Foundations of ASR

Steven Wegmann
Director Speech Research
International Computer Science Institute

29 June 2015
Outline

The main idea in the ASR formalism

High level view of HMMs and why they are used in ASR

HMM details and inference algorithms

ASR model training and decoding details
The main idea in the ASR formalism

Some notation:

- \( W = \) sequence of words (transcription)
- \( X = \) acoustic observations or evidence (MFCCs, \( \Delta s \), \( \Delta\Delta s \))

Given a particular acoustic utterance, \( x \), our goal is to compute for every possible transcription \( w \) the probability

\[
P(W = w \mid X = x)
\]

We use these probabilities to \textit{decode} or \textit{recognize} the utterance \( x \) by selecting the most likely transcription \( w^{\text{recog}} \) via:

\[
w^{\text{recog}} = \arg \max_w P(W = w \mid X = x)
\]
The main idea (cont’d)

We use Bayes’ Theorem

\[ P(w \mid x) = \frac{P(x \mid w)P(w)}{P(x)} \]

This decomposes the problem into two probability models

- The acoustic model (AM) gives \( P(x \mid w) \)
- The language model (LM) gives \( P(w) \)
- The term \( P(x) \) is constant so it is (usually) ignored
The main idea (cont’d)

The mainstream choices for these models

- AM: hidden Markov model (HMM)
- LM: smoothed n-gram model (earlier lecture)

We can view $P(w | x)$ as the posterior on word sequences given the acoustic observations

- Where $P(w)$ is the prior on word sequences
- $P(x | w)$ is the likelihood of $x$ given $w$

Note that we are not modeling $P(w | x)$ directly

- Why not?
Generative vs Discriminative classifiers

What we’ve just described is an example of a generative classifier

- Model \( P(X \mid W = w) \) separately for each class \( W = w \)
- \( X \) is random, \( W = w \) is given
- Stronger model assumptions
- Uses maximum likelihood estimation
- Estimation is “easy”

A discriminative classifier models \( P(W \mid X = x) \) directly

- \( W \) is random, \( X = x \) is given
- Weaker model assumptions
- Uses conditional maximum likelihood estimation
- Estimation is “hard”
Our acoustic models are hidden Markov models

We want a probability model for a sequence of frames given a particular word sequence, i.e.

\[ P(x \mid w) \]

In the last 25 years, hidden Markov models (HMM’s) have provided the most successful form for this model.

This is spite of the fact that they make very strong independence assumptions which are clearly violated in the case of speech data.
Notational quibbles

In this talk $X$ is always a continuous random variable

- So $P(X = x \mid w) = P(x \mid w)$ is a probability density function
- I would normally write $f(x \mid w)$ to emphasize this
- In this talk I will use $P$ for notational simplicity
- It’s bad because it makes densities look like probabilities: sorry!

In practice computations are done in the log domain

- So $\log P(X = x \mid w)$ instead of $P(X = x \mid w)$
- This is because most of the quantities are very very small
  - Lots of multiplies of small quantities
  - In the log domain these becomes sums
- However I’m leaving out the logs for clarity
Roughly speaking, what is an HMM?

An HMM consists of two synchronized stochastic processes

- An unobservable Markov chain of hidden states $S_t$
- An observed process $X_t$: in our case the acoustic observations

Each $S_t$ is a discrete random variable, while the $X_t$ can be either a discrete or continuous random variable

The hidden chain ‘explains’ the observed process, because each $S_t$ emits $X_t$

- Using a state dependent probability distribution
Why use HMMs for speech recognition?

We will sketch two motivations for HMMs

- In particular: the hidden state sequence
- To help build some intuition about HMMs

The first: based on the physiology of speech production

The second: based on properties of the model

The spectrogram serves both motivations

- It is essentially what we observe
Spectrograms

A spectrogram is a particularly useful way to display speech data

It displays a sequence of spectral envelopes computed from the signal

- Slide a 25ms analysis window in 10ms frame steps
- Apply a Hamming window
- Compute the FFT in the window to get the energy at each frequency

The usual front end (MFCCs) starts with this same processing

- It—and most other frontends—are transformations of this picture
A spectrogram (from MIT 6.345 Spring '03)

Two plus seven is less than ten
Physiological motivation: picture
Physiological motivation: idea

When we speak a sentence our production mechanism traverses an essentially hidden sequence of configurations involving our

- Glottis
- Vocal tract
- Tongue
- Lips . . .

View the HMM’s hidden state sequence as a discrete approximation to these continuous sequences of configurations.

Spectrograms give us hope that relatively few states are necessary for a reasonable approximation.
Second motivation: HMM’s point of view

HMMs were designed to solve the following ‘segmentation’ problem

- We observe a slowly varying stochastic process produced by a hidden group of states
- We know that the hidden states emit in bunches
- The task is to identify which states emitted the observed data

Spectrogram gives us hope again
The spectrogram segmentation problem

Two plus seven is less than ten
Historical notes

L. Baum and colleagues at the IDA in the 60’s

- Developed HMMs to solve problems in cryptography
- Jim Baker was an intern at IDA while a Princeton undergrad
- Jim went to CMU for his PhD where he applied HMMs to ASR (1970’s)
- Jim eventually cofounded Dragon Systems
- Baum and Simons (a colleague) both went on to found early hedge funds

Fred Jelinek independently applied HMMs to ASR at IBM

- First used HMMs in the 60’s to solve a problem involving hard drive read errors
  - May have used the work of Stratonovich?
  - Stratonovich developed many HMM properties in the late 50’s but published them in Russian
HMM Details: discrete time stochastic process

A discrete time stochastic process is a sequence of random variables

- \((X_0, X_1, \ldots, X_t, \ldots)\)
- Where all the \(X_t\) share the same sample space
- The \(X_t\) may be vectors

In ASR there are canonical stochastic processes

- All related to spoken utterances
- The observed sequence of speech frames: \((X_t)\)
- The corresponding hidden sequence of HMM states: \((S_t)\)
- The sequence of words: \((W_i)\)
HMM Details: Markov chains

A Markov chain is a discrete time stochastic process \((S_t)\)

- The \(S_t\) are discrete random variables satisfying
  \[
P(S_{t+1} = s_{t+1} \mid S_t = s_t, \ldots, S_0 = s_0) = P(S_{t+1} = s_{t+1} \mid S_t = s_t)
  \]

- Called the (first order) Markov assumption

For a Markov chain: the future depends on the past only through the moment!

We also assume that the \(S_t\) take on a finite set of values

- I.e., the sample space \(S\) is finite
- Abuse notation and write \(S = |S|\)
- So \(S = \{1, \ldots, |S|\}\)
A Markov chain is called stationary if

\[ P(S_{t+1} \mid S_t) = P(S_t \mid S_{t-1}) = \cdots = P(S_1 \mid S_0) \]

The transition probabilities define a \( S \times S \) matrix \( A \):

\[ A_{ij} = P(S_{t+1} = j \mid S_t = i) \quad \forall t \geq 0 \quad \forall i, j \in S \]

There is also an initial distribution \( \pi \):

\[ \pi(i) = P(S_0 = i) \quad \forall i \in S \]
Most HMMs in ASR use this Markov chain

There are 5 states

- State 0 is the initial state
- State 4 is the final (accepting) state
- \( \pi(0) = 1 \) and \( A_{01} = 1 \)

At state 1 there are only two possible transitions

- Stay put with probability \( A_{11} \)
- Advance to state 2 with probability \( A_{12} \)

\( A_{11} + A_{12} = 1 \)
HMM definition

A hidden Markov model consists of

- An observed stochastic process $X_0, \ldots, X_T$
- A hidden stationary Markov chain $S_0, \ldots, S_T$
- Their joint probability distribution $P(x_0, \ldots, x_T, s_0, \ldots, s_T)$

Satisfying these (strong!) assumptions

- Conditional independence:
  - Given $S_0, \ldots, S_T$, the $X_i$ are independent
  - Given $S_i$, $X_i$ is independent of $S_j$ when $j \neq i$

- Stationarity: the distribution of $X_i$ given $S_i$ does not depend on $i$
  - These are called the state’s output distributions
  - One distribution per state: $\{ P(x \mid s) \}_{s=1}^{S}$
  - Not specified by the model, i.e., we choose
Simple toy example using an isolated word

Use this HMM to model the word “COW”

States 0 (start) and 4 (accept) do not emit frames (think FSA)

States C, O, W emit frames according to the (as yet unspecified) distributions

\[
P(x_t \mid s_t = C), \ P(x_t \mid s_t = O), \ \text{and} \ P(x_t \mid s_t = W)
\]
Suppose we have an 5 frame utterance of COW

- \((x_0, x_1, x_2, x_3, x_4)\)

What are the allowable state sequences and their probabilities?

- CCCOW
- CCOOW
- CCOWW
- COOOW
- COOOWW
- COWWW
Toy example (cont’d)

For the probability CCCOW multiply the following:

- \(1.0 \times P(x_1 \mid C)\)
- \(0.5 \times P(x_2 \mid C)\)
- \(0.5 \times P(x_3 \mid C)\)
- \(0.5 \times P(x_4 \mid O)\)
- \(0.2 \times P(x_5 \mid W) \times 0.9\)
- You do the rest!
In principal there are $3^T$ possible state sequences for a length $T$ utterance with this HMM

- Don’t count non-emitting start and accept
- In reality most of those sequences are not allowed: zero prob
- Exercise: the number of state sequences is

$$\sum_{k=1}^{T-2} k = \frac{(T - 2)(T - 1)}{2}$$
HMM inference problems/algorithms

The joint distribution

\[ P(x_0, \ldots, x_T, s_0, \ldots, s_T) \]

The marginal distribution (forward algorithm)

\[ P(x_0, \ldots, x_T) = \sum_{s_0, \ldots, s_T} P(x_0, \ldots, x_T, s_0, \ldots, s_T) \]

The conditional distributions (forward-backward algorithm)

\[ P(s_t \mid x_0, \ldots, x_t) \]

The maximum likelihood state sequence (Viterbi algorithm)

\[ \hat{s}_{ML} = \arg \max_{s_0, \ldots, s_T} P(x_0, \ldots, x_T, s_0, \ldots, s_T) \]
A major reason for the HMM’s success

There are efficient (linear in $T$) algorithms for inference

Recursive factorization and dynamic programming are key to these algorithms

Strong model assumptions allow major simplifications

Make repeated use of the chain rule for probabilities

\[
P(A, B, C) = P(C \mid A, B)P(A, B) = P(C \mid A, B)P(B \mid A)P(A)
\]
HMM’s joint distribution

First simple decomposition

\[ P(x_0, \ldots, x_T, s_0, \ldots, s_T) = P(x_0, \ldots, x_T \mid s_0, \ldots, s_T)P(s_0, \ldots, s_T) \]

Use recursive factorization on the Markov chain

\[ P(s_0, \ldots, s_T) = P(s_T \mid s_0, \ldots, s_{T-1})P(s_0, \ldots, s_{T-1}) \]
\[ = P(s_T \mid s_0, \ldots, s_{T-1})P(s_{T-1} \mid s_{T-2}, \ldots, s_0) \times \]
\[ \ldots P(s_1 \mid s_0)P(s_0) \]
\[ = P(s_T \mid s_{T-1})P(S_{T-1} \mid s_{T-2}) \ldots P(s_1 \mid s_0)P(s_0) \]
\[ = A_{s_{T-1}s_T}A_{s_{T-2}s_{T-1}} \ldots A_{s_0s_1} \pi(s_0) \]
\[ = \pi(s_0) \prod_{t=0}^{T-1} A_{s_ts_{t+1}} \]
HMM’s joint distribution (cont’d)

Apply conditional independence assumptions

\[ P(x_0, \ldots, x_T \mid s_0, \ldots, s_T) = \prod_{t=0}^{T} P(x_t \mid s_0, \ldots, s_T) = \prod_{t=0}^{T} P(x_t \mid s_t) \]

Putting it all together

\[ P(x_0, \ldots, x_T, s_0, \ldots, s_T) = \pi(s_0) \prod_{t=0}^{T-1} A_{s_t s_{t+1}} \prod_{t=0}^{T} P(x_t \mid s_t) \]
HMM’s marginal distribution

Sum over all possible state sequences

\[
P(x_0, \ldots, x_T) = \sum_{(s_0, \ldots, s_T) \in S \times \cdots \times S} \pi(s_0) \prod_{t=0}^{T-1} A_{s_t s_{t+1}} \prod_{t=0}^{T} P(x_t \mid s_t)
\]

It is not feasible to sum over all state sequences

- $S^{T+1}$ state sequences!
- Instead, computation is done using the forward algorithm
The forward algorithm

Define the forward probabilities

\[ \alpha_t(s_t) = P(x_0, \ldots, x_t, s_t) \]

Note that

\[ P(x_0, \ldots, x_T) = \sum_{s_T=1}^S P(x_0, \ldots, x_T, s_T) = \sum_{s_T=1}^S \alpha_T(s_T) \]

We will derive the recursion

\[ \alpha_t(s_t) = P(x_t \mid s_t) \sum_{s_{t-1}=1}^S A_{s_{t-1}s_t} \alpha_{t-1}(s_{t-1}) \]
The forward algorithm: dynamic programming

To compute $P(x_0, \ldots, x_T)$

- Initialize: $\alpha_0(s_0) = P(x_0, s_0) = \pi(s_0)P(x_0 \mid s_0) \forall s_0 \in S$

- At t: compute $\alpha_t(s_t) \forall s_0 \in S$ using the results from $t - 1$ via

$$
\alpha_t(s_t) = P(x_t \mid s_t) \sum_{s_{t-1}=1}^{S} A_{s_{t-1}s_t} \alpha_{t-1}(s_{t-1})
$$

- Terminate: at $T$

$$
P(x_0, \ldots, x_T) = \sum_{s_T=1}^{S} \alpha_T(s_T)
$$

Dynamic programming reduces exponential time to linear time

- $O(S^T) \rightarrow O(TS^2)$

- For HMMs used in ASR: time is actually $O(TS)$
The forward algorithm: derivation

The chain rule, conditional independence, and transition matrix $A$ give

\[
\alpha_t(s_t) = P(x_0, \ldots, x_t, s_t) = \sum_{s_{t-1}=1}^{S} P(x_0, \ldots, x_t, s_t, s_{t-1})
\]

\[
= \sum_{s_{t-1}=1}^{S} \left[ P(x_t \mid x_0, \ldots, x_{t-1}, s_t, s_{t-1}) P(s_t \mid x_0, \ldots, x_{t-1}, s_{t-1}) \times P(x_0, \ldots, x_{t-1}, s_{t-1}) \right]
\]

\[
= P(x_t \mid s_t) \sum_{s_{t-1}=1}^{S} P(s_t \mid s_{t-1}) \alpha_{t-1}(s_{t-1})
\]

\[
= P(x_t \mid s_t) \sum_{s_{t-1}=1}^{S} A_{s_{t-1}s_t} \alpha_{t-1}(s_{t-1})
\]
Toy example again

Exercise: work out the forward algorithm using a 5 frame utterance of COW

\[(x_0, x_1, x_2, x_3, x_4)\]
Toy example (cont’d)

Related exercise: show that the allowable paths form a trellis

\[
\begin{array}{cccccc}
0 & \rightarrow & C & \rightarrow & C & \rightarrow & C \\
\downarrow & & \downarrow & & \downarrow & & \downarrow \\
O & \rightarrow & O & \rightarrow & O & \rightarrow & O \\
\downarrow & & \downarrow & & \downarrow & & \downarrow \\
W & \rightarrow & W & \rightarrow & W & \rightarrow & 4 \\
\end{array}
\]

\[x_0 \quad x_1 \quad x_2 \quad x_3 \quad x_4\]
The forward-backward algorithm

We want to compute

\[ P(s_t \mid x_0, \ldots, x_T) \forall t \]

Introducing the backward probabilities

\[ \beta_t(s_t) = P(x_{t+i}, \ldots, x_T \mid s_t) \]

Exercise: these satisfy the recursion

\[ \beta_t(s_t) = \sum_{s_{t+1}=1}^{S} P(x_{t+1} \mid s_{t+1}) A_{s_ts_{t+1}} \beta_{t+1}(s_{t+1}) \]
The forward-backward algorithm (cont’d)

The independence assumptions imply

\[ \gamma_t(s_t) \equiv P(x_0, \ldots, x_t, x_{t+1}, \ldots, x_T, s_t) \]
\[ = P(x_0, \ldots, x_t, s_t)P(x_0, \ldots, x_T, s_t \mid x_0, \ldots, x_t, s_t) \]
\[ = P(x_0, \ldots, x_t, s_t)P(x_{t+1}, \ldots, x_T \mid x_0, \ldots, x_t, s_t) \]
\[ = P(x_0, \ldots, x_t, s_t)P(x_{t+1}, \ldots, x_T \mid s_t) \]
\[ = \alpha_t(s_t)\beta_t(s_t) \]

Thus

\[ P(s_t \mid x_0, \ldots, x_T) = \frac{P(x_0, \ldots, x_T, s_t)}{P(x_0, \ldots, x_T)} \]
\[ = \frac{\gamma_t(s_t)}{\sum_{s=1}^{S} \gamma_t(s)} \]
The forward-backward algorithm: dynamic programming

To compute $P(s_t \mid x_0, \ldots, x_T)$

- Run the forward algorithm
- Initialize: $\beta_T(s) = 1 \forall s \in S$
- At $t$: compute $\beta_t(s_t) \forall s_0 \in S$ using the results from $t + 1$ via

$$\beta_t(s_t) = \sum_{s_{t+1}=1}^S P(x_{t+1} \mid s_{t+1}) A_{s_t s_{t+1}} \beta_{t+1}(s_{t+1})$$

- Terminate: at $t = 0$, set $\gamma_t(s_t) = \alpha_t(s_t) \beta_t(s_t)$ then $\forall t$

$$P(s_t \mid x_0, \ldots, x_T) = \frac{\gamma_t(s_t)}{\sum_{s=1}^S \gamma_t(s)}$$
The Viterbi algorithm: brief review

The goal is to find the maximum likelihood state sequence

\[ \hat{s}_{ML} = \arg\max_{s_0, \ldots, s_T} P(x_0, \ldots, x_T, s_0, \ldots, s_T) \]

The Viterbi algorithm is

- Used for decoding/recognition
- Used for Model parameter estimation
- A modified version of the forward algorithm
  - Sums are replaced by Max: the “Viterbi approximation”
  - An extra term is introduced to store the “back-trace”
We introduce a quantity similar to $\alpha_t(s_t)$:

$$
vit_t(s_t) = \max_{s_0, \ldots, s_{t-1}} P(x_0, \ldots, x_t, s_0, \ldots, s_{t-1}, s_t)
$$

The following recursions hold

$$
vit_t(s_t) = P(x_t \mid s_t) \max_{s \in S} A_{ss_t} \ vit_{t-1}(s)
$$

$$
bp_t(s_t) = P(x_t \mid s_t) \ arg\max_{s \in S} A_{ss_t} \ bp_{t-1}(s)
$$
In ASR HMMs are usually used to model phones.

We want the hidden states to model a portion of the speech that is consistently realized:

- Usually across a large vocabulary
- Usually across many speakers

Instead of using phones, we actually use *phones in context*, often triphones:

- The realization of a phone depends on context
- Start with a dictionary entry: cat k ae t
- Corresponding triphone sequence is: sil-k+ae k-ae+t ae-t+sil
- Longer contexts, e.g., pentaphones, are useful too
- In principle one HMM for each triphone
- The three HMM states: start, middle, end of phone
  - Regions of local stationarity
Phones in context

Two plus seven is less than ten
The triphones need to be clustered

Problem: typical language uses 50 phonemes so 125K possible triphones

- Triphone distribution—like words—is approximately Zipfian
  - Small population of frequent triphones
  - Long tail of rare or non-occurring triphones
- Even in huge training sets, many triphones will never occur
- What models should we use for unobserved triphones?
  - They may occur during recognition!

Solution: top down decision tree clustering is performed on the triphone HMM states

- Each triphone state is in exactly one cluster
- Decision trees ask a series of questions, e.g.,
  - Is the left context a vowel?
  - Is the right context a stop?
What we need for ASR

A vocabulary

- The list of words that can be recognized

A dictionary

- A pronunciation for each word in terms of phonemes

AM and LM

- AM: HMMs for each triphone
- LM: models word sequences in vocabulary
  - Often estimated from 100’s of billions of words

The next slide shows the generative model

- Exploited in recognition and training
The standard ASR formalism

Notation: $o = x$ observations
Remarks on recognition/decoding

The two main approaches to decoding differ in how their search space is constructed

- Dynamic decoders
- Static WFST decoders
- Both use the Viterbi algorithm to conduct their search

Dynamic decoders follow the previous slide very closely

- Expert knowledge/experimentation has been applied to optimize the search algorithms
- Much has been published but there are many unpublished secrets!
Remarks on recognition/decoding (cont’d)

WFST-based decoders use WFSTs to represent the search space

- Standard WFST theory/operations are used to algorithmically optimize the search space
- The actual decode on this space is vanilla Viterbi
Acoustic model training overview: requirements

A large acoustic training corpus

- Many thousands of hours of audio data
  - Drawn from a large speaker population
  - Similar to the target application
- Corresponding word-level transcripts

A dictionary that covers all the words in the training corpus
The standard ASR formalism: training

Notation: $o = x$ observations
Acoustic model training overview: basic recipe

Train monophone HMMs

▶ Use transcripts and dictionary to get the monophone sequences: one HMM per monophone

Initialize triphone HMMs

▶ Use the context in the phone sequences to construct corresponding triphone sequences
  ▶ “a b c” maps to “a+b a-b+c b-c”
▶ Clone triphone models
  ▶ HMM parameters for a-b+c come from b
▶ Re-estimate triphone parameters

Cluster HMM triphone states using decision trees

Re-estimate clustered HMMs
Output distributions

While there are many potential choices for $P(x_t \mid s_t)$

- Mixture models with normal components have been dominant until recently
- Often called “Gaussian” mixture models (GMMs)
  - Gives Gauss (his usage in 1810) undeserved credit
  - De Moivre first in 1733
  - Even Laplace was earlier than Gauss (in the 1770’s)

Neural networks are currently the preferred choice

- The hybrid HMM/NNETs framework was developed in the late 80’s/early 90’s by Morgan and Bourlard (with many other key contributors)
- A later lecture will cover HMM/NNETs
- HMM/GMMs are still very useful/instructive
A $D$-dimensional normal with diagonal covariance

- Parameters: mean vector $\mu$ and variance vector $\sigma^2$
- Write $\mu_d$ for the $d^{th}$ component
- $\mathcal{N}(\mu, \sigma^2)$ denotes the distribution
- $\varphi(x; \mu, \sigma^2)$ denotes the probability density function (p.d.f.)

$$
\varphi(x; \mu, \sigma^2) = \frac{1}{\sqrt{(2\pi)^D \prod_{d=1}^{D} \sigma^2_d}} \exp \left( -\frac{1}{2} \sum_{d=1}^{D} \frac{(x - \mu_d)^2}{\sigma^2_d} \right)
$$

Mixture models of normals, i.e., GMMs

- $M$ component weights, means, variances $\{w_m, \mu_m, \sigma_m^2\}_{m=1}^{M}$
- $\sum_{m=1}^{M} w_m = 1$
- P.d.f.: $\sum_{m=1}^{M} w_m \varphi(x; \mu_m, \sigma_m^2)$
Multivariate normal distributions: review

Maximum likelihood estimates (MLEs)

- A sample of $T$, $D$-dim training examples $\{x_t\}_{t=1}^T$
- $x_{t,d}$ is example $t$’s $d^{th}$ component
- MLE for the mean:
  \[ \hat{\mu} = \frac{1}{T} \sum_{t=1}^{T} x_t \]
- MLE for the variance in dim $d$:
  \[ \hat{\sigma}^2_d = \frac{1}{T} \sum_{t=1}^{T} (x_{t,d} - \hat{\mu}_d)^2 \]

We are skipping MLEs for GMMs
HMMs with normal output distributions

For each state $s \in S$

- $P(x \mid s) = \varphi(x; \mu_s, \sigma_s^2)$
- State dependent means and variances $\{\mu_s, \sigma_s^2\}_{s=1}^S$

Given a training sample $\{x_t\}_{t=1}^T$ if we knew what states at each time $t$, $s_t$, were responsible for the $x_t$ then the MLEs would be obvious

- But we don’t since the state identities are hidden

Two approximate solutions to this problem

- Viterbi training
- Baum-Welch (BW) training
HMM with normal output distributions

Let $\theta$ denote the HMM model parameters

- The state transition matrix $A$
- State means and variances $\{\mu_s, \sigma^2_s\}_{s=1}^S$
- Surface the existence of $\theta$ in the probability distributions $P_\theta$

The Baum-Welch and Viterbi estimation algorithms

- Are iterative
- Produce sequences of model parameter estimates
  - $\hat{\theta}_0, \hat{\theta}_1, \ldots, \hat{\theta}_n$

We are skipping the estimates for transitions and mixture models

- Functionally equivalent and easy to work out
Estimate $\hat{\theta}_{i+1}$ using $\hat{\theta}_i$

**Viterbi training**
- Uses the maximum likelihood estimate for the state identities
- Uses the Viterbi algorithm and $P_{\hat{\theta}_i}$
- A hard, single choice for which state generates each frame
- Kaldi uses Viterbi training

**Baum-Welch training**
- Uses the distributions $P_{\hat{\theta}_i}(s_t \mid x_1, \ldots, x_T)$
- Uses the forward-backward algorithm
- A soft, fractional choice which state generates each frame
- HTK uses Baum-Welch training
Viterbi training: basic idea

First use the Viterbi algorithm to obtain

\[ \hat{s} = \arg \max_{s_1, \ldots, s_T} P_{\hat{\theta}_i}(x_1, \ldots, x_T, s_1, \ldots, s_T) \]

- \( \hat{s} \) depends on \( \hat{\theta}_i \) so should write \( \hat{s}_{\hat{\theta}_i} \)
  - We don’t to keep notation “cleaner”
- View this as an assignment of frames to states
- I.e., \( x_t \) is generated by state \( \hat{s}_t \)
- Called a Viterbi or forced state-level alignment
- This means that state \( k \)’s training data is the set of frames

\[ \{x_t : 1 \leq t \leq T \text{ and } \delta_{k, \hat{s}_t} = 1\} \]

- Frame \( x_t \)’s count for state \( k \) is \( \delta_{k, \hat{s}_t} \) (depends on \( \hat{\theta}_i \))
Viterbi training: the new estimate $\hat{\theta}_{i+1}$

We use the MLEs obtained on each state’s training data to obtain the new parameter estimates $\hat{\theta}_{i+1}$

- MLE for state $k$’s mean:

$$\hat{\mu}_k = \frac{\sum_{t=1}^{T} \delta_{k,\hat{s}_t} x_t}{\sum_{t=1}^{T} \delta_{k,\hat{s}_t}}$$

- MLE for state $k$’s variance in dim $d$:

$$\hat{\sigma}^2_{k,d} = \frac{\sum_{t=1}^{T} \delta_{k,\hat{s}_t} (x_t,d - \hat{\mu}_{k,d})^2}{\sum_{t=1}^{T} \delta_{k,\hat{s}_t}}$$
Baum-Welch training: basic idea

First use the forward-backward algorithm to obtain

\[ P_{\hat{\theta}_i}(s_t \mid x_1, \ldots, x_T) \]

- View this as a probabilistic assignment of frames to states
- I.e., \( x_t \) is generated by state \( k \) with prob
  \[ P_{\hat{\theta}_i}(s_t = k \mid x_1, \ldots, x_T) \]
- Frame \( x_t \)'s count for state \( k \) is \( P_{\hat{\theta}_i}(s_t = k \mid x_1, \ldots, x_T) \)
- The counts are now fractional instead of integral
Baum-Welch training: the new estimate $\hat{\theta}_{i+1}$

We use the MLEs obtained on each state’s training data to obtain the new parameter estimates $\hat{\theta}_{i+1}$

- MLE for state $k$’s mean:

$$
\hat{\mu}_k = \frac{\sum_{t=1}^{T} P_{\hat{\theta}_i}(s_t = k \mid x_1, \ldots, x_T) x_t}{\sum_{t=1}^{T} P_{\hat{\theta}_i}(s_t = k \mid x_1, \ldots, x_T)}
$$

- MLE for state $k$’s variance in dim $d$:

$$
\hat{\sigma}^2_{k,d} = \frac{\sum_{t=1}^{T} P_{\hat{\theta}_i}(s_t = k \mid x_1, \ldots, x_T)(x_{t,d} - \hat{\mu}_{k,d})^2}{\sum_{t=1}^{T} P_{\hat{\theta}_i}(s_t = k \mid x_1, \ldots, x_T)}
$$
How do we initialize, i.e., choose $\theta_0$?

Usually via a flat start

- State transition probabilities use the uniform distribution
  - E.g. $A_{1,1} = A_{1,2} = 0.5$

- Use the training data total mean and variance for all the states means and variances:

\[
\bar{\mu} = \frac{1}{T} \sum_{t=1}^{T} x_t
\]

\[
\bar{\sigma}_d^2 = \frac{1}{T} \sum_{t=1}^{T} (x_{t,d} - \bar{\mu}_d)^2 \quad \forall d
\]
What are Viterbi/Baum-Welch training optimizing?

Baum-Welch training

- Baum-Welch model/parameter selection criterion is the training data likelihood:

\[ \mathcal{L}(\theta) = \sum_{s_1, \ldots, s_T} P_\theta(x_1, \ldots, x_T, s_1, \ldots, s_T) \]

- Produces a parameter sequence \((\hat{\theta}_0, \hat{\theta}_1, \ldots, \hat{\theta}_n)\) satisfying

\[ \mathcal{L}(\hat{\theta}_0) \leq \mathcal{L}(\hat{\theta}_1) \leq \cdots \leq \mathcal{L}(\hat{\theta}_n) \]

- Baum-Welch is an example of an Expectation-Maximum algorithm
  - Many interesting/important properties
  - Beyond the scope of this introduction
Viterbi training

- Viterbi model/parameter selection criterion is an approximation to the training data likelihood:

\[ \mathcal{V} \mathcal{L}(\theta) = \max_{s_1, \ldots, s_T} P_\theta(x_1, \ldots, x_T, s_1, \ldots, s_T) \]

- Produces a parameter sequence \((\hat{\theta}_0, \hat{\theta}_1, \ldots, \hat{\theta}_n)\) satisfying

\[ \mathcal{V} \mathcal{L}(\hat{\theta}_0) \leq \mathcal{V} \mathcal{L}(\hat{\theta}_1) \leq \cdots \leq \mathcal{V} \mathcal{L}(\hat{\theta}_n) \]

- This is straightforward to verify: exercise!
A short, incomplete list of things we didn’t cover

**Discriminative training**

- Alternatives to MLE using different model selection criteria
- E.g., MMI and MPE

**Model adaptation**

- E.g., given a small sample of data from a new speaker: can we leverage extant models to more accurately predict future samples?
- Bayesian methods are ideal
- MAP, MLLR, etc.

**Reducing inter-speaker variability at training/test**

- Vocal tract length normalization (VTLN)
- Speaker adaptive training (SAT)