Discriminative Language and Acoustic Modeling for Large Vocabulary Continuous Speech Recognition

Murat Saraçlar

Electrical and Electronics Engineering Department,
Boğaziçi University, Istanbul, Turkey

Currently Visiting Research Scientist at IBM TJ Watson Research Center

December 3, 2012

Introduction to ASR

Aim:

To automatically produce the transcription of the input speech.

$W = \{w_1 \ldots \text{nice cat} \ldots w_n\}, w_i \in \mathcal{V}$: fixed and finite vocabulary

$A = \{a_1 a_2 \ldots a_t\}, a_i \in \mathcal{A}$: 39 dimensional MFCC features

$$\hat{W} = \underset{W}{\operatorname{argmax}} P(W|A) = \underset{W}{\operatorname{argmax}} \frac{P(A|W)P(W)}{P(A)} = \underset{W}{\operatorname{argmax}} P(A|W)P(W)$$

Introduction to ASR

Aim:

To automatically produce the transcription of the input speech.

$W = \{w_1 \ldots \text{nice cat} \ldots w_n\}$, $w_i \in \mathcal{V}$: fixed and finite vocabulary $\longrightarrow$ (Out-of-Vocabulary (OOV) words)

$A = \{a_1 a_2 \ldots a_t\}$, $a_i \in \mathcal{A}$: 39 dimensional MFCC features

$$\hat{W} = \underset{W}{\operatorname{argmax}}\, P(W|A) = \underset{W}{\operatorname{argmax}}\, \frac{P(A|W)P(W)}{P(A)} = \underset{W}{\operatorname{argmax}}\, P(A|W)P(W)$$

Introduction to ASR

Aim:

To automatically produce the transcription of the input speech.

$W = \{w_1 \ldots \text{nice cat} \ldots w_n\}$, $w_i \in \mathcal{V}$: fixed and finite vocabulary $\longrightarrow$ (Out-of-Vocabulary (OOV) words)

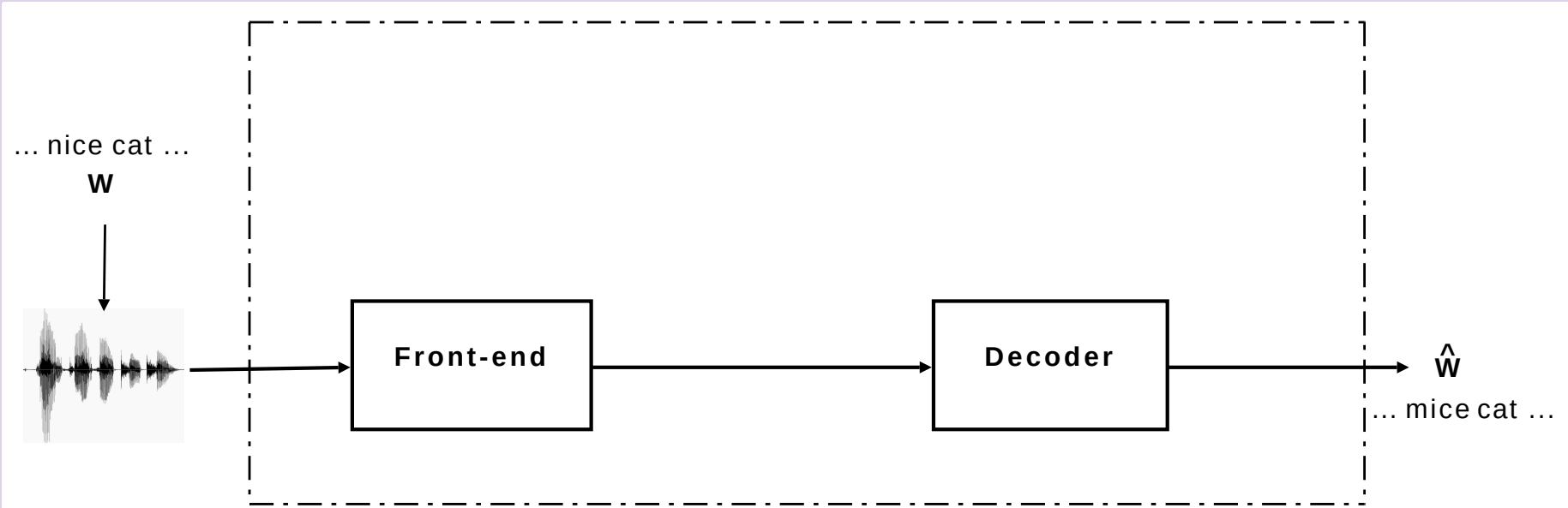
$A = \{a_1 a_2 \ldots a_t\}$, $a_i \in \mathcal{A}$: 39 dimensional MFCC features

$$\hat{W} = \underset{W}{\operatorname{argmax}}\, P(W|A) = \underset{W}{\operatorname{argmax}}\, \frac{P(A|W)P(W)}{P(A)} = \underset{W}{\operatorname{argmax}}\, P(A|W)P(W)$$

Introduction to ASR

Aim:

To automatically produce the transcription of the input speech.



$W = \{w_1 \ldots \text{nice cat} \ldots w_n\}$, $w_i \in \mathcal{V}$: fixed and finite vocabulary $\longrightarrow$ (Out-of-Vocabulary (OOV) words)

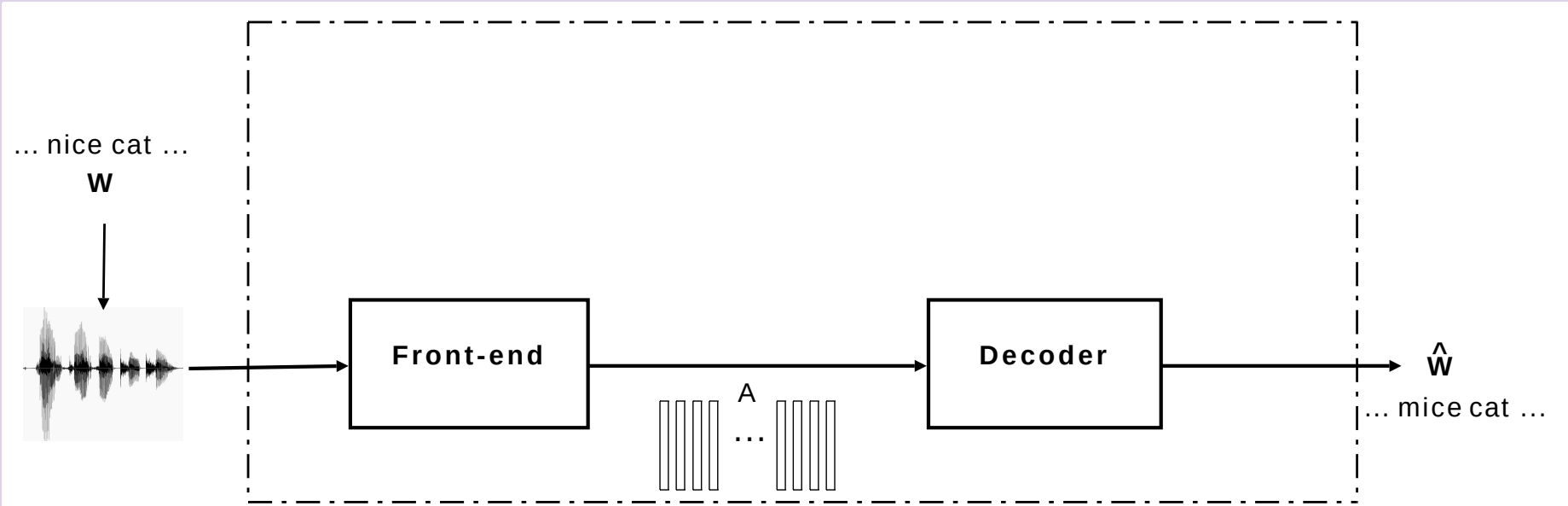
$A = \{a_1 a_2 \ldots a_t\}$, $a_i \in \mathcal{A}$: 39 dimensional MFCC features

$$\hat{W} = \underset{W}{\operatorname{argmax}}\, P(W|A) = \underset{W}{\operatorname{argmax}}\, \frac{P(A|W)P(W)}{P(A)} = \underset{W}{\operatorname{argmax}}\, P(A|W)P(W)$$

Introduction to ASR

Aim:

To automatically produce the transcription of the input speech.



$W = \{w_1 \ldots \text{nice cat} \ldots w_n\}$, $w_i \in \mathcal{V}$: fixed and finite vocabulary $\longrightarrow$ (Out-of-Vocabulary (OOV) words)

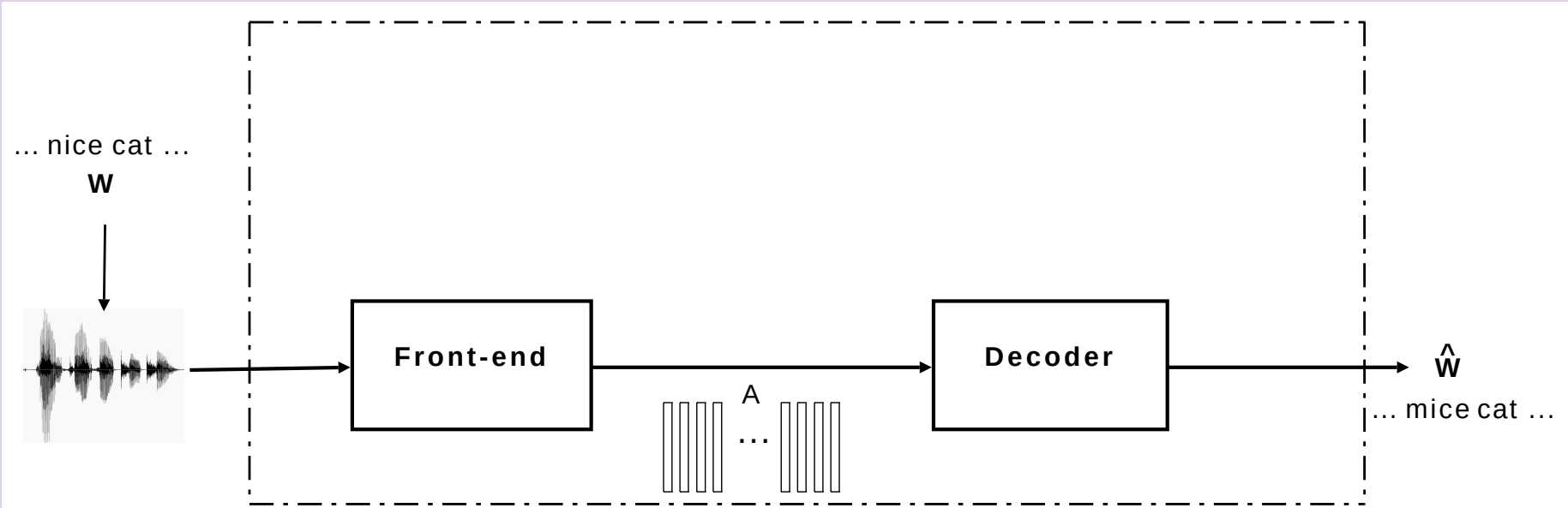
$A = \{a_1 a_2 \ldots a_t\}$, $a_i \in \mathcal{A}$: 39 dimensional MFCC features

$$\hat{W} = \underset{W}{\operatorname{argmax}}\, P(W|A) = \underset{W}{\operatorname{argmax}}\, \frac{P(A|W)P(W)}{P(A)} = \underset{W}{\operatorname{argmax}}\, P(A|W)P(W)$$

Introduction to ASR

Aim:

To automatically produce the transcription of the input speech.



$W = \{w_1 \ldots \text{nice cat} \ldots w_n\}$, $w_i \in \mathcal{V}$: fixed and finite vocabulary $\longrightarrow$ (Out-of-Vocabulary (OOV) words)

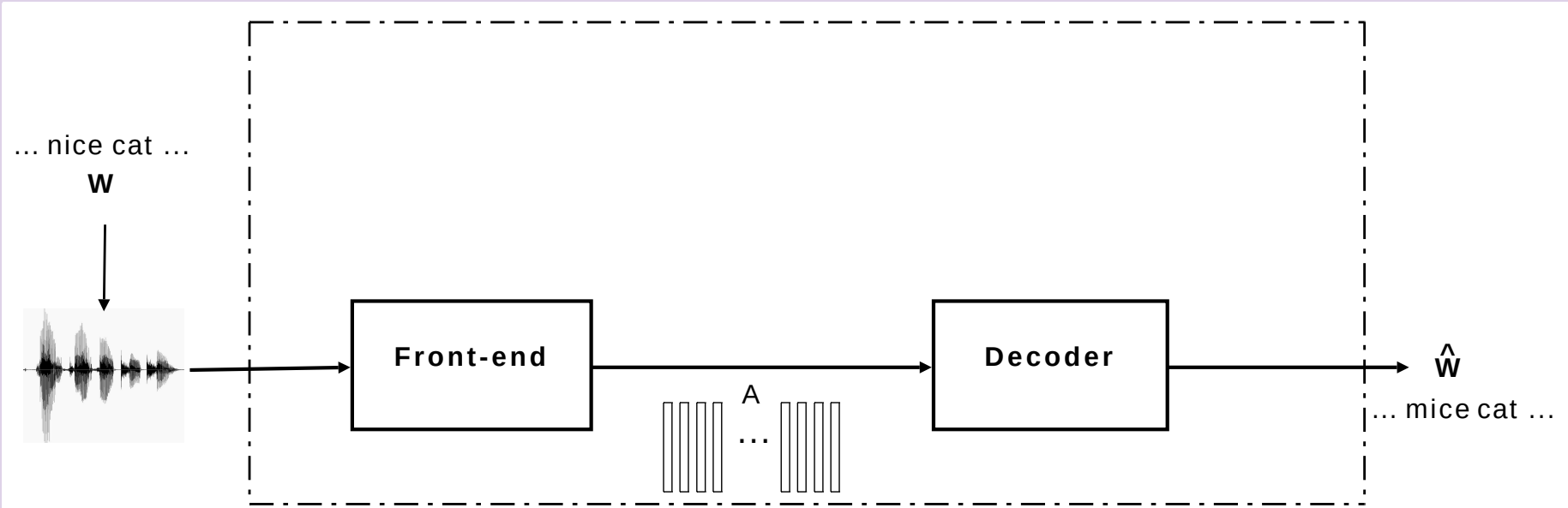
$A = \{a_1 a_2 \ldots a_t\}$, $a_i \in \mathcal{A}$: 39 dimensional MFCC features

$$\hat{W} = \underset{W}{\operatorname{argmax}}\, P(W|A) = \underset{W}{\operatorname{argmax}}\, \frac{P(A|W)P(W)}{P(A)} = \underset{W}{\operatorname{argmax}}\, P(A|W)P(W)$$

Introduction to ASR

Aim:

To automatically produce the transcription of the input speech.



$W = \{w_1 \ldots \text{nice cat} \ldots w_n\}$, $w_i \in \mathcal{V}$: fixed and finite vocabulary $\longrightarrow$ (Out-of-Vocabulary (OOV) words)

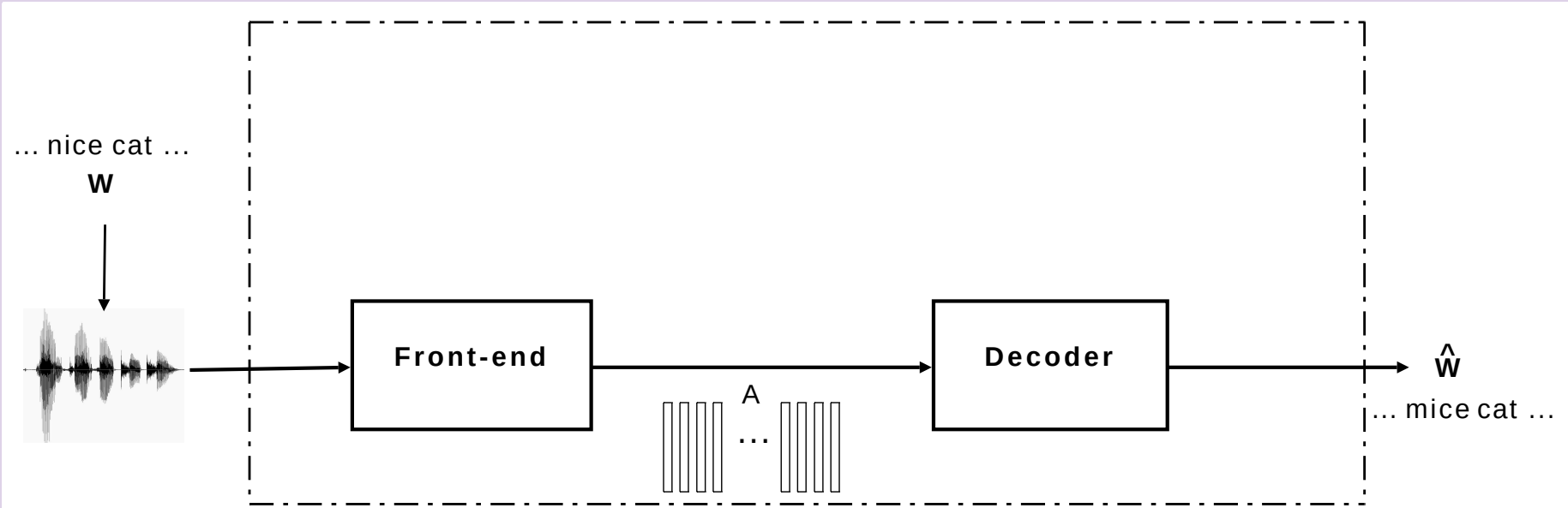
$A = \{a_1 a_2 \ldots a_t\}$, $a_i \in \mathcal{A}$: 39 dimensional MFCC features

$$\hat{W} = \underset{W}{\operatorname{argmax}}\, P(W|A) = \underset{W}{\operatorname{argmax}}\, \frac{P(A|W)P(W)}{P(A)} = \underset{W}{\operatorname{argmax}}\, P(A|W)P(W)$$

Introduction to ASR

Aim:

To automatically produce the transcription of the input speech.



$W = \{w_1 \ldots \text{nice cat} \ldots w_n\}$, $w_i \in \mathcal{V}$: fixed and finite vocabulary $\longrightarrow$ (Out-of-Vocabulary (OOV) words)

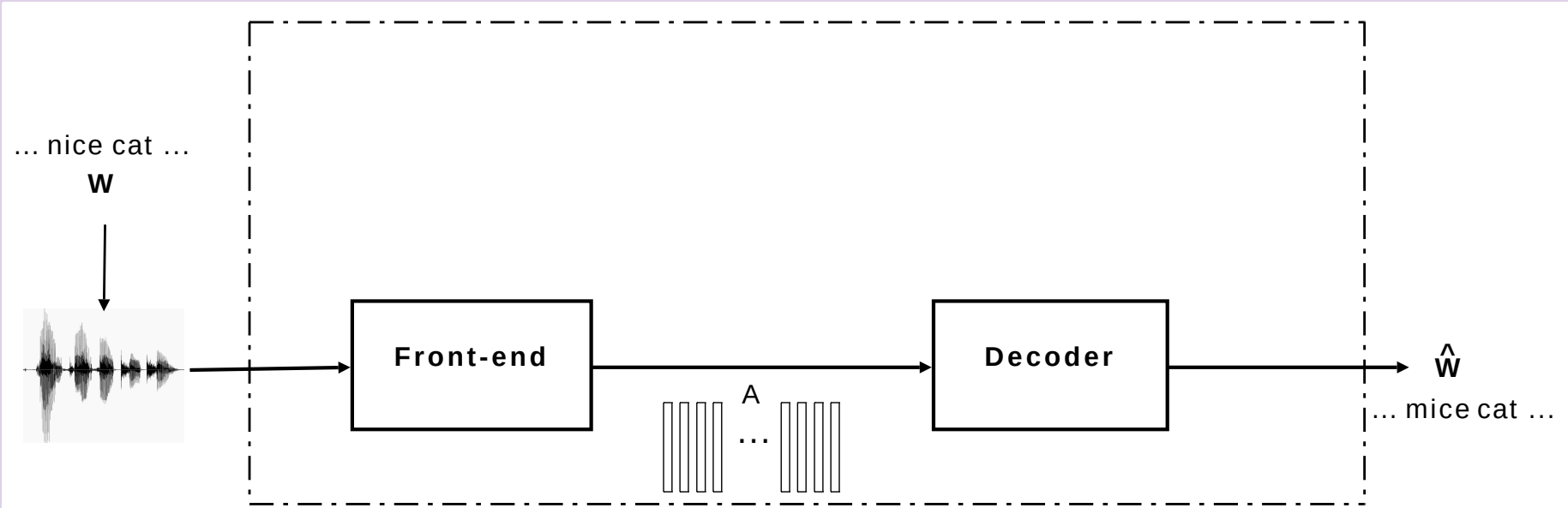
$A = \{a_1 a_2 \ldots a_t\}$, $a_i \in \mathcal{A}$: 39 dimensional MFCC features

$$\hat{W} = \underset{W}{\text{argmax}}\, P(W|A) = \underset{W}{\text{argmax}}\, \frac{P(A|W)P(W)}{P(A)} = \underset{W}{\text{argmax}}\, P(A|W)P(W)$$

Introduction to ASR

Aim:

To automatically produce the transcription of the input speech.



$W = \{w_1 \ldots \text{nice cat} \ldots w_n\}$, $w_i \in \mathcal{V}$: fixed and finite vocabulary $\longrightarrow$ (Out-of-Vocabulary (OOV) words)

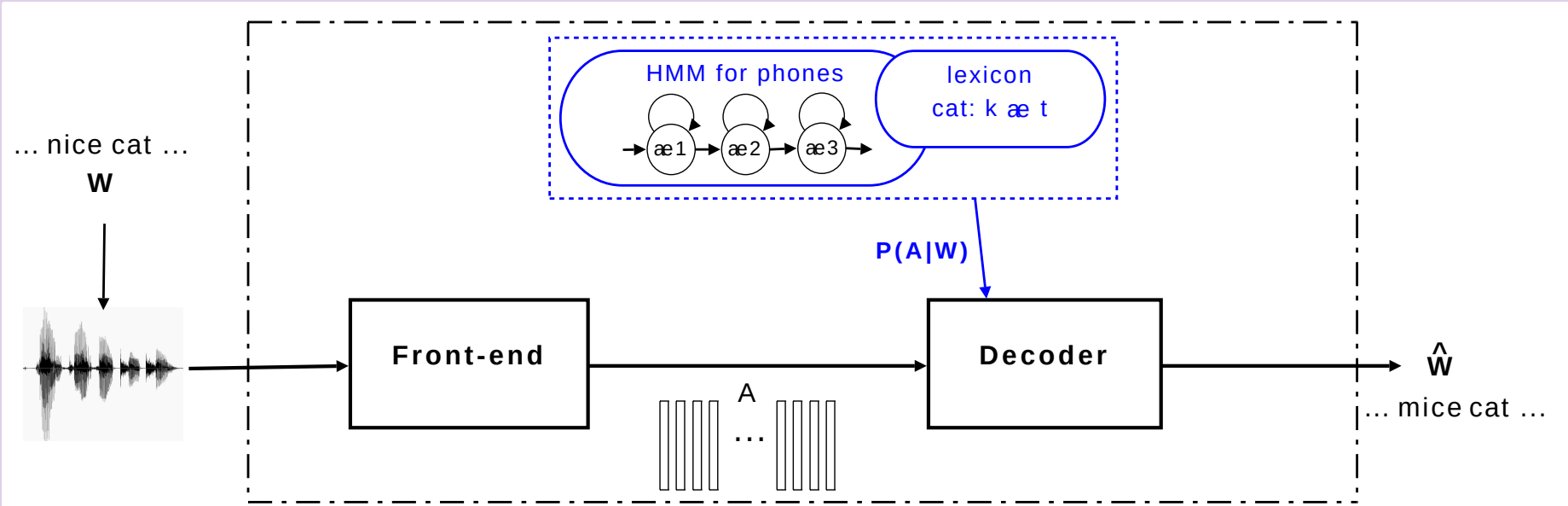
$A = \{a_1 a_2 \ldots a_t\}$, $a_i \in \mathcal{A}$: 39 dimensional MFCC features

$$\hat{W} = \underset{W}{\text{argmax}}\, P(W|A) = \underset{W}{\text{argmax}}\, \frac{P(A|W)P(W)}{P(A)} = \underset{W}{\text{argmax}}\, P(A|W)P(W)$$

Introduction to ASR

Aim:

To automatically produce the transcription of the input speech.



$W = \{w_1 \ldots \text{nice cat} \ldots w_n\}$, $w_i \in \mathcal{V}$: fixed and finite vocabulary $\longrightarrow$ (Out-of-Vocabulary (OOV) words)

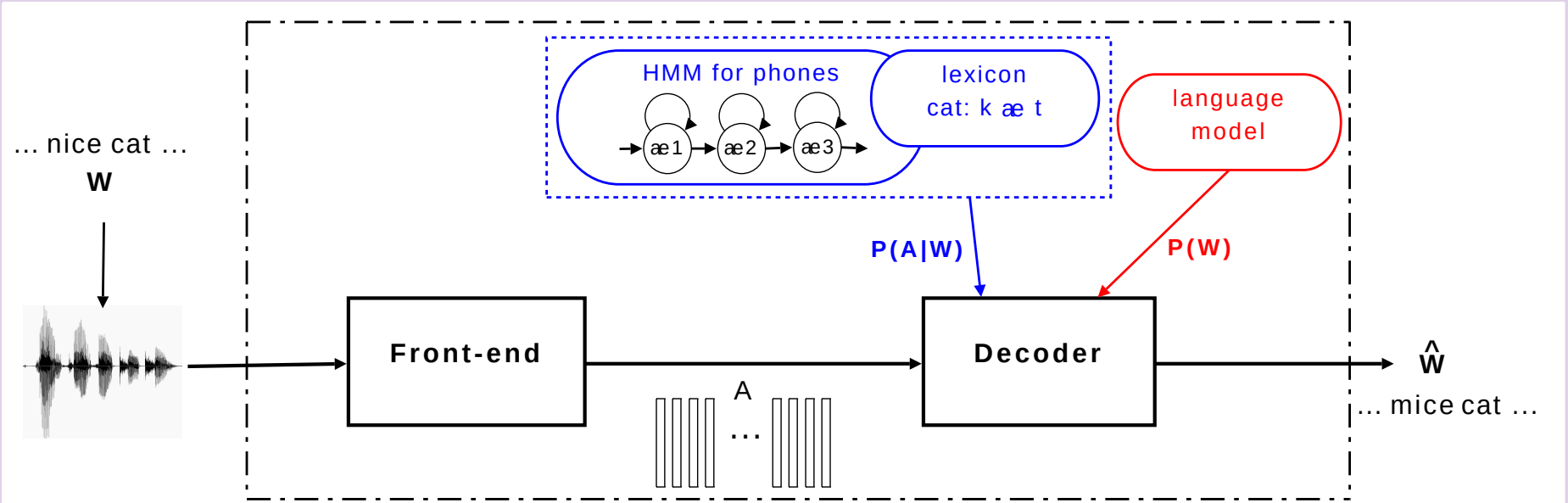
$A = \{a_1 a_2 \ldots a_t\}$, $a_i \in \mathcal{A}$: 39 dimensional MFCC features

$$\hat{W} = \underset{W}{\text{argmax}}\, P(W|A) = \underset{W}{\text{argmax}}\, \frac{P(A|W)P(W)}{P(A)} = \underset{W}{\text{argmax}}\, P(A|W)P(W)$$

Introduction to ASR

Aim:

To automatically produce the transcription of the input speech.



$W = \{w_1 \ldots \text{nice cat} \ldots w_n\}$, $w_i \in \mathcal{V}$: fixed and finite vocabulary $\longrightarrow$ (Out-of-Vocabulary (OOV) words)

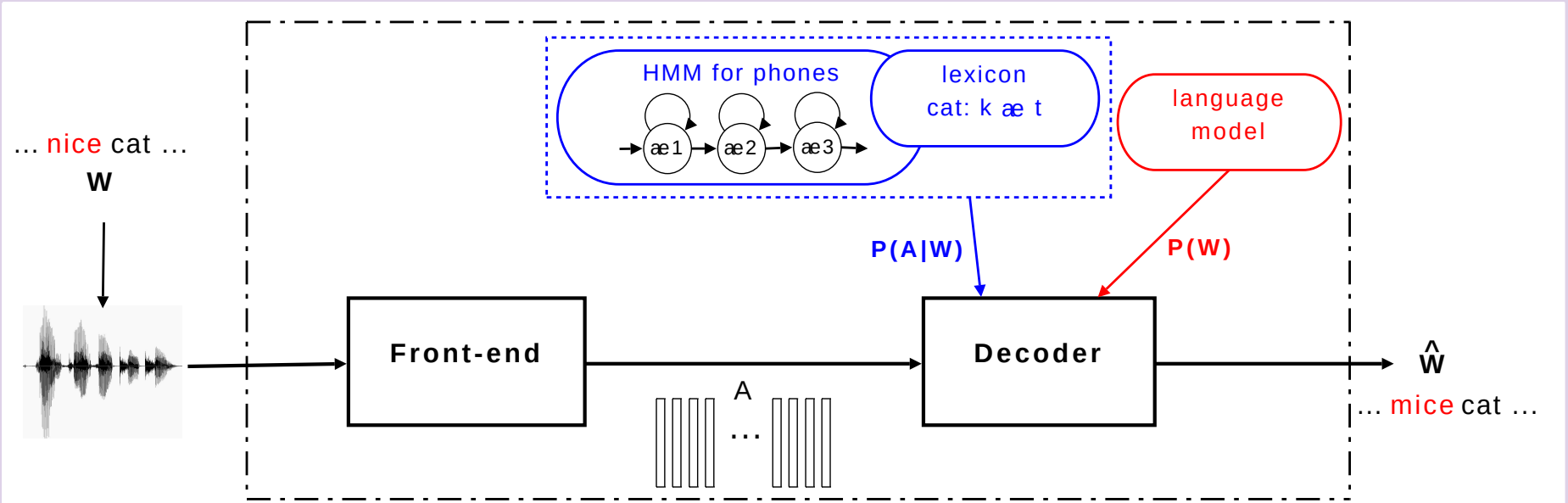
$A = \{a_1 a_2 \ldots a_t\}$, $a_i \in \mathcal{A}$: 39 dimensional MFCC features

$$\hat{W} = \underset{W}{\text{argmax}}\, P(W|A) = \underset{W}{\text{argmax}}\, \frac{P(A|W)P(W)}{P(A)} = \underset{W}{\text{argmax}}\, P(A|W)P(W)$$

Introduction to ASR

Aim:

To automatically produce the transcription of the input speech.



$W = \{w_1 \ldots \text{nice cat} \ldots w_n\}$, $w_i \in \mathcal{V}$: fixed and finite vocabulary $\longrightarrow$ (Out-of-Vocabulary (OOV) words)

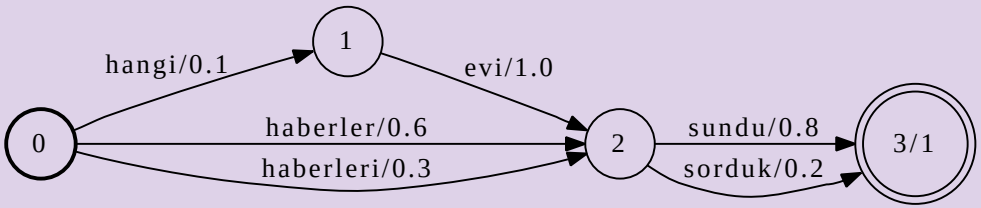
$A = \{a_1 a_2 \ldots a_t\}$, $a_i \in \mathcal{A}$: 39 dimensional MFCC features

$$\hat{W} = \underset{W}{\text{argmax}}\, P(W|A) = \underset{W}{\text{argmax}}\, \frac{P(A|W)P(W)}{P(A)} = \underset{W}{\text{argmax}}\, P(A|W)P(W)$$

ASR Output Example

ASR output for utterance corresponding to "o haberleri sundu"

Lattice Output



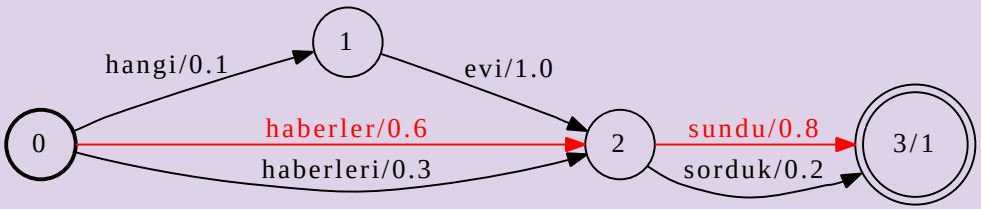
N-best list

1. haberler sundu 0.48
2. haberleri sundu 0.24
3. haberler sorduk 0.12
4. hangi evi sundu 0.08
5. haberleri sorduk 0.06
6. hangi evi sorduk 0.02

ASR Output Example

ASR output for utterance corresponding to "o haberleri sundu"

Lattice Output



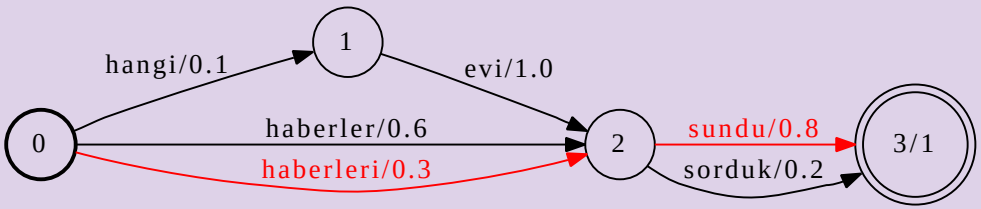
N-best list

1. haberler sundu 0.48 (1-best)
2. haberleri sundu 0.24
3. haberler sorduk 0.12
4. hangi evi sundu 0.08
5. haberleri sorduk 0.06
6. hangi evi sorduk 0.02

ASR Output Example

ASR output for utterance corresponding to "o haberleri sundu"

Lattice Output



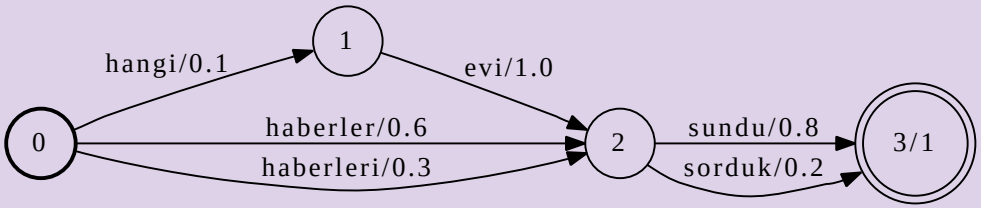
N-best list

1. haberler sundu 0.48
2. haberleri sundu 0.24 (Oracle)
3. haberler sorduk 0.12
4. hangi evi sundu 0.08
5. haberleri sorduk 0.06
6. hangi evi sorduk 0.02

ASR Output Example

ASR output for utterance corresponding to "o haberleri sundu"

Lattice Output



N-best list

1. haberler sundu 0.48
2. haberleri sundu 0.24
3. haberler sorduk 0.12
4. hangi evi sundu 0.08
5. haberleri sorduk 0.06
6. hangi evi sorduk 0.02

Multi-pass ASR systems and rescoring/reranking

- Typically the first pass models are simple and efficient
- The output of the first pass can be rescored or reranked with more accurate but more complicated models.

How to Evaluate ASR?

return to ASR slide

Word Error Rate (WER)

Ratio of total number of errors in an hypothesis string to total number of words in the reference string

$$\text{WER}(\%) = \frac{\#D + \#S + \#I}{\#N} \times 100$$

$\#D$: number of deletions
$\#S$: number of substitutions
$\#I$: number of insertions
$\#N$: number of words in the reference

The number of errors in each hypothesis string is calculated with the minimum edit distance algorithm.

Introduction to SLMs

Statistical Language Models (SLMs)

SLM assigns a prior probability, $P(W)$, to every word string.

n-gram Language Models

$W = \{w_1 \dots w_N\} =$ I saw a nice cat

$$\begin{aligned} P(W) &= P(\text{I saw a nice cat}) \\ &= P(\text{I})P(\text{saw}|\text{I})P(\text{a}|\text{I saw})P(\text{nice}|\text{I saw a})P(\text{cat}|\text{I was a nice}) \end{aligned}$$

n-gram parameters $\approx \mathcal{V}^n$: estimated from text corpus with MLE

Introduction to SLMs

Statistical Language Models (SLMs)

SLM assigns a prior probability, $P(W)$, to every word string.

n-gram Language Models

$W = \{w_1 \dots w_N\} =$ I saw a nice cat

$$\begin{aligned} P(W) &= P(\text{I saw a nice cat}) \\ &= P(\text{I})P(\text{saw}|\text{I})P(\text{a}|\text{I saw})P(\text{nice}|\text{I saw a})P(\text{cat}|\text{I was a nice}) \end{aligned}$$

n-gram parameters $\approx \mathcal{V}^n$: estimated from text corpus with MLE

Introduction to SLMs

Statistical Language Models (SLMs)

SLM assigns a prior probability, $P(W)$, to every word string.

n-gram Language Models

$W = \{w_1 \dots w_N\} =$ I saw a nice cat

$$\begin{aligned} P(W) &= P(\text{I saw a nice cat}) \\ &= P(\text{I})P(\text{saw}|\text{I})P(\text{a}|\text{I saw})P(\text{nice}|\text{I saw a})P(\text{cat}|\text{I was a nice}) \end{aligned}$$

n-gram parameters $\approx \mathcal{V}^n$: estimated from text corpus with MLE

Introduction to SLMs

Statistical Language Models (SLMs)

SLM assigns a prior probability, $P(W)$, to every word string.

n-gram Language Models

$W = \{w_1 \dots w_N\} =$ I saw a nice cat

$$\begin{aligned} P(W) &= P(\text{I saw a nice cat}) \\ &= P(\text{I})P(\text{saw}|\text{I})P(\text{a}|\text{I saw})P(\text{nice}|\text{I saw a})P(\underbrace{\text{cat}}_{\textit{predicted}}|\underbrace{\text{I saw a nice}}_{\text{history}}) \end{aligned}$$

n-gram parameters $\approx \mathcal{V}^n$: estimated from text corpus with MLE

Introduction to SLMs

Statistical Language Models (SLMs)

SLM assigns a prior probability, $P(W)$, to every word string.

n-gram Language Models

$W = \{w_1 \dots w_N\} =$ I saw a nice cat

$$\begin{aligned} P(W) &= P(\text{I saw a nice cat}) \\ &= P(\text{I})P(\text{saw}|\text{I})P(\text{a}|\text{I saw})P(\text{nice}|\text{I saw a})P(\underbrace{\text{cat}}_{\textit{predicted}}|\underbrace{\text{I saw a nice}}_{\text{history}}) \end{aligned}$$

if $n = 2 \rightarrow$ (called 2-gram)

n-gram parameters $\approx \mathcal{V}^n$: estimated from text corpus with MLE

Introduction to SLMs

Statistical Language Models (SLMs)

SLM assigns a prior probability, $P(W)$, to every word string.

n-gram Language Models

$W = \{w_1 \dots w_N\} =$ I saw a nice cat

$$\begin{aligned} P(W) &= P(\text{I saw a nice cat}) \\ &= P(\text{I})P(\text{saw}|\text{I})P(\text{a}|\text{I saw})P(\text{nice}|\text{I saw a})P(\underbrace{\text{cat}}_{\textit{predicted}}|\underbrace{\text{I saw a nice}}_{\text{history}}) \end{aligned}$$

$$\begin{aligned} \text{if } n = 2 &\rightarrow \text{(called 2-gram)} \\ P(W) &\approx P(\text{I})P(\text{saw}|\text{I})P(\text{a}|\text{saw})P(\text{nice}|\text{a})P(\text{cat}|\text{nice}) \end{aligned}$$

n-gram parameters $\approx \mathcal{V}^n$: estimated from text corpus with MLE

Introduction to SLMs

Statistical Language Models (SLMs)

SLM assigns a prior probability, $P(W)$, to every word string.

n-gram Language Models

$W = \{w_1 \dots w_N\} =$ I saw a nice cat

$$\begin{aligned} P(W) &= P(\text{I saw a nice cat}) \\ &= P(\text{I})P(\text{saw}|\text{I})P(\text{a}|\text{I saw})P(\text{nice}|\text{I saw a})P(\underbrace{\text{cat}}_{\textit{predicted}}|\underbrace{\text{I saw a nice}}_{\text{history}}) \end{aligned}$$

$$\begin{aligned} \text{if } n = 2 &\rightarrow \text{(called 2-gram)} \\ P(W) &\approx P(\text{I})P(\text{saw}|\text{I})P(\text{a}|\text{saw})P(\text{nice}|\text{a})P(\text{cat}|\underbrace{\text{nice}}_{\text{history}}) \end{aligned}$$

n-gram parameters $\approx \mathcal{V}^n$: estimated from text corpus with MLE

Introduction to SLMs

Statistical Language Models (SLMs)

SLM assigns a prior probability, $P(W)$, to every word string.

n-gram Language Models

$W = \{w_1 \dots w_N\} =$ I saw a nice cat

$$\begin{aligned} P(W) &= P(\text{I saw a nice cat}) \\ &= P(\text{I})P(\text{saw}|\text{I})P(\text{a}|\text{I saw})P(\text{nice}|\text{I saw a})P(\underbrace{\text{cat}}_{\textit{predicted}}|\underbrace{\text{I saw a nice}}_{\text{history}}) \end{aligned}$$

$$\begin{aligned} \text{if } n = 2 &\rightarrow \text{(called 2-gram)} \\ P(W) &\approx P(\text{I})P(\text{saw}|\text{I})P(\text{a}|\text{saw})P(\text{nice}|\text{a})P(\text{cat}|\underbrace{\text{nice}}_{\text{history}}) \end{aligned}$$

n-gram parameters $\approx \mathcal{V}^n$: estimated from text corpus with MLE

Generative LM vs Discriminative LM

Generative LM: *n*-grams

- *n*-gram probabilities are calculated with MLE.

Discriminative LM:

- A complementary approach to the generative model.
- Trained on utterances with their transcripts to optimize the error rate.
- Advantages:
  - Learning from positive and negative examples (“the of”)
  - Easy to incorporate any feature (morphological, syntactic, semantic)
- Disadvantages:
  - Requires both acoustic data and its transcriptions
  - More expensive to estimate

Generative LM vs Discriminative LM

Generative LM: *n*-grams

- *n*-gram probabilities are calculated with MLE.

Discriminative LM:

- A complementary approach to the generative model.
- Trained on utterances with their transcripts to optimize the error rate.
- Advantages:
  - Learning from positive and negative examples (“the of”)
  - Easy to incorporate any feature (morphological, syntactic, semantic)
- Disadvantages:
  - Requires both acoustic data and its transcriptions
  - More expensive to estimate

DLM – Linear Models

Want to learn a mapping from inputs $\boldsymbol{x} \in \mathcal{X}$ to outputs $\boldsymbol{y} \in \mathcal{Y}$, e.g. utterances to transcription.

Definitions

- Training examples (x_i, y_i) for $i = 1 \ldots N$
- A function **GEN** which enumerates a set of candidates **GEN**$(\boldsymbol{x})$ for an input $\boldsymbol{x}$ (baseline recognizer)
- A **representation** $\boldsymbol{\Phi}$ mapping each $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$ to a feature vector $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^{\boldsymbol{d}}$
- A **parameter vector** $\bar{\alpha} \in \mathbb{R}^{\boldsymbol{d}}$

n-grams as DLM Features

n-gram Features

$\Phi(x, y) =$ the number of times an *n*-gram is seen in y

Word *n*-gram Features

y: language modeling 0.2

$\Phi_1(x, y) =$ number of times "language" is seen in y

$\Phi_2(x, y) =$ number of times "modeling" is seen in y

$\Phi_3(x, y) =$ number of times "language modeling" is seen in y

n-grams as DLM Features

n-gram Features

$\Phi(x, y) =$ the number of times an *n*-gram is seen in y

Word *n*-gram Features

y: `language modeling 0.2`

$\Phi_i(x, y)$ = number of times "`language`" is seen in y

$\Phi_j(x, y)$ = number of times "`modeling`" is seen in y

$\Phi_k(x, y)$ = number of times "`language modeling`" is seen in y

n-grams as DLM Features

n-gram Features

$\Phi(x, y) =$ the number of times an *n*-gram is seen in y

Word *n*-gram Features

y: `language modeling 0.2`

$\Phi_i(x, y)$ = number of times "`language`" is seen in y

$\Phi_j(x, y)$ = number of times "`modeling`" is seen in y

$\Phi_k(x, y)$ = number of times "`language modeling`" is seen in y

$\Phi(x, y) =$

n-grams as DLM Features

n-gram Features

$\Phi(x, y) =$ the number of times an n-gram is seen in y

Word n-gram Features

y: `language modeling` 0.2

$\Phi_i(x, y)$ = number of times "`language`" is seen in y

$\Phi_j(x, y)$ = number of times "`modeling`" is seen in y

$\Phi_k(x, y)$ = number of times "`language modeling`" is seen in y

$$\Phi(x, y) = \begin{pmatrix} \log 0.2 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}_{d \times 1}$$

n-grams as DLM Features

n-gram Features

$\Phi(x, y) =$ the number of times an n-gram is seen in y

Word n-gram Features

y: `language modeling` 0.2

$\Phi_i(x, y)$ = number of times "`language`" is seen in y

$\Phi_j(x, y)$ = number of times "`modeling`" is seen in y

$\Phi_k(x, y)$ = number of times "`language modeling`" is seen in y

$$\Phi(x, y) = \begin{pmatrix} \log 0.2 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}_{d \times 1}$$

n-grams as DLM Features

n-gram Features

$\Phi(x, y) =$ the number of times an n-gram is seen in y

Word n-gram Features

y: `language modeling 0.2`

$\Phi_i(x, y)$ = number of times "`language`" is seen in y

$\Phi_j(x, y)$ = number of times "`modeling`" is seen in y

$\Phi_k(x, y)$ = number of times "`language modeling`" is seen in y

$$\Phi(x, y) = \begin{pmatrix} \log 0.2 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}_{d \times 1}$$

n-grams as DLM Features

n-gram Features

$\Phi(x, y) =$ the number of times an *n*-gram is seen in y

Word *n*-gram Features

y: `language modeling 0.2`

$\Phi_i(x, y)$ = number of times "`language`" is seen in y

$\Phi_j(x, y)$ = number of times "`modeling`" is seen in y

$\Phi_k(x, y)$ = number of times "`language modeling`" is seen in y

$$\Phi(x, y) = \begin{pmatrix} \log 0.2 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}_{d \times 1} \longrightarrow \Phi_0(x, y) : \text{log probability of x and y in the } N\text{-best list}$$

n-grams as DLM Features

n-gram Features

$\Phi(x, y) =$ the number of times an *n*-gram is seen in y

Word *n*-gram Features

y: `language modeling 0.2`

$\Phi_i(x, y)$ = number of times "`language`" is seen in y

$\Phi_j(x, y)$ = number of times "`modeling`" is seen in y

$\Phi_k(x, y)$ = number of times "`language modeling`" is seen in y

$$\Phi(x, y) = \begin{pmatrix} \log 0.2 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}_{d \times 1} \begin{array}{l} \longrightarrow \Phi_0(x, y) : \text{log probability of x and y in the } N\text{-best list} \\ \\ \\ \longrightarrow \Phi_i(x, y) \\ \\ \longrightarrow \Phi_j(x, y) \\ \longrightarrow \Phi_k(x, y) \end{array}$$

DLM – Definitions

- $\Phi(x, y)$: Feature vector
- $\bar{\alpha}$: Feature parameters
- $F(x) = \underset{y \in GEN(x)}{\text{argmax}} \; \Phi(x, y) \cdot \bar{\alpha}$

- Training: Estimating $\bar{\alpha}$
- Decoding: Searching for y that maximizes $\Phi(x, y) \cdot \bar{\alpha}$

DLM – Definitions

- $\Phi(x, y)$: Feature vector $\rightarrow x$: Utterance y: Candidate hypothesis
- $\bar{\alpha}$: Feature parameters
- $F(x) = \underset{y \in GEN(x)}{\text{argmax}} \; \Phi(x, y) \cdot \bar{\alpha}$

- Training: Estimating $\bar{\alpha}$
- Decoding: Searching for y that maximizes $\Phi(x, y) \cdot \bar{\alpha}$

DLM – Definitions

- $\Phi(x, y)$: Feature vector $\rightarrow x$: Utterance y: Candidate hypothesis
- $\bar{\alpha}$: Feature parameters $\rightarrow$ Discriminative training
- $F(x) = \underset{y \in GEN(x)}{\text{argmax}} \; \Phi(x, y) \cdot \bar{\alpha}$

- Training: Estimating $\bar{\alpha}$
- Decoding: Searching for y that maximizes $\Phi(x, y) \cdot \bar{\alpha}$

DLM – Definitions

- $\Phi(x, y)$: Feature vector $\rightarrow x$: Utterance y: Candidate hypothesis
- $\bar{\alpha}$: Feature parameters $\rightarrow$ Discriminative training
- $F(x) = \underset{y \in GEN(x)}{\operatorname{argmax}} \Phi(x, y) \cdot \bar{\alpha} \rightarrow GEN(x)$: Hypotheses in N-best list of x

- Training: Estimating $\bar{\alpha}$
- Decoding: Searching for y that maximizes $\Phi(x, y) \cdot \bar{\alpha}$

DLM – Definitions

- $\Phi(x, y)$: Feature vector $\rightarrow x$: Utterance y: Candidate hypothesis
- $\bar{\alpha}$: Feature parameters $\rightarrow$ Discriminative training
- $F(x) = \underset{y \in GEN(x)}{\operatorname{argmax}} \Phi(x, y) \cdot \bar{\alpha} \rightarrow GEN(x)$: Hypotheses in N-best list of x

- Training: Estimating $\bar{\alpha}$
- Decoding: Searching for y that maximizes $\Phi(x, y) \cdot \bar{\alpha}$

DLM – Definitions

- $\Phi(x, y)$: Feature vector $\rightarrow x$: Utterance y: Candidate hypothesis
- $\bar{\alpha}$: Feature parameters $\rightarrow$ Discriminative training
- $F(x) = \underset{y \in GEN(x)}{\operatorname{argmax}} \Phi(x, y) \cdot \bar{\alpha} \rightarrow GEN(x)$: Hypotheses in N-best list of x

- Training: Estimating $\bar{\alpha} \rightarrow$ perceptron algorithm
- Decoding: Searching for y that maximizes $\Phi(x, y) \cdot \bar{\alpha}$

Perceptron Algorithm

Inputs: Training examples (x_i, y_i) for $i = 1 \ldots N$

Initilization: Set $\bar{\alpha} = 0$

Algorithm:

For $t = 1 \ldots T$

For $i = 1 \ldots N$

Calculate $z_i = \underset{z \in GEN(x_i)}{\text{argmax}} \langle \Phi(x_i, z), \bar{\alpha} \rangle$

If$(z_i \neq y_i)$ then $\bar{\alpha} = \bar{\alpha} + \Phi(x_i, y_i) - \Phi(x_i, z_i)$

Output: Parameters $\bar{\alpha}$

Averaged Perceptron Algorithm

Inputs: Training examples (x_i, y_i) for $i = 1 \ldots N$

Initilization: Set $\bar{\alpha} = 0$

Algorithm:

For $t = 1 \ldots T$

For $i = 1 \ldots N$

Calculate $z_i = \underset{z \in GEN(x_i)}{\text{argmax}} \langle \Phi(x_i, z), \bar{\alpha} \rangle$

If$(z_i \neq y_i)$ then $\bar{\alpha} = \bar{\alpha} + \Phi(x_i, y_i) - \Phi(x_i, z_i)$; $\bar{\alpha}_{avg} = \bar{\alpha}_{avg} + \bar{\alpha}$

Output: Parameters $\bar{\alpha}_{avg}/NT$

Averaged Perceptron Discussion

- Important to control training data
- Trigrams, lattices better, but not that much
- Using oracle best-path makes a difference

Global Conditional Log-linear Models (GCLM)

Define a conditional distribution over the members of **GEN**(x) for a given input x:

$$p_{\bar{\alpha}}(y|x) = \frac{1}{Z(x,\bar{\alpha})} \exp\left(\Phi(x,y)\cdot\bar{\alpha}\right)$$

where

$$Z(x,\bar{\alpha}) = \sum_{y\in\textbf{GEN}(x)} \exp\left(\Phi(x,y)\cdot\bar{\alpha}\right)$$

Note that

$$\underset{y\in\textbf{GEN}(x)}{\operatorname{argmax}}\ p_{\bar{\alpha}}(y|x) = \underset{y\in\textbf{GEN}(x)}{\operatorname{argmax}}\ \Phi(x,y)\cdot\bar{\alpha}$$

so that for testing this is equivalent to linear models.

GCLM Estimation

Objective Function

Choose $\bar{\alpha}$ to maximize the conditional log-likelihood of the training data:

$$LL(\bar{\alpha}) = \sum_{i=1}^{N} \log p_{\bar{\alpha}}(y_i|x_i) = \sum_{i=1}^{N} [\Phi(x_i, y_i) \cdot \bar{\alpha} - \log Z(x_i, \bar{\alpha})]$$

Use a zero-mean Gaussian prior on the parameters resulting in the regularized objective function:

$$LL_R(\bar{\alpha}) = \sum_{i=1}^{N} [\Phi(x_i, y_i) \cdot \bar{\alpha} - \log Z(x_i, \bar{\alpha})] - \frac{||\bar{\alpha}||^2}{2\sigma^2}$$

The value σ is typically estimated using held-out data.

GCLM Estimation

Optimization

- The objective function is convex and there is a globally optimum solution.
- A general optimization method (e.g. *limited memory variable metric* method) can be used to optimize LL_R.
- The optimizer needs the function value and the gradient:

$$\frac{\partial LL_R}{\partial \alpha_s} = \sum_{i=1}^{N} \left[\Phi_s(x_i, y_i) - \sum_{y \in \mathbf{GEN}(x_i)} p_{\bar{\alpha}}(y|x_i) \Phi_s(x_i, y) \right] - \frac{\alpha_s}{\sigma^2}$$

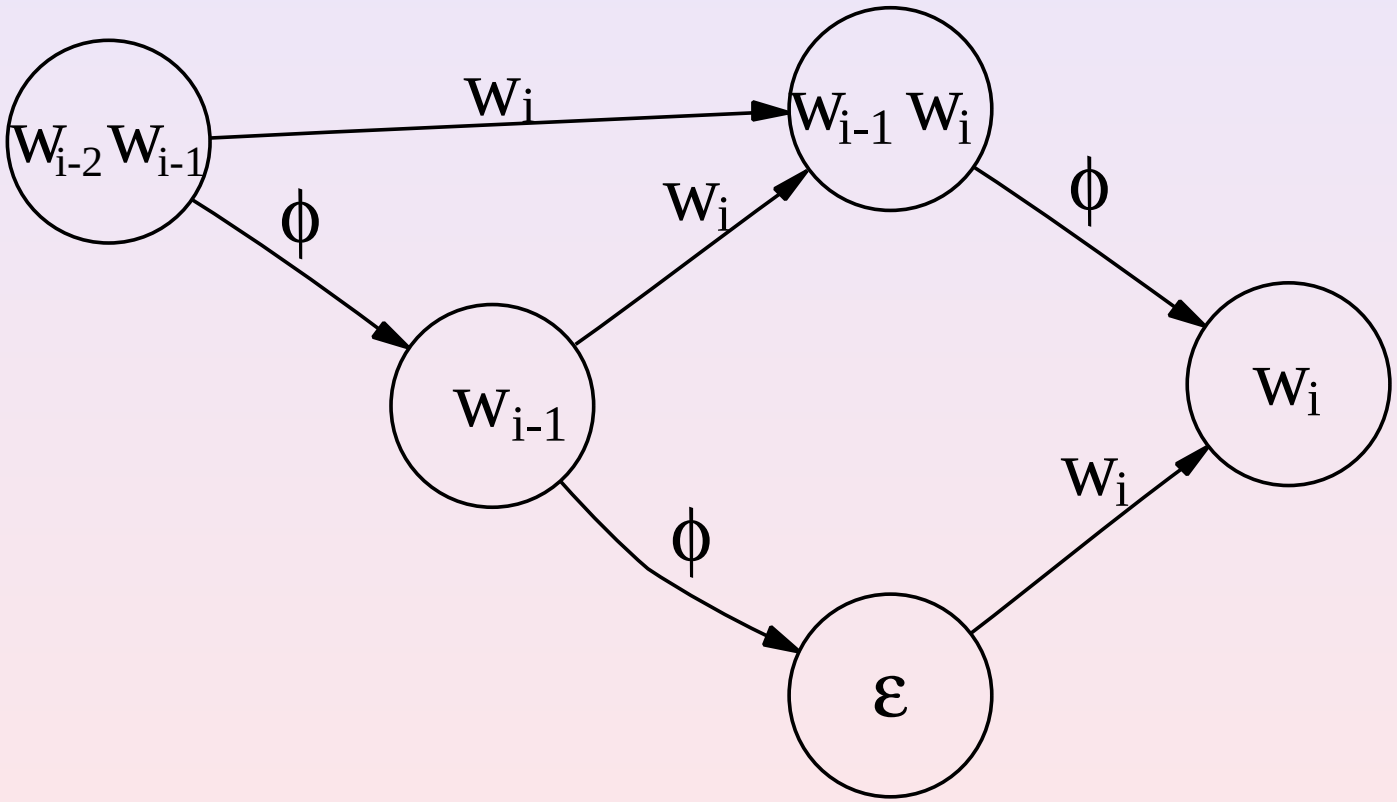
GCLM Discussion

- For training, the most expensive part is computing the objective function and its gradient. Fortunately this can be parallelized.
- GCLM parameter estimation yields improvement over the perceptron algorithm.
- Using the feature set selected by the perceptron algorithm is more efficient.

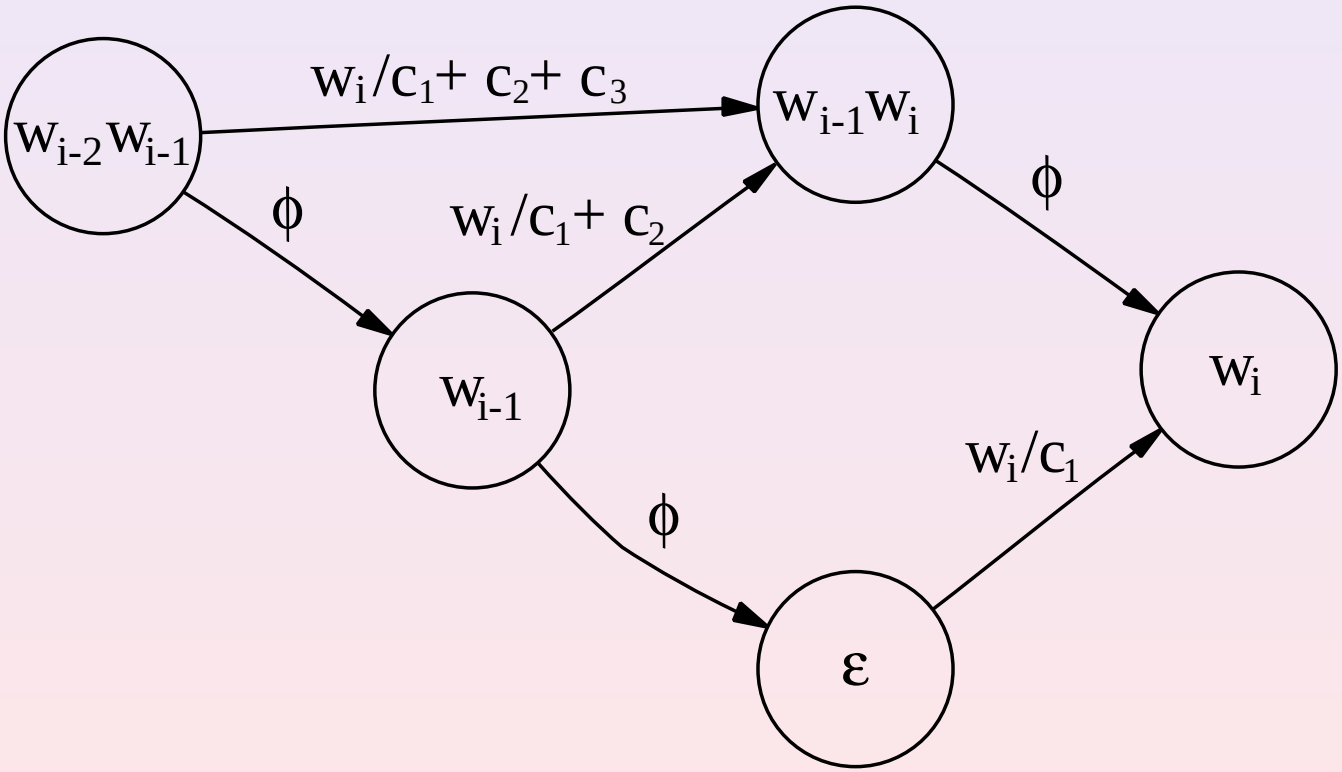
Implementation Using Weighted Finite State Transducers (WFSTs)

- The components of an ASR system can be represented as WFSTs
- These WFSTs can be composed into a single ASR search network that can be optimized using WFST operations (HoCoLoG)
- In particular an n-gram LM can be efficiently represented as a WFST
- Same can be done for DLMs
- Counting can be implemented as WFST composition
- It is also possible to disciminatively train the weights of the WFST representing the ASR search network

WFST encoding of backoff n-gram



WFST encoding of n-gram linear models



Advanced Techniques for DLM

1. Better feature extraction
2. Better parameter estimation
3. Being able to train DLMs using only text data [JHU WS11]
4. Being able to train DLMs using only acoustic data

DLM Features

- *n*-gram features
- Sub-lexical features
- POS features
- Syntactic parse features
- Topic related features
- Duration features

Other Learning Algorithms for DLM

- Minimum Classification Error (MCE)
- Boosting
- Ranking SVM
- Loss sensitive perceptron
- Perceptron with uneven margins (PCUM)
- Minimum Error Rate Training (MERT)
- Minimum Sample Risk (MSR)
- Weighted GCLM (WGCLM)
- Round Robin Duel Discrimination (R2D2)
- ...

Ideas behind some DLM variants

- Consider all hypotheses (as in GCLM) in the objective function
- Introduce a weight (e.g. error rate) for each hypothesis in the objective function
- Directly optimize the error rate
- Maximize the margin explicitly
- Pairwise discrimination, equivalent to (re)ranking

WER Sensitive Perceptron Algorithm

Definition: Edit Distance Difference

$$\Delta(y_i, z_i) = \text{Edit\_Distance}(z_i) - \text{Edit\_Distance}(y_i)$$

Inputs: Training examples (x_i, y_i) for $i = 1 \ldots N$

Initilization: Set $\bar{\alpha} = 0$

Algorithm:

For $t = 1 \ldots T$

For $i = 1 \ldots N$

Calculate $z_i = \underset{z \in GEN(x_i)}{\text{argmax}} \langle \Phi(x_i, z), \bar{\alpha} \rangle$

If $(z_i \neq y_i)$ then $\bar{\alpha} = \bar{\alpha} + \Delta(y_i, z_i)(\Phi(x_i, y_i) - \Phi(x_i, z_i))$;

Output: Parameters $\bar{\alpha}$

WER Sensitive Perceptron Loss Function

For the perceptron, we have an error when the best hypothesis is not the oracle.

Perceptron Loss Function

$$\mathcal{L}_{\text{SER}} = \sum_i [\![\bar{\alpha} \cdot \Phi(x_i, z_i) - \bar{\alpha} \cdot \Phi(x_i, y_i)]\!]$$

However, the actual error is the difference between the number of word errors.

WER Sensitive Perceptron Loss Function

$$\mathcal{L}_{\text{WER}} = \sum_i \Delta(y_i, z_i)[\![\bar{\alpha} \cdot \Phi(x_i, z_i) - \bar{\alpha} \cdot \Phi(x_i, y_i)]\!]$$

Conclusion

- Discriminative n-gram modeling yields substantial improvements over baseline system
- Efficient to use, very little real-time cost
- Improvements carry over to utterance classification
- Improvements are additive to AM adaptation techniques
- Has worked in every domain and language investigated

The Acoustic Model

$$P(A|W) = \sum_S P(A|S) \sum_q P(S|q) \sum_B P(q|B)P(B|W)$$

- $P(B|W)$: Pronunciation model
- $P(q|B)$: Context dependency model
- $P(S|q)$: Duration model
- $P(A|S)$: Output model

$$P(A|S) = \prod_{t=1}^{T} P(a_t|s_t) = \prod_{t=1}^{T} \sum_{i \in G(s_t)} c_i \mathcal{N}(a_t; \mu_i, \sigma_i^2)$$

- The parameters $\theta = \{c_i, \mu_i, \sigma_i^2\}$ are estimated to maximize the likelihood $P(A|W)$.

Expectation Maximization Algorithm

- EM is a technique for MLE for incomplete data problems.
- Observed data: $\{A, W\}$
- Hidden data: $\{S\}$
- Complete data: $\{A, W, S\}$
- Starting with initial parameters $\theta^{(0)}$ iterate the following steps:
  - E-Step: Compute the auxiliary function Q

$$Q(\theta; \theta^{(k)}) = E\left[\log P_\theta(A, W, S)|A, W; \theta^{(k)}\right]$$

  - M-Step: Re-estimate parameters θ

$$\theta^{(k+1)} = \arg\max_\theta Q(\theta; \theta^{(k)})$$

EM iterations increase the likelihood

- Log Likelihood : $L(\theta) = \log P_\theta(A, W) = \log \sum_S P_\theta(A, W, S)$
- $Q(\theta; \theta^{(k)}) = \sum_S P_{\theta^{(k)}}(S|A, W) \log P_\theta(A, W, S)$
- Claim : if $Q(\theta; \theta^{(k)}) \geq Q(\theta^{(k)}; \theta^{(k)})$ then $L(\theta) \geq L(\theta^{(k)})$
- Proof :

$$\begin{aligned}
L(\theta) - L(\theta^{(k)}) &= \log \sum_S P_\theta(A, W, S) - \log P_{\theta^{(k)}}(A, W) \\
&= \log \sum_S P_{\theta^{(k)}}(S|A, W) \frac{P_\theta(A, W, S)}{P_{\theta^{(k)}}(A, W, S)} \\
&\geq \sum_S P_{\theta^{(k)}}(S|A, W) \log \frac{P_\theta(A, W, S)}{P_{\theta^{(k)}}(A, W, S)} \\
&= Q(\theta; \theta^{(k)}) - Q(\theta^{(k)}; \theta^{(k)})
\end{aligned}$$

ML parameter update formulas for Gaussians

$$\theta^{(k+1)} = \arg\max_{\theta} \sum_{S} P_{\theta^{(k)}}(S|A,W) \log P_{\theta}(A,W,S)$$

- $P_\theta(A|S) = \prod_t P_\theta(a_t|s_t) = \prod_t \mathcal{N}(a_t; \mu_{s_t}, \sigma^2_{s_t})$
- $\theta = \{\mu_s, \sigma^2_s\}$
- $\gamma_s(t) = P_{\theta^{(k)}}(s_t = s|A,W) = \sum_{S:s_t=s} P_{\theta^{(k)}}(S|A,W)$

$$\hat{\mu}_s = \frac{\sum_t \gamma_s(t) a_t}{\sum_t \gamma_s(t)}$$

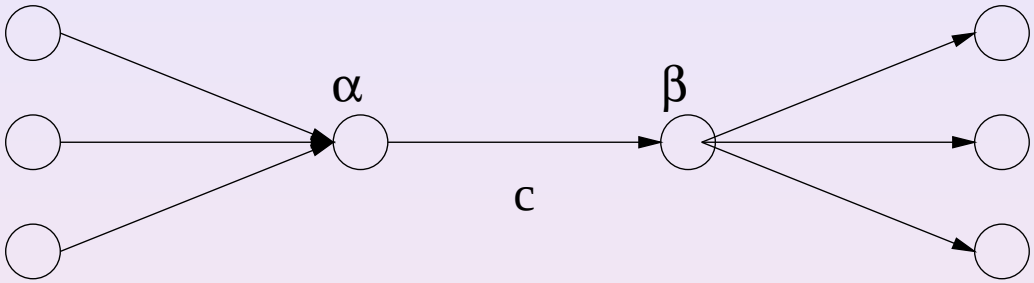
$$\hat{\sigma}^2_s = \frac{\sum_t \gamma_s(t) a_t^2}{\sum_t \gamma_s(t)} - \hat{\mu}^2_s$$

Computing the Posterior Probabilities $\gamma_s(t)$

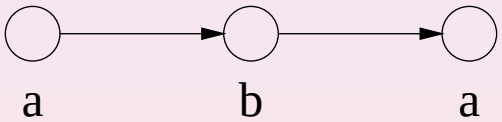
$$\gamma_s(t) = \frac{\sum_{S \in \mathcal{S}: s_t = s} P_{\theta^{(k)}}(A, W, S)}{\sum_{S \in \mathcal{S}} P_{\theta^{(k)}}(A, W, S)}$$

- For HMMs an efficient algorithm exists (Forward-Backward).
- Baum-Welch Training: use all possible state sequences $\mathcal{S}$. Accurate but expensive.
- Viterbi Training: use the most likely state sequence $\mathcal{S}_1$. Approximate but cheaper.
- N-Best Training: use the most likely N state sequences $\mathcal{S}_N$. More accurate than Viterbi but inefficient.
- Lattice Training: use the most likely state sequences $\mathcal{S}_L$. More efficient than N-Best.

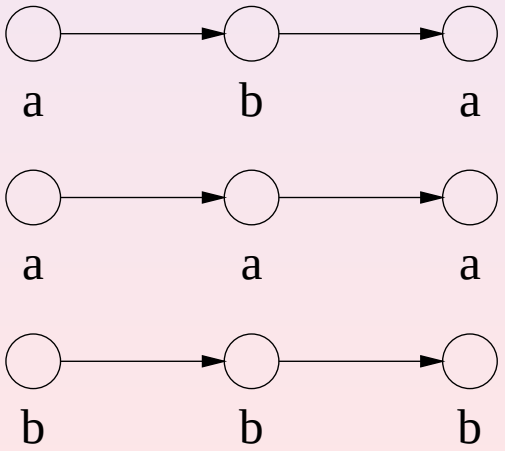
Forward-Backward



Viterbi Training



N-Best Training



Discriminative Techniques for Parameter Estimation

- The ultimate goal is to recognize (test) utterances correctly.
- MLE training tries to maximize the joint likelihood of acoustics and correct word sequences for the training set. This might increase the likelihood of other word sequences as well.
- Another reasonable alternative is to maximize the correct recognition rate of the training set. These discriminative training schemes consider not only the correct word sequence but also the competing word sequences.
- There are at least two major difficulties with discriminative approaches: computation, generalization.

Objective Functions for Parameter Estimation

- (Joint) Maximum Likelihood (ML) :

$$\hat{\Theta} = \arg\max_{\Theta} P_{\Theta}(A, W) = \arg\max_{\Theta} \sum_{n=1}^{N} \log P_{\Theta}(A_n, W_n) \quad (1)$$

- Conditional Maximum Likelihood (CML) :

$$\hat{\Theta} = \arg\max_{\Theta} P_{\Theta}(W|A) = \arg\max_{\Theta} \sum_{n=1}^{N} \log \frac{P_{\Theta}(A_n, W_n)}{P_{\Theta}(A_n)} \quad (2)$$

- Maximum Mutual Information (MMI) :

$$\hat{\Theta} = \arg\max_{\Theta} I_{\Theta}(A, W) \approx \arg\max_{\Theta} \sum_{n=1}^{N} \log \frac{P_{\Theta}(A_n, W_n)}{P_{\Theta}(A_n)P(W_n)} \quad (3)$$

A Unifying Approach for a Class of Discriminative Training Criteria

$$F_D(\theta) = \sum_{n=1}^{N} f\left(\log \frac{[P_\theta(A_n|W_n)P(W_n)]^\alpha}{\sum_{W'}[P_\theta(A_n|W')P(W')]^\alpha}\right)$$

- For CML/MMI: $\alpha = 1$ and $f(x) = x$.
- For (an equivalent version of) MCE: α controls the contribution of competing hypotheses and

$$f(x) = \frac{-1}{1 + e^{\beta x}}$$

  Also the sum over W' excludes W_n.
- Hopeless utterances are weighted down in MCE.

Parameter Estimation for CML/MMIE

$$\begin{aligned}\hat{\Theta} &= \arg\max_{\Theta} \log P_{\Theta}(W|A) \\ &= \arg\max_{\Theta} \log \frac{P_{\Theta}(A|W)P(W)}{\sum_{W} P_{\Theta}(A|W)P(W)}\end{aligned}$$

- For LVCSR, sum over all word sequences is prohibitively expensive; instead sum over only the most likely word sequences using a word lattice.
- For optimization use:
  1. Gradient Descent Methods
  2. Extended Baum-Welch (EBW) Algorithm (Gopalakrishnan et al, 1989; Normandin 1991)

The Extended Baum-Welch Algorithm (EBW)

$$\hat{\Theta} = \arg\max_{\Theta} \log \frac{\sum_S P_\Theta(A|S)P(S|W)P(W)}{\sum_{W' \in \mathcal{W}} \sum_{S'} P_\Theta(A|S')P(S'|W')P(W')}$$

- $\gamma_s(t) = P_{\theta^{(k)}}(s_t = s|A, W)$
- $\gamma'_s(t) = P_{\theta^{(k)}}(s_t = s|A)$
- Auxiliary function $Q_D(\theta; \theta^{(k)})$ (similar to Q)

$$\begin{aligned} Q_D(\theta; \theta^{(k)}) &= \sum_s \sum_t [\gamma_s(t) - \gamma'_s(t)] \log P_\theta(a_t|s_t) \\ &\quad + \sum_s D_s \int da P_{\theta^{(k)}}(a|s) \log P_\theta(a|s) \end{aligned}$$

- Convergence requires large enough $\{D_s\}$.

CML parameter update formulae for Gaussians

$$\theta^{(k+1)} = \arg\max_{\theta} Q_D(\theta; \theta^{(k)})$$

- $\theta = \{\mu_s, \sigma_s^2\}$ and $P_\theta(a_t|s_t) = \mathcal{N}(a_t; \mu_{s_t}, \sigma_{s_t}^2)$
- $\gamma_s(t) = \frac{\sum_{S:s_t=s} P_{\theta^{(k)}}(S,A,W)}{\sum_{S'} P_{\theta^{(k)}}(S',A,W)}$
- $\gamma'_s(t) = \frac{\sum_{S:s_t=s} \sum_{W' \in \mathcal{W}_S} P_{\theta^{(k)}}(S,A,W')}{\sum_{S'} \sum_{W'' \in \mathcal{W}_{S'}} P_{\theta^{(k)}}(S',A,W'')}$

$$\hat{\mu}_s = \frac{\sum_t [\gamma_s(t) - \gamma'_s(t)] a_t + D_s \mu_s}{\sum_t [\gamma_s(t) - \gamma'_s(t)] + D_s}$$

$$\hat{\sigma}_s^2 = \frac{\sum_t [\gamma_s(t) - \gamma'_s(t)] a_t^2 + D_s(\mu_s^2 + \sigma_s^2)}{\sum_t [\gamma_s(t) - \gamma'_s(t)] + D_s} - \hat{\mu}_s^2$$

Tricks of the Trade

- During recognition a trigram LM $[P(W) = \prod_i P(w_i|w_{i-1}w_{i-2})]$ is used, however using a unigram LM $[P(W) = \prod_i P(w_i)]$ during training gives better results.
- During recognition a scale factor λ is used.

$$\hat{W} = \arg\max_W P(A|W)P(W)^{\lambda} = \arg\max_W P(A|W)^{\frac{1}{\lambda}}P(W)$$

During training the acoustic model should be scaled down with the same factor.

$$P(A,W) = P(A|W)^{\frac{1}{\lambda}}P(W)$$

- D_s are chosen to guarantee positive variances.

Summary for MMIE/CMLE of the Acoustic Model

1. Generate word lattices representing the alternate hypotheses using existing acoustic and language models.
2. Generate state level segmentation of the truth and the alternate hypotheses based on the word lattices.
   In other words, compute $P_\theta(S, A, W)$.
3. Compute the posterior probabilities ($\gamma_s(t)$ and $\gamma'_s(t)$).
4. Accumulate sufficient statistics ($\gamma_s(t), \gamma_s(t)a_t, \gamma_s(t)a_t^2$).
5. Determine $\{D_s\}$.
6. Update model parameters Θ.
7. If not done iterating go to Step 2 (or 1).