Automatic Speech Recognition (II)

borrowing from
Daniel Jurafsky and James Martin

Outline for ASR

- ASR Architecture
  - The Noisy Channel Model
- Five easy pieces of an ASR system
  1) Language Model
  2) Lexicon/Pronunciation Model (HMM)
  3) Feature Extraction
  4) Acoustic Model
  5) Decoder
- Training
- Evaluation

Acoustic Modeling (= Phone detection)

- Given a 39-dimensional vector corresponding to the observation of one frame o_i
- And given a phone q we want to detect
- Compute $p(o_i|q)$
- Most popular method:
  - GMM (Gaussian mixture models)
- Other methods
  - Neural nets, CRFs, SVM, etc

Problem: how to apply HMM model to continuous observations?

- We have assumed that the output alphabet V has a finite number of symbols
- But spectral feature vectors are real-valued!
- How to deal with real-valued features?
  - Decoding: Given o_t, how to compute $P(o_t|q)$
  - Learning: How to modify EM to deal with real-valued features

Better than VQ

- vector quantization is insufficient for real ASR
- Instead: Assume the possible values of the observation feature vector o_t are normally distributed.
- Represent the observation likelihood function $b_j(o_t)$ as a Gaussian with mean μ_j and variance σ_j^2

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

Using a (univariate Gaussian) as an acoustic likelihood estimator

- Let's suppose our observation was a single real-valued feature (instead of 39D vector)
- Then if we had learned a Gaussian over the distribution of values of this feature
- We could compute the likelihood of any given observation o_t as follows:

$$b_j(o_t) = \frac{1}{\sqrt{2\pi\sigma_j^2}} exp\left(-\frac{(o_t - \mu_j)^2}{2\sigma_j^2}\right)$$

Training a Univariate Gaussian

- A (single) Gaussian is characterized by a mean and a variance
- Imagine that we had some training data in which each state was labeled
- We could just compute the mean and variance from the data:

$$\mu_i = \frac{1}{T}\sum_{t=1}^{T} o_t \quad s.t.\ o_t \ \text{is state } i$$

$$\sigma_i^2 = \frac{1}{T}\sum_{t=1}^{T} (o_t - \mu_i)^2 \quad s.t.\ o_t \ \text{is state } i$$

Training Univariate Gaussians

- But we don't know which observation was produced by which state!
- What we want: to assign each observation vector o_t to every possible state i, prorated by the probability the the HMM was in state i at time t.
- The probability of being in state i at time t is $\xi_t(i)$!!

$$\overline{\mu}_i = \frac{\sum_{t=1}^{T} \xi_t(i) o_t}{\sum_{t=1}^{T} \xi_t(i)} \qquad \overline{\sigma}^2{}_i = \frac{\sum_{t=1}^{T} \xi_t(i)(o_t - \mu_i)^2}{\sum_{t=1}^{T} \xi_t(i)}$$

Multivariate Gaussians

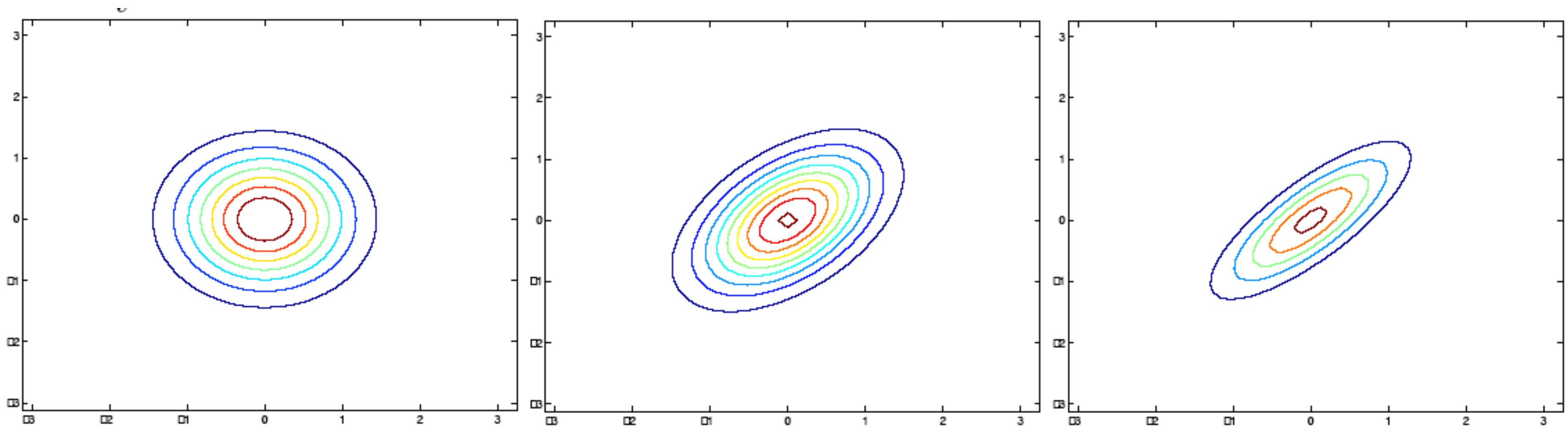
- Instead of a single mean μ and variance σ:

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

- Vector of means μ and covariance matrix Σ

$$f(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} \mid \Sigma \mid^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

Gaussian Intuitions: off-diagonal

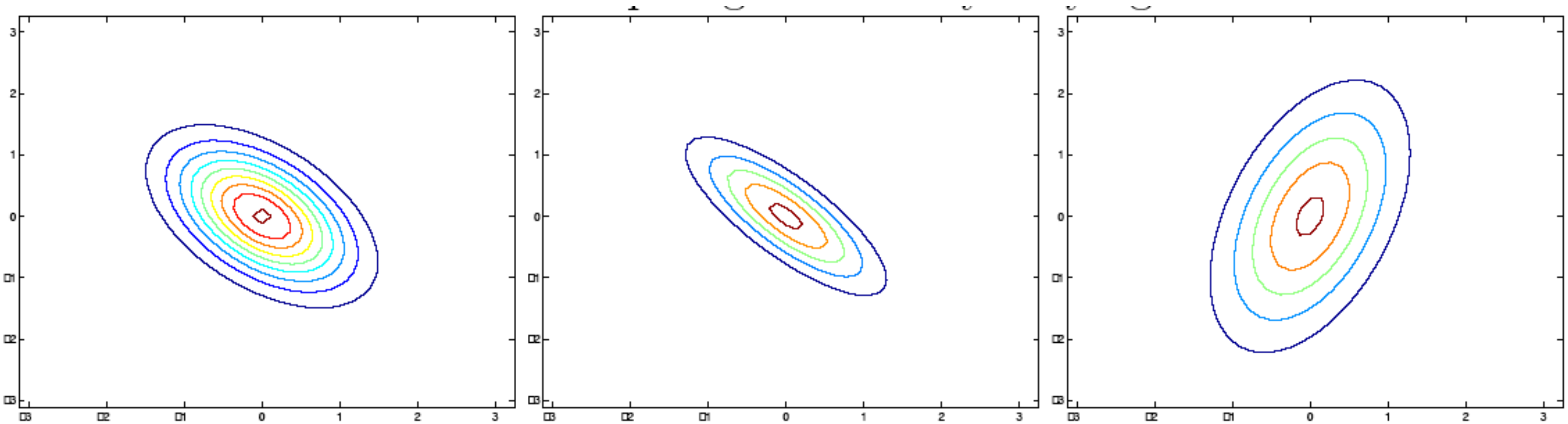


$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

- As we increase the off-diagonal entries, more correlation between value of x and value of y

Text and figures from Andrew Ng's lecture notes for CS229

Gaussian Intuitions: off-diagonal and diagonal



$$\Sigma = \begin{bmatrix} 1 & \text{-}0.5 \\ \text{-}0.5 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & \text{-}0.8 \\ \text{-}0.8 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 3 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

- Decreasing non-diagonal entries (#1-2)
- Increasing variance of one dimension in diagonal (#3)

Text and figures from Andrew Ng's lecture notes for CS229

But: assume diagonal covariance

- I.e., assume that the features in the feature vector are uncorrelated
- This isn't true for FFT features, but is true for MFCC features, as we will see.
- Computation and storage much cheaper if diagonal covariance.
- I.e. only diagonal entries are non-zero
- Diagonal contains the variance of each dimension σ_{ii}^2
- So this means we consider the variance of each acoustic feature (dimension) separately

Diagonal covariance

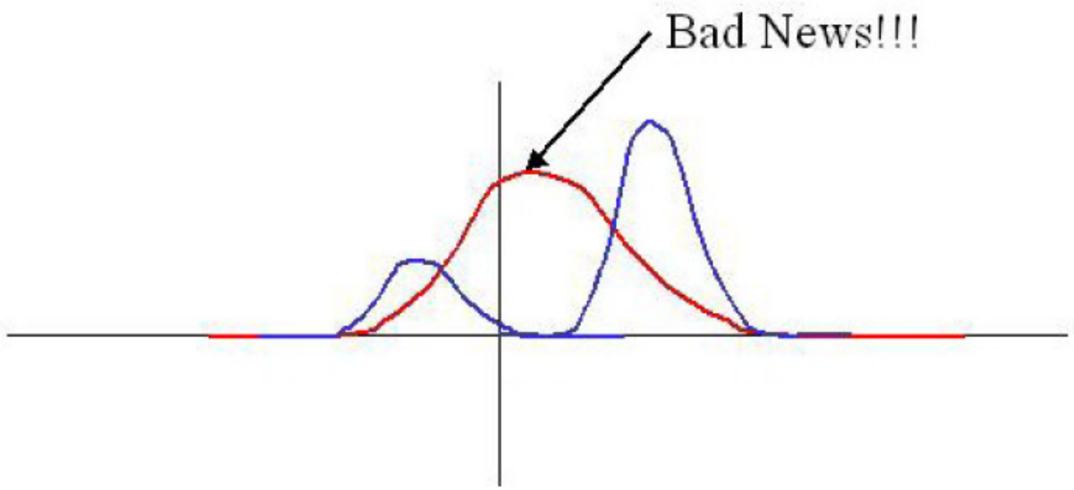
- Diagonal contains the variance of each dimension σ_{ii}^2
- So this means we consider the variance of each acoustic feature (dimension) separately

$$f(x \mid \mu, \sigma) = \prod_{d=1}^{D} \frac{1}{\sigma_j \sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{x_d - \mu_d}{\sigma_d} \right)^2 \right)$$

$$f(x \mid \mu, \sigma) = \frac{1}{2\pi^{D/2} \prod_{d=1}^{D} \sigma_d^2} \exp\left(-\frac{1}{2} \sum_{d=1}^{D} \frac{(x_d - \mu_d)^2}{\sigma_d^2}\right)$$

But we're not there yet

- Single Gaussian may do a bad job of modeling distribution in any dimension:



- Solution: Mixtures of Gaussians

Figure from Chen, Picheney et al slides

Mixtures of Gaussians

- M mixtures of Gaussians:

$$f(x \mid \mu_{jk}, \Sigma_{jk}) = \sum_{k=1}^{M} c_{jk} N(x, \mu_{jk}, \Sigma_{jk})$$

$$b_j(o_t) = \sum_{k=1}^{M} c_{jk} N(o_t, \mu_{jk}, \Sigma_{jk})$$

- For diagonal covariance:

$$b_j(o_t) = \sum_{k=1}^{M} \frac{c_{jk}}{2\pi^{D/2} \prod_{d=1}^{D} \sigma_{jkd}^{\;2}} \exp\left(-\frac{1}{2} \sum_{d=1}^{D} \frac{(x_{jkd} - \mu_{jkd})^2}{\sigma_{jkd}^{\;2}}\right)$$

GMMs

- Summary: each state has a likelihood function parameterized by:
  - M Mixture weights
  - M Mean Vectors of dimensionality D
  - Either
    - M Covariance Matrices of DxD
  - Or more likely
    - M Diagonal Covariance Matrices of DxD
    - which is equivalent to
    - M Variance Vectors of dimensionality D

Where we are

- Given: A wave file
- Goal: output a string of words
- What we know: the **acoustic model**
  - How to turn the wavefile into a sequence of acoustic feature vectors, one every 10 ms
  - If we had a complete phonetic labeling of the training set, we know how to train a gaussian "phone detector" for each phone.
  - We also know how to represent each word as a sequence of phones
- What we knew from Chapter 4: **the language model**
- Next:
  - Seeing all this back in the context of HMMs
  - Search: how to combine the language model and the acoustic model to produce a sequence of words

Decoding

- In principle:

$$\hat{W} = \underset{W \in \mathcal{L}}{\operatorname{argmax}} \overbrace{P(O|W)}^{\text{likelihood}} \; \overbrace{P(W)}^{\text{prior}}$$

- In practice:

$$\hat{W} = \underset{W \in \mathcal{L}}{\operatorname{argmax}} P(O|W)P(W)^{LMSF}$$

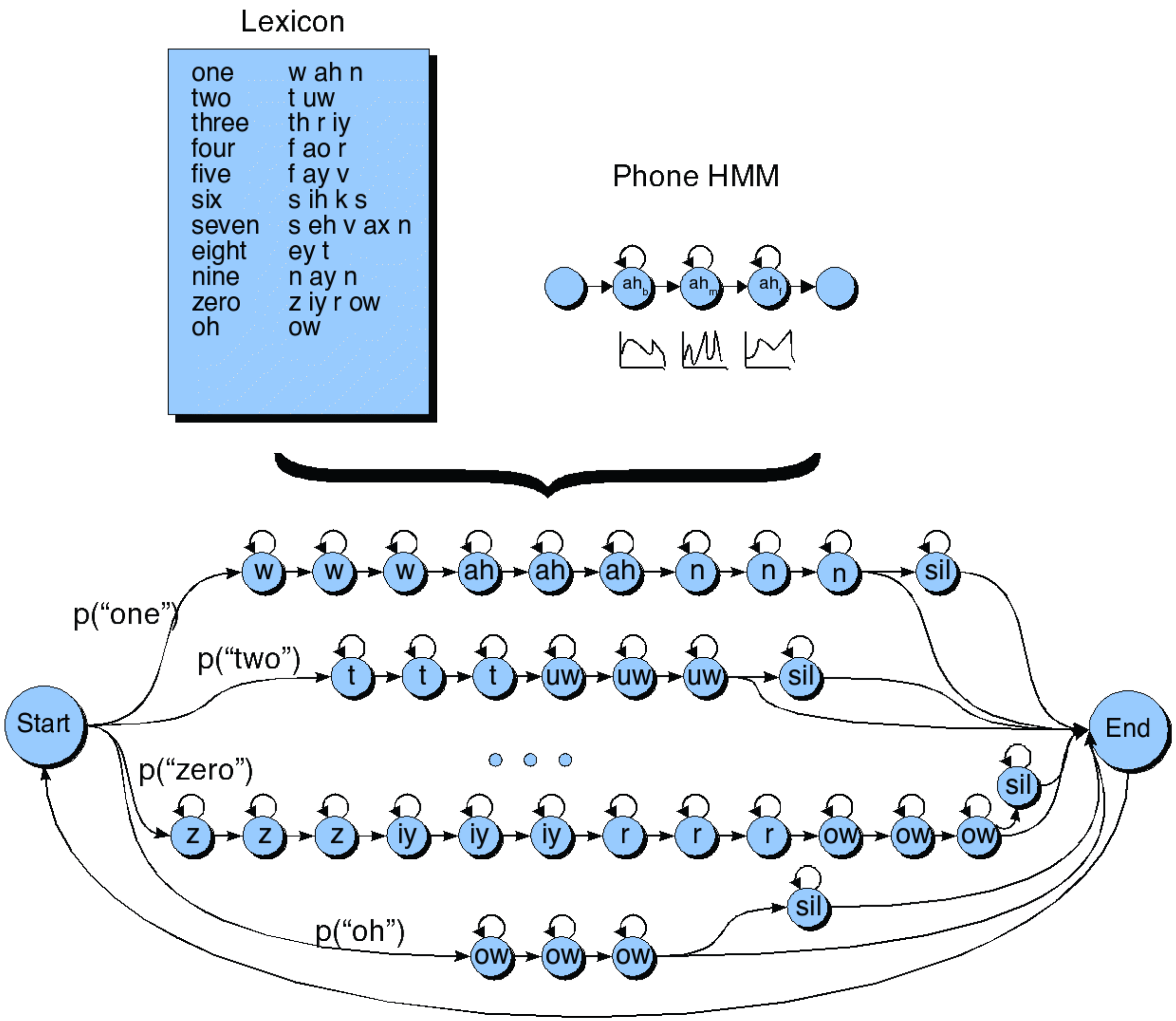
$$\hat{W} = \underset{W \in \mathcal{L}}{\operatorname{argmax}} P(O|W)P(W)^{LMSF} WIP^{N}$$

$$\hat{W} = \underset{W \in \mathcal{L}}{\operatorname{argmax}} \log P(O|W) + LMSF \times \log P(W) + N \times \log WIP$$

HMMs for speech

$Q = q_1 q_2 \ldots q_N$	a set of **states** corresponding to **subphones**
$A = a_{01} a_{02} \ldots a_{n1} \ldots a_{nn}$	a **transition probability matrix** A, each a_{ij} representing the probability for each subphone of taking a **self-loop** or going to the next subphone. Together, Q and A implement a **pronunciation lexicon**, an HMM state graph structure for each word that the system is capable of recognizing.
$B = b_i(o_t)$	A set of **observation likelihoods:**, also called **emission probabilities**, each expressing the probability of a cepstral feature vector (observation o_t) being generated from subphone state i.

HMM for digit recognition task



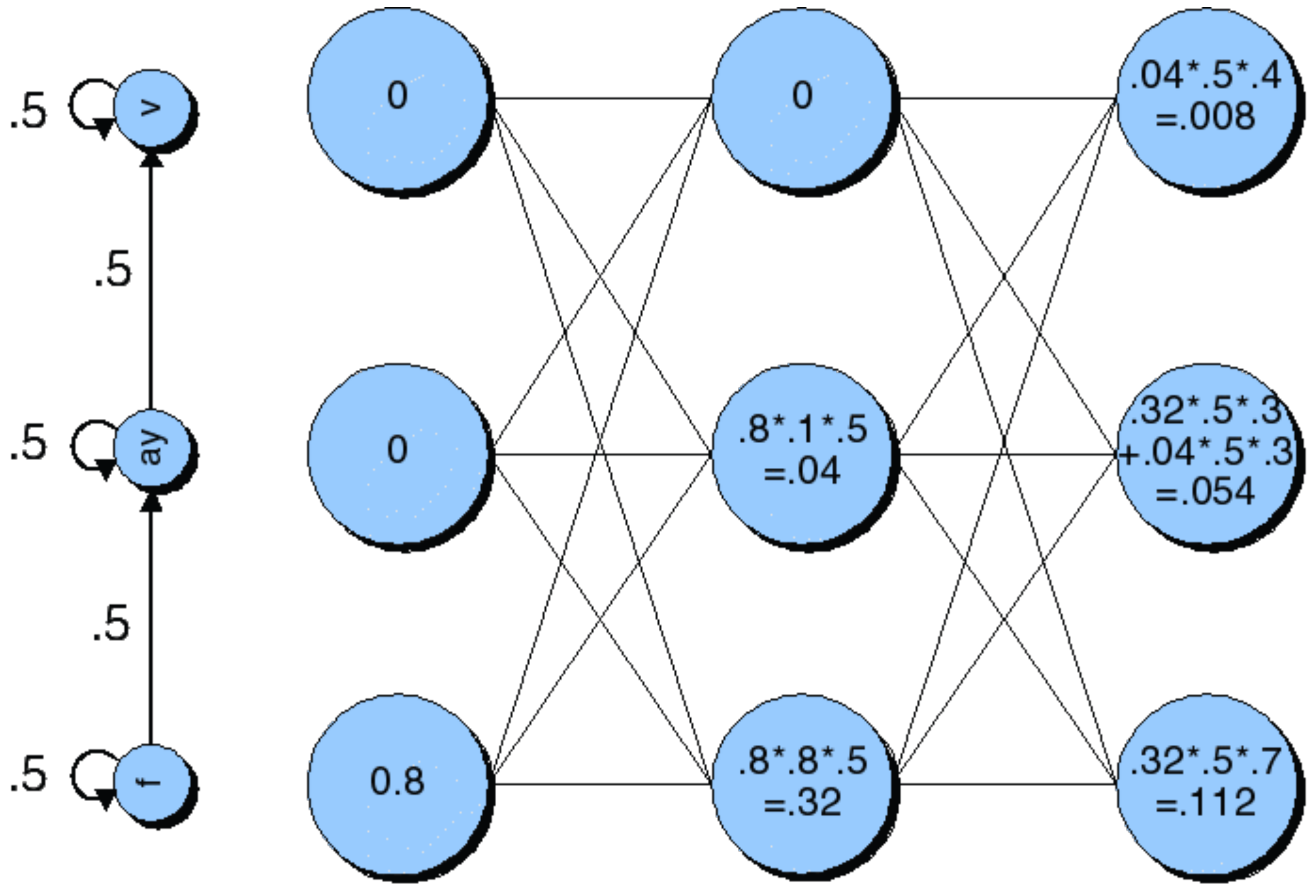
The Evaluation (forward) problem for speech

- The observation sequence O is a series of MFCC vectors
- The hidden states W are the phones and words
- For a given phone/word string W, our job is to evaluate P(O|W)
- Intuition: how likely is the input to have been generated by just that word string W

Evaluation for speech: Summing over all different paths!

- f ay ay ay ay v v v v
- f f ay ay ay ay v v v
- f f f f ay ay ay ay v
- f f ay ay ay ay ay ay v
- f f ay ay ay ay ay ay ay ay v
- f f ay v v v v v v v

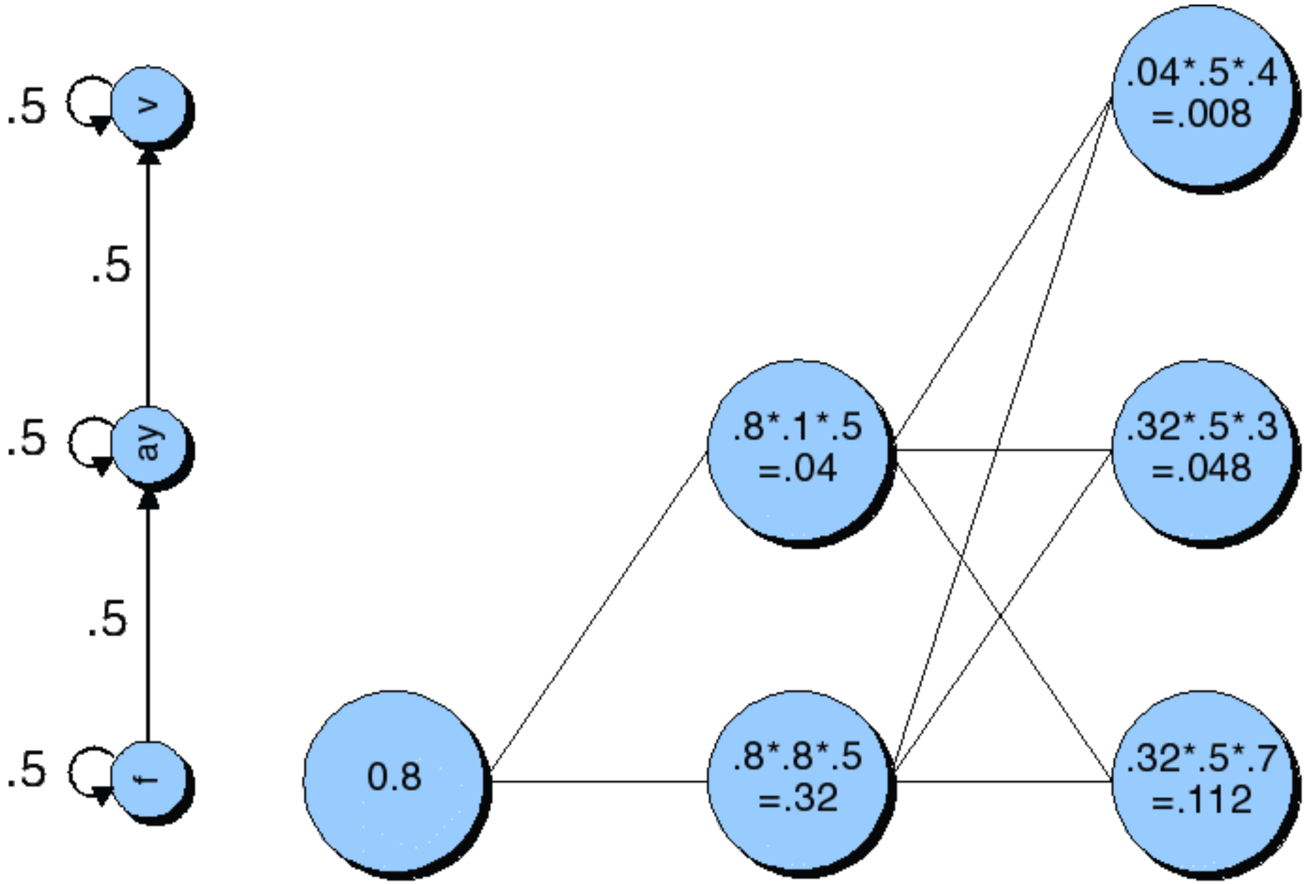
The forward lattice for “five”



The forward trellis for “five”

Time	1	2	3	4	5	6	7	8	9	10
V	0	0	0.008	0.0093	0.0114	0.00703	0.00345	0.00306	0.00206	0.00117
AY	0	0.04	0.054	0.0664	0.0355	0.016	0.00676	0.00208	0.000532	0.000109
F	0.8	0.32	0.112	0.0224	0.00448	0.000896	0.000179	4.48e-05	1.12e-05	2.8e-06
B: *f*	0.8	0.8	0.7	0.4	0.4	0.4	0.4	0.5	0.5	0.5
B: *ay*	0.1	0.1	0.3	0.8	0.8	0.8	0.8	0.6	0.5	0.4
B: *v*	0.6	0.6	0.4	0.3	0.3	0.3	0.3	0.6	0.8	0.9
B: *p*	0.4	0.4	0.2	0.1	0.1	0.1	0.1	0.1	0.3	0.3
B: *iy*	0.1	0.1	0.3	0.6	0.6	0.6	0.6	0.5	0.5	0.4

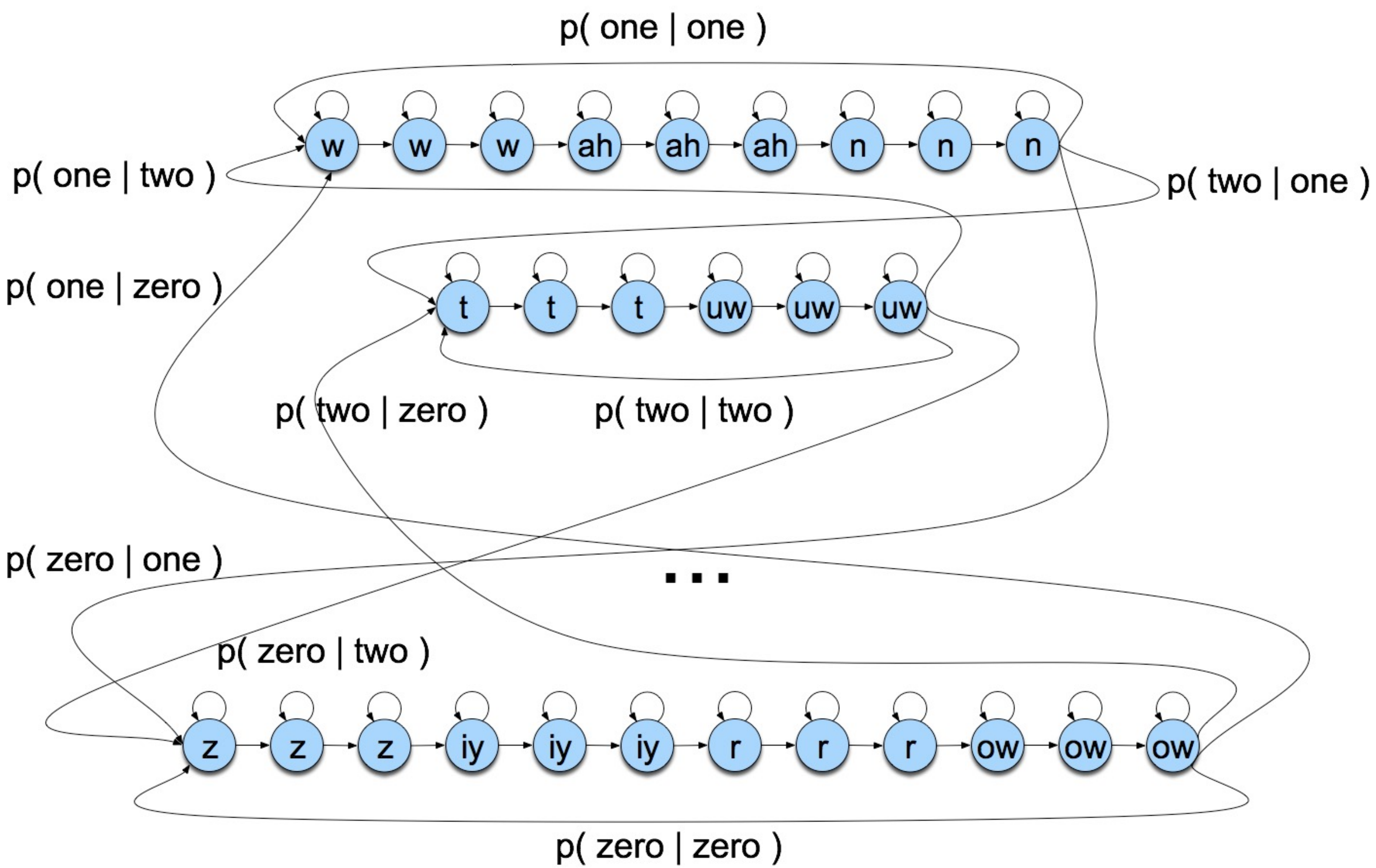
Viterbi trellis for “five”



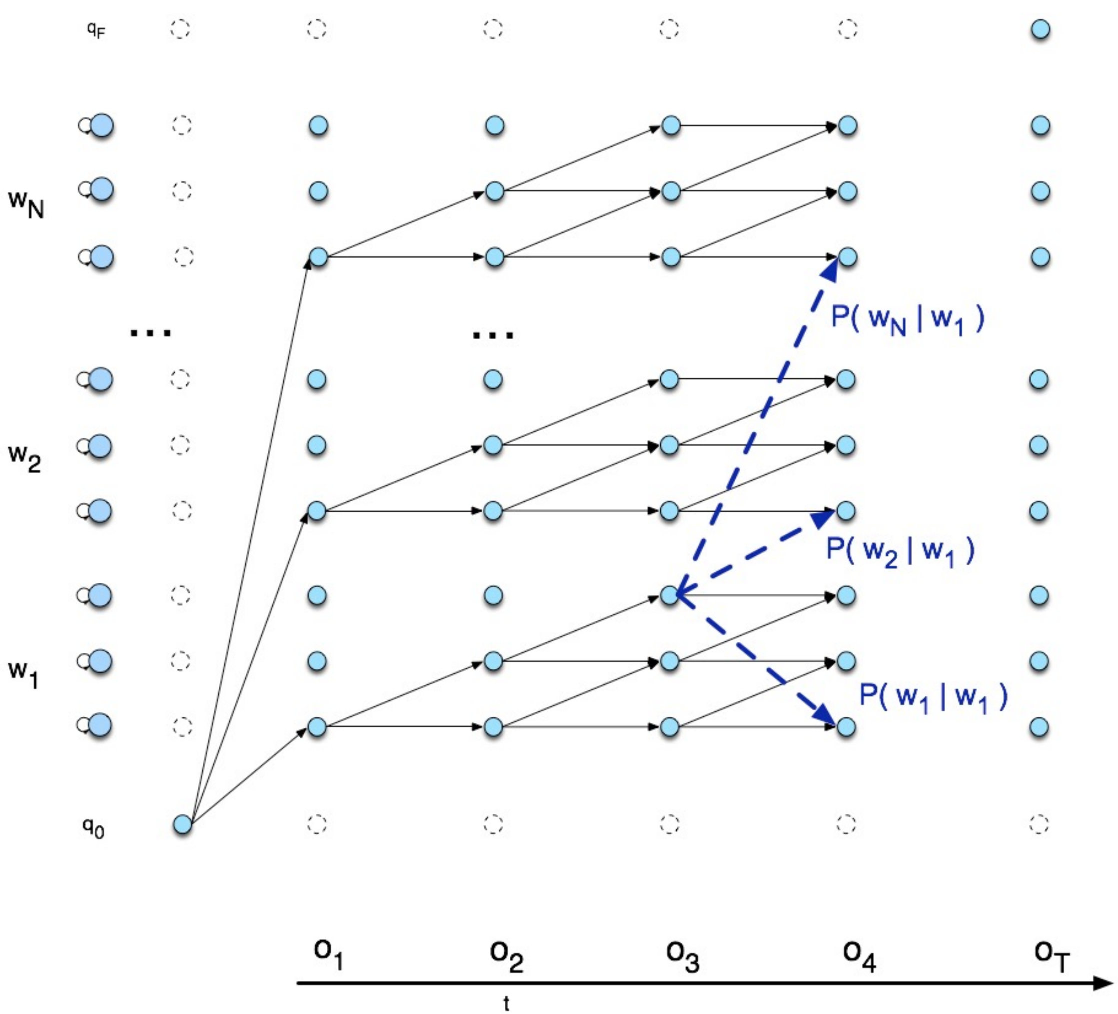
Viterbi trellis for "five"

V	0	0	0.008	0.0072	0.00672	0.00403	0.00188	0.00161	0.000667	0.000493
AY	0	0.04	0.048	0.0448	0.0269	0.0125	0.00538	0.00167	0.000428	8.78e-05
F	0.8	0.32	0.112	0.0224	0.00448	0.000896	0.000179	4.48e-05	1.12e-05	2.8e-06
Time	1	2	3	4	5	6	7	8	9	10
B	*f* 0.8 *ay* 0.1 *v* 0.6 *p* 0.4 *iy* 0.1	*f* 0.8 *ay* 0.1 *v* 0.6 *p* 0.4 *iy* 0.1	*f* 0.7 *ay* 0.3 *v* 0.4 *p* 0.2 *iy* 0.3	*f* 0.4 *ay* 0.8 *v* 0.3 *p* 0.1 *iy* 0.6	*f* 0.4 *ay* 0.8 *v* 0.3 *p* 0.1 *iy* 0.6	*f* 0.4 *ay* 0.8 *v* 0.3 *p* 0.1 *iy* 0.6	*f* 0.4 *ay* 0.8 *v* 0.3 *p* 0.1 *iy* 0.6	*f* 0.5 *ay* 0.6 *v* 0.6 *p* 0.1 *iy* 0.5	*f* 0.5 *ay* 0.5 *v* 0.8 *p* 0.3 *iy* 0.5	*f* 0.5 *ay* 0.4 *v* 0.9 *p* 0.3 *iy* 0.4

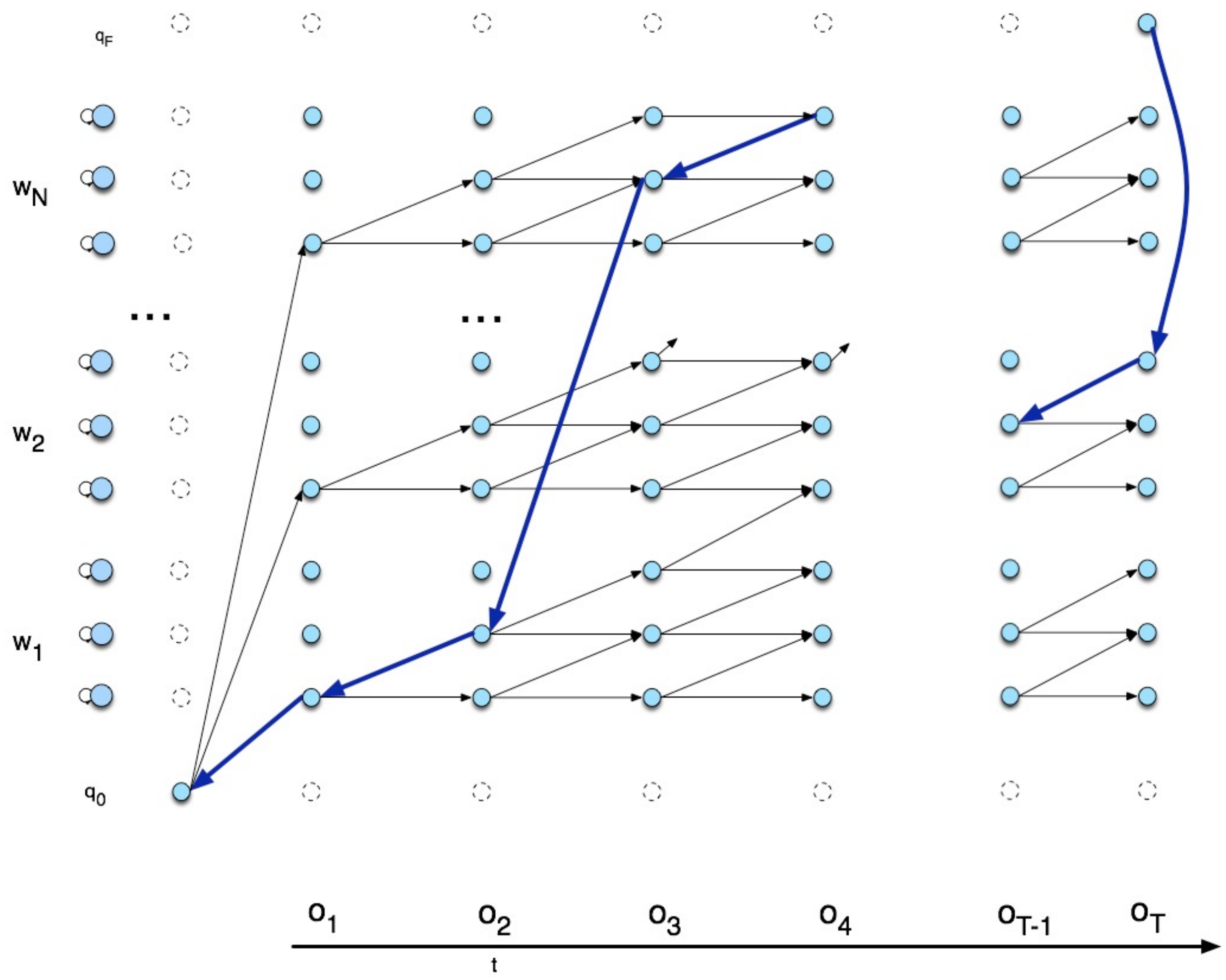
Search space with bigrams



Viterbi trellis



Viterbi backtrace



Evaluation

- How to evaluate the word string output by a speech recognizer?

Word Error Rate

- Word Error Rate =

100 (Insertions+Substitutions + Deletions)

Total Word in Correct Transcript

Aligment example:

REF: portable **** PHONE UPSTAIRS last night so

HYP: portable FORM OF STORES last night so

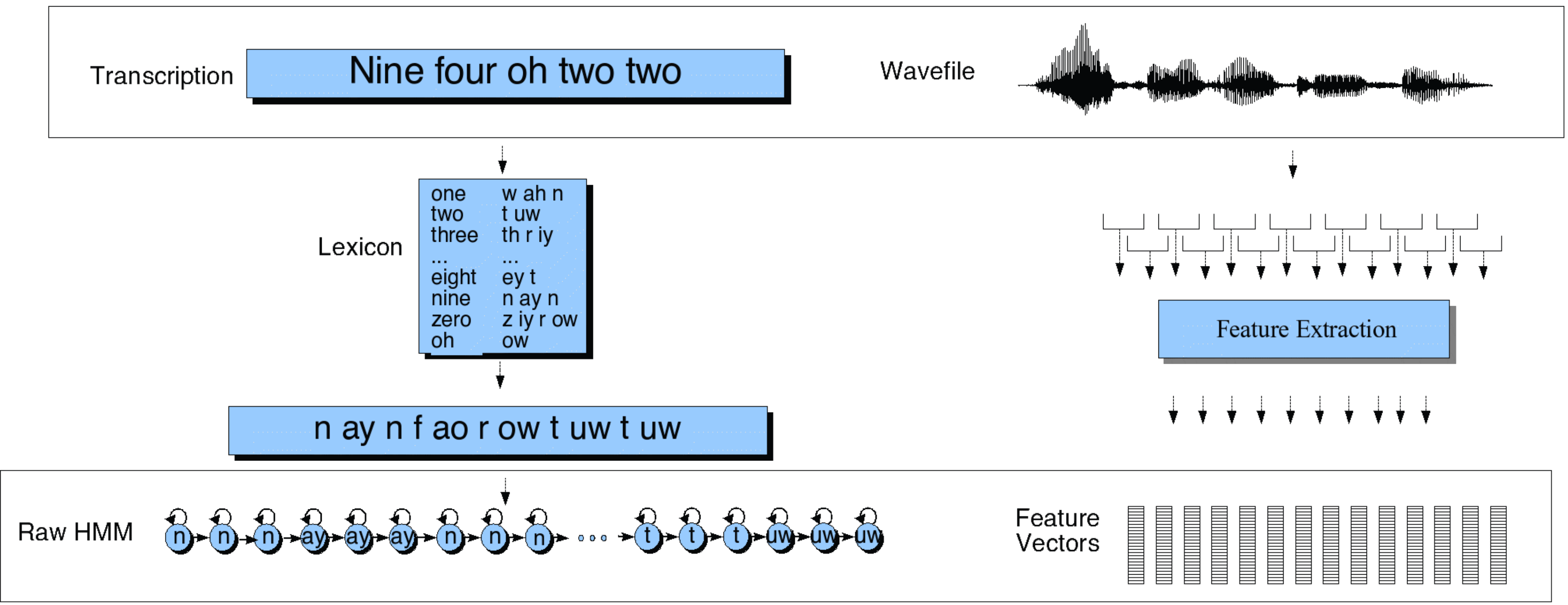
Eval I S S

WER = 100 (1+2+0)/6 = 50%

Better metrics than WER?

- WER has been useful
- But should we be more concerned with meaning ("semantic error rate")?
  - Good idea, but hard to agree on
  - Has been applied in dialogue systems, where desired semantic output is more clear

Training


Transcription
Nine four oh two two
Wavefile
Lexicon
one w ah n
two t uw
three th r iy
... ...
eight ey t
nine n ay n
zero z iy r ow
oh ow
Feature Extraction
n ay n f ao r ow t uw t uw
Raw HMM
n n n ay ay ay n n n ... t t t uw uw uw
Feature Vectors

Summary: ASR Architecture

- Five easy pieces: ASR Noisy Channel architecture
  1) Feature Extraction:
     39 "MFCC" features
  2) Acoustic Model:
     Gaussians for computing $p(o|q)$
  3) Lexicon/Pronunciation Model
     - HMM: what phones can follow each other
  4) Language Model
     - N-grams for computing $p(w_i|w_{i-1})$
  5) Decoder
     - Viterbi algorithm: dynamic programming for combining all these to get word sequence from speech!

Summary

- ASR Architecture
  - The Noisy Channel Model
- Five easy pieces of an ASR system
  1) Language Model
  2) Lexicon/Pronunciation Model (HMM)
  3) Feature Extraction
  4) Acoustic Model
  5) Decoder
- Training
- Evaluation