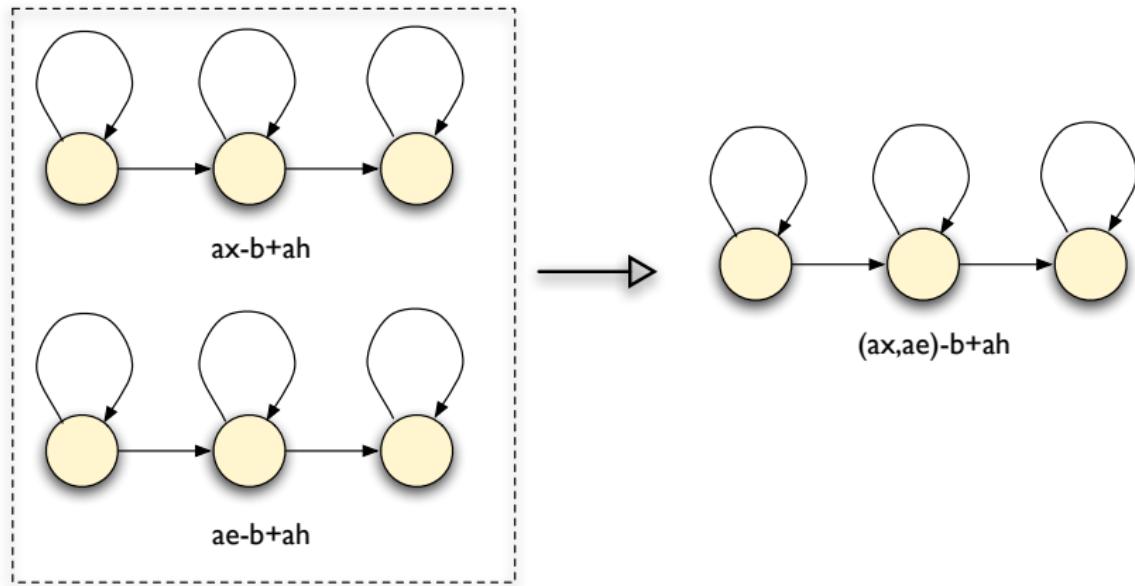
# Parameter Sharing

- **Basic idea** Explicitly share models or parameters between different contexts
    - enables **training data** to be shared between the models
    - enables models to share parameters
  - Sharing can take place at different levels
- ① Sharing Gaussians: all distributions share the same set of Gaussians but have different mixture weights (**tied mixtures**)
  - ② Sharing states: allow different models to share the same states (**state clustering**)
  - ③ Sharing models: merge those context-dependent models that are the most similar (**generalised triphones**)

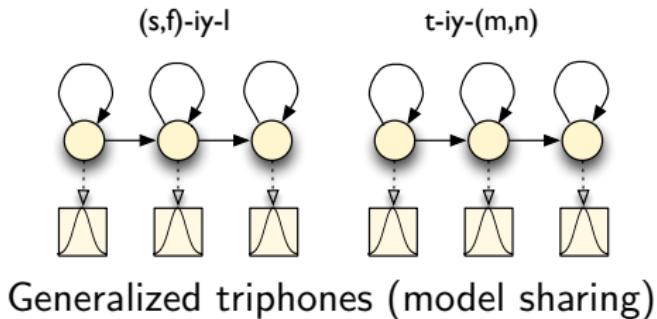
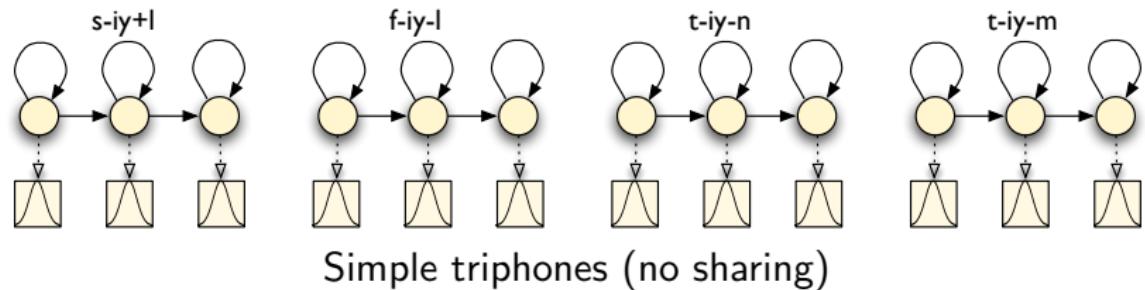
## Sharing Models: Generalized triphones

- **Basic idea** Merge similar context-dependent models
- Bottom-up merging: Compare allophone models with different triphone contexts and merge those that are similar
- Merged models will be estimated from more data than individual models: more accurate models, fewer models in total
- The resultant merged models are referred to as generalized triphones

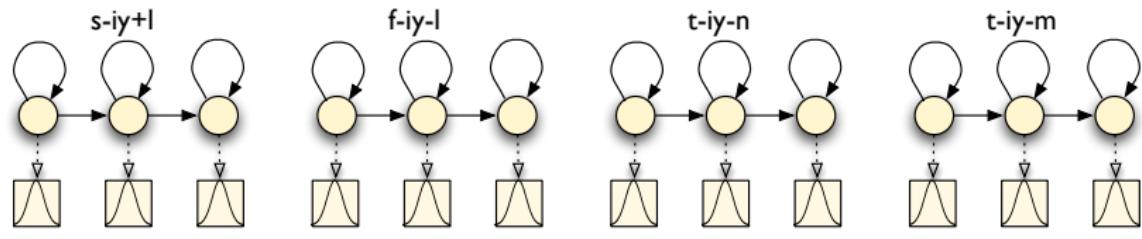
## Example: Generalized Triphones



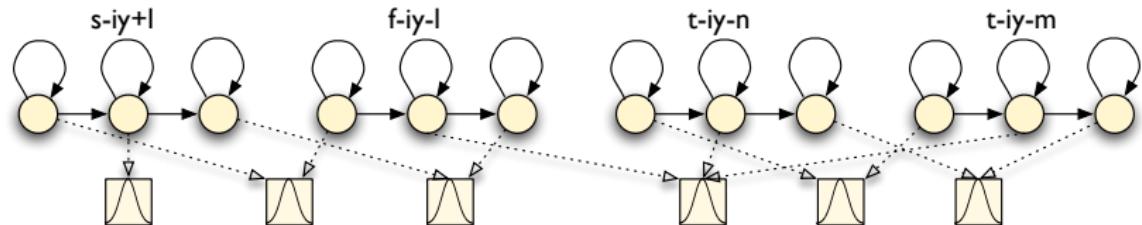
## Example : Generalized triphones



# State Clustering



Simple triphones (no sharing)



State-clustered triphones (state sharing)

## Sharing States: State clustering

- **Basic idea** States which are responsible for acoustically similar data are shared
- By clustering similar states, the training data associated with individual states may be pooled together – results in better parameter estimates for the state
  - ① Create a set of context dependent models for a parent phone
  - ② Cluster and tie similar states, ensuring that each resultant clustered state is responsible for “enough” training data (ie setting a minimum state occupation count)
- More flexible than clustering whole models: left and right contexts may be clustered separately

## Good contexts to share

- Which states should be clustered together?
- Bottom-up clustering, for triphones of the same parent phone
  - ① Create raw triphone models for each observed triphone context
  - ② Cluster states as before
- Drawback: unable to solve **unseen triphone problem**
- Top-down clustering: start with a parent context independent model then successively split models to create context dependent models

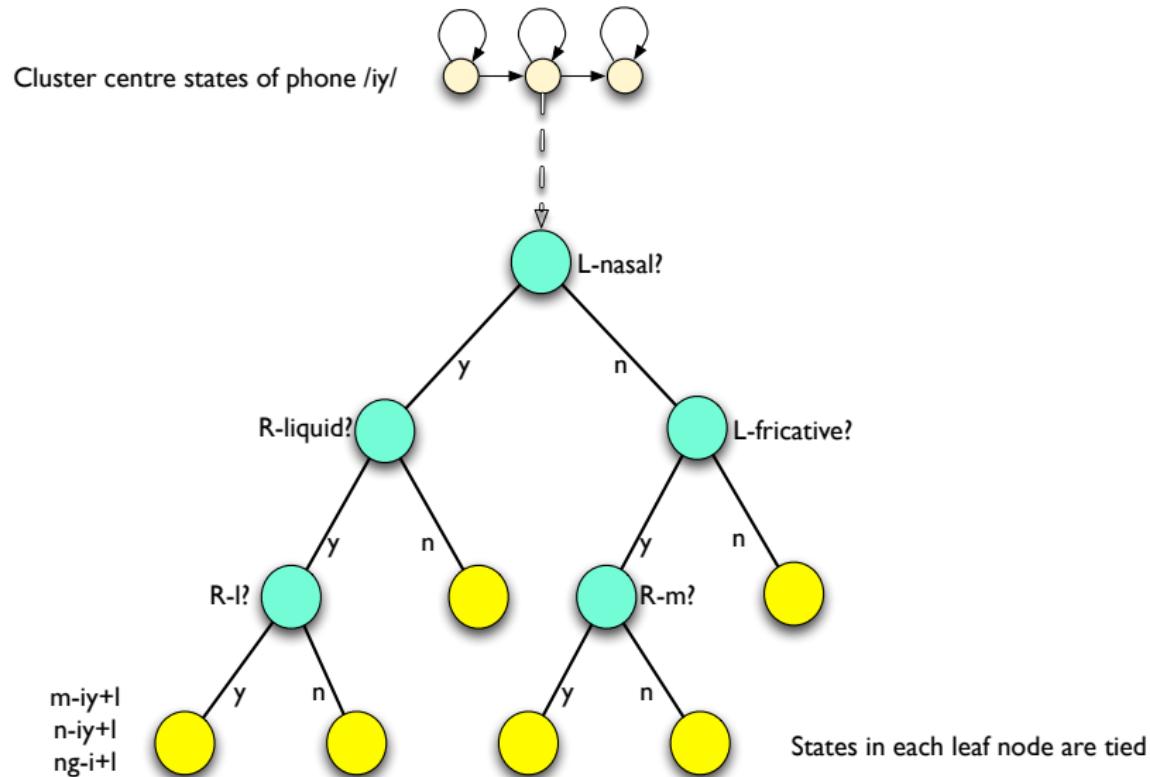
$$\text{Gain} = (L(\mathbf{S}_1) + L(\mathbf{S}_2)) - L(\mathbf{S})$$

## Phonetic decision trees

# Phonetic Decision Trees

- **Basic idea** Build a decision tree for each state of each parent phone, with yes/no questions at each node
- At the root of the tree, all states are shared
- Questions split the pool of states, the resultant state clusters are given by the leaves of the tree
- Example questions:
  - Is the left context a nasal?
  - Is the right context a central stop?
- The questions at each node are chosen from a large set of predefined questions
- Choose the question which maximizes the likelihood of the data given the state clusters
- Stop splitting if either: (a) the likelihood does not increase by more than a predefined threshold; or (b) the amount of data associated with a split node would below a threshold

# Phonetic Decision Tree



# Phonetic questions

- Ask questions of the form: does phone at offset  $s$  have feature  $f$ ?
- Offsets are  $+/-1$  for triphone context
- Example general questions:
  - Stop: b d g p t k
  - Nasal: m n ng
  - Fricative: ch dh f jh s sh th v z zh
  - Liquid: l r w y
  - Vowel: aa ae ah ao aw ax axr ay eh er ...
- Example consonant questions: Un/voiced, front/central/back, voiced stop, ....
- Example vowel questions: front, central, back, long, short, diphthong, rounded, ....
- Kaldi – generates all questions automatically using a top down binary clustering

# Most useful phonetic questions

- All states of all models:
  - +Vowel -Vowel +Unrounded -UnFortisLenis
  - +UnFortisLenis +r
- Entry state of all models:
  - UnFortisLenis -Vowel -Nasal -CentralFront
  - Unrounded -Fortis
- Exit state of all consonants:
  - +Vowel +Unrounded +High +ee +Rounded +Syllabic

(for Wall St Journal read speech – Young, Odell and Woodland 1994)

## Likelihood of a state cluster (1)

- **Basic idea** Compute the log likelihood of the data associated with a pool of states
- All states pooled in a single cluster at the root
- All states have Gaussian output pdf
- Let  $\mathbf{S} = \{s_1, s_2, \dots, s_K\}$  be a pool of  $K$  states forming a cluster, sharing a common mean  $\mu_S$  and covariance  $\Sigma_S$
- Let  $\mathbf{X}$  be the set of training data
- Let  $\gamma_s(\mathbf{x})$  be the probability that  $\mathbf{x} \in \mathbf{X}$  was generated by state  $s$  (i.e. state occupation probability)
- The log likelihood of the data associated with cluster  $\mathbf{S}$  is:

$$L(\mathbf{S}) = \sum_{s \in \mathbf{S}} \sum_{\mathbf{x} \in \mathbf{X}} \log P(\mathbf{x} | \mu_S, \Sigma_S) \gamma_s(\mathbf{x})$$

## Likelihood of a state cluster (2)

- Don't need to iterate through all data for each state
- If the output pdfs are Gaussian it can be shown that

$$L(\mathbf{S}) = -\frac{1}{2} \left( \log(2\pi)^d |\boldsymbol{\Sigma}_S| \right) + d \sum_{s \in \mathbf{S}} \sum_{x \in \mathbf{X}} \gamma_s(x)$$

where  $d$  is the dimension of the data

- Thus  $L(\mathbf{S})$  depends on only
  - the pooled state variance  $\boldsymbol{\Sigma}_S$  - can be computed from the means and variances of the individual states in the pool
  - and the state occupation probabilities already computed when forward-backward was carried out

## State splitting (1)

- **Basic idea** Use the likelihood of the parent state and of the split states to choose the splitting question
- Split  $\mathbf{S}$  into two partitions  $\mathbf{S}_y$  and  $\mathbf{S}_n$  using a question about the phonetic context
- Each partition is now clustered together to form a single Gaussian output distribution with mean  $\mu_{S_y}$  and covariance  $\Sigma_{S_y}$  (for partition  $S_y$ )
- The likelihood of the data after partition is given by  $L(\mathbf{S}_y) + L(\mathbf{S}_n)$
- The total likelihood of the partitioned data will increase by

$$\Delta = L(\mathbf{S}_y) + L(\mathbf{S}_n) - L(\mathbf{S})$$

## State splitting (2)

- **Basic idea** Use the likelihood of the parent state and of the split states to choose the splitting question

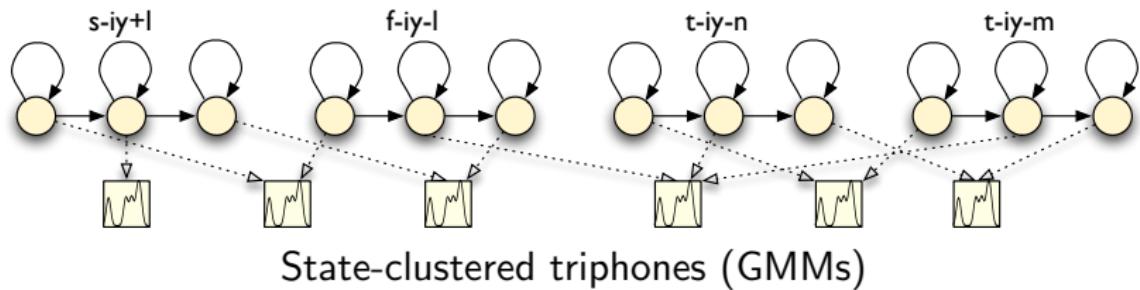
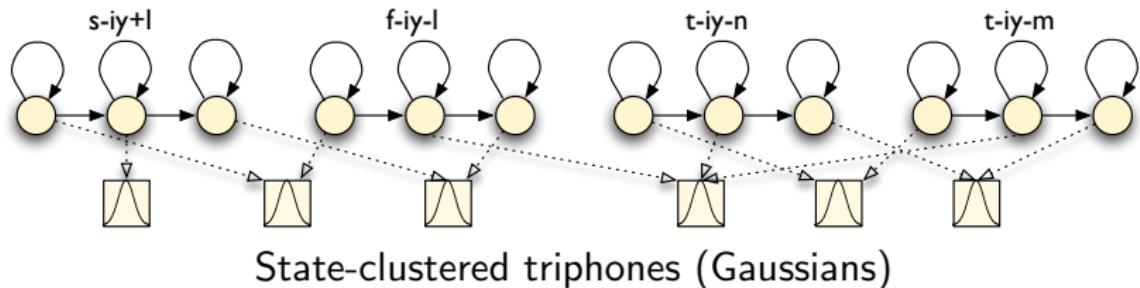
$$\Delta = L(\mathbf{S}_y) + L(\mathbf{S}_n) - L(\mathbf{S})$$

- Cycle through all possible questions, compute  $\Delta$  for each and choose the question for which  $\Delta$  is biggest
- Continue by splitting each of the new clusters  $\mathbf{S}_y$  and  $\mathbf{S}_n$
- Terminate when
  - ① Maximum  $\Delta$  falls below a threshold
  - ② The amount of data associated with a split node falls below a threshold
- For a Gaussian output distribution: State likelihood estimates can be estimated using just the *state occupation counts* (obtained at alignment) and the parameters of the Gaussian – no need to use the acoustic data
- State occupation count: sum of state occupation probabilities for a state over time

## “Mixing up”

- **Basic idea** Transforming an HMM-based system based on Gaussian distributions to one based on mixtures of Gaussians
- The above methods for state clustering assume that the state outputs are Gaussians – this makes the computations **much** simpler
- BUT: Gaussian mixtures offer much better acoustic models than Gaussians
- Solution:
  - Perform state clustering using Gaussian distributions
  - Split the Gaussian distributions in the clustered states, by cloning and perturbing the means by a small fraction of the standard deviation, and retrain.
  - Repeat by splitting the dominant (highest state occupation count) mixture components in each state

# “Mixing up”



## Summary: Context-dependent phone models

- Share parameters through state clustering
- Cluster states using phonetic decision trees for each state of parent phone
- Use Gaussian distributions when state clustering
- Then split Gaussians and retrain to obtain a GMM state clustered system

## References: context-dependent phone models

- c1980: First proposed by Bahl et al (IBM)
- Schwartz et al (1985): first paper using triphone models
- Lee (1990): generalized triphones
- Bellegarda (1990), Huang (1992): tied mixture modelling
- Bahl et al (1991): phonetic decision trees first proposed
- Young and Woodland (1994): state clustering
- Young et al (1994): decision tree-based state clustering
- Povey, 2012: Lecture on phonetic context-dependency in Kaldi

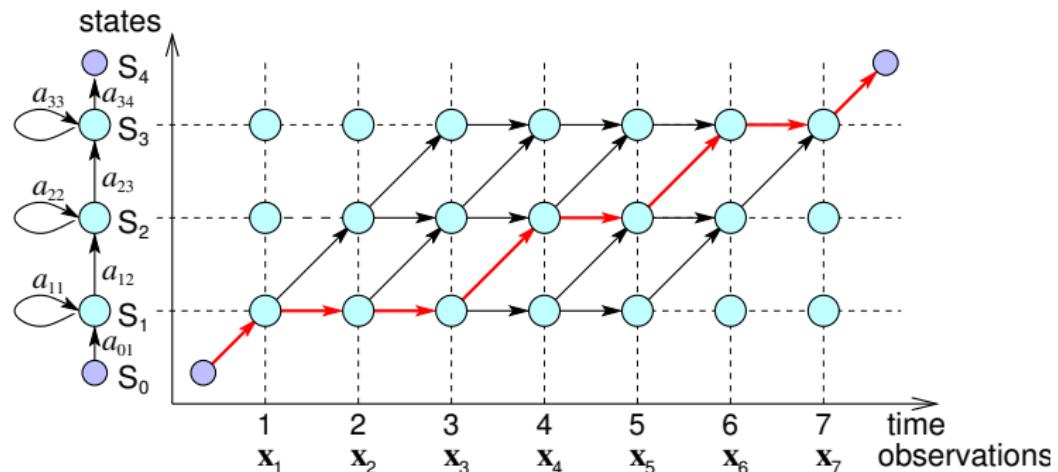
# Introduction to Neural Networks

Steve Renals

Automatic Speech Recognition – ASR Lecture 7  
5 February 2018

# Local Phonetic Scores and Sequence Modelling

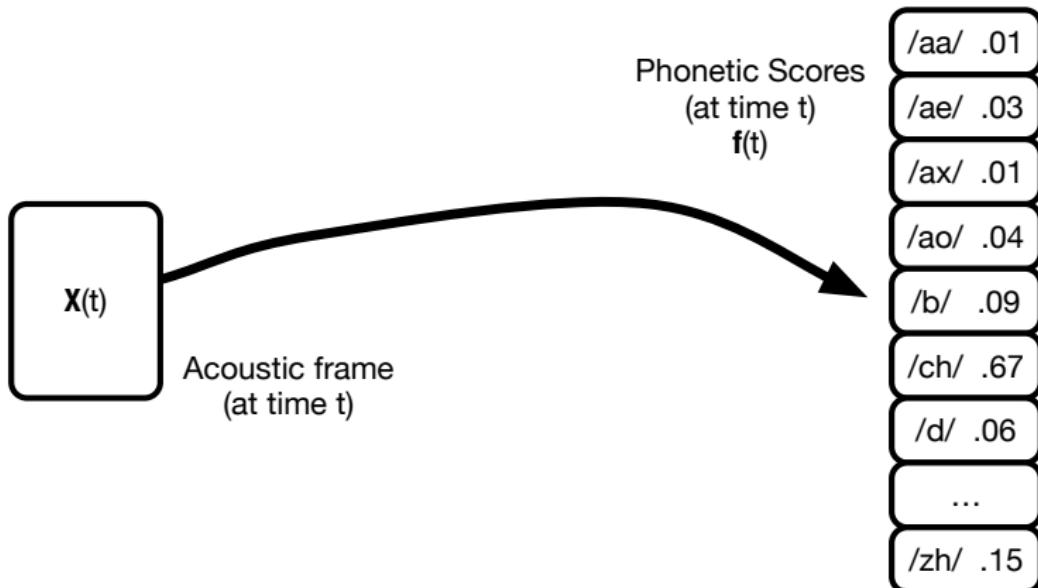
- DTW - local distances (Euclidean)
- HMM - emission probabilities (Gaussian or GMM)



- Compute the phonetic score(acoustic-frame, phone-model) – this does the detailed matching at the frame-level
- Chain phonetic scores together in a sequence - DTW, HMM

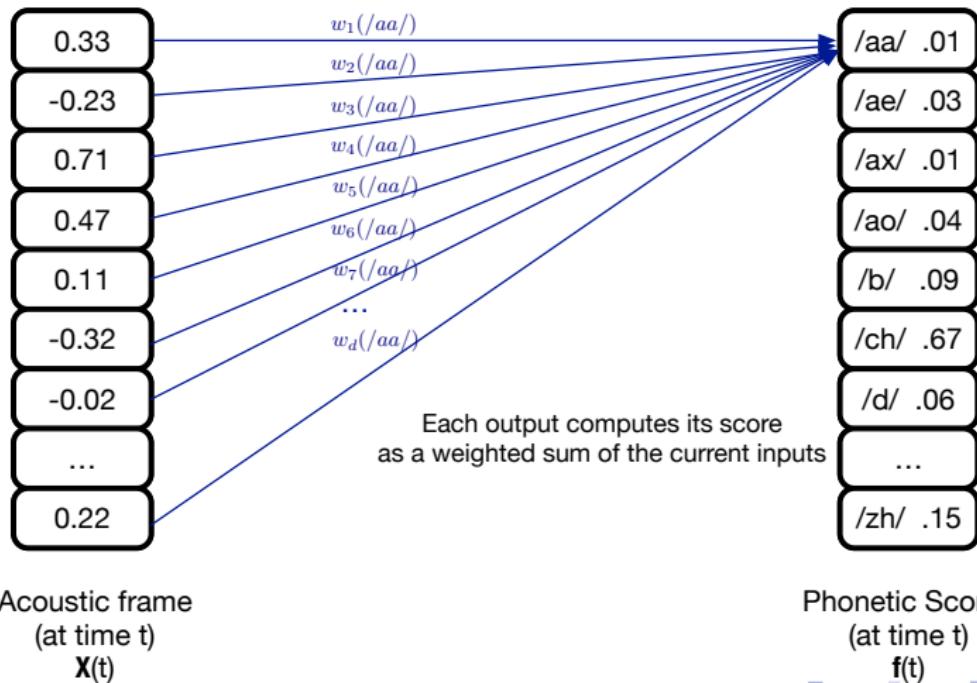
# Phonetic scores

Task: given an input acoustic frame, output a score for each phone



# Phonetic scores

Compute the phonetic scores using a single layer neural network  
(linear regression!)



# Phonetic scores

Compute the phonetic scores using a single layer neural network

- Write the estimated phonetic scores as a vector  
 $\mathbf{f} = (f_1, f_2, \dots, f_Q)$
- Then if the acoustic frame at time  $t$  is  $\mathbf{X} = (x_1, x_2, \dots, x_d)$ :

$$f_j = w_{j1}x_1 + w_{j2}x_2 + \dots + w_{jd}x_d + b_j$$

or, write it using summation notation:

$$f_j = \sum_{i=1}^d w_{ji}x_i + b_j$$

or, write it as vectors:

$$\mathbf{f} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

where we call  $\mathbf{W}$  the *weight matrix*, and  $\mathbf{b}$  the *bias vector*.

- Check your understanding:

What are the dimensions of  $\mathbf{W}$  and  $\mathbf{b}$ ?

# Error function

$$f(t) = \mathbf{Wx}(t) + \mathbf{b}$$

estimated      observed  
↓                  ↓  
trained            trained

How do we learn the *parameters  $\mathbf{W}$  and  $\mathbf{b}$ ?*

- **Minimise an Error Function:** Define a function which is 0 when the output  $\mathbf{f}(n)$  equals the *target output  $\mathbf{r}(n)$*  for all  $n$
- **Target output:** for TIMIT the target output corresponds to the phone label for each frame
- **Mean square error:** define the error function  $E$  as the mean square difference between output and the target:

$$E = \frac{1}{2} \cdot \frac{1}{N} \sum_{n=1}^N \|\mathbf{f}(n) - \mathbf{r}(n)\|^2$$

where there are  $N$  frames of training data in total

# Notes on the error function

- $f$  is a function of the acoustic data  $x$  and the weights and biases of the network ( $W$  and  $b$ )
- This means that as well as depending on the training data ( $x$  and  $r$ ),  $E$  is also a function of the weights and biases, since it is a function of  $f$
- We want to minimise the error function given a fixed training set: we must set  $W$  and  $b$  to minimise  $E$
- **Weight space:** given the training set we can imagine a space where every possible value of  $W$  and  $b$  results in a specific value of  $E$ . We want to find the minimum of  $E$  in this weight space.
- **Gradient descent:** find the minimum iteratively – given a current point in weight space find the direction of steepest descent, and change  $W$  and  $b$  to move in that direction

# Gradient Descent

- Iterative update – after seeing some training data, we adjust the weights and biases to reduce the error. Repeat until convergence.
- To update a parameter so as to reduce the error, we move downhill in the direction of steepest descent. Thus to train a network we must compute the gradient of the error with respect to the weights and biases:

$$\begin{pmatrix} \frac{\partial E}{\partial w_{10}} & \cdot & \frac{\partial E}{\partial w_{1i}} & \cdot & \frac{\partial E}{\partial w_{1d}} \\ & \ddots & & & \\ \frac{\partial E}{\partial w_{j0}} & \cdot & \frac{\partial E}{\partial w_{ji}} & \cdot & \frac{\partial E}{\partial w_{jd}} \\ & \ddots & & & \\ \frac{\partial E}{\partial w_{Q0}} & \cdot & \frac{\partial E}{\partial w_{Qi}} & \cdot & \frac{\partial E}{\partial w_{Qd}} \end{pmatrix} \quad \left( \begin{array}{cccc} \frac{\partial E}{\partial b_1} & \cdot & \frac{\partial E}{\partial b_j} & \cdot & \frac{\partial E}{\partial b_Q} \end{array} \right)$$

# Gradient Descent Procedure

- ① Initialise weights and biases with small random numbers
- ② For each batch of training data
  - ① Initialise total gradients:  $\Delta w_{ki} = 0$ ,  $\Delta b_k = 0$
  - ② For each training example  $n$  in the batch:
    - Compute the error  $E^n$
    - For all  $k, i$ : Compute the gradients  $\partial E^n / \partial w_{ki}$ ,  $\partial E^n / \partial b_k$
    - Update the total gradients by accumulating the gradients for example  $n$

$$\Delta w_{ki} \leftarrow \Delta w_{ki} + \frac{\partial E^n}{\partial w_{ki}} \quad \forall k, i$$

$$\Delta b_k \leftarrow \Delta b_k + \frac{\partial E^n}{\partial b_k} \quad \forall k$$

- ③ Update weights:

$$w_{ki} \leftarrow w_{ki} - \eta \Delta w_{ki} \quad \forall k, i$$

$$b_k \leftarrow b_k - \eta \Delta b_k \quad \forall k$$

Terminate either after a fixed number of epochs, or when the error stops decreasing by more than a threshold.

# Gradient in SLN

How do we compute the gradients  $\frac{\partial E^n}{\partial w_{ki}}$  and  $\frac{\partial E^n}{\partial b_k}$ ?

$$E^n = \frac{1}{2} \sum_{k=1}^K (f_k^n - r_k^n)^2 = \frac{1}{2} \sum_{k=1}^K \left( \sum_{i=1}^d (w_{ki}x_i^n + b_k) - r_k^n \right)^2$$

$$\boxed{\frac{\partial E^n}{\partial w_{ki}}} = (f_k^n - r_k^n)x_i^n = \boxed{g_k^n x_i^n} \quad \boxed{g_k^n = f_k^n - r_k^n}$$

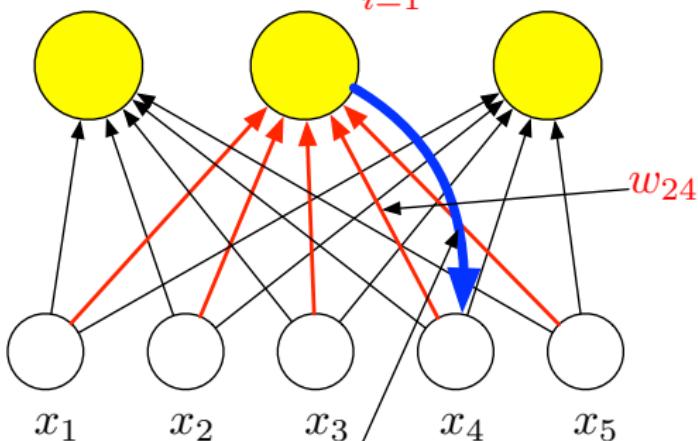
**Update rule:** Update a weight  $w_{ki}$  using the gradient of the error with respect to that weight: the product of the difference between the actual and target outputs for an example ( $f_k^n - r_k^n$ ) and the value of the unit at the input to the weight ( $x_i$ ).

*Check your understanding:* Show that the gradient for the bias is

$$\frac{\partial E^n}{\partial b_k} = g_k^n$$

# Applying gradient descent to a single-layer network

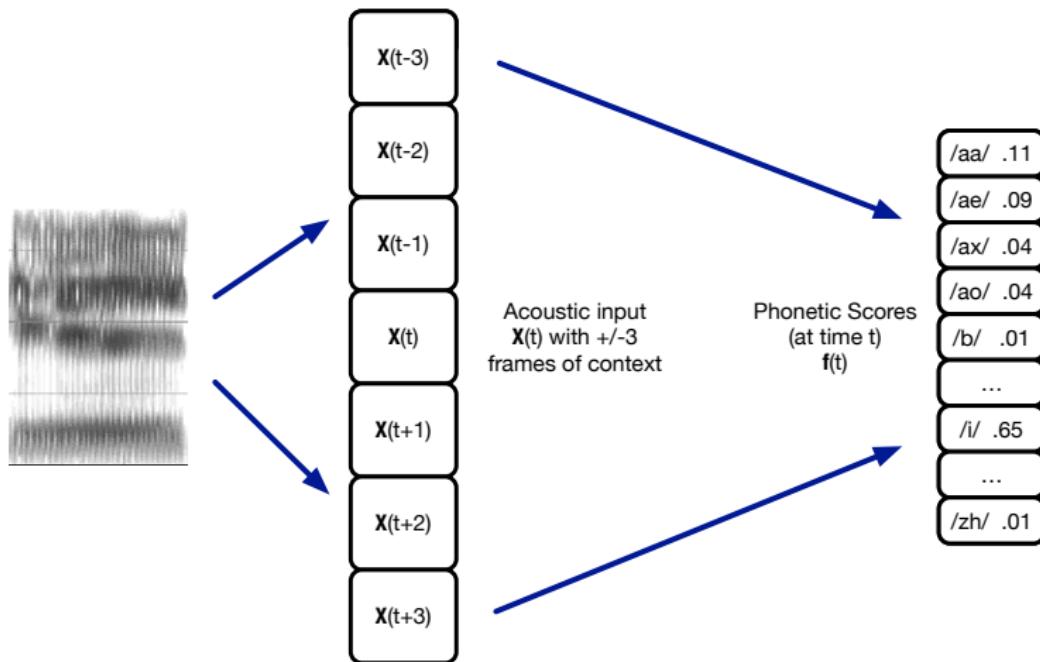
$$f_2 = \sum_{i=1}^5 w_{2i} x_i$$



$$\Delta w_{24} = \sum_n (f_2^n - r_2^n) x_4^n$$

# Acoustic context

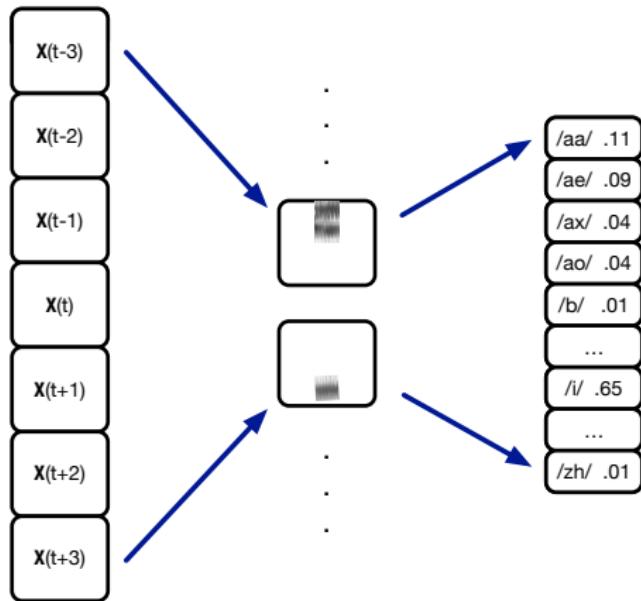
Use multiple frames of acoustic context



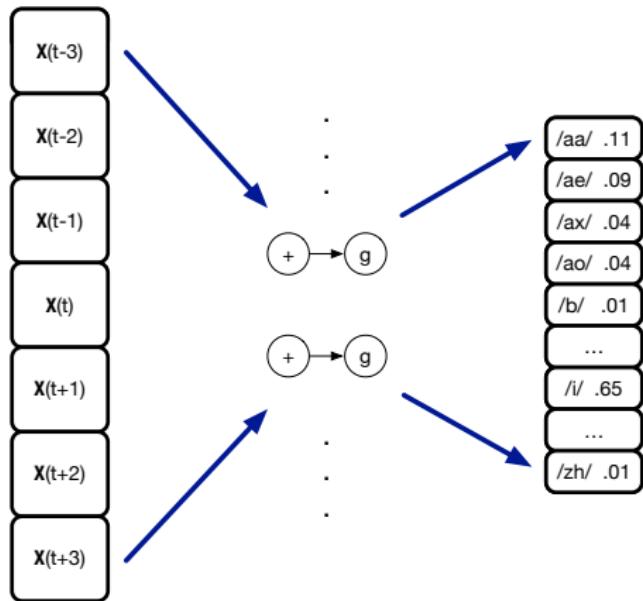
# Hidden units

- Single layer networks have limited computational power – each output unit is trained to match a spectrogram directly (a kind of discriminative template matching)
- But there is a lot of variation in speech (as previously discussed) – rate, coarticulation, speaker characteristics, acoustic environment
- Introduce an intermediate feature representation – “hidden units” – more robust than template matching
- Intermediate features represented by hidden units

# Hidden units extracting features



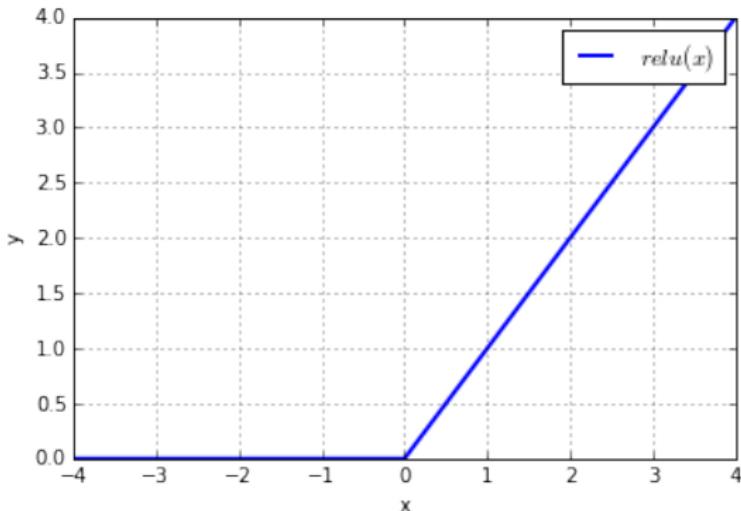
# Hidden Units



$$h_j = \text{relu} \left( \sum_{i=1}^d w_{ji} x_i + b_j \right)$$

$$f_k = \text{softmax} \left( \sum_{j=1}^H v_{kj} h_j + b_k \right)$$

# Rectified Linear Unit – ReLU



$$\text{relu}(x) = \max(0, x)$$

Derivative:  $\text{relu}'(x) = \frac{d}{dx} \text{relu}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$

# Softmax

$$y_k = \frac{\exp(a_k)}{\sum_{j=1}^K \exp(a_j)}$$

$$a_k = \sum_{j=1}^H v_{kj} h_j + b_k$$

- This form of activation has the following properties
  - Each output will be between 0 and 1
  - The denominator ensures that the  $K$  outputs will sum to 1
- Using softmax we can interpret the network output  $y_k^n$  as an estimate of  $P(k|\mathbf{x}^n)$

# Cross-entropy error function

- Cross-entropy error function:

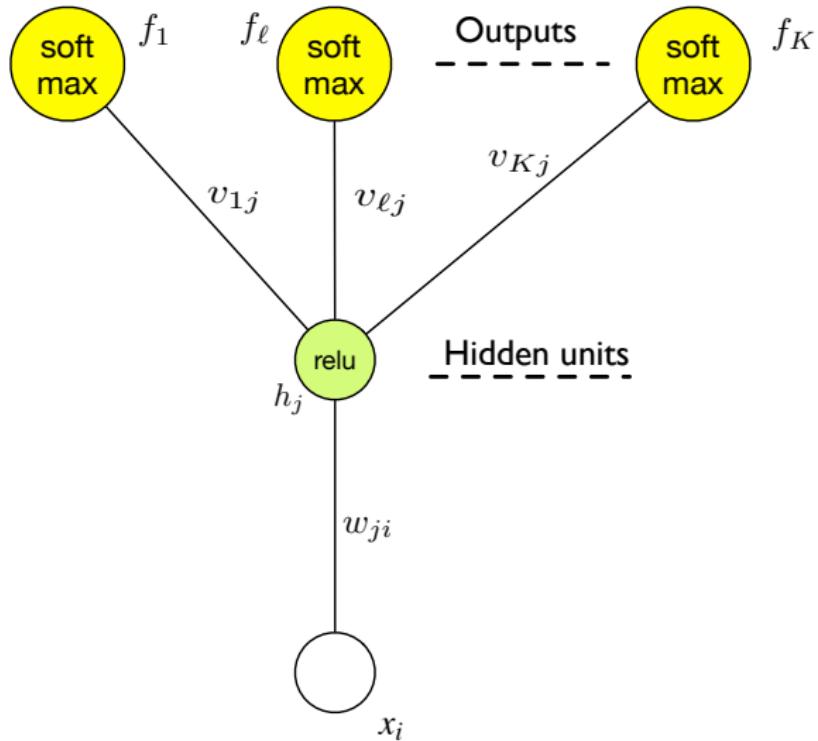
$$E^n = - \sum_{k=1}^C r_k^n \ln y_k^n$$

Optimise the weights  $\mathbf{W}$  to maximise the log probability – or to minimise the negative log probability.

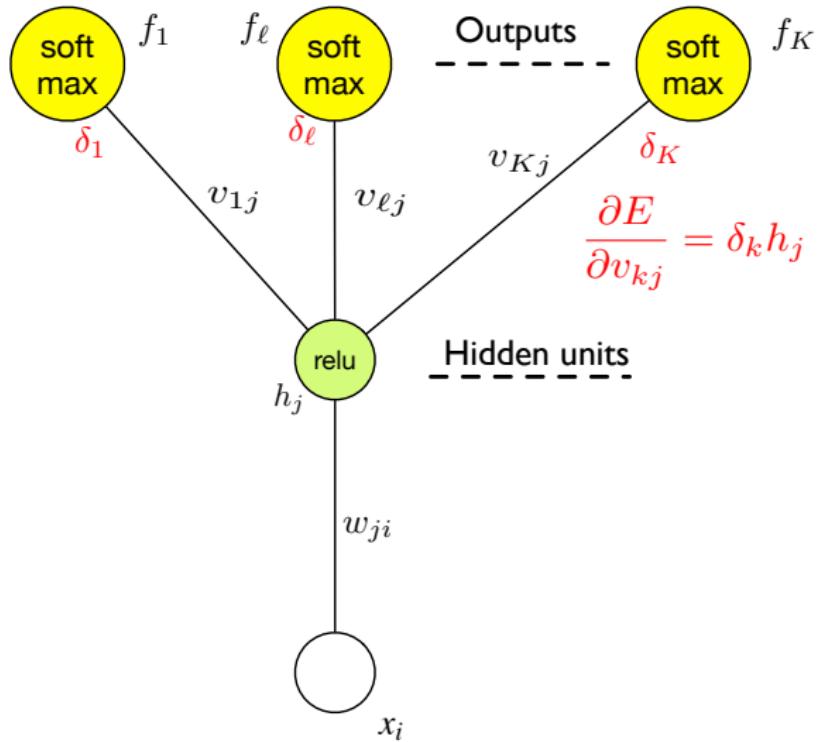
- A neat thing about softmax: if we train with cross-entropy error function, we get a simple form for the gradients of the output weights:

$$\boxed{\frac{\partial E^n}{\partial v_{kj}}} = \underbrace{(y_k - r_k)}_{g_k} h_j$$

# Training multilayered networks – output layer



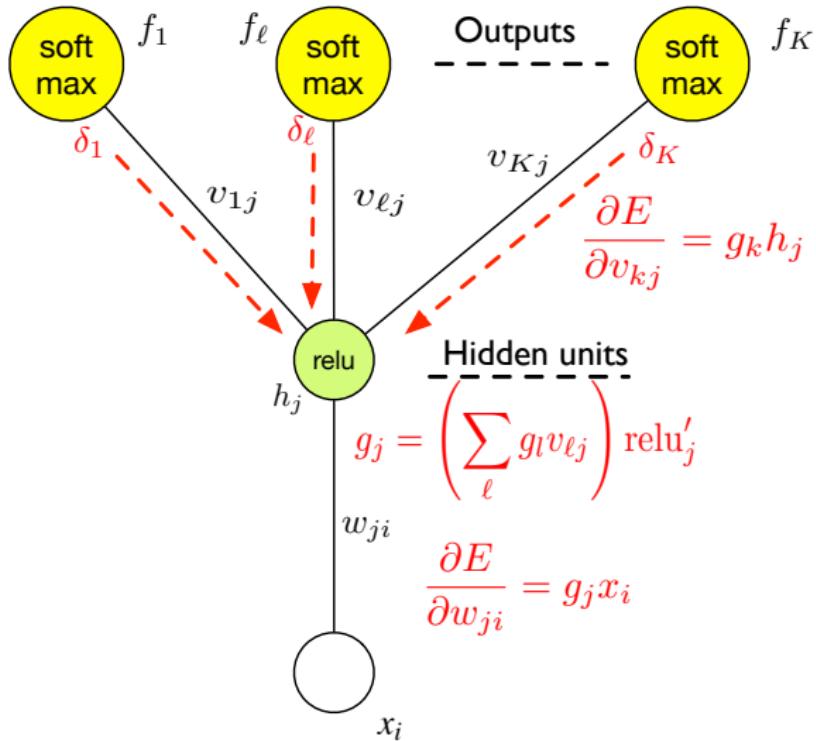
# Training multilayered networks – output layer



# Backprop

- Hidden units make training the weights more complicated, since the hidden units affect the error function indirectly via all the outputs
- The Credit assignment problem: what is the “error” of a hidden unit? how important is input-hidden weight  $w_{ji}$  to output unit  $k$ ?
- Solution: *back-propagate* the deltas through the network
- $g_j$  for a hidden unit is the weighted sum of the deltas of the connected output units. (Propagate the  $g$  values backwards through the network)
- Backprop provides way of estimating the error of each hidden unit

# Backprop



# Back-propagation of error

- The back-propagation of error algorithm is summarised as follows:
  - ① Apply an input vectors from the training set,  $\mathbf{x}$ , to the network and forward propagate to obtain the output vector  $\mathbf{f}$
  - ② Using the target vector  $\mathbf{r}$  compute the error  $E^n$
  - ③ Evaluate the error signals  $g_k$  for each output unit
  - ④ Evaluate the error signals  $g_j$  for each hidden unit using back-propagation of error
  - ⑤ Evaluate the derivatives for each training pattern
- Back-propagation can be extended to multiple hidden layers, in each case computing the  $g_s$  for the current layer as a weighted sum of the  $g_s$  of the next layer

# Summary and Reading

- Single-layer and multi-layer neural networks
- Error functions, weight space and gradient descent training
- Multilayer networks and back-propagation
- Transfer functions – sigmoid and softmax
- Acoustic context
- M Nielsen, *Neural Networks and Deep Learning*,  
<http://neuralnetworksanddeeplearning.com> (chapters 1, 2, and 3)

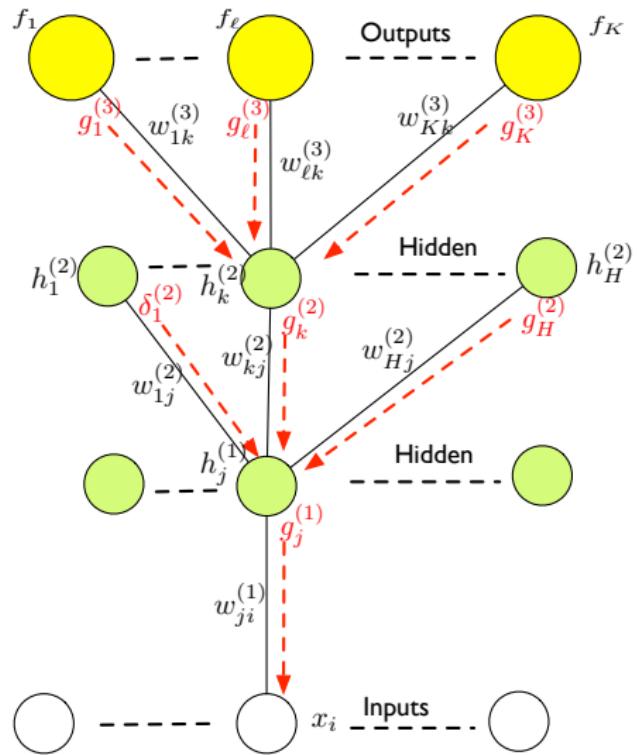
**Next lecture:** Neural network acoustic models

# Neural Networks for Acoustic Modelling part 1

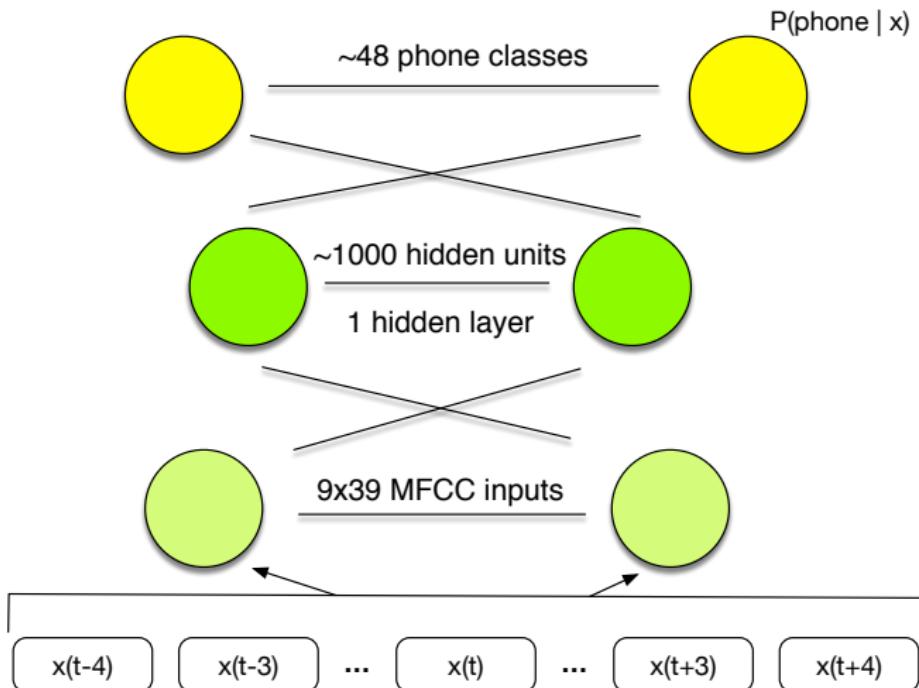
Steve Renals

Automatic Speech Recognition – ASR Lecture 8  
8 February 2018

# Recap: Training NNs using back-propagation of error



# Neural networks for phone classification



# Neural networks for phone classification

- Phone recognition task – e.g. TIMIT corpus
  - 630 speakers (462 train, 168 test) each reading 10 sentences (usually use 8 sentences per speaker, since 2 sentences are the same for all speakers)
  - Speech is labelled by hand at the phone level (time-aligned)
  - 61-phone set, usually reduced to 48/39 phones
- Phone recognition tasks
  - Frame classification – classify each frame of data
  - Phone classification – classify each segment of data (segmentation into unlabelled phones is given)
  - Phone recognition – segment the data and label each segment (the usual speech recognition task)
- Frame classification – straightforward with a neural network
  - train using labelled frames
  - test a frame at a time, assigning the label to the output with the highest score

# Neural networks for phone recognition

- Train a neural network to associate a phone label with a frame of acoustic data (+ context)
- Can interpret the output of the network as  $P(\text{phone} \mid \text{acoustic-frame})$
- **Hybrid NN/HMM systems:** in an HMM, replace the GMMs used to estimate output pdfs with the outputs of neural networks
- One-state per phone HMM system:
  - Train an NN as a phone classifier (= phone probability estimator)
  - Use NN to obtain output probabilities in Viterbi algorithm to find most probable sequence of phones (words)

## Posterior probability estimation

- Consider a neural network trained as a classifier – each output corresponds to a class.
- When applying a trained network to test data, it can be shown that the value of output corresponding to class  $q$  given an input  $\mathbf{x}$ , is an estimate of the posterior probability  $P(q|\mathbf{x})$
- Using Bayes Rule we can relate the posterior  $P(q|\mathbf{x})$  to the likelihood  $p(\mathbf{x}|q)$  used as an output probability in an HMM:

$$P(q|\mathbf{x}) = \frac{p(\mathbf{x}|q)P(q)}{p(\mathbf{x})}$$

(this is assuming 1 state per phone  $q$ )

## Scaled likelihoods

- If we would like to use NN outputs as output probabilities in an HMM, then we would like probabilities (or densities) of the form  $p(\mathbf{x}|q)$  – likelihoods.

We can write *scaled likelihoods* as:

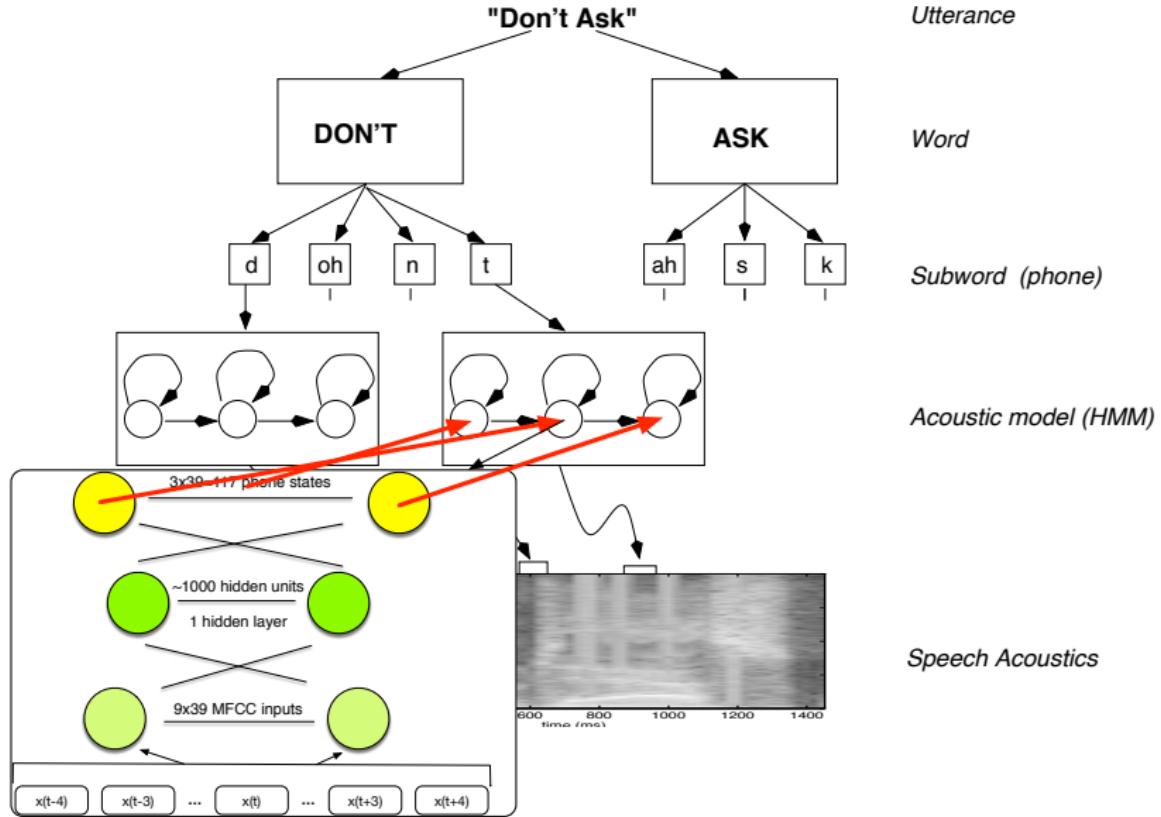
$$\frac{P(q|\mathbf{x})}{p(q)} = \frac{p(\mathbf{x}|q)}{p(\mathbf{x})}$$

- Scaled likelihoods can be obtained by “dividing by the priors” – divide each network output  $P(q|\mathbf{x})$  by  $P(q)$ , the relative frequency of class  $q$  in the training data
- Using  $p(\mathbf{x}|q)/p(\mathbf{x})$  rather than  $p(\mathbf{x}|q)$  is OK since  $p(\mathbf{x})$  does not depend on the class  $q$
- We can use the scaled likelihoods obtained from a neural network in place of the usual likelihoods obtained from a GMM

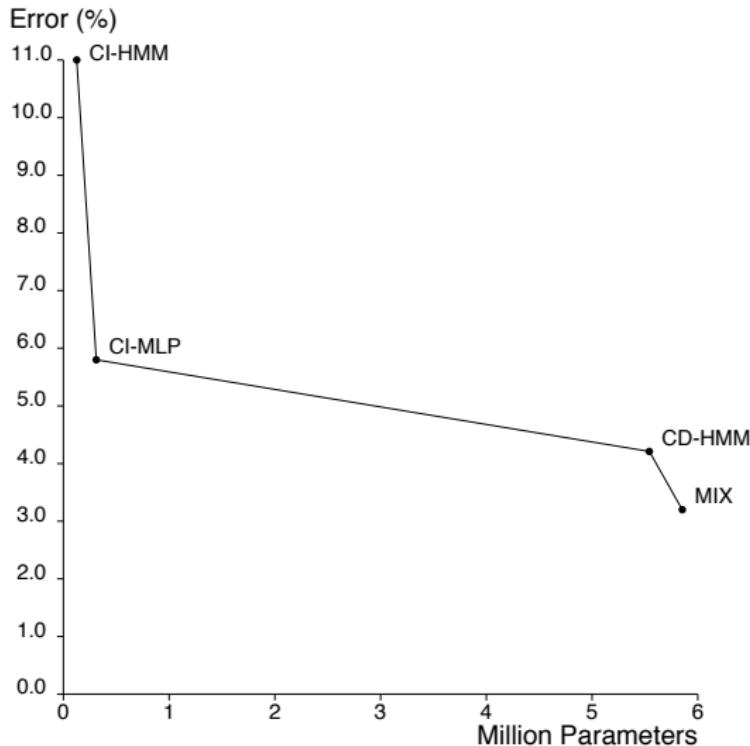
# Hybrid NN/HMM

- If we have a  $K$ -state HMM system, then we train a  $K$ -output NN to estimate the scaled likelihoods used in a hybrid system
- For TIMIT, using a 1 state per phone systems, we obtain scaled likelihoods from a NN trained to classify phones
- For continuous speech recognition we can use:
  - 1 state per phone models
  - 3 state CI models (so we would have an NN with  $39 \times 3 = 117$  outputs)
  - State-clustered models, with one NN output per tied state (this can lead to networks with many outputs!)
- Scaled likelihood and dividing by the priors
  - One can interpret computing the scaled likelihoods as factoring out the prior estimates for each phone based on the acoustic training data. The HMM can then integrate better prior estimates based on the language model and lexicon

# Hybrid NN/HMM

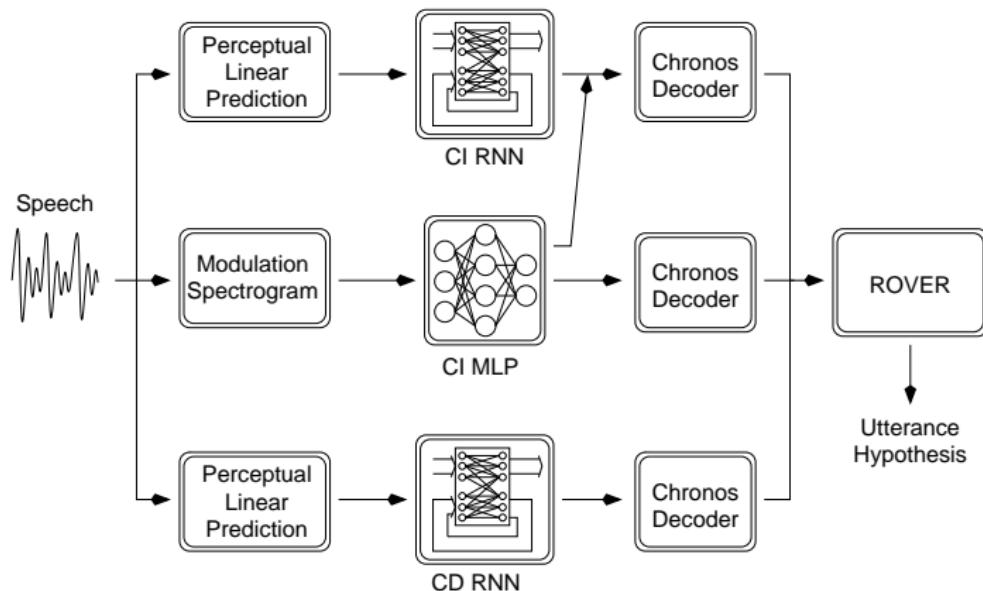


# Monophone HMM/NN hybrid system (1993)



Renals, Morgan, Cohen & Franco, ICASSP 1992

# Monophone HMM/NN hybrid system (1998)

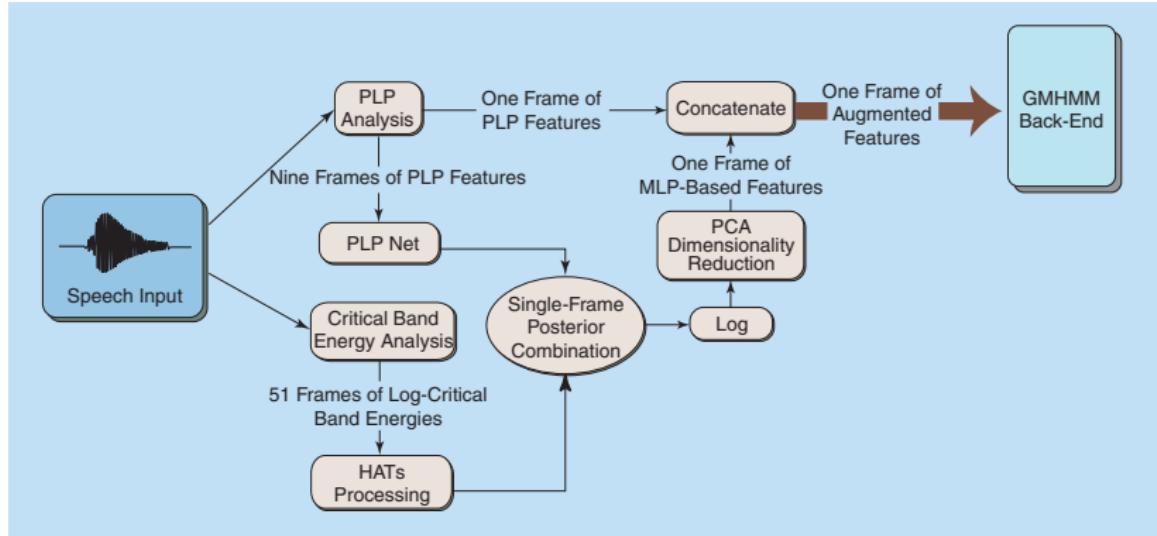


- Broadcast news transcription (1998) – 20.8% WER
- (best GMM-based system, 13.5%)
- Cook et al, DARPA, 1999

# Tandem features (posteriorgrams)

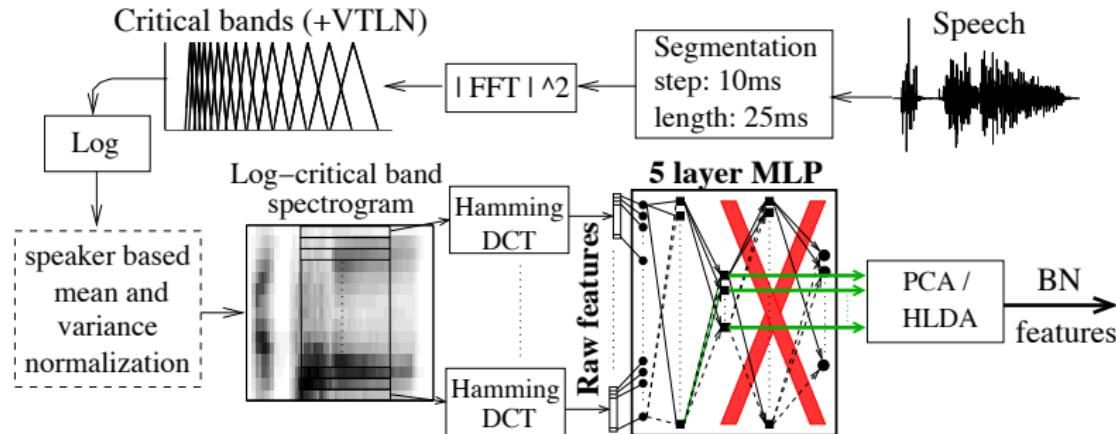
- Use NN probability estimates as an additional input *feature stream* in an HMM/GMM system — (*Tandem* features (i.e. NN + acoustics), posteriorgrams)
- Advantages of tandem features
  - can be estimated using a large amount of temporal context (eg up to  $\pm 25$  frames)
  - encode phone discrimination information
  - only weakly correlated with PLP or MFCC features
- Tandem features: reduce dimensionality of NN outputs using PCA, then concatenate with acoustic features (e.g. MFCCs)
  - PCA also decorrelates feature vector components – important for GMM-based systems

# Tandem features



Morgan et al (2005)

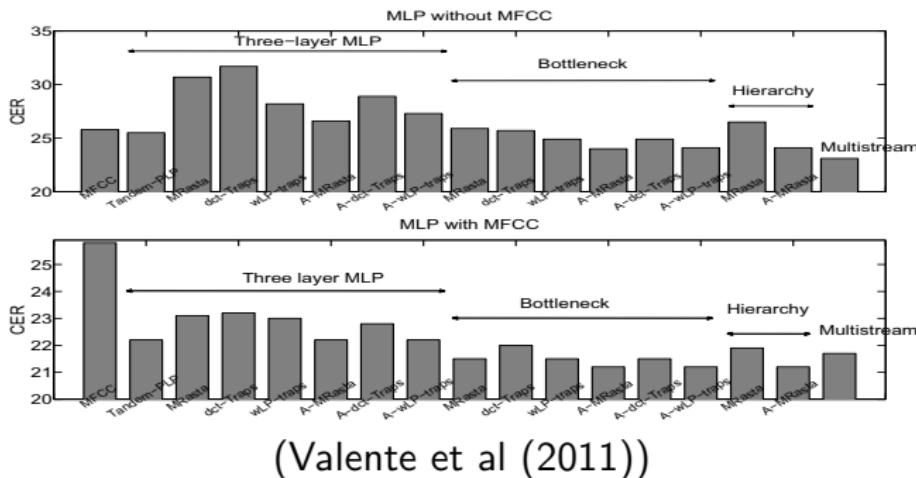
# Bottleneck features



Grezl and Fousek (2008)

- Use a “bottleneck” hidden layer to provide features for a HMM/GMM system
- Decorrelate the hidden layer using PCA (or similar)

# Experimental comparison of tandem and bottleneck features



(Valente et al (2011))

- Results on a Mandarin broadcast news transcription task, using an HMM/GMM system
- Explores many different acoustic features for the NN
- Posteriorgram/bottleneck features alone (top)
- Concatenating NN features with MFCCs (bottom)

# HMM/NN vs HMM/GMM

- Advantages of NN:
  - Can easily model **correlated features**
    - Correlated feature vector components (eg spectral features)
    - Input context – multiple frames of data at input
  - **More flexible** than GMMs – not made of (nearly) local components); GMMs inefficient for non-linear class boundaries
  - NNs can **model multiple events** in the input simultaneously – different sets of hidden units modelling each event; GMMs assume each frame generated by a single mixture component.
  - NNs can **learn richer representations** and learn ‘higher-level’ features (tandem, posteriograms, bottleneck features)

# HMM/NN vs HMM/GMM

- Advantages of NN:
  - Can easily model **correlated features**
    - Correlated feature vector components (eg spectral features)
    - Input context – multiple frames of data at input
  - **More flexible** than GMMs – not made of (nearly) local components); GMMs inefficient for non-linear class boundaries
  - NNs can **model multiple events** in the input simultaneously – different sets of hidden units modelling each event; GMMs assume each frame generated by a single mixture component.
  - NNs can **learn richer representations** and learn ‘higher-level’ features (tandem, posteriograms, bottleneck features)
- Disadvantages of NN:
  - Until ~ 2012:
    - Context-independent (monophone) models, weak speaker adaptation algorithms
    - NN systems less complex than GMMs (fewer parameters): RNN – < 100k parameters, MLP – ~ 1M parameters
  - Computationally expensive - more difficult to parallelise training than GMM systems

# Summary

- Hybrid neural network / HMM systems – using NN acoustic models to compute the output probabilities for HMMs
  - NNs trained as a phone classifier estimate posterior probabilities  $P(\text{phone} \mid \text{acoustic-frame})$
  - Scaled likelihoods – divide by the phone priors to obtain scaled likelihoods to use as HMM output probabilities
- Neural network features – append features obtained from a trained NN to acoustic features (e.g. MFCCs)
  - Tandem / posteriorgram: use the (transformed) output of an NN trained as a phone classifier as additional features for a GMM system
  - Bottleneck features: use the (transformed) hidden layer output of an NN trained as a phone classifier as additional features for a GMM system

**Next lecture:** Deep neural network acoustic models

# Reading

- N Morgan and H Bourlard (May 1995). “Continuous speech recognition: Introduction to the hybrid HMM/connectionist approach”, *IEEE Signal Processing Mag.*, **12**(3), 24–42.  
<http://ieeexplore.ieee.org/document/382443>
- N Morgan et al (Sep 2005). “Pushing the envelope – aside”, *IEEE Signal Processing Mag.*, **22**(5), 81–88.  
<http://ieeexplore.ieee.org/document/1511826>
- F Grezl and P Fousek (2008). “Optimizing bottleneck features for LVCSR”, Proc ICASSP–2008.  
<http://ieeexplore.ieee.org/document/4518713>
- F Valente et al (2011). “Analysis and Comparison of Recent MLP Features for LVCSR Systems”, Proc Interspeech–2011.  
[http://www.isca-speech.org/archive/interspeech\\_2011/i11\\_1245.html](http://www.isca-speech.org/archive/interspeech_2011/i11_1245.html)

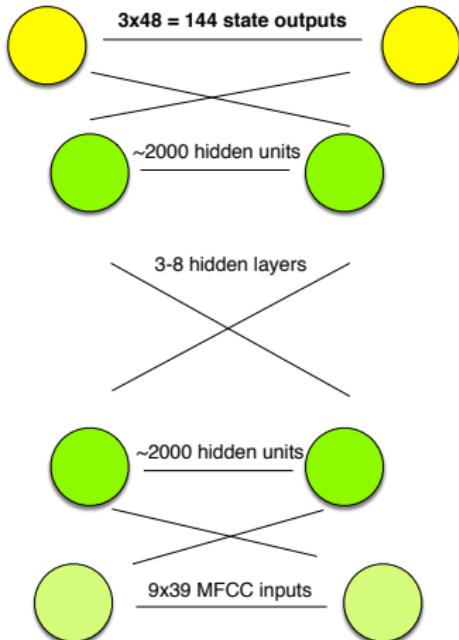
# Neural Networks for Acoustic Modelling part 2

Steve Renals

Automatic Speech Recognition – ASR Lecture 9  
12 February 2018

# DNN Acoustic Models

# Deep neural network for TIMIT



- **Deeper:** Deep neural network architecture – multiple hidden layers
- **Wider:** Use HMM state alignment as outputs rather than hand-labelled phones – 3-state HMMs, so  $3 \times 48$  states
- Can use *pretraining* to improve training accuracy of models with many hidden layers
- Training many hidden layers is computationally expensive – use GPUs to provide the computational power

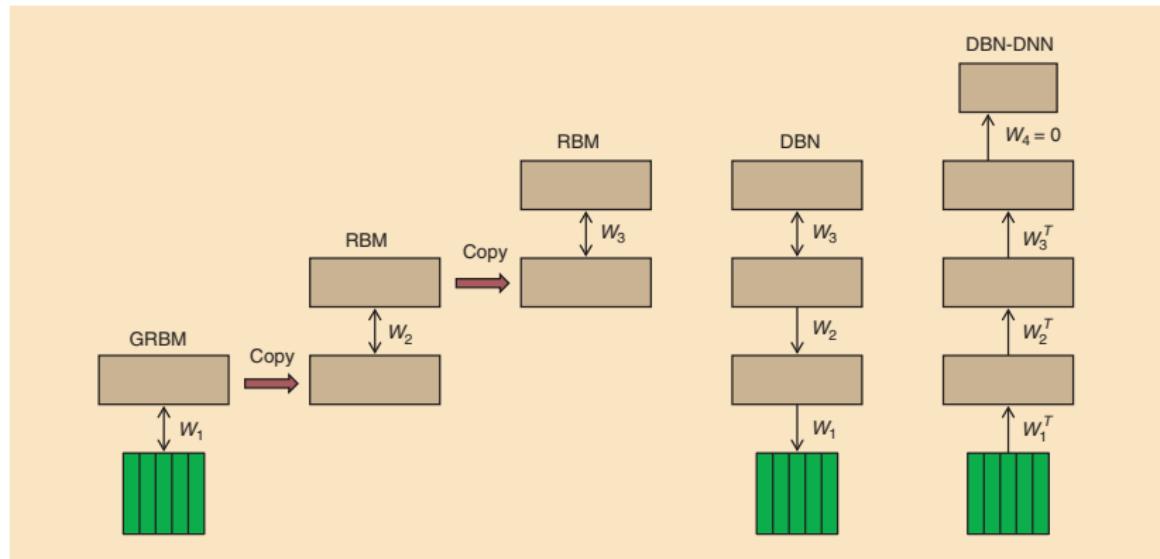
# Pretraining

- Training multi-hidden layers directly with gradient descent is difficult — sensitive to initialisation, gradients can be very small after propagating back through several layers.
- **Unsupervised pretraining**
  - Train a stacked restricted Boltzmann machine generative model (unsupervised, contrastive divergence training), then finetune with backprop
  - Train a stacked autoencoder, then finetune with backprop

## Layer-by-layer training

- Successively train deeper networks, each time replacing output layer with hidden layer and new output layer

# Unsupervised pretraining



Hinton et al (2012)

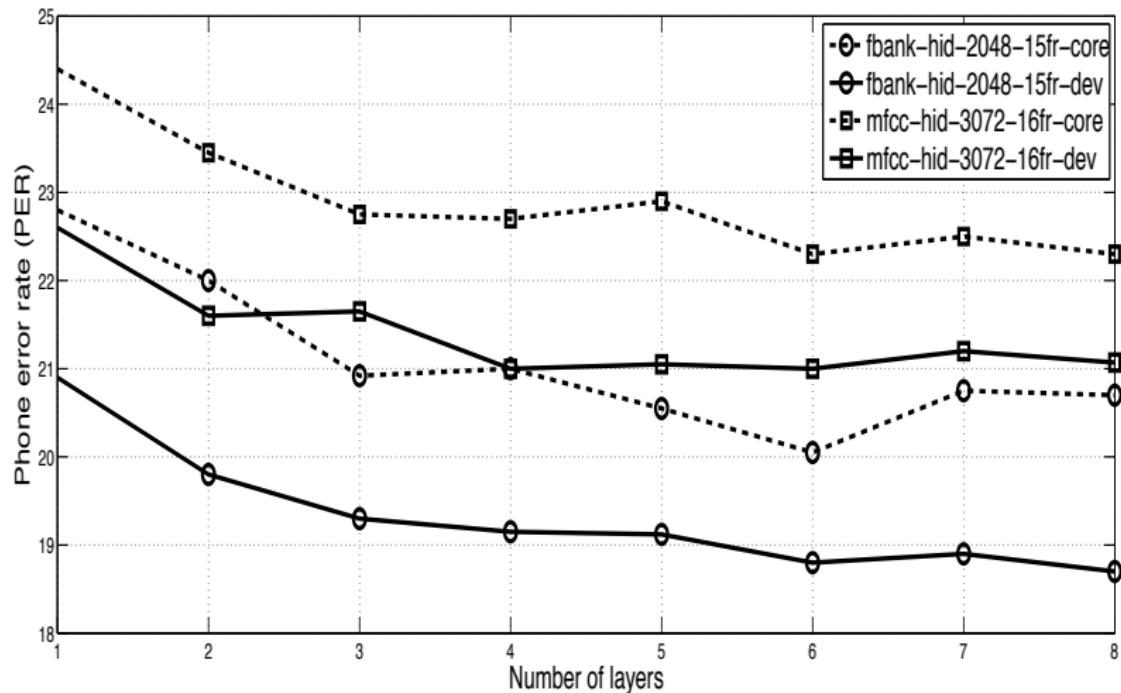
# Hybrid HMM/DNN phone recognition (TIMIT)

- Train a ‘baseline’ three state monophone HMM/GMM system (61 phones, 3 state HMMs) and Viterbi align to provide DNN training targets (time state alignment)
- The HMM/DNN system uses the same set of states as the HMM/GMM system — DNN has 183 ( $61 \times 3$ ) outputs
- Hidden layers — many experiments, exact sizes not highly critical
  - 3–8 hidden layers
  - 1024–3072 units per hidden layer
- Multiple hidden layers always work better than one hidden layer
- Pretraining always results in lower error rates
- Best systems have lower phone error rate than best HMM/GMM systems (using state-of-the-art techniques such as discriminative training, speaker adaptive training)

# Acoustic features for NN acoustic models

- GMMs: filter bank features (spectral domain) not used as they are strongly correlated with each other – would either require
  - full covariance matrix Gaussians
  - many diagonal covariance Gaussians
- DNNs do not require the components of the feature vector to be uncorrelated
  - Can directly use multiple frames of input context (this has been done in NN/HMM systems since 1990, and is crucial to make them work)
  - Can potentially use feature vectors with correlated components (e.g. filter banks)
- Experiments indicate that filter bank features result in greater accuracy than MFCCs

# TIMIT phone error rates: effect of depth and feature type

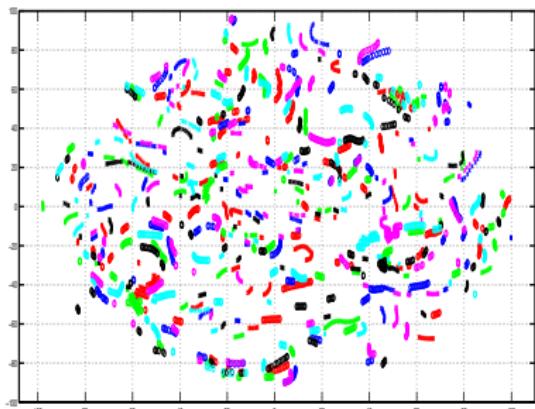


(Mohamed et al (2012))

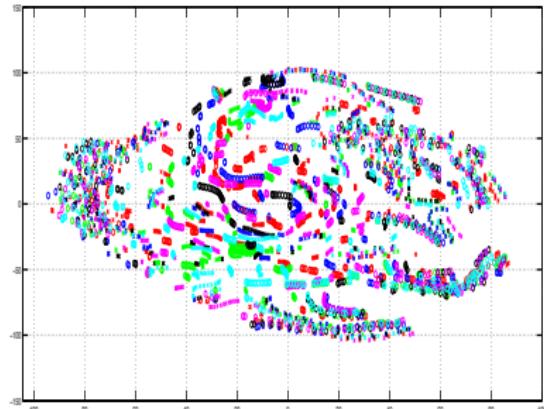
# Visualising neural networks

- How to visualise NN layers? “t-SNE” (stochastic neighbour embedding using t-distribution) projects high dimension vectors (e.g. the values of all the units in a layer) into 2 dimensions
- t-SNE projection aims to keep points that are close in high dimensions close in 2 dimensions by comparing distributions over pairwise distances between the high dimensional and 2 dimensional spaces – the optimisation is over the positions of points in the 2-d space

# Feature vector (input layer): t-SNE visualisation



MFCC



FBANK

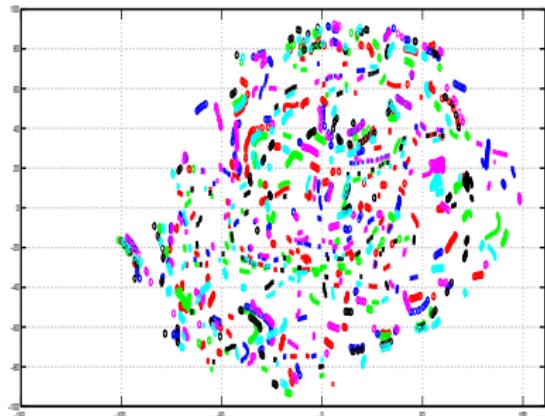
(Mohamed et al (2012))

Visualisation of 2 utterances (cross and circle) spoken by 6 speakers (colours)

MFCCs are more scattered than FBANK

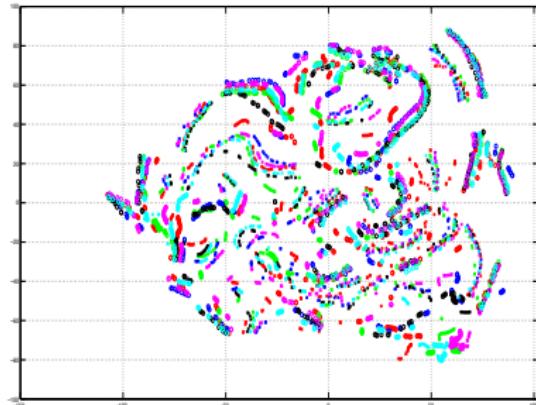
FBANK has more local structure than MFCCs

# First hidden layer: t-SNE visualisation



MFCC

(Mohamed et al (2012))

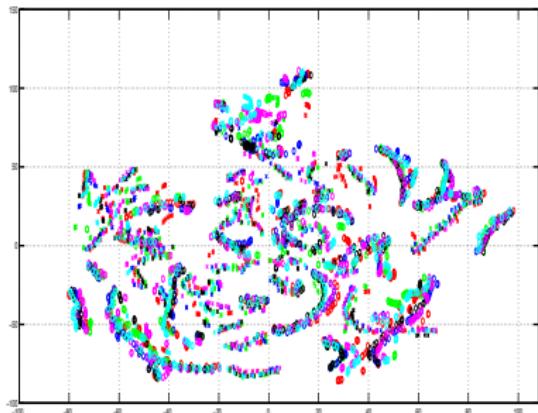


FBANK

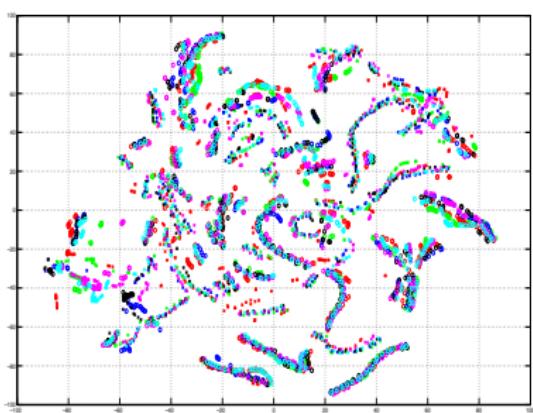
Visualisation of 2 utterances (cross and circle) spoken by 6 speakers (colours)

Hidden layer vectors start to align more between speakers for FBANK

# Eighth hidden layer: t-SNE visualisation



MFCC



FBANK

(Mohamed et al (2012))

Visualisation of 2 utterances (cross and circle) spoken by 6 speakers (colours)

In the final hidden layer, the hidden layer outputs for the same phone are well-aligned across speakers for both MFCC and FBANK – but stronger for FBANK

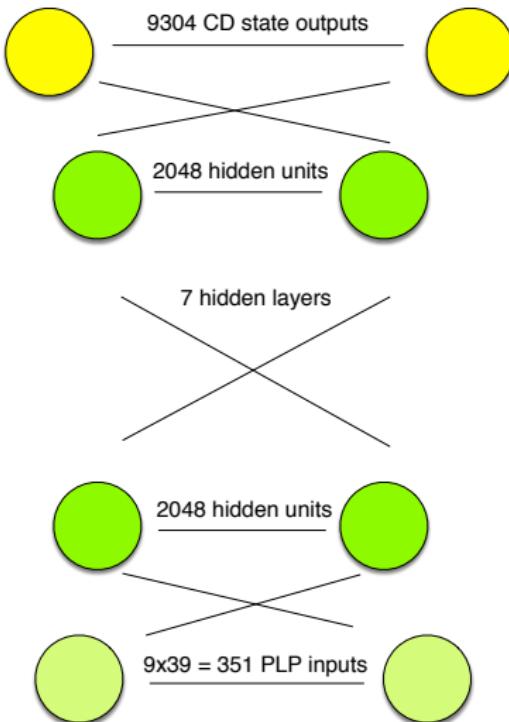
# Visualising neural networks

- How to visualise NN layers? “t-SNE” (stochastic neighbour embedding using t-distribution) projects high dimension vectors (e.g. the values of all the units in a layer) into 2 dimensions
- t-SNE projection aims to keep points that are close in high dimensions close in 2 dimensions by comparing distributions over pairwise distances between the high dimensional and 2 dimensional spaces – the optimisation is over the positions of points in the 2-d space

Are the differences due to FBANK being higher dimension ( $41 \times 3 = 123$ ) than MFCC ( $13 \times 3 = 39$ )?

- No – Using higher dimension MFCCs, or just adding noisy dimensions to MFCCs results in higher error rate
- Why? – In FBANK the useful information is distributed over all the features; in MFCC it is concentrated in the first few.

# DNN acoustic model for Switchboard



(Hinton et al (2012))

## Example: hybrid HMM/DNN large vocabulary conversational speech recognition (Switchboard)

- Recognition of American English conversational telephone speech (Switchboard)
- Baseline context-dependent HMM/GMM system
  - 9,304 tied states
  - Discriminatively trained (BMMI — similar to MPE)
  - 39-dimension PLP (+ derivatives) features
  - Trained on 309 hours of speech
- Hybrid HMM/DNN system
  - Context-dependent — 9304 output units obtained from Viterbi alignment of HMM/GMM system
  - 7 hidden layers, 2048 units per layer
- DNN-based system results in significant word error rate reduction compared with GMM-based system
- Pretraining not necessary on larger tasks (empirical result)

# DNN vs GMM on large vocabulary tasks (Experiments from 2012)

[TABLE 3] A COMPARISON OF THE PERCENTAGE WERs USING DNN-HMMs AND GMM-HMMs ON FIVE DIFFERENT LARGE VOCABULARY TASKS.

TASK	HOURS OF TRAINING DATA	DNN-HMM	GMM-HMM WITH SAME DATA	GMM-HMM WITH MORE DATA
SWITCHBOARD (TEST SET 1)	309	18.5	27.4	18.6 (2,000 H)
SWITCHBOARD (TEST SET 2)	309	16.1	23.6	17.1 (2,000 H)
ENGLISH BROADCAST NEWS	50	17.5	18.8	
BING VOICE SEARCH (SENTENCE ERROR RATES)	24	30.4	36.2	
GOOGLE VOICE INPUT	5,870	12.3		16.0 (>> 5,870 H)
YOUTUBE	1,400	47.6	52.3	

(Hinton et al (2012))

# Summary

- DNN/HMM systems (hybrid systems) give a significant improvement over GMM/HMM systems
- Compared with 1990s NN/HMM systems, DNN/HMM systems
  - model context-dependent tied states with a much wider output layer
  - are deeper – more hidden layers
  - can use correlated features (e.g. FBANK)

**Next lecture:** Lexicon and language model

# Reading

- G Hinton et al (Nov 2012). “Deep neural networks for acoustic modeling in speech recognition”, *IEEE Signal Processing Magazine*, **29**(6), 82–97.  
<http://ieeexplore.ieee.org/document/6296526>
- A Mohamed et al (2012). “Unserstanding how deep belief networks perform acoustic modelling”, Proc ICASSP-2012. [http://www.cs.toronto.edu/~asamir/papers/icassp12\\_dbn.pdf](http://www.cs.toronto.edu/~asamir/papers/icassp12_dbn.pdf)

# Lexicon and Language Model

Steve Renals

Automatic Speech Recognition – ASR Lecture 10  
15 February 2018

# Three levels of model

- **Acoustic model**  $P(X | Q)$

Probability of the acoustics given the phone states:  
context-dependent HMMs using state clustering, phonetic  
decision trees, etc.

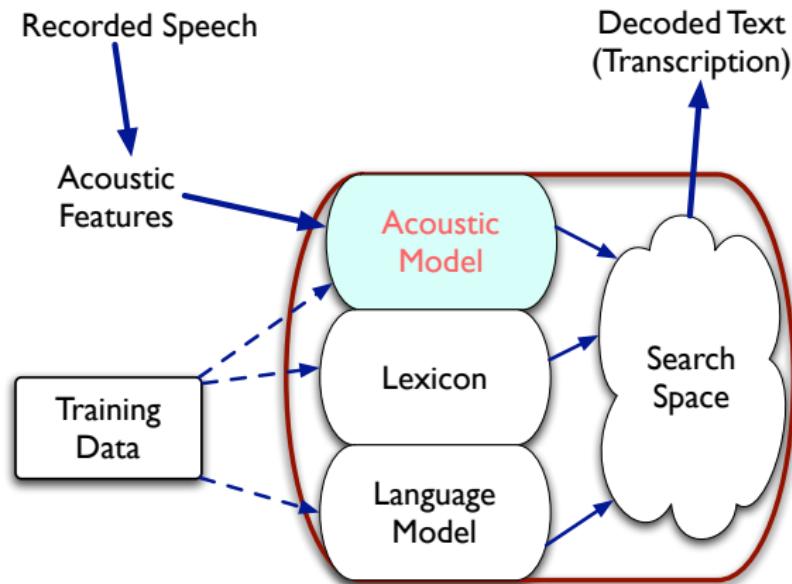
- **Pronunciation model**  $P(Q | W)$

Probability of the phone states given the words; may be as  
simple a dictionary of pronunciations, or a more complex  
model

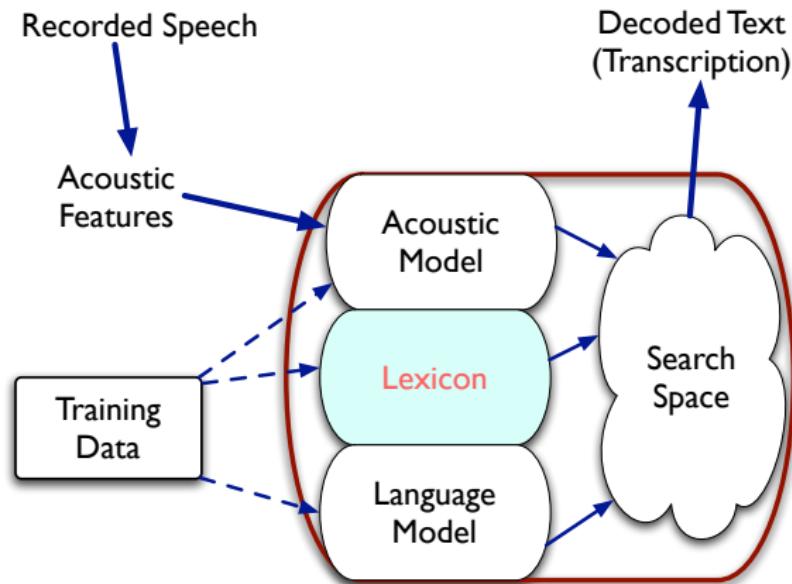
- **Language model**  $P(W)$

Probability of a sequence of words. Typically an  $n$ -gram

# HMM Speech Recognition



# HMM Speech Recognition



# Pronunciation dictionary

- Words and their pronunciations provide the link between sub-word HMMs and language models
- Written by human experts
- Typically based on phones

# Pronunciation dictionary

- Words and their pronunciations provide the link between sub-word HMMs and language models
- Written by human experts
- Typically based on phones
- Constructing a dictionary involves
  - ① Selection of the words in the dictionary—want to ensure high coverage of words in test data
  - ② Representation of the pronunciation(s) of each word

# Pronunciation dictionary

- Words and their pronunciations provide the link between sub-word HMMs and language models
- Written by human experts
- Typically based on phones
- Constructing a dictionary involves
  - ① Selection of the words in the dictionary—want to ensure high coverage of words in test data
  - ② Representation of the pronunciation(s) of each word
- Explicit modelling of pronunciation variation

# Out-of-vocabulary (OOV) rate

- OOV rate: percent of word tokens in test data that are not contained in the ASR system dictionary
- Training vocabulary requires pronunciations for *all* words in training data (since training requires an HMM to be constructed for each training utterance)
- Select the recognition vocabulary to minimize the OOV rate (by testing on development data)
- Recognition vocabulary may be different to training vocabulary
- Empirical result: each OOV word results in 1.5–2 extra errors (>1 due to the loss of contextual information)

# Multilingual aspects

- Many languages are morphologically richer than English: this has a major effect of vocabulary construction and language modelling
- **Compounding** (eg German): decompose compound words into constituent parts, and carry out pronunciation and language modelling on the decomposed parts
- **Highly inflected languages** (eg Arabic, Slavic languages): specific components for modelling inflection (eg factored language models)
- **Inflecting and compounding languages** (eg Finnish)
- All approaches aim to reduce ASR errors by reducing the OOV rate through modelling at the morph level; also addresses data sparsity

# Single and multiple pronunciations

- Words may have multiple pronunciations:
  - ➊ Accent, dialect: *tomato, zebra*  
global changes to dictionary based on consistent pronunciation variations
  - ➋ Phonological phenomena: *handbag/ h ae m b ae g*  
*I can't stay / [ah k ae n s t ay]*
  - ➌ Part of speech: *project, excuse*

# Single and multiple pronunciations

- Words may have multiple pronunciations:
  - ➊ Accent, dialect: *tomato, zebra*  
global changes to dictionary based on consistent pronunciation variations
  - ➋ Phonological phenomena: *handbag/ h ae m b ae g*  
*I can't stay / [ah k ae n s t ay]*
  - ➌ Part of speech: *project, excuse*
- This seems to imply many pronunciations per word, including:
  - ➊ Global transform based on speaker characteristics
  - ➋ Context-dependent pronunciation models, encoding of phonological phenomena

# Single and multiple pronunciations

- Words may have multiple pronunciations:
  - ➊ Accent, dialect: *tomato, zebra*  
global changes to dictionary based on consistent pronunciation variations
  - ➋ Phonological phenomena: *handbag/ h ae m b ae g*  
*I can't stay / [ah k ae n s t ay]*
  - ➌ Part of speech: *project, excuse*
- This seems to imply many pronunciations per word, including:
  - ➊ Global transform based on speaker characteristics
  - ➋ Context-dependent pronunciation models, encoding of phonological phenomena
- **BUT** state-of-the-art large vocabulary systems average about 1.1 pronunciations per word: most words have a single pronunciation

# Consistency vs Fidelity

- **Empirical finding:** adding pronunciation variants can result in reduced accuracy
- Adding pronunciations gives more “flexibility” to word models and increases the number of potential ambiguities—more possible state sequences to match the observed acoustics

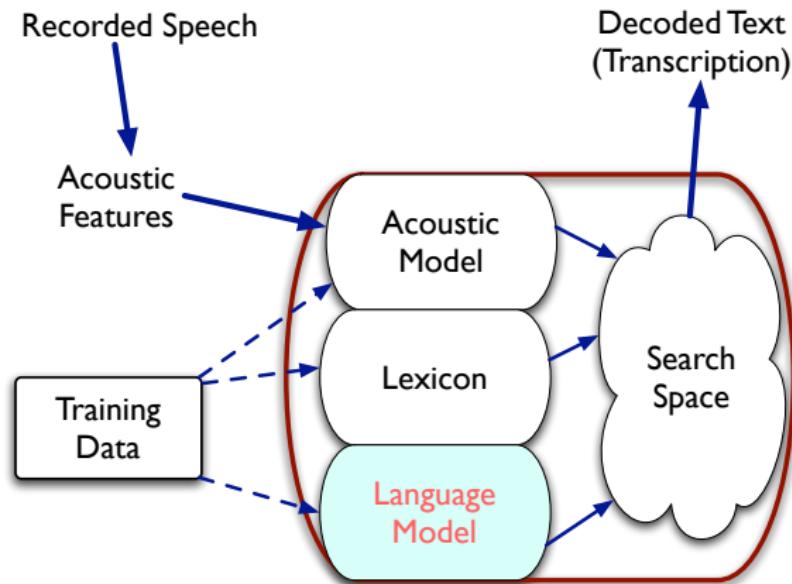
# Consistency vs Fidelity

- **Empirical finding:** adding pronunciation variants can result in reduced accuracy
- Adding pronunciations gives more “flexibility” to word models and increases the number of potential ambiguities—more possible state sequences to match the observed acoustics
- Speech recognition uses a **consistent** rather than a **faithful** representation of pronunciations
- A consistent representation requires only that the same word has the same phonemic representation (possibly with alternates): the training data need only be transcribed at the word level
- A faithful phonemic representation requires a detailed phonetic transcription of the training speech (much too expensive for large training data sets)

# Current topics in pronunciation modelling

- Automatic learning of pronunciation variations or alternative pronunciations for some words – e.g. learning probability distribution over possible pronunciations generated by grapheme-to-phoneme models
  - Automatic learning of pronunciations of new words based on an initial seed lexicon
- Joint learning of the inventory of subword units and the pronunciation lexicon
- Sub-phonetic / articulatory feature model
- Grapheme-based modelling: model at the character level and remove the problem of pronunciation modelling entirely

# HMM Speech Recognition



# Statistical language models

- **Basic idea** The language model is the prior probability of the word sequence  $P(W)$
- Statistical language models: cover “ungrammatical” utterances, computationally efficient, trainable from huge amounts of data, can assign a probability to a sentence fragment as well as a whole sentence
- Until very recently **n-grams** were the state-of-the-art language model for ASR
  - Unsophisticated, linguistically implausible
  - Short, finite context
  - Model solely at the shallow word level
  - But: wide coverage, able to deal with “ungrammatical” strings, statistical and scaleable
- In an n-gram, the probability of a word depends only on the identity of that word and of the preceding n-1 words. These short sequences of n words are called n-grams.

# Bigram language model

- Word sequence  $\mathbf{W} = w_1, w_2, \dots, w_M$

$$P(\mathbf{W}) = P(w_1)P(w_2 | w_1)P(w_3 | w_1, w_2) \\ \dots P(w_M | w_1, w_2, \dots, w_{M-1})$$

- Bigram approximation—consider only one word of context:

$$P(\mathbf{W}) \simeq P(w_1)P(w_2 | w_1)P(w_3 | w_2) \dots P(w_M | w_{M-1})$$

# Bigram language model

- Word sequence  $\mathbf{W} = w_1, w_2, \dots, w_M$

$$P(\mathbf{W}) = P(w_1)P(w_2 | w_1)P(w_3 | w_1, w_2) \\ \dots P(w_M | w_1, w_2, \dots, w_{M-1})$$

- Bigram approximation—consider only one word of context:

$$P(\mathbf{W}) \simeq P(w_1)P(w_2 | w_1)P(w_3 | w_2) \dots P(w_M | w_{M-1})$$

- Parameters of a bigram are the conditional probabilities  $P(w_j | w_i)$
- Maximum likelihood estimates by counting:

$$P(w_j | w_i) \sim \frac{c(w_i, w_j)}{c(w_i)}$$

where  $c(w_i, w_j)$  is the number of observations of  $w_i$  followed by  $w_j$ , and  $c(w_i)$  is the number of observations of  $w_i$  (irrespective of what follows)

# The zero probability problem

- Maximum likelihood estimation is based on counts of words in the training data
- If a n-gram is not observed, it will have a count of 0—and the maximum likelihood probability estimate will be 0
- The **zero probability** problem: just because something does not occur in the training data does not mean that it will not occur
- As n grows larger, so the data grow sparser, and the more zero counts there will be

# The zero probability problem

- Maximum likelihood estimation is based on counts of words in the training data
- If a n-gram is not observed, it will have a count of 0—and the maximum likelihood probability estimate will be 0
- The **zero probability** problem: just because something does not occur in the training data does not mean that it will not occur
- As  $n$  grows larger, so the data grow sparser, and the more zero counts there will be
- Solution: **smooth** the probability estimates so that unobserved events do not have a zero probability
- Since probabilities sum to 1, this means that some probability is redistributed from observed to unobserved n-grams

# Smoothing language models

- What is the probability of an unseen n-gram?

# Smoothing language models

- What is the probability of an unseen n-gram?
- Add-one smoothing: add one to all counts and renormalize.
  - “Discounts” non-zero counts and redistributes to zero counts
  - Since most n-grams are unseen (for large n more types than tokens!) this gives too much probability to unseen n-grams (discussed in Manning and Schütze)
- Absolute discounting: subtract a constant from the observed (non-zero count) n-grams, and redistribute this subtracted probability over the unseen n-grams (zero counts)
- Kneser-Ney smoothing: family of smoothing methods based on absolute discounting that are at the state of the art (Goodman, 2001)

# Backing off

- How is the probability distributed over unseen events?
- Basic idea: estimate the probability of an unseen n-gram using the (n-1)-gram estimate
- Use successively less context: trigram → bigram → unigram
- Back-off models redistribute the probability “freed” by discounting the n-gram counts

# Backing off

- How is the probability distributed over unseen events?
- Basic idea: estimate the probability of an unseen n-gram using the (n-1)-gram estimate
- Use successively less context: trigram → bigram → unigram
- Back-off models redistribute the probability “freed” by discounting the n-gram counts
- For a bigram

$$\begin{aligned} P(w_j | w_i) &= \frac{c(w_i, w_j) - D}{c(w_i)} \quad \text{if } c(w_i, w_j) > c \\ &= P(w_j) b_{w_i} \quad \text{otherwise} \end{aligned}$$

$c$  is the count threshold, and  $D$  is the discount.  $b_{w_i}$  is the backoff weight required for normalization

# Interpolation

- **Basic idea:** Mix the probability estimates from all the estimators: estimate the trigram probability by mixing together trigram, bigram, unigram estimates
- Simple interpolation

$$\hat{P}(w_n \mid w_{n-2}, w_{n-1}) = \lambda_3 P(w_n \mid w_{n-2}, w_{n-1}) + \lambda_2 P(w_n \mid w_{n-1}) + \lambda_1 P(w_n)$$

With  $\sum_i \lambda_i = 1$

- Interpolation with coefficients conditioned on the context

$$\begin{aligned} \hat{P}(w_n \mid w_{n-2}, w_{n-1}) &= \\ &\lambda_3(w_{n-2}, w_{n-1}) P(w_n \mid w_{n-2}, w_{n-1}) + \\ &\lambda_2(w_{n-2}, w_{n-1}) P(w_n \mid w_{n-1}) + \lambda_1(w_{n-2}, w_{n-1}) P(w_n) \end{aligned}$$

- Set  $\lambda$  values to maximise the likelihood of the interpolated language model generating a *held-out* corpus (possible to use EM to do this)

# Perplexity

- Measure the quality of a language model by how well it predicts a test set  $W$  (i.e. estimated probability of word sequence)
- Perplexity ( $PP(W)$ ) – inverse probability of the test set  $W$ , normalized by the number of words  $N$

$$PP(W) = P(W)^{\frac{-1}{N}} = P(w_1 w_2 \dots w_N)^{\frac{-1}{N}}$$

- Perplexity of a bigram LM

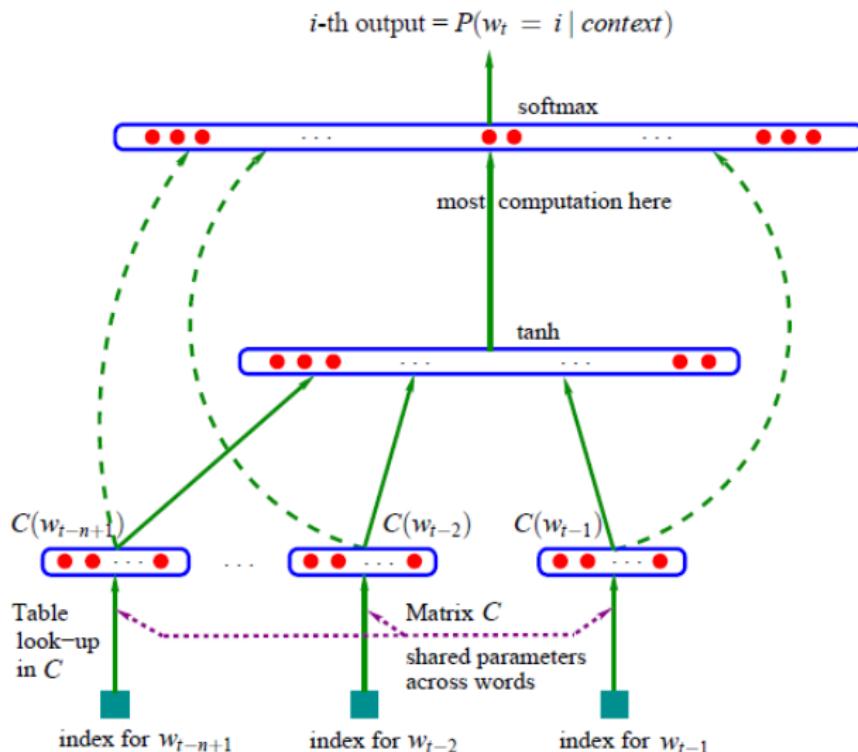
$$PP(W) = (P(w_1)P(w_2|w_1)P(w_3|w_2)\dots P(w_N|w_{N-1}))^{\frac{-1}{N}}$$

- Example perplexities for different n-gram LMs trained on Wall St Journal (38M words)
  - Unigram – 962
  - Bigram – 170
  - Trigram – 109

# Distributed representation for language modelling

- Each word is associated with a learned *distributed representation* (feature vector)
- Use a neural network to estimate the conditional probability of the next word given the distributed representations of the context words
- Learn the distributed representations and the weights of the conditional probability estimate jointly by maximising the log likelihood of the training data
- Similar words (distributionally) will have similar feature vectors — small change in feature vector will result in small change in probability estimate (since the NN is a smooth function)

# Neural Probabilistic Language Model



Bengio et al (2006)

# Neural Probabilistic Language Model

- Train using stochastic gradient ascent to maximise log likelihood
- Number of free parameters (weights) scales
  - Linearly with vocabulary size
  - Linearly with context size
- Can be (linearly) interpolated with n-gram model
- Perplexity results on AP News (14M words training).  
 $|V| = 18k$

model	n	perplexity
NPLM(100,60)	6	109
n-gram (KN)	3	127
n-gram (KN)	4	119
n-gram (KN)	5	117

# Shortlists

- Reduce computation by only including the  $s$  most frequent words at the output — the *shortlist* ( $S$ ) (full vocabulary still used for context)
- Use an n-gram model to estimate probabilities of words not in the shortlist
- Neural network thus redistributes probability for the words in the shortlist

$$P_S(h_t) = \sum_{w \in S} P(w|h_t)$$

$$P(w_t|h_t) = \begin{cases} P_{NN}(w_t|h_t)P_S(h_t) & \text{if } w_t \in S \\ P_{KN}(w_t|h_t) & \text{else} \end{cases}$$

- In a  $|V| = 50k$  task a 1024 word shortlist covers 89% of 4-grams, 4096 words covers 97%

# NPLM — ASR results

Speech recognition results on Switchboard

7M / 12M / 27M words in domain data.

500M words background data (broadcast news)

Vocab size  $|V| = 51k$ , Shortlist size  $|S| = 12k$

	WER/%		
in-domain words	7M	12M	27M
KN (in-domain)	25.3	23.0	20.0
NN (in-domain)	24.5	22.2	19.1
KN (+b/g)	24.1	22.3	19.3
NN (+b/g)	23.7	21.8	18.9

# Summary

- Pronunciation dictionaries
- n-gram language models
- Neural network language models

# Reading

- Jurafsky and Martin, chapter 4
- Y Bengio et al (2006), “Neural probabilistic language models” (sections 6.1, 6.2, 6.3, 6.6, 6.7, 6.8), Studies in Fuzziness and Soft Computing Volume 194, Springer, chapter 6. [http://link.springer.com/chapter/10.1007/3-540-33486-6\\_6](http://link.springer.com/chapter/10.1007/3-540-33486-6_6)

# Speaker Adaptation

Steve Renals

Automatic Speech Recognition – ASR Lecture 11  
22 March 2018

# Speaker independent / dependent / adaptive

- Speaker independent (SI) systems have long been the focus for research in transcription, dialogue systems, etc.
- Speaker dependent (SD) systems can result in word error rates 2–3 times lower than SI systems (given the same amount of training data)
- A Speaker adaptive (SA) system... we would like
  - Error rates similar to SD systems
  - Building on an SI system
  - Requiring only a small fraction of the speaker-specific training data used by an SD system

# Speaker-specific variation

- **Acoustic model**

- Speaking styles
- Accents
- Speech production anatomy (eg length of the vocal tract)

Also non-speaker variation, such as channel conditions  
(telephone, reverberant room, close talking mic) and  
application domain

Speaker adaptation of acoustic models aims to reduce the  
mismatch between test data and the models

# Speaker-specific variation

- **Acoustic model**
  - Speaking styles
  - Accents
  - Speech production anatomy (eg length of the vocal tract)

Also non-speaker variation, such as channel conditions (telephone, reverberant room, close talking mic) and application domain

Speaker adaptation of acoustic models aims to reduce the mismatch between test data and the models

- **Pronunciation model:** speaker-specific, consistent change in pronunciation
- **Language model:** user-specific documents (exploited in personal dictation systems)

# Modes of adaptation

- Supervised or unsupervised
  - **Supervised:** the word level transcription of the adaptation data is known
  - **Unsupervised:** no transcription provided
- Static or dynamic
  - **Static:** Adaptation data presented to the system in a block before the final system is estimated (eg enrolment in a dictation system)
  - **Dynamic:** Adaptation data incrementally available, models must be adapted before all adaptation data is available (eg spoken dialogue system)
- Desirable properties for speaker adaptation
  - **Compact:** relatively few speaker-dependent parameters
  - **Unsupervised:** does not require labelled adaptation data
  - **Efficient:** low computational requirements
  - **Flexible:** applicable to different model variants

# Approaches to adaptation

- **Model based:** Adapt the parameters of the acoustic models to better match the observed data
  - Maximum a posteriori (MAP) adaptation of HMM/GMM parameters
  - Maximum likelihood linear regression (MLLR) of GMM parameters
  - Learning Hidden Unit Contributions (LHUC) for neural networks
- **Speaker normalization:** Normalize the acoustic data to reduce mismatch with the acoustic models
  - Vocal Tract Length Normalization (VTLN)
  - Constrained MLLR (cMLLR) — model-based normalisation
- **Speaker space:** Estimate multiple sets of acoustic models, characterizing new speakers in terms of these model sets
  - Cluster-adaptive training
  - Eigenvoices
  - Speaker codes

# Approaches to adaptation

- **Model based:** Adapt the parameters of the acoustic models to better match the observed data
  - **Maximum a posteriori (MAP) adaptation** of HMM/GMM parameters
  - **Maximum likelihood linear regression (MLLR)** of GMM parameters
  - **Learning Hidden Unit Contributions (LHUC)** for neural networks
- **Speaker normalization:** Normalize the acoustic data to reduce mismatch with the acoustic models
  - Vocal Tract Length Normalization (VTLN)
  - **Constrained MLLR (cMLLR)** — model-based normalisation
- **Speaker space:** Estimate multiple sets of acoustic models, characterizing new speakers in terms of these model sets
  - Cluster-adaptive training
  - Eigenvoices
  - **Speaker codes**

# Model-based adaptation: MAP training

- **Basic idea** MAP training balances the parameters estimated on the SI data with estimates from the new data
- Consider the mean of the  $m$ th Gaussian in the  $j$ th state,  $\mu_{mj}$ 
  - ML estimation of SI model:

$$\mu_{mj} = \frac{\sum_n \gamma_{jm}(n) \mathbf{x}_n}{\sum_n \gamma_{jm}(n)}$$

where  $\gamma_{jm}(n)$  is the component occupation probability

# Model-based adaptation: MAP training

- **Basic idea** MAP training balances the parameters estimated on the SI data with estimates from the new data
- Consider the mean of the  $m$ th Gaussian in the  $j$ th state,  $\mu_{mj}$ 
  - ML estimation of SI model:

$$\mu_{mj} = \frac{\sum_n \gamma_{jm}(n) \mathbf{x}_n}{\sum_n \gamma_{jm}(n)}$$

where  $\gamma_{jm}(n)$  is the component occupation probability

- **MAP estimate** for the adapted model:

$$\hat{\mu} = \frac{\tau \mu_0 + \sum_n \gamma(n) \mathbf{x}_n}{\tau + \sum_n \gamma(n)}$$

- $\tau$  controls the balance between SI estimate and the adaptation date Typically  $0 \leq \tau \leq 20$
- $\mathbf{x}_n$  is the adaptation vector at time  $n$
- $\gamma(n)$  the probability of this Gaussian at this time

# Model-based adaptation: MAP training

- **Basic idea** MAP training balances the parameters estimated on the SI data with estimates from the new data
- Consider the mean of the  $m$ th Gaussian in the  $j$ th state,  $\mu_{mj}$ 
  - ML estimation of SI model:

$$\mu_{mj} = \frac{\sum_n \gamma_{jm}(n) \mathbf{x}_n}{\sum_n \gamma_{jm}(n)}$$

where  $\gamma_{jm}(n)$  is the component occupation probability

- **MAP estimate** for the adapted model:

$$\hat{\mu} = \frac{\tau \mu_0 + \sum_n \gamma(n) \mathbf{x}_n}{\tau + \sum_n \gamma(n)}$$

- $\tau$  controls the balance between SI estimate and the adaptation date Typically  $0 \leq \tau \leq 20$
- $\mathbf{x}_n$  is the adaptation vector at time  $n$
- $\gamma(n)$  the probability of this Gaussian at this time

# Model-based adaptation: MAP training

- **Basic idea** MAP training balances the parameters estimated on the SI data with estimates from the new data
- Consider the mean of the  $m$ th Gaussian in the  $j$ th state,  $\mu_{mj}$ 
  - ML estimation of SI model:

$$\mu_{mj} = \frac{\sum_n \gamma_{jm}(n) \mathbf{x}_n}{\sum_n \gamma_{jm}(n)}$$

where  $\gamma_{jm}(n)$  is the component occupation probability

- **MAP estimate** for the adapted model:

$$\hat{\mu} = \frac{\tau \mu_0 + \sum_n \gamma(n) \mathbf{x}_n}{\tau + \sum_n \gamma(n)}$$

- $\tau$  controls the balance between SI estimate and the adaptation date Typically  $0 \leq \tau \leq 20$
- $\mathbf{x}_n$  is the adaptation vector at time  $n$
- $\gamma(n)$  the probability of this Gaussian at this time
- As the amount of training data increases, so the MAP estimate converges to the ML estimate

# The MLLR family

- **Basic idea** Rather than directly adapting the model parameters, to apply to the Gaussian means and covariances
- MAP training only adapts parameters belonging to observed components – with many Gaussians and a small amount of adaptation data, most Gaussians are not adapted
- Solution: share adaptation parameters across Gaussians – each adaptation data point can then affect many of (or even all) the Gaussians in the system
- Since there are relatively few adaptation parameters, estimation is robust
- **Maximum Likelihood Linear Regression (MLLR)** – use a linear transform to share adaptation parameters across Gaussians

# MLLR: Maximum Likelihood Linear Regression

- MLLR adapts the means of the Gaussians by applying an affine (linear) transform of mean parameters

$$\hat{\mu} = \mathbf{A}\mu + \mathbf{b}$$

If the observation vectors are  $d$ -dimension, then  $\mathbf{A}$  is a  $d \times d$  matrix and  $\mathbf{b}$  is  $d$ -dimension vector

- If we define  $\mathbf{W} = [\mathbf{b} \mathbf{A}]$  and  $\boldsymbol{\eta} = [1 \mu^T]^T$ , then we can write:

$$\hat{\mu} = \mathbf{W}\boldsymbol{\eta}$$

- In MLLR,  $\mathbf{W}$  is estimated so as to maximize the likelihood of the adaptation data
- A single transform  $\mathbf{W}$  can be shared across a set of Gaussian components (even all of them!)

# How many transforms?

- A set of Gaussian components that share a transform is called a *regression class*
- In practice the number of regression is often small: one per context-independent phone class, one per broad class, two (speech/non-speech), or just a single transform for all Gaussians
- The number of regression classes may also be obtained automatically by constructing a *regression class tree*
  - Each node in the tree represents a regression class sharing a transform
  - For an adaptation set, work down the tree until arriving at the most specific set of nodes for which there is sufficient data
  - Regression class tree constructed in a similar way to state clustering tree

## Estimating the transforms

- The linear transformation matrix  $W$  is obtained by finding its setting which optimizes the log likelihood
- **Mean adaptation:** Log likelihood

$$L = \sum_r \sum_n \gamma_r(n) \log \left( K_r \exp \left( -\frac{1}{2} (\mathbf{x}_n - \mathbf{W}\boldsymbol{\eta}_r)^T \boldsymbol{\Sigma}_r^{-1} (\mathbf{x}_n - \mathbf{W}\boldsymbol{\eta}_r) \right) \right)$$

where  $r$  ranges over the components belonging to the regression class

- Differentiating  $L$  and setting to 0 results in an equation for  $\mathbf{W}$ : there is no closed form solution if  $\boldsymbol{\Sigma}$  is full covariance; can be solved if  $\boldsymbol{\Sigma}$  is diagonal (but requires a matrix inversion)
- Variance adaptation is also possible
- See Gales and Woodland (1996), Gales (1998) for details

## MLLR in practice

- Mean-only MLLR results in 10–15% relative reduction in WER
- Few regression classes and well-estimated transforms work best in practice
- Robust adaptation available with about 1 minute of speech; performance similar to SD models available with 30 minutes of adaptation data
- Such linear transforms can account for any systematic (linear) variation from the speaker independent models, for example those caused by channel effects.

# Constrained MLLR (cMLLR)

- Basic idea use the same linear transform for both mean and covariance

$$\hat{\mu} = \mathbf{A}'\boldsymbol{\mu} - \mathbf{b}'$$
$$\hat{\Sigma} = \mathbf{A}'\boldsymbol{\Sigma}\mathbf{A}'^T$$

- No closed form solution but can be solved iteratively
- Log likelihood for cMLLR

$$L = \mathcal{N}(\mathbf{Ax}_n + \mathbf{b}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) + \log(|\mathbf{A}|) \quad \mathbf{A}' = \mathbf{A}^{-1}; \mathbf{b}' = \mathbf{Ab}$$

**Equivalent to applying the linear transform to the data!**

Also called *fMLLR* (*feature space MLLR*)

- Iterative solution amenable to online/dynamic adaptation, by using just one iteration for each increment
- Similar improvement in accuracy to standard MLLR

# Speaker-adaptive training (SAT)

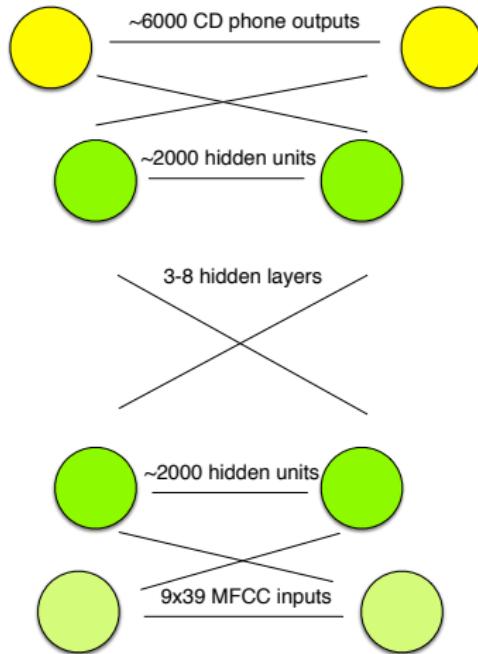
- **Basic idea** Rather than SI seed (canonical) models, construct models designed for adaptation
- Estimate parameters of canonical models by training MLLR mean transforms for each training speaker
- Train using the MLLR transform for each speaker; interleave Gaussian parameter estimation and MLLR transform estimation
- SAT results in much higher training likelihoods, and improved recognition results
- But: increased training complexity and storage requirements
- SAT using cMLLR, corresponds to a type of speaker normalization at training time

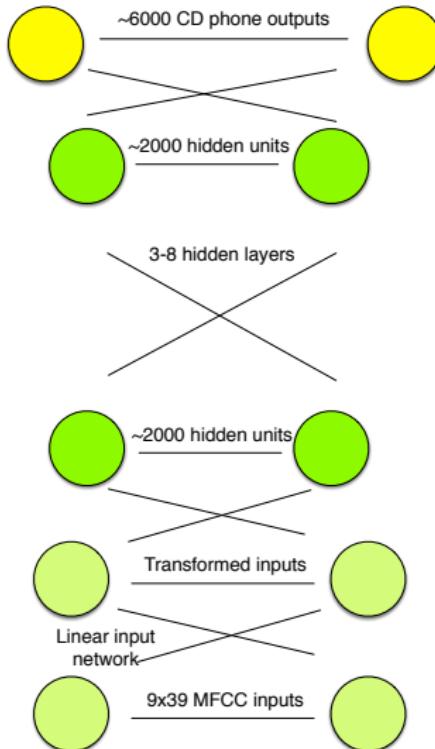
# Speaker adaptation in hybrid HMM/NN systems: CMLLR feature transformation

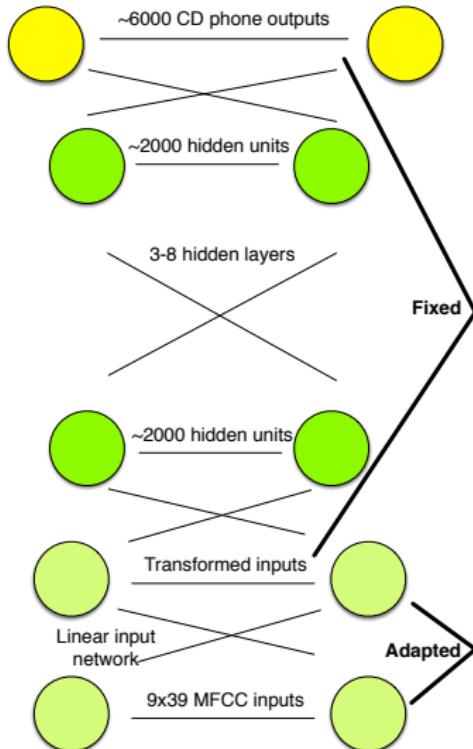
- **Basic idea:** If HMM/GMM system is used to estimate a single constrained MLLR adaptation transform, this can be viewed as a feature space transform
- Use the HMM/GMM system with the same tied state space to estimate a single CMLLR transform for a given speaker, and use this to transform the input speech to the DNN for the target speaker
- Can operate unsupervised (since the GMM system estimates the transform)
- Limited to a single transform (regression class)

# Speaker adaptation in hybrid HMM/NN systems: LIN – Linear Input Network

- **Basic idea:** single linear input layer trained to map input speaker-dependent speech to speaker-independent network
- Training: linear input network (LIN) can either be fixed as the identity or (adaptive training) be trained along with the other parameters
- Testing: freeze the main (speaker-independent) network and propagate gradients for speech from the target speaker to the LIN, which is updated — linear transform learned for each speaker
- Requires supervised training data

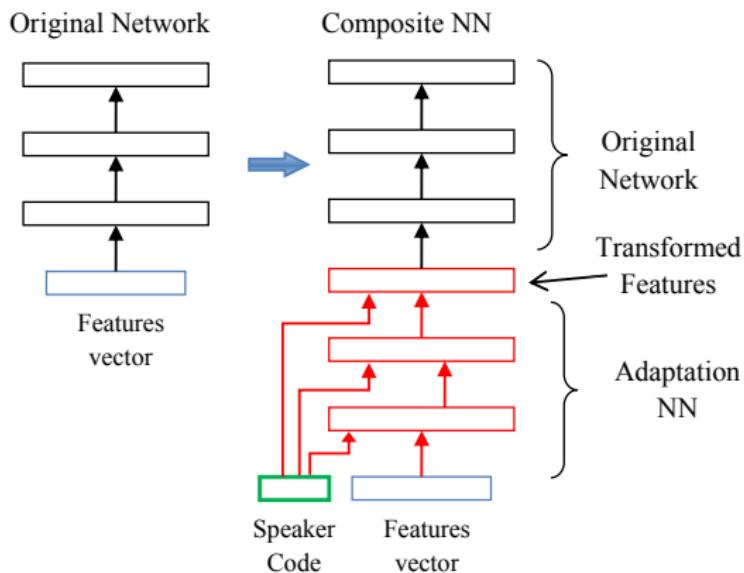






# Speaker adaptation in hybrid HMM/NN systems: Speaker codes

- **Basic idea:** Learn a short speaker code vector for each talker



# Speaker adaptation in hybrid HMM/NN systems: i-vectors

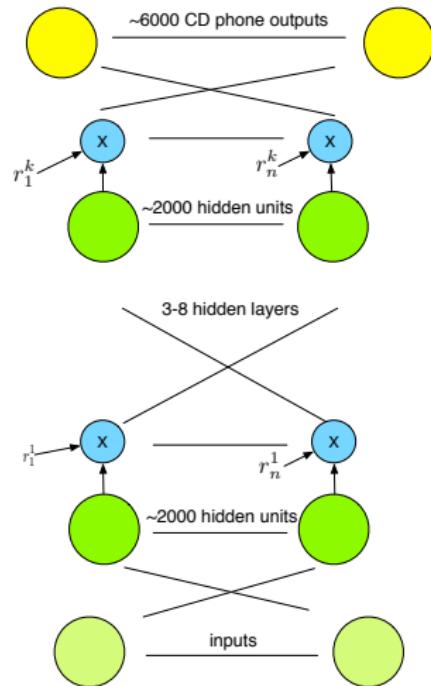
- **Basic idea:** Use i-vectors (speaker identity vectors) as speaker code
- i-vectors are fixed-dimensional representations  $\lambda_s$ , representing speaker  $s$ 
  - a GMM is trained on all the data
  - i-vector is used (along with a matrix  $M$  sometimes called the “total variability matrix”) to model the difference between the means trained on all data  $\mu_0$  and the speaker specific mean  $\mu_s$

$$\mu_s = \mu_0 + M\lambda_s$$

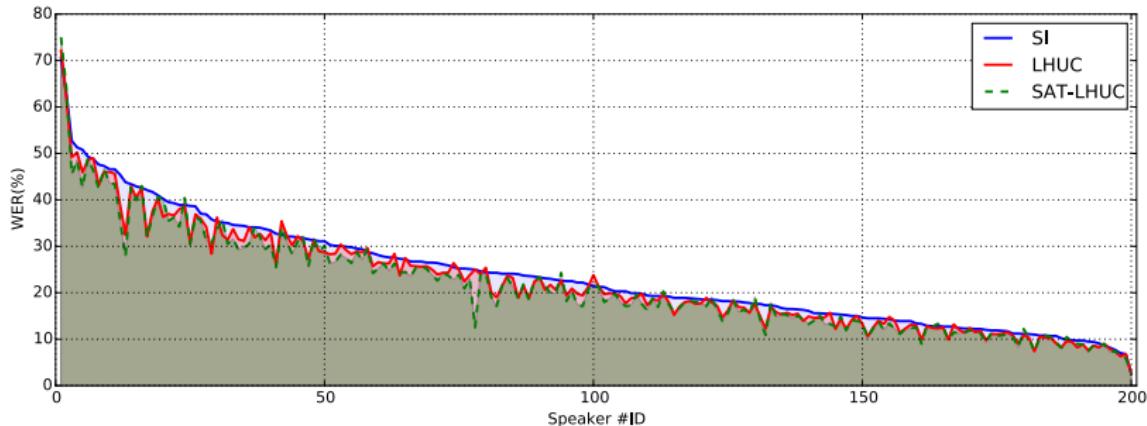
- use an EM style algorithm in which  $\lambda_s$  and  $M$  are alternatively estimated
- this is a factor analysis approach which was developed for speaker identification
- i-vectors typically 40-100 dimensions

# Speaker adaptation in hybrid HMM/NN systems: LHUC – Learning Hidden Unit Contributions

- **Basic idea:** Add a learnable speaker dependent amplitude to each hidden unit
- Speaker independent: amplitudes set to 1
- Speaker dependent: learn amplitudes from data, per speaker

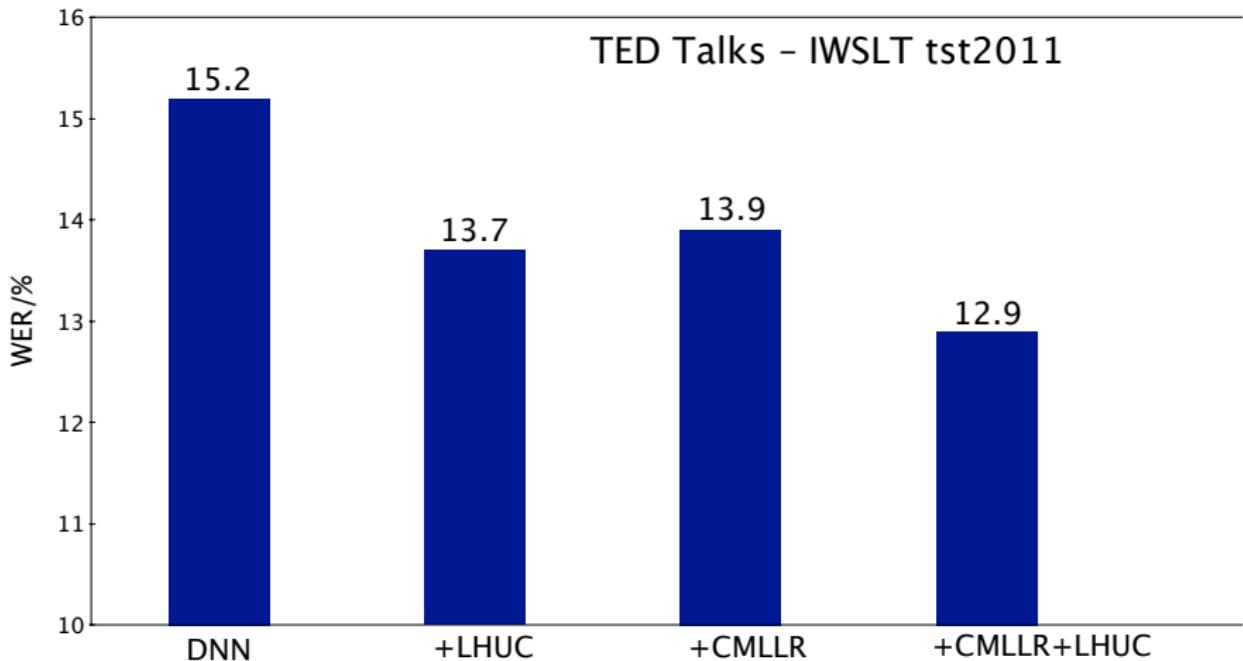


# LHUC adaptation by speaker



Results on speakers across AMI, TED, Switchboard corpora

# Speaker adaptation in hybrid HMM/NN systems: Experimental Results on TED



## Speaker Adaptation

- One of the most intensive areas of speech recognition research since the early 1990s
- HMM/GMM
  - Substantial progress, resulting in significant, additive, consistent reductions in word error rate
  - Close mathematical links between different approaches
  - Linear transforms at the heart of many approaches
  - MLLR family is very effective
- HMM/NN
  - Open research topic
  - GMM-based feature space transforms (cMLLR) and LHUC are effective (and complementary)

# Reading

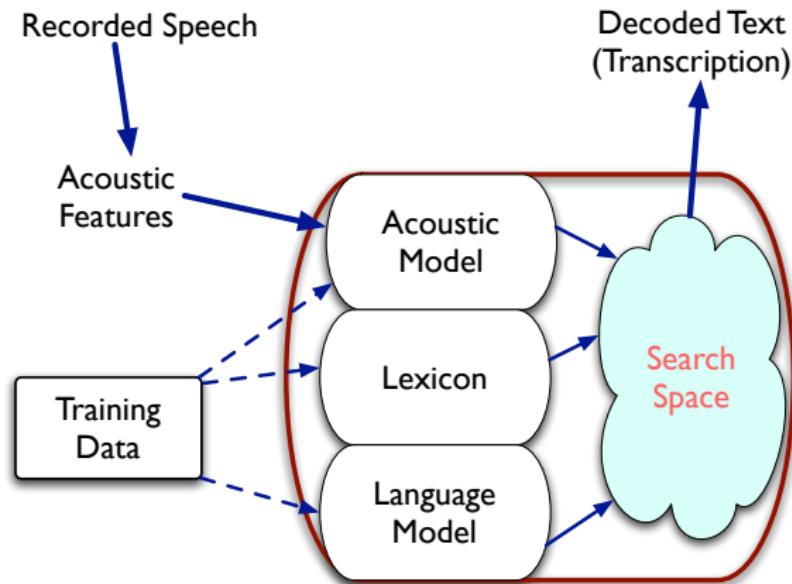
- Gales and Young (2007), "The Application of Hidden Markov Models in Speech Recognition", *Foundations and Trends in Signal Processing*, 1 (3), 195–304: section 5.  
<http://mi.eng.cam.ac.uk/~sjy/papers/gayo07.pdf>
- Woodland (2001), "Speaker adaptation for continuous density HMMs: A review", ISCA ITRW on Adaptation Methods for Speech Recognition.  
[http://www.isca-speech.org/archive\\_open/archive\\_papers/adaptation/adap\\_011.pdf](http://www.isca-speech.org/archive_open/archive_papers/adaptation/adap_011.pdf)
- Abdel-Hamid and Jiang (2013), "Fast speaker adaptation of hybrid NN/HMM model for speech recognition based on discriminative learning of speaker code", ICASSP-2013.  
<https://dx.doi.org/10.1109/ICASSP.2013.6639211>
- Saon et al (2013), "Speaker adaptation of neural network acoustic models using i-vectors", ASRU-2013.  
<https://doi.org/10.1109/ASRU.2013.6707705>
- Swietojanski and Renals (2014), "Learning Hidden Unit Contributions for Unsupervised Acoustic Model Adaptation", IEEE Trans. ASLP, 24:1450–1463. [http://www.research.ed.ac.uk/portal/files/25128271/lhuc\\_final\\_taslp16.pdf](http://www.research.ed.ac.uk/portal/files/25128271/lhuc_final_taslp16.pdf)

# Decoding, Alignment, and WFSTs

Steve Renals

Automatic Speech Recognition – ASR Lecture 12  
26 March 2018

# HMM Speech Recognition



# The Search Problem in ASR

- Find the most probable word sequence  $\hat{W} = w_1, w_2, \dots, w_M$  given the acoustic observations  $\mathbf{X} = \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ :

$$\begin{aligned}\hat{W} &= \arg \max_W P(W|\mathbf{X}) \\ &= \arg \max_W \underbrace{p(\mathbf{X} | W)}_{\text{acoustic model}} \underbrace{P(W)}_{\text{language model}}\end{aligned}$$

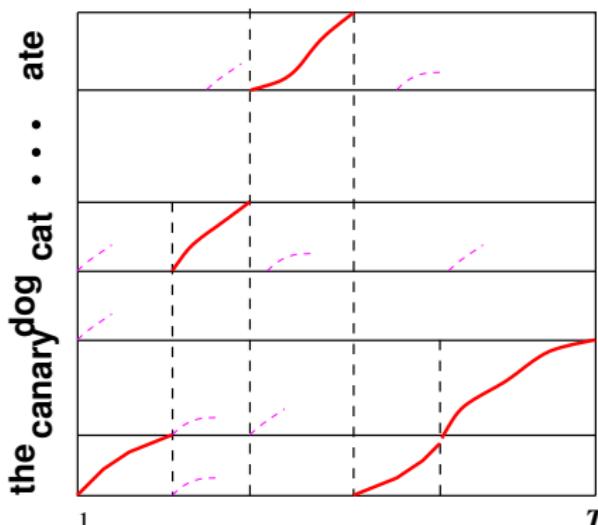
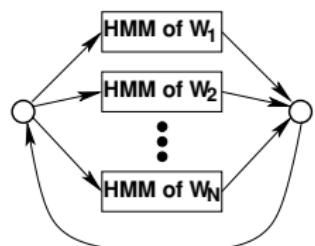
- Words are composed of state sequences so this problem corresponds to finding the most probable allowable state sequence (given the constraints of pronunciation lexicon and language model) - **Viterbi decoding**
- In a large vocabulary task evaluating all possible word sequences is infeasible (even using an efficient exact algorithm)
  - Reduce the size of the search space through pruning unlikely hypotheses
  - Eliminate repeated computations

# Connected Word Recognition

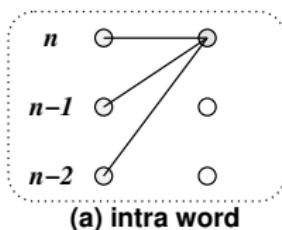
- The number of words in the utterance is not known
- Word boundaries are not known:  $V$  words may potentially start at each frame

# Connected Word Recognition

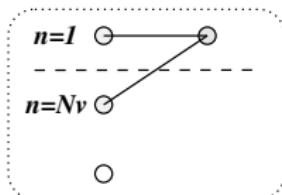
- The number of words in the utterance is not known
- Word boundaries are not known:  $V$  words may potentially start at each frame



speech: "the cat ate the canary"

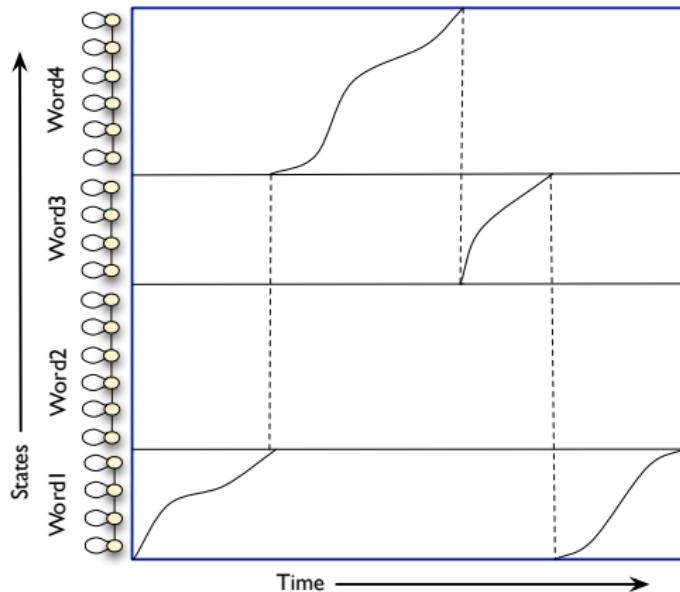


(a) intra word

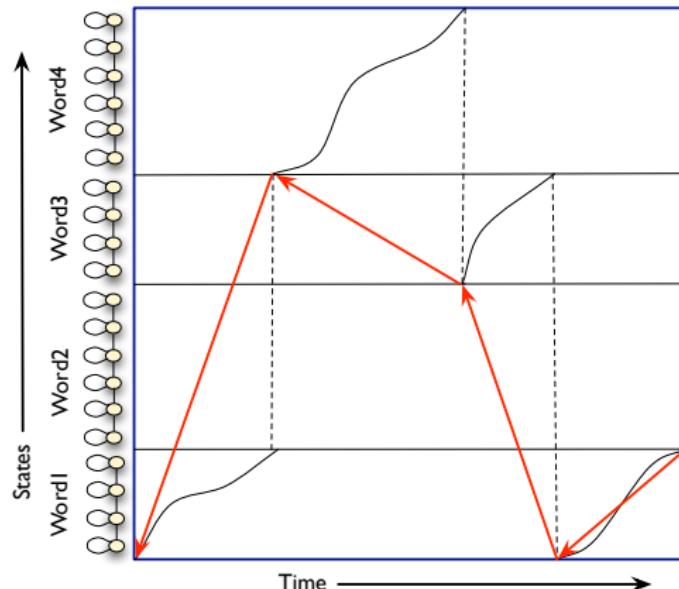


(b) intra/inter word

# Time Alignment Path

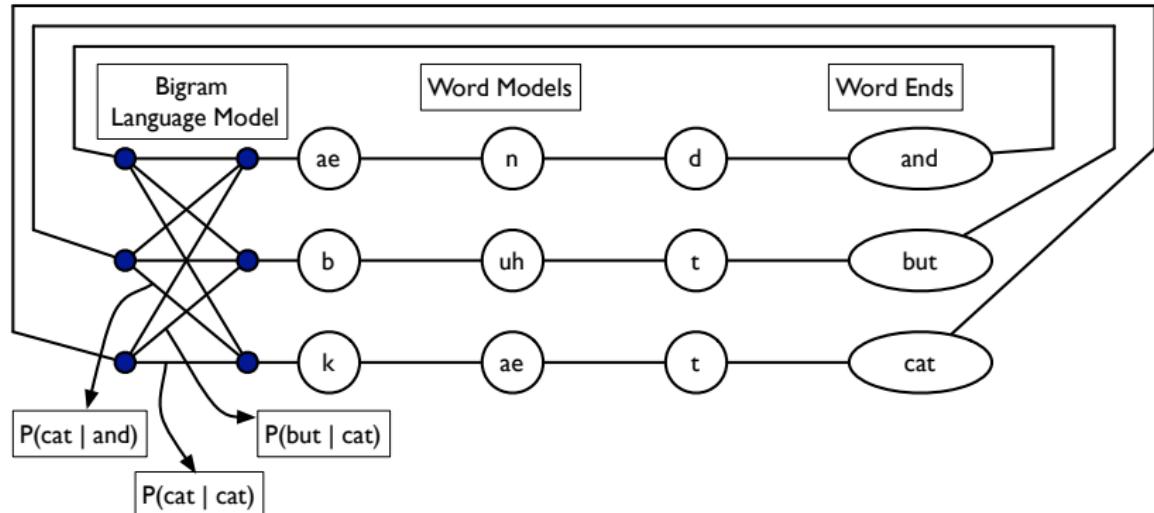


# Backtrace to Obtain Word Sequence



- Backpointer array keeps track of word sequence for a path:  
 $\text{backpointer[word][wordStartFrame]} = (\text{prevWord}, \text{prevWordStartFrame})$
- Backtrace through backpointer array to obtain the word sequence for a path

# Incorporating a bigram language model



Trigram or longer span models require a word history.

# Computational Issues

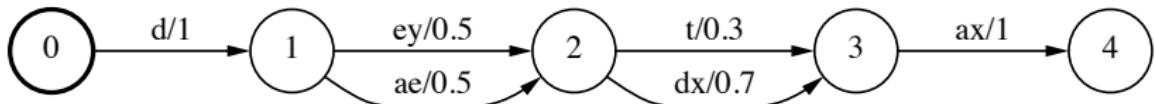
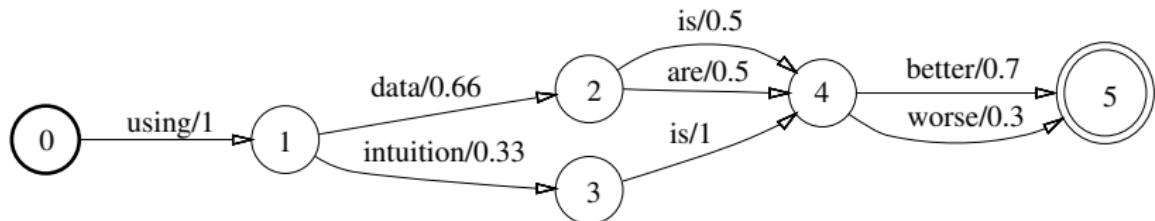
- Viterbi decoding performs an exact search in an efficient manner
- Exact search is not possible for large vocabulary tasks
  - Cross-word triphones need to be handled carefully since the acoustic score of a word-final phone depends on the initial phone of the next word
  - Long-span language models (eg trigrams) greatly increase the size of the search space
- Solutions:
  - Beam search (prune low probability hypotheses)
  - Dynamic search structures
  - Multipass search ( $\rightarrow$  two-stage decoding)
  - Best-first search ( $\rightarrow$  stack decoding / A\* search)

- Viterbi decoding performs an exact search in an efficient manner
- Exact search is not possible for large vocabulary tasks
  - Cross-word triphones need to be handled carefully since the acoustic score of a word-final phone depends on the initial phone of the next word
  - Long-span language models (eg trigrams) greatly increase the size of the search space
- Solutions:
  - Beam search (prune low probability hypotheses)
  - Dynamic search structures
  - Multipass search ( $\rightarrow$  two-stage decoding)
  - Best-first search ( $\rightarrow$  stack decoding / A\* search)
- An alternative approach: Weighted Finite State Transducers (WFST)

# Weighted Finite State Transducers

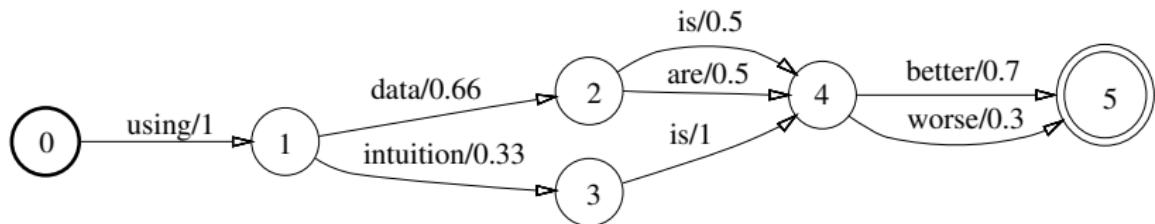
- Used by Kaldi
- Weighted finite state automaton that transduces an input sequence to an output sequence (Mohri et al 2008)
- States connected by transitions. Each transition has
  - input label
  - output label
  - weight

# Weighted Finite State Acceptors

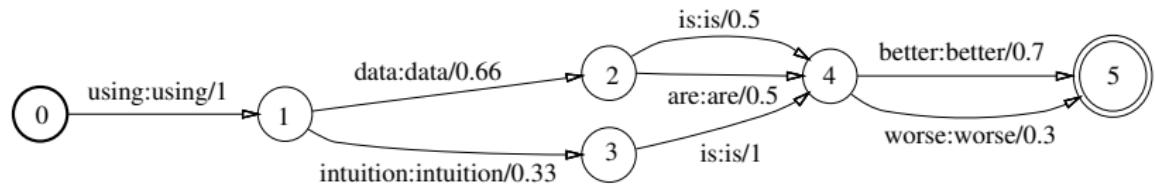


# Weighted Finite State Transducers

Acceptor

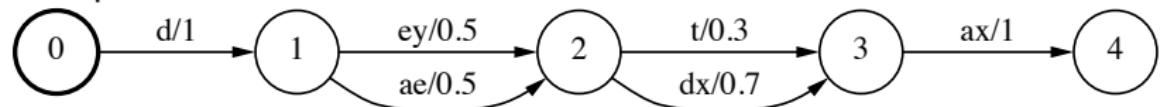


Transducer

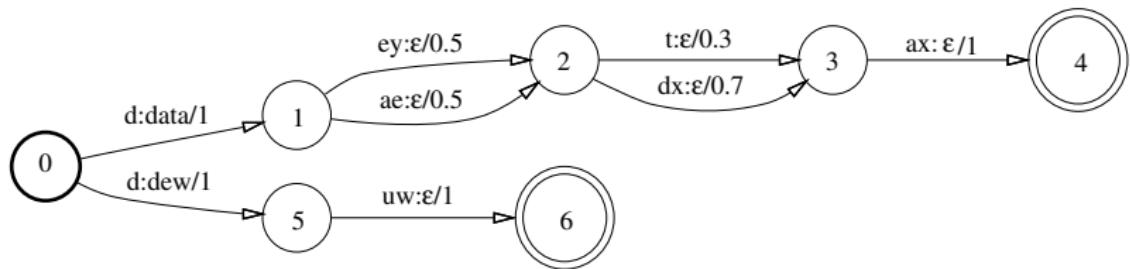


# Weighted Finite State Transducers

Acceptor



Transducer



# WFST Algorithms

**Composition** Combine transducers at different levels. For example if  $G$  is a finite state grammar and  $L$  is a pronunciation dictionary then  $L \circ G$  transduces a phone string to word strings allowed by the grammar

**Determinisation** Ensure that each state has no more than a single output transition for a given input label

**Minimisation** transforms a transducer to an equivalent transducer with the fewest possible states and transitions

# Applying WFSTs to speech recognition

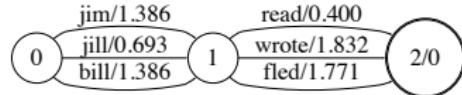
- Represent the following components as WFSTs

	transducer	input sequence	output sequence
G	word-level grammar	words	words
L	pronunciation lexicon	phones	words
C	context-dependency	CD phones	phones
H	HMM	HMM states	CD phones

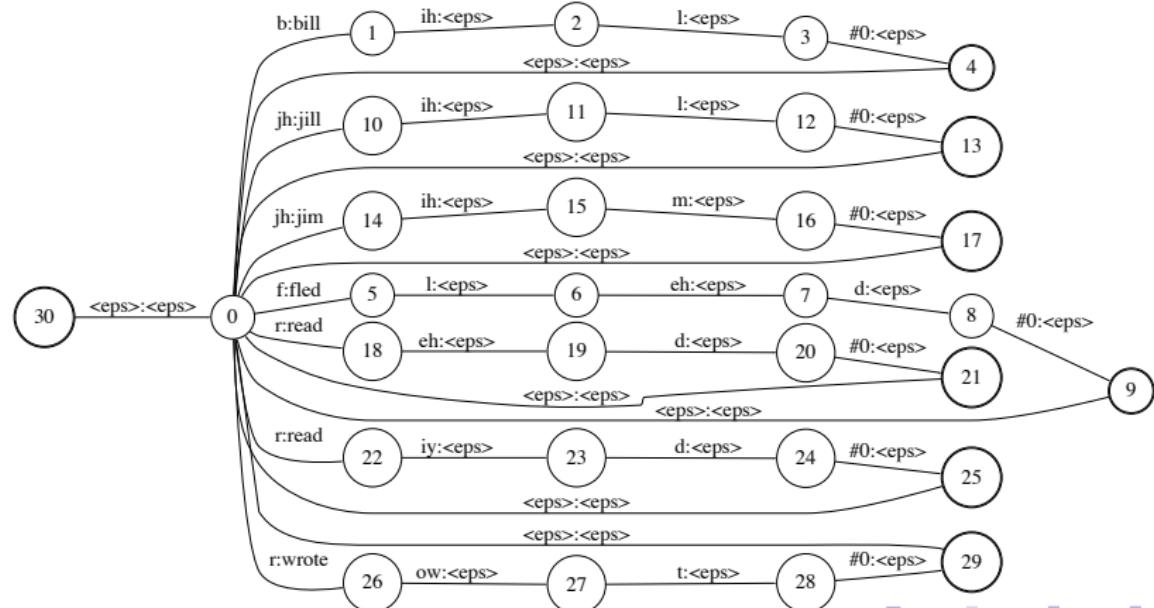
- Composing  $L$  and  $G$  results in a transducer  $L \circ G$  that maps a phone sequence to a word sequence
- $H \circ C \circ L \circ G$  results in a transducer that maps from HMM states to a word sequence

$L$ ,  $G$

$G$



$L$

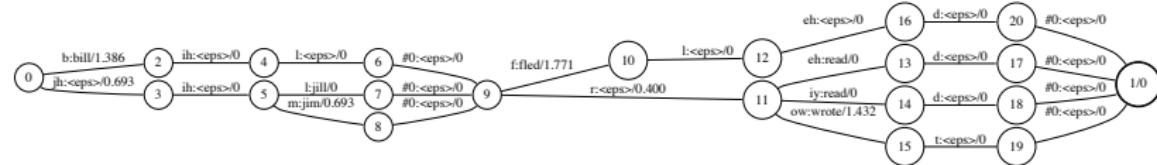


$$L \circ G, \det(L \circ G), \min(\det(L \circ G))$$

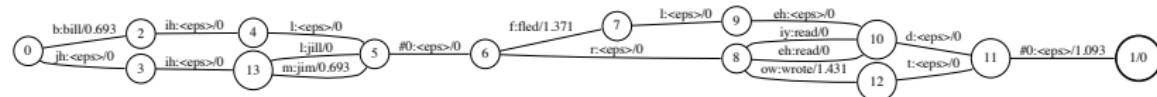
$$L \circ G$$



$$\det(L \circ G)$$

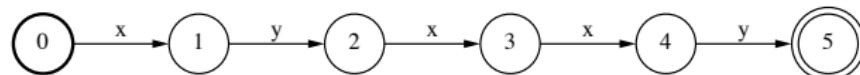


$$\min(\det(L \circ G))$$

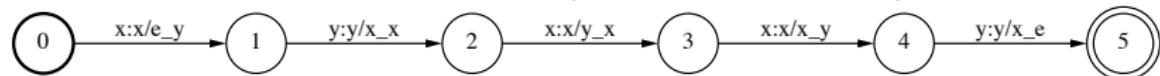


# Context dependency transducer $C$

Context-independent “string”



Context-dependency transducer (weights not shown)



( $x/e_y - x$  with left context  $e$  (start/end) and right context  $y$ )

# Decoding using WFSTs

- We can represent the HMM acoustic model, pronunciation lexicon and n-gram language model as four transducers:  $H$ ,  $C$ ,  $L$ ,  $G$
- Combining the transducers gives an overall “decoding graph” for our ASR system – but minimisation and determination means it is much smaller than naively combining the transducers
- But it is important in which order the algorithms are combined otherwise the transducers may “blow-up” – basically after each composition, first determinise then minimise
- In Kaldi, ignoring one or two details

$$HCLG = \min(\det(H \circ \min(\det(C \circ \min(\det(L \circ G)))))))$$

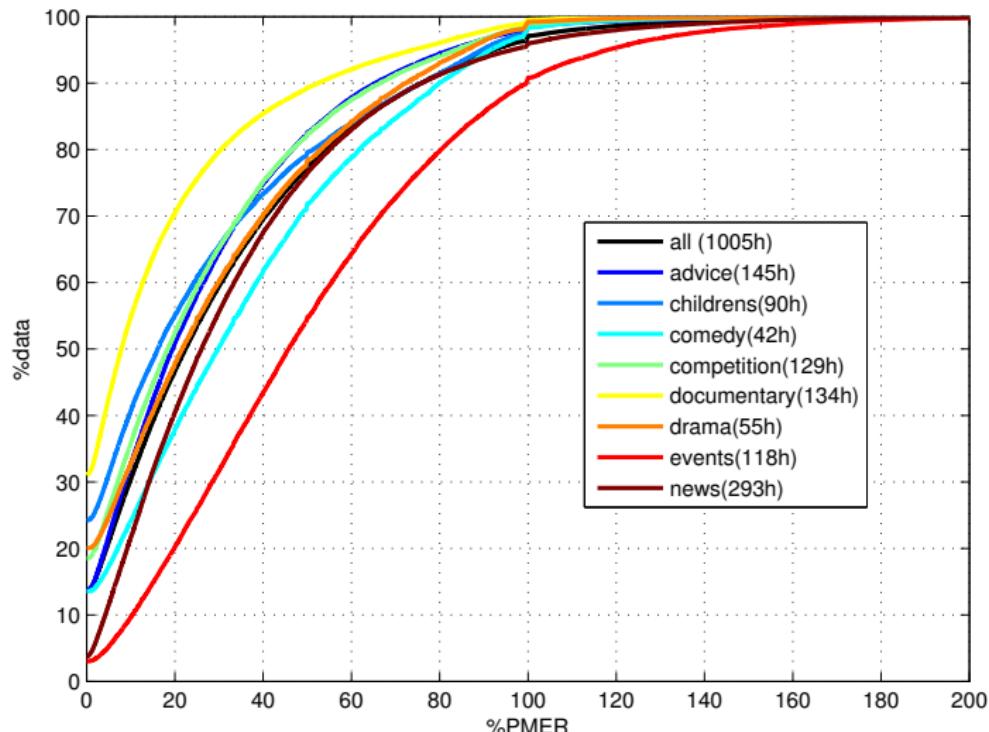
# Alignment

- Alignment is the task of matching a recording to a transcript
- In many circumstances the available transcript differs from a verbatim transcript: for example, captions/subtitles for a TV programme may not include every word spoken, or may include paraphrasing
- Performing alignment using such transcripts is of great practical use
  - time-aligning subtitles to the broadcast
  - using the data for speech recognition training (*lightly supervised training*)
- In lightly supervised training we need to use the alignment to identify reliable labels and learn from them – without also learning from unreliable labels, or past mistakes

# Alignment using a biased language model

- Basic idea - transcribe the recording using a language model biased towards the transcript
  - ➊ Train a *biased* language model on the supplied transcript, interpolated (smoothed) with a background LM
$$p(w_t|h_t) = \lambda p_{bias}(w_t|h_t) + (1 - \lambda)p_{bg}(w_t|h_t)$$
  - ➋ Decode the training data with a pre-existing acoustic model, and the biased LM
  - ➌ Align the captions with the ASR output
- For lightly supervised training – select utterances where there is a good match between the captions and the automatic output

# Data selection from subtitled TV recordings



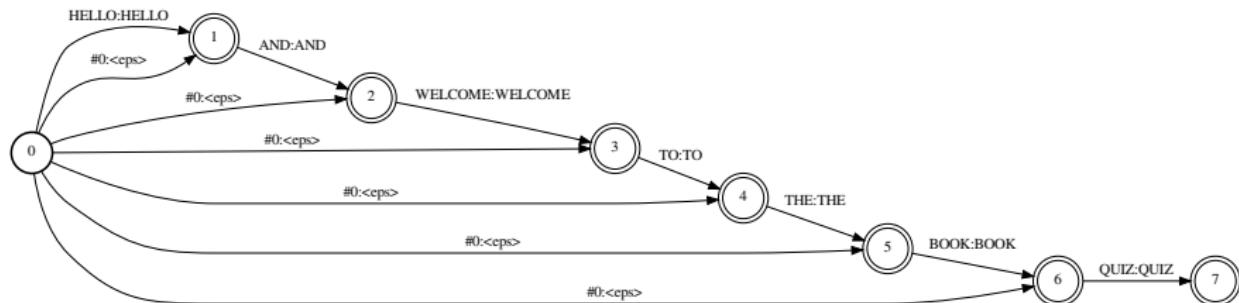
MGB Challenge 1 training data (BBC TV)

## An alternative alignment method

- The biased LM approach is quite computationally costly; it can also lead to bias towards data that we can already recognise well
- We have used an alternative approach based on constructing weighted finite state transducers for each utterance
- This allows us to use much stronger constraints – based on the captions – at decoding time

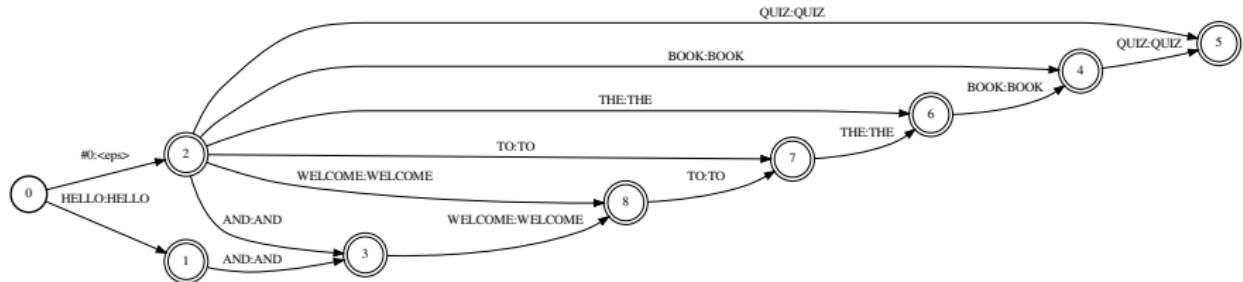
# Alignment with WFSTs

A  $G$  transducer that allows any substring of the original captions – known as a *factor transducer*



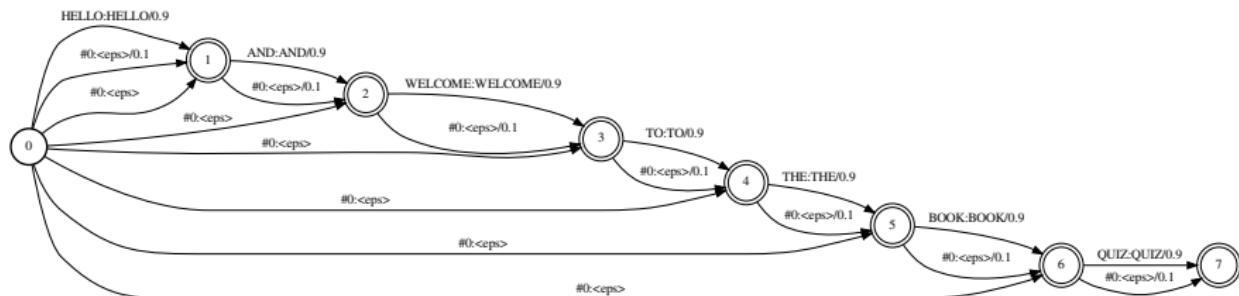
# Alignment with WFSTs

A determinized version of the  $G$  transducer



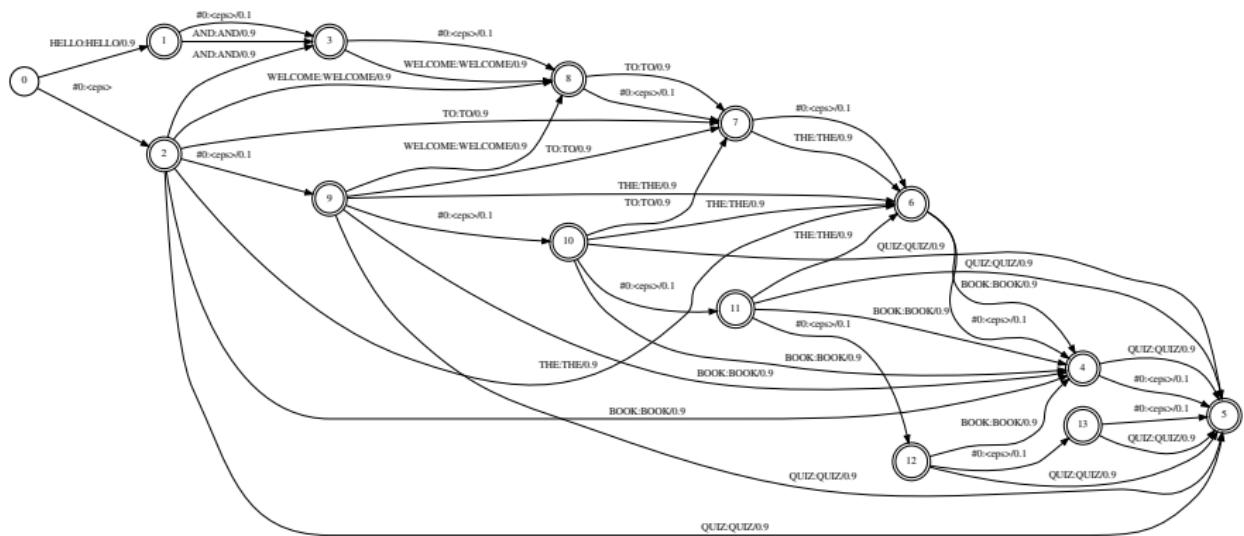
# Alignment with WFSTs

What about when word appears in the captions that was not actually spoken? We need to alter the design to be robust to this by allowing deletions (at a cost)



# Alignment with WFSTs

A determinized version



# The complete alignment process

- ① Decode with a factor-transducer for each programme
- ② Align the output to the original captions
- ③ Re-segment the data, to potentially include missed speech
- ④ Decode again with utterance-specific factor transducers, allowing word-skips

# Evaluating alignment

- To evaluate we need gold-standard (verbatim) transcriptions as well as the captions to be aligned
- Evaluate the alignment of the captions with respect to a forced alignment of the gold-standard verbatim transcription
- Words spoken but not in the captions are ignored
- For words in both, systems judged correct if supplied timings are correct within a 100ms window
- Evaluated in terms of f-score

$$P = \frac{N_{match}}{N_{hyp}}, R = \frac{N_{match}}{N_{ref}}, F = 2 \times \frac{P \times R}{P + R}$$

# Alignment results on MGB

System	Precision	Recall	F-score
Preliminary DNN AMs			
Pass 1 FT	0.8816	0.7629	0.8180
+ force align	0.8290	0.7855	0.8066
Pass 2 FT+skip	0.8679	0.8563	0.8620
Final DNN AMs			
Pass 1	0.9009	0.8128	0.8546
Pass 2 FT+skip	0.8856	0.9013	0.8934

# Summary

- Search (decoding) in ASR involves finding the correct word sequence given a sample recording
- Weighted finite state transducer (WFST) framework – provides a well-justified way to combine models at different levels
- WFST algorithms - composition, determinisation, minimisation
- Kaldi represents a speech recogniser as an HCLG transducer – combining 4 transducers to map from HMM states to word sequences
- WFSTs provide a way to represent various problems in speech recognition, eg alignment

# Reading

- Mohri et al (2008). “Speech recognition with weighted finite-state transducers.” In Springer Handbook of Speech Processing, pp. 559-584. Springer.  
<http://www.cs.nyu.edu/~mohri/pub/hbka.pdf>
- WFSTs in Kaldi. <http://danielpovey.com/files/Lecture4.pdf>
- Bell and Renals (2015), “A system for automatic alignment of broadcast media captions using weighted finite-state transducers,” ASRU. <https://doi.org/10.1109/ASRU.2015.7404861>
- Moreno and Alberti (2009), “A factor automaton approach for the forced alignment of long speech recordings,” ICASSP.  
<https://doi.org/10.1109/ICASSP.2009.4960722>
- Braunschweiler et al (2010), “Lightly supervised recognition for automatic alignment of large coherent speech recordings,” Interspeech. [http://www.isca-speech.org/archive/interspeech\\_2010/i10\\_2222.html](http://www.isca-speech.org/archive/interspeech_2010/i10_2222.html)

# Multilingual Speech Recognition

Steve Renals

Automatic Speech Recognition – ASR Lecture 13  
29 March 2018

# Languages of the World

- Over 6,000 languages globally....
- In Europe alone
  - 24 official languages and 5 “semi-official” languages
  - Over 100 further regional/minority languages
  - If we rank the 50 most used languages in Europe, then there are over 50 million speakers of languages 26-50 (Finnish – Montenegrin)
- 3,000 of the world’s languages are endangered
- Google cloud speech API covers about 70 languages\* and a further 50 accents/dialects of those languages; Apple Siri covers 21 languages and a further 21 accents/dialects

(\*: it was 45 this time last year!)

# Under-resourced languages

Under-resourced (or low-resourced) languages have some or all of the following characteristics

- limited web presence
- lack of linguistic expertise
- lack of digital resources: acoustic and text corpora, pronunciation lexica, ...

Under-resourced languages thus provide a challenge for speech technology

See Besacier et al (2014) for more

# Speech recognition of under-resourced languages

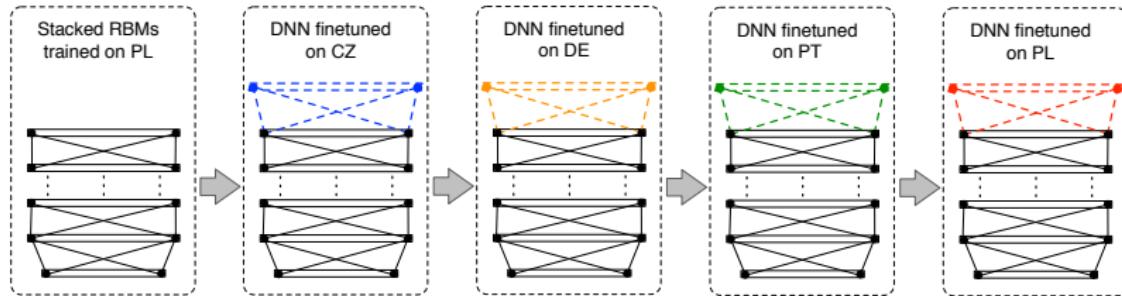
- Training acoustic and language models with limited training data
- Transferring knowledge between languages
- Constructing pronunciation lexica
- Dealing with language specific characteristics (e.g. morphology)

# Multilingual and cross-lingual acoustic models

How to share information from acoustic models in different languages?

- General principal – use neural network hidden layers to learn a **multilingual representation**
- Hidden layers are multilingual – shared between languages
- Output layer is monolingual language specific
- **Hat swap** use a network with multilingual hidden representations directly in a hybrid DNN/HMM systems
- **Block softmax** train a network with an output layer for each language, but shared hidden layers
- **Multilingual bottleneck** use a bottleneck hidden layer (trained in a multilingual) way as features for either a GMM- or NN-based system

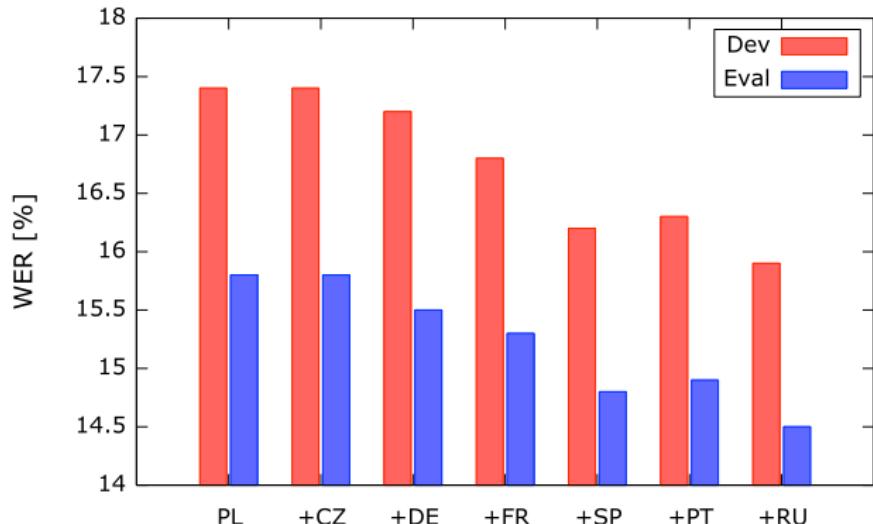
# Hat Swap – architecture



Ghoshal et al, 2013

# Hat Swap – experiment

## Recognition of GlobalPhone Polish

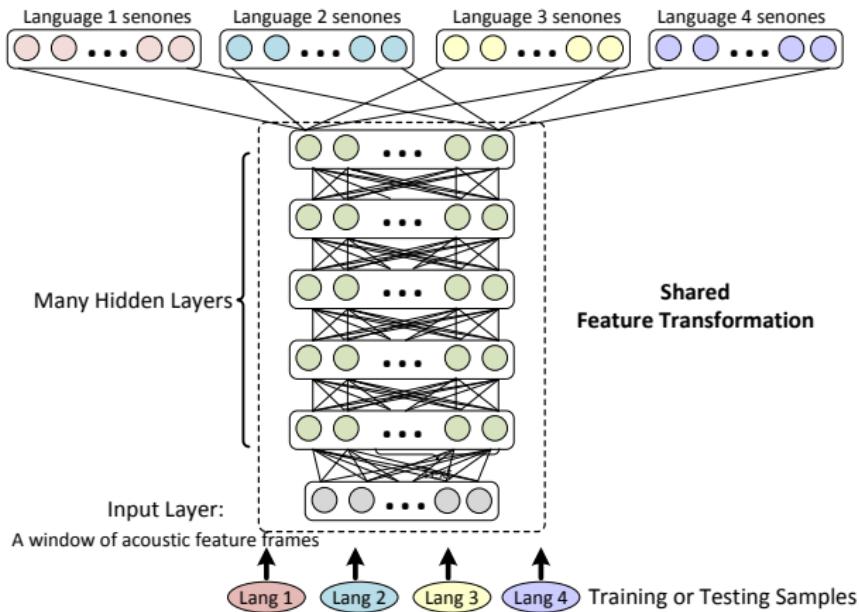


Ghoshal et al, 2013

# Block softmax

- In block softmax we train one network for all languages:
  - separate output layer for each language
  - shared hidden layers
- Each training input is propagated forward to the output layer of the corresponding language – only that output layer is used to compute the error used to train the network for that input
- Since the hidden layers are shared, they must learn features relevant to all the output layers (languages)
- Can view block softmax as a parallel version of hat swap

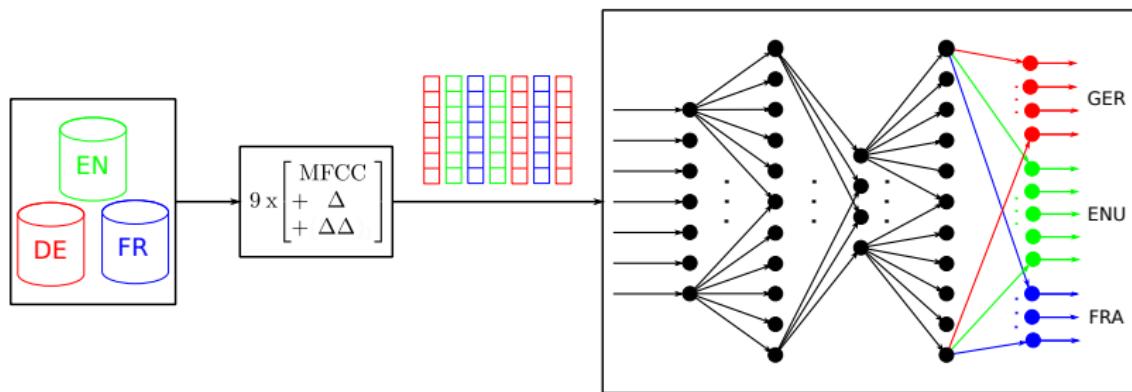
# Block softmax – architecture



Huang et al, 2013

NB: A senone is a context-dependent tied state

# Multilingual bottleneck features – architecture



Tüske et al, 2013

# Multilingual bottleneck features – experiments

GMM-based acoustic models. (Similar results obtained using multilingual bottleneck features with NN-based acoustic models.)

Test language	WER [%]	MFCC	MFCC+BN		
			Bottleneck trained on		
			GER	ENU	FRA
GER	29.97		27.50 (8.2)	29.63 (1.1)	30.38 (-1.4)
ENU	21.69		21.31 (1.8)	18.85 (13.1)	22.63 (-4.3)
FRA	37.78		37.76 (0.1)	38.72 (-2.5)	33.95 (10.1)

Test language	WER [%]	MFCC	MFCC+BN			
			BN trained on			
			GER	ENU	GER + FRA	GER + ENU + FRA
GER	34.58		33.39 (3.4)	34.07 (1.5)	32.74 (5.3)	31.72 (8.3)
ENU	26.14		23.54 (9.9)	24.81 (5.1)	23.68 (9.4)	21.79 (16.6)
FRA	43.52		40.51 (6.9)	43.65 (-0.3)	41.96 (3.6)	39.98 (8.1)

(Mismatched acoustic environment)

Tüske et al, 2013

# Graphemes and phonemes

- Can represent pronunciations as a sequence of graphemes (letters) rather than a sequence of phones
- Advantages of grapheme-based pronunciations
  - No need to construct/generate phone-based pronunciations
  - Can use unicode attributes to assist in decision tree construction
- Disadvantages: not always direct link between graphemes and sounds (most of in English)

# Grapheme-based ASR results for 6 low-resource languages

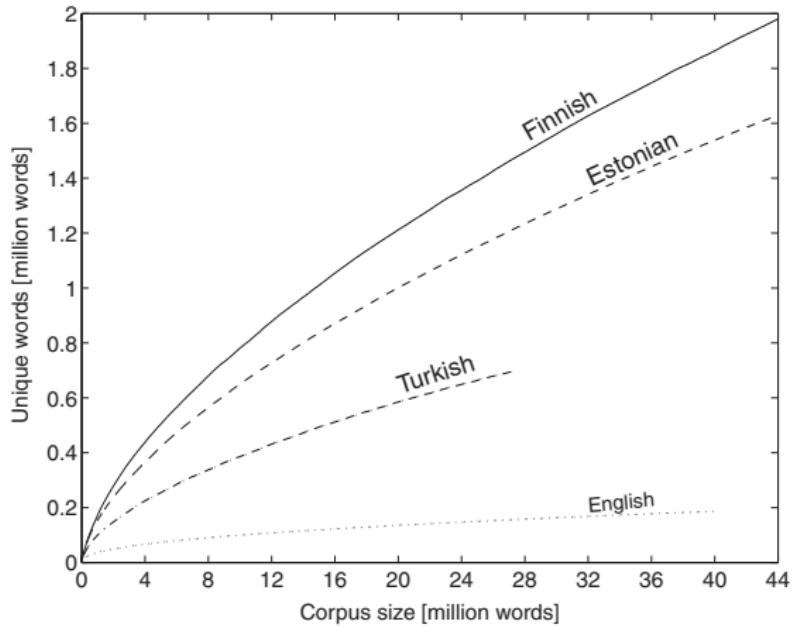
Language	ID	System	WER (%)		
			tg	+cn	cnc
Kurmanji Kurdish	205	Phonetic	67.6	65.8	64.1
		Graphemic	67.0	65.3	
Tok Pisin	207	Phonetic	41.8	40.6	39.4
		Graphemic	42.1	41.1	
Cebuano	301	Phonetic	55.5	54.0	52.6
		Graphemic	55.5	54.2	
Kazakh	302	Phonetic	54.9	53.5	51.5
		Graphemic	54.0	52.7	
Telugu	303	Phonetic	70.6	69.1	67.5
		Graphemic	70.9	69.5	
Lithuanian	304	Phonetic	51.5	50.2	48.3
		Graphemic	50.9	49.5	

IARPA Babel, 40h acoustic training data per language,  
monolingual training; cnc is confusion network combination,  
combining the grapheme- and phone-based systems  
Gales et al (2015)

# Morphology

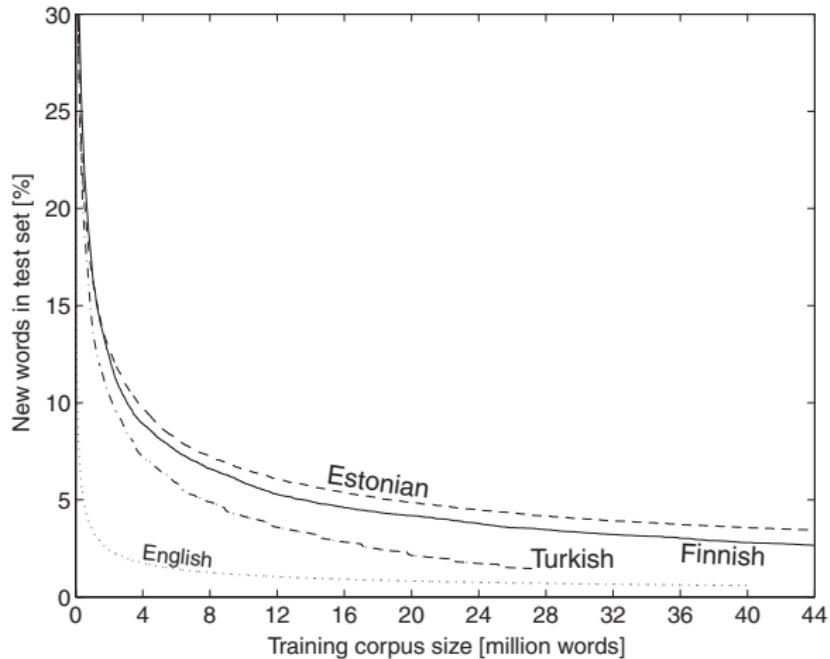
- Many languages are morphologically richer than English: this has a major effect of vocabulary construction and language modelling
- **Compounding** (eg German): decompose compound words into constituent parts, and carry out pronunciation and language modelling on the decomposed parts
- **Highly inflected languages** (eg Arabic, Slavic languages): specific components for modelling inflection (eg factored language models)
- **Inflecting and compounding languages** (eg Finnish)
- All approaches aim to reduce ASR errors by reducing the OOV rate through modelling at the morph level; also addresses data sparsity

# Vocabulary size for different languages



Creutz et al (2007)

# OOV Rate for different languages



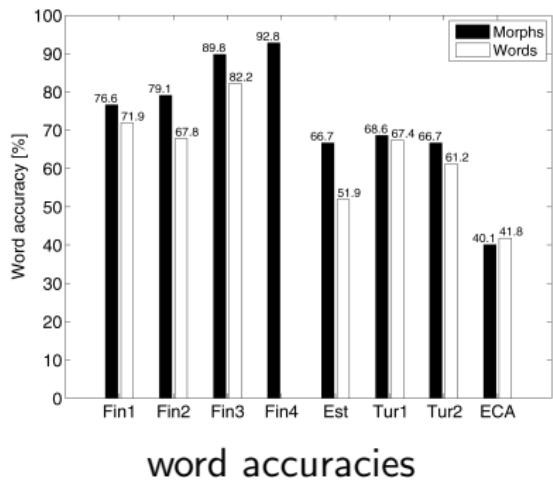
Creutz et al (2007)

# Segmenting into morphs

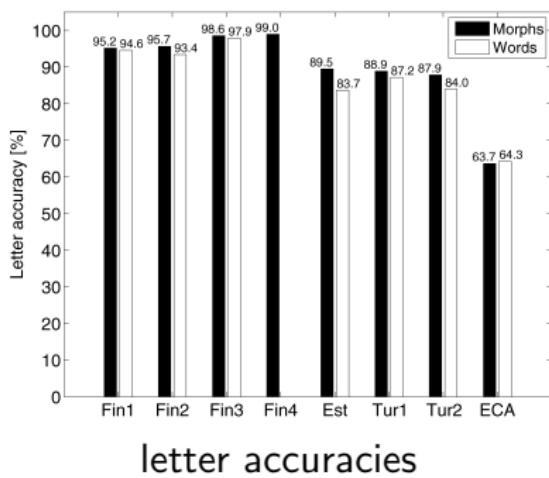
- Linguistic rule-based approaches – require a lot of work for an under-resourced language!
- Automatic approaches – use automatically segment and cluster words into their constituent morphs
- Morfessor (<http://www.cis.hut.fi/projects/morpho/>)
  - “Morfessor is an unsupervised data-driven method for the segmentation of words into morpheme-like units.”
  - Aims to identify frequently occurring substrings of letters within either a word list (type-based) or a corpus of text (token-based)
  - Uses a probabilistic framework to balance between few, short morphs and many, longer morphs
- Morph-based language modelling uses morphs instead of words – may require longer context (since multiple morphs correspond to one word)

# Morph-based vs Word-based ASR

Speech recognition accuracies on Finnish (Fin1-Fin4), Estonian (Est), Turkish (Tur), and Egyptian Arabic (ECA) tasks, using morph-based and word-based n-gram language models.



word accuracies



letter accuracies

Creutz et al (2007)

# Speech recognition systems for low-resource languages

- Transferring data between acoustic models based on multilingual hidden representations
- Grapheme-based pronunciation lexica
- Morph-based language modeling

# Reading

- L Besacier et al (2014). "Automatic speech recognition for under-resourced languages: A survey", *Speech Communication*, 56:85–100. <http://www.sciencedirect.com/science/article/pii/S0167639313000988>
- Z Tüske et al (2013). "Investigation on cross- and multilingual MLP features under matched and mismatched acoustical conditions", ICASSP. <http://ieeexplore.ieee.org/abstract/document/6639090/>
- A Ghoshal et al (2013). "Multilingual training of deep neural networks", ICASSP-2013. <http://ieeexplore.ieee.org/abstract/document/6639084/>
- J-T Huang et al (2013). "Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers", ICASSP. <http://ieeexplore.ieee.org/abstract/document/6639081/>.
- M Gales et al (2015). "Unicode-based graphemic systems for limited resource languages", ICASSP. <http://ieeexplore.ieee.org/document/7178960/>
- M Creutz et al (2007). "Morph-based speech recognition and modeling OOV words across languages", *ACM Trans Speech and Language Processing*, 5(1). <http://doi.acm.org/10.1145/1322391.1322394>

# Sequence Discriminative Training

Steve Renals

Automatic Speech Recognition – ASR Lecture 14  
2 April 2018

## Recall: Maximum likelihood estimation of HMMs

- Maximum likelihood estimation (MLE) sets the parameters so as to maximize an objective function  $F_{\text{MLE}}$ :

$$F_{\text{MLE}} = \sum_{u=1}^U \log P_\lambda(\mathbf{X}_u \mid M(W_u))$$

for training utterances  $\mathbf{X}_1 \dots \mathbf{X}_U$  where  $W_u$  is the word sequence given by the transcription of the  $u$ th utterance,  $M(W_u)$  is the corresponding HMM, and  $\lambda$  is the set of HMM parameters

# Maximum mutual information estimation

- Maximum mutual information estimation (MMIE) aims to directly maximise the posterior probability (sometimes called conditional maximum likelihood). Using the same notation as before, with  $P(w)$  representing the language model probability of word sequence  $w$ :

$$\begin{aligned} F_{\text{MMIE}} &= \sum_{u=1}^U \log P_\lambda(M(W_u) \mid \mathbf{X}_u) \\ &= \sum_{u=1}^U \log \frac{P_\lambda(\mathbf{X}_u \mid M(W_u))P(W_u)}{\sum_{w'} P_\lambda(\mathbf{X}_u \mid M(w'))P(w')} \end{aligned}$$

# Maximum mutual information estimation

- Maximum mutual information estimation (MMIE) aims to directly maximise the posterior probability (sometimes called conditional maximum likelihood). Using the same notation as before, with  $P(w)$  representing the language model probability of word sequence  $w$ :

$$F_{\text{MMIE}} = \sum_{u=1}^U \log P_\lambda(M(W_u) | \mathbf{X}_u)$$

$$F_{\text{MLE}} = \sum_{u=1}^U \log \frac{P_\lambda(\mathbf{X}_u | M(W_u))P(W_u)}{\sum_{w'} P_\lambda(\mathbf{X}_u | M(w'))P(w')}$$

# Maximum mutual information estimation

$$F_{\text{MMIE}} = \sum_{u=1}^U \log \frac{P_\lambda(\mathbf{X}_u \mid M(W_u)) P(W_u)}{\sum_{w'} P_\lambda(\mathbf{X}_u \mid M(w')) P(w')}$$

# Maximum mutual information estimation

$$F_{\text{MMIE}} = \sum_{u=1}^U \log \frac{P_\lambda(\mathbf{X}_u \mid M(W_u)) P(W_u)}{\sum_{w'} P_\lambda(\mathbf{X}_u \mid M(w')) P(w')}$$

- **Numerator:** likelihood of data given correct word sequence (“clamped” to reference alignment)

# Maximum mutual information estimation

$$F_{\text{MMIE}} = \sum_{u=1}^U \log \frac{P_\lambda(\mathbf{X}_u \mid M(W_u)) P(W_u)}{\sum_{w'} P_\lambda(\mathbf{X}_u \mid M(w')) P(w')}$$

- **Numerator:** likelihood of data given correct word sequence (“clamped” to reference alignment)
- **Denominator:** total likelihood of the data given all possible word sequences – equivalent to summing over all possible word sequences estimated by the full acoustic and language models in recognition. (“free”)

# Maximum mutual information estimation

$$F_{\text{MMIE}} = \sum_{u=1}^U \log \frac{P_\lambda(\mathbf{X}_u \mid M(W_u)) P(W_u)}{\sum_{w'} P_\lambda(\mathbf{X}_u \mid M(w')) P(w')}$$

- **Numerator:** likelihood of data given correct word sequence (“clamped” to reference alignment)
- **Denominator:** total likelihood of the data given all possible word sequences – equivalent to summing over all possible word sequences estimated by the full acoustic and language models in recognition. (“free”)
- The objective function  $F_{\text{MMIE}}$  is optimised by making the correct word sequence likely (maximise the numerator), and all other word sequences unlikely (minimise the denominator)

# Sequence training and lattices

- Computing the denominator involves summing over all possible word sequences – estimate by generating lattices, and summing over all words in the lattice
- In practice also compute numerator statistics using lattices (useful for summing multiple pronunciations)
- Generate numerator and denominator lattices for every training utterance
- Denominator lattice uses recognition setup (with a weaker language model)
- Each word in the lattice is decoded to give a phone segmentation, and forward-backward is then used to compute the state occupation probabilities
- Lattices not usually re-computed during training

# MMIE is sequence discriminative training

- **Sequence:** like forward-backward (MLE) training, the overall objective function is at the sequence level – maximise the posterior probability of the word sequence given the acoustics  $P_\lambda(M(W_u) | \mathbf{X}_u)$
- **Discriminative:** unlike forward-backward (MLE) training the overall objective function for MMIE is discriminative – to maximise MMI:
  - Maximise the numerator by increasing the likelihood of data given the correct word sequence
  - Minimise the denominator by decreasing the total likelihood of the data given all possible word sequences

This results in “pushing up” the correct word sequence, while “pulling down” the rest

## MPE: Minimum phone error

- Basic idea adjust the optimization criterion so it is directly related to word error rate
- Minimum phone error (MPE) criterion

# MPE: Minimum phone error

- Basic idea adjust the optimization criterion so it is directly related to word error rate
- Minimum phone error (MPE) criterion

$$F_{\text{MPE}} = \sum_{u=1}^U \log \frac{\sum_W P_\lambda(\mathbf{X}_u | M(W))P(W)A(W, W_u)}{\sum_{W'} P_\lambda(\mathbf{X}_u | M(W'))P(W')}$$

- $A(W, W_u)$  is the phone transcription accuracy of the sentence  $W$  given the reference  $W_u$

## MPE: Minimum phone error

- Basic idea adjust the optimization criterion so it is directly related to word error rate
- Minimum phone error (MPE) criterion

$$F_{\text{MMIE}} = \sum_{u=1}^U \log \frac{\sum_W P_\lambda(\mathbf{X}_u | M(W_u)) P(W_u) A(W, W_u)}{\sum_{W'} P_\lambda(\mathbf{X}_u | M(W')) P(W')}$$

- $A(W, W_u)$  is the phone transcription accuracy of the sentence  $W$  given the reference  $W_u$

## MPE: Minimum phone error

- Basic idea adjust the optimization criterion so it is directly related to word error rate
- Minimum phone error (MPE) criterion

$$F_{\text{MPE}} = \sum_{u=1}^U \log \frac{\sum_W P_\lambda(\mathbf{X}_u | M(W))P(W)A(W, W_u)}{\sum_{W'} P_\lambda(\mathbf{X}_u | M(W'))P(W')}$$

- $A(W, W_u)$  is the phone transcription accuracy of the sentence  $W$  given the reference  $W_u$
- $F_{\text{MPE}}$  is a weighted average over all possible sentences  $w$  of the raw phone accuracy
- Although MPE optimizes a phone accuracy level, it does so in the context of a word-level system: it is optimized by finding probable sentences with low phone error rates

# HMM/DNN systems

- DNN-based systems are discriminative – the cross-entropy (CE) training criterion with softmax output layer “pushes up” the correct label, and “pulls down” competing labels
- CE is a frame-based criterion – we would like a sequence level training criterion for DNNs, operating at the word sequence level
- Can we train DNN systems with an MMI-type objective function?

# HMM/DNN systems

- DNN-based systems are discriminative – the cross-entropy (CE) training criterion with softmax output layer “pushes up” the correct label, and “pulls down” competing labels
- CE is a frame-based criterion – we would like a sequence level training criterion for DNNs, operating at the word sequence level
- Can we train DNN systems with an MMI-type objective function? – **Yes**

# Sequence training of hybrid HMM/DNN systems

- Forward- and back-propagation equations are structurally similar to forward and backward recursions in HMM training
- Initially train DNN framewise using cross-entropy (CE) error function
  - Use CE-trained model to generate alignments and lattices for sequence training
  - Use CE-trained weights to initialise weights for sequence training
- Train using back-propagation with sequence training objective function (e.g. MMI)

# Sequence training results on Switchboard (Kaldi)

Results on Switchboard “Hub 5 '00” test set, trained on 300h training set, comparing maximum likelihood (ML) and discriminative (BMMI) trained GMMs with framewise cross-entropy (CE) and sequence trained (MMI) DNNs. GMM systems use speaker adaptive training (SAT).

All systems had 8859 tied triphone states.

GMMs – 200k Gaussians

DNNs – 6 hidden layers each with 2048 hidden units

	SWB	CHE	Total
GMM ML (+SAT)	21.2	36.4	28.8
GMM BMMI (+SAT)	18.6	33.0	25.8
DNN CE	14.2	25.7	20.0
DNN MMI	12.9	24.6	18.8

Veseley et al, 2013.

# Lattice-Free MMI (LF-MMI) 1

- Sequence training of NN systems requires initially training a CE model to give a (very good) weight initialisation and to generate lattices for the denominator computation
- Lattice-free MMI (Povey et al, 2016) (sometimes called the 'Chain' model)
  - Avoids the need to pre-compute lattices for the denominator
  - Avoids the requirement to train using frame-based CE loss function, before sequence training
- Denominator calculation directly applies forward-backward computations to the denominator; speed-ups:
  - *phone-level* language model (typically 4-gram) (rather than word-level)
  - process training input in 1 second chunks (for GPU memory reasons)
  - Use 30ms frame rate at the output
  - Use a simpler HMM topology (hence fewer states, and a smaller output layer)

# Lattice-Free MMI (LF-MMI) 2

- LF-MMI is vulnerable to overfitting:
  - L2 regularization on the network output (aims to prevent over-confident likelihood estimations)
  - Multitask training: train the network with two output layers – one trained using MMI, the other trained at the frame-level using CE. Only the MMI output layer is used for recognition, but the network learns to optimise both MMI and CE.
- LF-MMI in practice
  - Faster than conventional training – subsampling at output layer (30ms frame rate), smaller networks (fewer HMM states)
  - Similar word error rates to sequence training
  - In practice LF-MMI is more sensitive to noisy training transcripts compared with frame based CE or conventional sequence training

# LF-MMI word error rates on various ASR tasks

pre ASR Data Set	Size	CE	CE → sMBR	LF-MMI	Rel. Δ
AMI-IHM	80 hrs	25.1%	23.8%	22.4%	6%
AMI-SDM	80 hrs	50.9%	48.9%	46.1%	6%
TED-LIUM*	118 hrs	12.1%	11.3%	11.2%	0%
Switchboard	300 hrs	18.2%	16.9%	15.5%	8%
LibriSpeech	960 hrs	4.97%	4.56%	4.28%	6%
Fisher + Switchboard	2100 hrs	15.4%	14.5%	13.3%	8%

TDNN acoustic models

Similar architecture across LVCSR tasks

Povey et al, 2016

# Summary

- Sequence training: discriminatively optimise GMM or DNN to a sentence (sequence) level criterion rather than a frame level criterion
  - ML training of HMM/GMM – sequence-level, not discriminative
  - CE training of HMM/NN – discriminative at the frame level
  - MMI training of HMM/GMM or HMM/NN – discriminative at the sequence level
- Usually initialise sequence discriminative training
  - HMM/GMM – first train using ML, followed by MMI
  - HMM/NN – first train at frame level (CE), followed by MMI
- Sequence discriminative training is computationally costly – need to compute the “denominator lattices”
- Lattice-free MMI for HMM/NN
  - avoids the need to compute denominator lattices
  - avoids the need to first apply CE training

# Reading

- HMM discriminative training: Sec 27.3.1 of: S Young (2008), "HMMs and Related Speech Recognition Technologies", in *Springer Handbook of Speech Processing*, Benesty, Sondhi and Huang (eds), chapter 27, 539–557. <http://www.inf.ed.ac.uk/teaching/courses/asr/2010-11/restrict/Young.pdf>
- NN sequence training: K Vesely et al (2013), "Sequence-discriminative training of deep neural networks", Interspeech-2013, [http://homepages.inf.ed.ac.uk/aghoshal/pubs/is13-dnn\\_seq.pdf](http://homepages.inf.ed.ac.uk/aghoshal/pubs/is13-dnn_seq.pdf)
- Lattice-free MMI: D Povey et al (2016), "Purely sequence-trained neural networks for ASR based on lattice-free MMI", Interspeech-2016. [http://www.danielpovey.com/files/2016\\_interspeech\\_mmi.pdf](http://www.danielpovey.com/files/2016_interspeech_mmi.pdf); slides – [http://www.danielpovey.com/files/2016\\_interspeech\\_mmi\\_presentation.pptx](http://www.danielpovey.com/files/2016_interspeech_mmi_presentation.pptx)

# End-to-end systems: Deep Speech and CTC

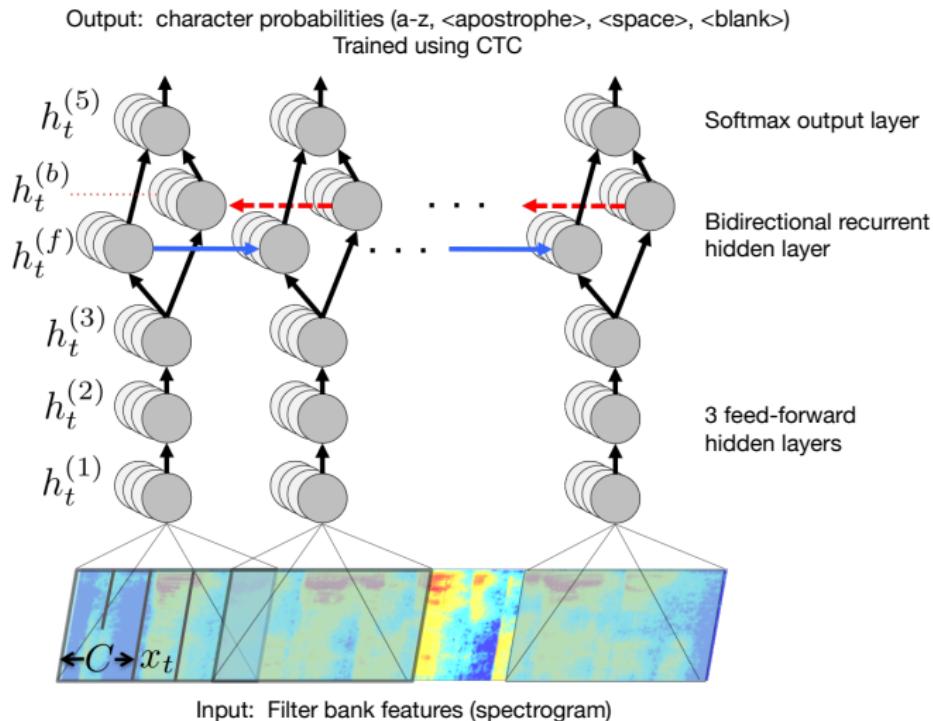
Steve Renals

Automatic Speech Recognition – ASR Lecture 15  
5 April 2018

# End-to-end systems

- End-to-end systems are systems which learn to directly map from an input sequence  $X$  to an output sequence  $Y$ , estimating  $P(Y|X)$
- ML trained HMMs are kind of end-to-end system – the HMM estimates  $P(X|Y)$  but when combined with a language model gives an estimate of  $P(Y|X)$
- Sequence discriminative training of HMMs (using GMMs or DNNs) can be regarded as end-to-end
  - But training is quite complicated – need to estimate the denominator (total likelihood) using lattices, first train conventionally (ML for GMMs, CE for NNs) then finetune using sequence discriminative training
  - Lattice-free MMI is one way to address these issues
- Other approaches based on recurrent networks which directly map input to output sequences
  - CTC – Connectionist Temporal Classification
  - Encoder-decoder approaches

# Deep Speech



Hannun et al, "Deep Speech: Scaling up end-to-end speech recognition",  
<https://arxiv.org/abs/1412.5567>.

# Deep Speech: Results

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]	<b>10.4</b>	n/a	n/a
<b>Deep Speech SWB</b>	20.0	31.8	25.9
<b>Deep Speech SWB + FSH</b>	12.6	<b>19.3</b>	<b>16.0</b>

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.

# Deep Speech Training

- Maps from acoustic frames to character sequences
- CTC loss function
- Makes good use of large training data
  - Synthetic additional training data by jittering the signal and adding noise
- Many computational optimisations
- n-gram language model to impose word-level constraints
- Competitive results on standard tasks

- Maps from acoustic frames to character sequences
- **CTC loss function**
- Makes good use of large training data
  - Synthetic additional training data by jittering the signal and adding noise
- Many computational optimisations
- n-gram language model to impose word-level constraints
- Competitive results on standard tasks

# Connectionist Temporal Classification (CTC)

- Train a recurrent network to map from input sequence  $X$  to output sequence  $Y$ 
  - sequences can be different lengths
  - frame-level alignment (matching each input frame to an output token) not required
- CTC sums over all possible alignments (similar to forward-backward algorithm) – "alignment free"
- Possible to back-propagate gradients through CTC

This presentation of CTC based on Awni Hannun, "Sequence Modeling with CTC", *Distill.* <https://distill.pub/2017/ctc>

# CTC: Alignment

- Imagine mapping  $(x_1, x_2, x_3, x_4, x_5, x_6)$  to  $[a, b, c]$
- Possible alignments:  $aaabbc, aabbcc, abbbbc, \dots$
- However
  - Don't always want to map every input frame to an output symbol (e.g. silence)
  - Want to be able to have two identical symbols adjacent to each other e.g.  $[h, e, l, l, o]$
- Solve this with an additional *blank* symbol ( $\epsilon$ )
  - Blanks removed from the output sequence
  - To model the same character in a row, separate with a blank

# CTC: Alignment example

h h e  $\epsilon$   $\epsilon$  | | |  $\epsilon$  | | o

First, merge repeat characters.

h e  $\epsilon$  |  $\epsilon$  | o

Then, remove any  $\epsilon$  tokens.

h e | | | o

The remaining characters are the output.

h e | | o

# CTC: Valid and invalid alignments

Consider an output [c, a, t] with an input of length six

## Valid Alignments

$\epsilon \ C \ C \ \epsilon \ a \ t$

c c a a t t

c a  $\epsilon$   $\epsilon$   $\epsilon$  t

## Invalid Alignments

c  $\epsilon$  c  $\epsilon$  a t

c c a a t   

c  $\epsilon$   $\epsilon$   $\epsilon$  | t t

corresponds to  
 $Y = [c, c, a, t]$

has length 5

missing the 'a'

# CTC: Alignment properties

- Monotonic – Alignments are monotonic (left-to-right model); no re-ordering (unlike neural machine translation)
- Many-to-one – Alignments are many-to-one; many inputs can map to the same output (however many outputs cannot map to a single input)
- CTC doesn't find a single alignment, sums over all possible alignments

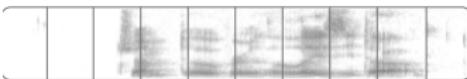
# CTC: Loss function

$$P(Y|X) = \sum_A \prod_t p(a_t|X)$$

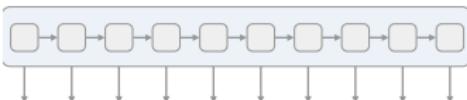
Estimate using an RNN

Sum over alignments using dynamic programming – similar structure as used in forward-backward algorithm and Viterbi

# CTC: Distribution over alignments



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



h	h	h	h	h	h	h	h	h	h	h
e	e	e	e	e	e	e	e	e	e	e
l	l	l	l	l	l	l	l	l	l	l
o	o	o	o	o	o	o	o	o	o	o
€	€	€	€	€	€	€	€	€	€	€

The input is fed into an RNN,  
for example.

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

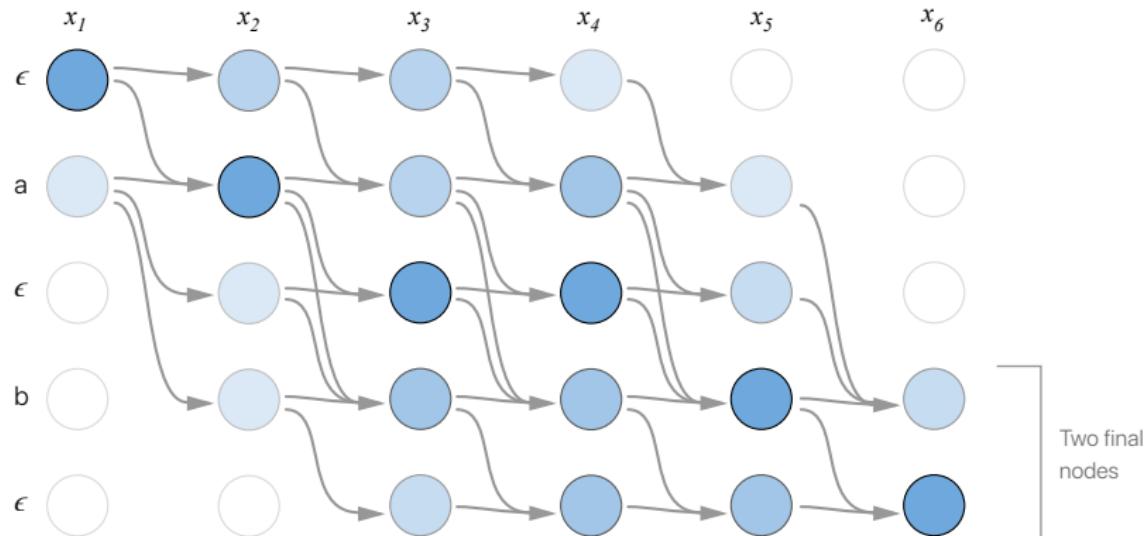
h	e	€	l	l	€	l	l	o	o
h	h	e	l	l	€	€	l	€	o
€	e	€	l	l	€	€	l	o	o

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

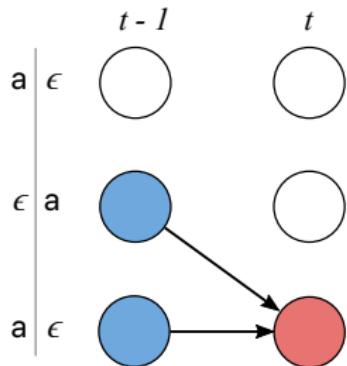
h	e	l	l	o
e	l	l	o	
h	e	l	o	

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

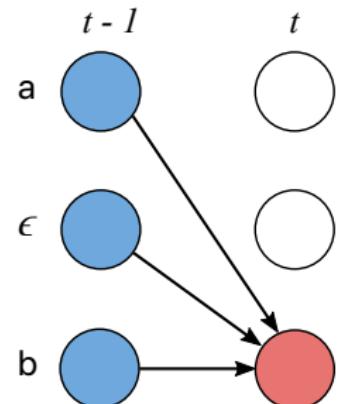
# CTC: Valid paths



# CTC: Allowed transitions



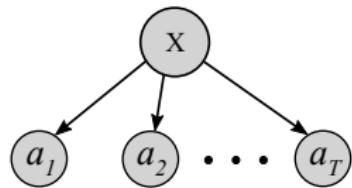
No skip transition allowed  
Previous token in output seq  
OR blank between repeat symbols



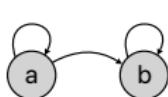
Skip transition allowed  
Previous token is a blank between  
different symbols

# Understanding CTC: Conditional independence assumption

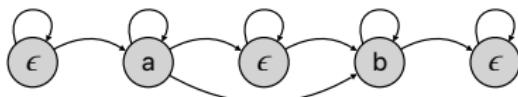
- Each output is dependent on the entire input sequence (in Deep Speech this is achieved using a bidirectional recurrent layer)
- Given the inputs, each output is independent of the other outputs (conditional independence)
- CTC does not learn a language model over the outputs, although a language model can be applied later
- Graphical model showing dependences in CTC:



# Understanding CTC: CTC and HMM



Left-to-right HMM



CTC HMM

- CTC can be interpreted as an HMM with additional (skippable) blank states, trained discriminatively

# Mozilla Deep Speech

- Mozilla have released an Open Source TensorFlow implementation of the Deep Speech architecture:
- [https://hacks.mozilla.org/2017/11/  
a-journey-to-10-word-error-rate/](https://hacks.mozilla.org/2017/11/a-journey-to-10-word-error-rate/)
- <https://github.com/mozilla/DeepSpeech>
- Close to state-of-the-art results on librispeech
- Mozilla Common Voice project: <https://voice.mozilla.org/en>

## Summary and reading

- CTC is an alternative approach to sequence discriminative training, typically applied to RNN systems
- Used in “Deep Speech” architecture for end-to-end speech recognition
- Reading
  - A Hannun et al (2014), “Deep Speech: Scaling up end-to-end speech recognition”, ArXiV:1412.5567.  
<https://arxiv.org/abs/1412.5567>
  - A Hannun (2017), “Sequence Modeling with CTC”, *Distill.*  
<https://distill.pub/2017/ctc>