Unit 3: Natural Language Learning

Part 1: Unsupervised Learning
(EM, forward-backward, inside-outside)

Fall 2011
Liang Huang (lhuang@isi.edu)
Review of Noisy-Channel Model

Application
- Machine Translation
- Optical Character Recognition (OCR)
- Part Of Speech (POS) tagging
- Speech recognition

Input
- $L_1$ word sequences
- actual text
- POS tag sequences
- word sequences

Output
- $L_2$ word sequences
- text with mistakes
- English words
- speech signal

$p(i)$
- $p(L_1)$ in a language model
- prob of language text
- prob of POS sequences

$p(o|i)$
- translation model
- model of OCR errors
- $p(w|t)$
- acoustic model
Example 1: Part-of-Speech Tagging

\[
P(t \ldots t | w \ldots w)
\sim P(t \ldots t) \cdot P(w \ldots w | t \ldots t)
\sim P(t_1) \cdot P(t_2 | t_1) \ldots P(t_n | t_{n-1}) \cdot P(w_1 | t_1) \ldots P(w_n | t_n)
\]

- use tag bigram as a language model
- channel model is context-indep.

\[
\text{Source: } t_i / P(t_i) \rightarrow t_2 / P(t_2 | t_1) \rightarrow \text{end} \rightarrow \emptyset
\]

\[
\text{Channel: } t_i : w_4 / P(w_4 | t_i) \rightarrow t_6 : w_9 / P(w_9 | t_6)
\]

\[
\text{New string: } \text{they can fish}
\]
Ideal vs. Available Data

WFSA $\xrightarrow{t \ldots t} \xrightarrow{w \ldots w}$ WFST

Monday, November 7, 2011
Cryptography

1. Generate $e_1, \ldots, e_n$ by $p(e_k | e_{k-1})$
2. For $i = 1 \to n$
   - Output $c_i$ by $p(c_i | e_i)$
Ideal vs. Available Data

Cryptography
1. generate $e_1, \ldots, e_n$ by $p(e_k | e_{k-1})$
2. for $i = 1$ to $n$
   output $c_i$ by $p(c_i | e_i)$

Spelling-to-Sound
1. generate $p_0, \ldots, p_n$
2. transform into $c_1, \ldots, c_m$ by WFSA
Ideal vs. Available Data

Cryptography
1. Generate $e_1, \ldots, e_n$ by $P(e_k | e_{k-1})$
2. For $i = 1$ to $n$
   - Output $c_i$ by $P(c_i | e_i)$

Spelling-to-sound
1. Generate phon, ..., phon
2. Transform into $c_1, \ldots, c_m$ by WFST

MT
1. Generate $e_1, \ldots, e_n$ by $P(e_k | e_{k-1})$
2. For $i = 1$ to $n$
   - Generate $f_i$ by $P(f_i | e_i)$
3. Permute all $f_i$ by $\frac{1}{n!}$
Ideal vs. Available Data

**HW3: ideal**

EY B AH L  1 2 3 4 4

AH B AW T  1 2 3 3 4 4

AH L ER T  1 2 3 3 4 4

EY S  1 1 2 2

**HW5: realistic**

EY B AH L  1 2 3 4 4

AH B AW T  1 2 3 3 4 4

AH L ER T  1 2 3 3 4 4

EY S  1 1 2 2
Ideal vs. Available Data

WFSA \rightarrow t \cdots t \rightarrow WFST \rightarrow w \cdots w
CRYPTOGRAPHY

1. generate $e_1, \ldots, e_n$ by $P(e_k | e_{k-1})$
2. for $i = 1$ to $n$
   output $c_i$ by $P(c_i | e_i)$
**Cryptography**
1. generate $e_1, \ldots, e_n$ by $P(e_k|e_{k-1})$
2. for $i = 1$ to $n$
   - output $c_i$ by $P(c_i|e_i)$

**Part of Speech**
1. generate $t_1, \ldots, t_n$ by $P(t_k|t_{k-1})$
2. for $i = 1$ to $n$
   - output $w_i$ by $P(w_i|t_i)$
Cryptography
1. generate \( e_1, \ldots, e_n \) by \( P(e_k | e_{k-1}) \)
2. for \( i = 1 \) to \( n \)
   - output \( c_i \) by \( P(c_i | e_i) \)

Part of Speech
1. generate \( t_1, \ldots, t_n \) by \( P(t_k | t_{k-1}) \)
2. for \( i = 1 \) to \( n \)
   - output \( w_i \) by \( P(w_i | t_i) \)

Spelling - To-Sound
1. generate \( \text{pho}, \ldots, \text{phon} \)
2. transform into \( c_1, \ldots, c_m \) by WFST

\[ \text{WFSA} \rightarrow t \ldots t \rightarrow \text{WFST} \rightarrow w \ldots w \]
Ideal vs. Available Data

CRYPTOGRAPHY
1. generate $e_1, \ldots, e_n$ by $P(e_k|e_{k-1})$
2. for $i = 1$ to $n$
   output $c_i$ by $P(c_i|e_i)$

PART OF SPEECH
1. generate $t_1, \ldots, t_n$ by $P(t_k|t_{k-1})$
2. for $i = 1$ to $n$
   output $w_i$ by $P(w_i|t_i)$

SPELLING-TO-SOUND
1. generate $\text{phon}$, $\ldots$, phon
2. transform into $c_1, \ldots, c_m$ by WFST

SPELLING-TO-SOUND (no examples)

(ws)
Incomplete Data / Model

- Complete data → ML training
  \( \text{argmax} \ P(\text{data} | m) \) → Complete model (parameters set)

- Incomplete data → Data generation
  (e.g., Viterbi)
  \( \text{argmax} \ P(d | m) \) → Complete data

- Incomplete data → EM

- Incomplete data → Incomplete model → EM

---

Idea: \( \text{argmax} \ P(\text{incomplete-data} | m) \)
Example: Cryptography. \[ \arg\max_m P(c_1 \ldots c_n | m) \]

\[ \arg\max_m \sum_{e_1 \ldots e_n} P(e_1 \ldots e_n) \cdot P(c_1 \ldots c_n | e_1 \ldots e_n, m) \]

\[ \arg\max_m \sum_{e_1 \ldots e_n} P(e_1 \ldots e_n) \cdot P(c_i | e_i, m) \ldots P(c_n | e_n, m) \]

Each choice of \( m \) yields a specific number! Some \( m \) are better than others!

Which is best?

Start with \( m \) such that \( P(c_i | e_j, m) = \frac{1}{2} \).

That gives a certain \( P(c_1 \ldots c_n | m) \).

Now, change \( m \) to \( m' \) such that

\[ P(c_1 \ldots c_n | m') > P(c_1 \ldots c_n | m) \]

(& repeat)
How to Change $m$? 1) Hard

Idea #1

Values for all parameters, including hidden ones

Viterbi

ML training “count & normalize”

Complete data

Suggests iterative procedure.

Initially:

\[ t(a|x) = 0.5 \]
\[ t(b|x) = 0.5 \]
\[ t(a|y) = 0.5 \]
\[ t(b|y) = 0.5 \]

Viterbi: a a a b a a b a a

\{ NOTE: other decodings are equally good. (tie break) \}
How to Change $m$? 1) Hard

Idea #1

Values for all parameters, including hidden ones

ML training "count & normalize"

Viterbi

Complete data

Suggests iterative procedure.

Initially:

- $t(a|x) = 0.5$
- $t(b|x) = 0.5$
- $t(a|y) = 0.5$
- $t(b|y) = 0.5$

Viterbi:

```
a a a b a a b a a
y x x x x x y x x
```

Revised:

- $t(a|x) = 6/7$
- $t(b|x) = 1/7$
- $t(a|y) = 1/2$
- $t(b|y) = 1/2$

Note: other decodings are equally good. (tie break)
How to Change $m$? 1) Hard

viterbi: $a a a b a a b a a$

revised: $t(a|x) = 6/7$
$t(b|x) = 1/7$
$t(a|y) = 1/2$
$t(b|y) = 1/2$

revised
viterbi: $a a a b a a b a a$
How to Change $m$?

1) Hard

```
viterbi:   a a a b a a b a a
          y x x x x x y x x

revised:  $t(a|x) = 6/7$
          $t(b|x) = 1/7$
          $t(a|y) = 1/2$
          $t(b|y) = 1/2$

revised

viterbi:   a a a b a a b a a
          x x x y x y x x

revised:  $t(a|x) = 1$
          $t(b|x) = 0$
          $t(a|y) = 0$
          $t(b|y) = 1$

Stuck now in local minimum, as viterbi doesn't change.

⇒ VITERBI TRAINING```
How to Change $m$? 2) Soft

Idea #1

values for all parameters, including hidden ones

Viterbi

ML training “count & normalize”

Complete data

Idea #2

Values for all parameters

Composition

all ways of completing data, individually weighted

ML training “fractional counts”

EM TRAINING
## Fractional Counts

- distribution over all possible hallucinated hidden variables

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A I</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>AY</td>
<td>W AI N</td>
<td>0.333</td>
<td>0.333</td>
</tr>
<tr>
<td>W</td>
<td>0.667</td>
<td>0.333</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0.667</td>
<td>0.333</td>
<td></td>
</tr>
</tbody>
</table>

- fractional counts

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A I</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>AY</td>
<td>0.500</td>
<td>0.250</td>
<td>0.250</td>
</tr>
<tr>
<td>W</td>
<td>0.750</td>
<td>0.250</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0.750</td>
<td>0.250</td>
<td></td>
</tr>
</tbody>
</table>

eventually

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Monday, November 7, 2011
Fractional Counts

- how about
  \[ W \ E H \ T \]
  \[ W \ E T \ O \]
  \[ B \ I Y \]
  \[ \mid \mid \mid \]
  \[ B \ I I \]

- so EM can possibly: (1) learn something correct
  (2) learn something wrong (3) doesn’t learn anything

- but with lots of data => likely to learn something good
EM: slow version (non-DP)

- initialize the conditional prob. table to uniform

- repeat until converged:
  - E-step:
    - for each training example $x$ (here: (e...e, j...j) pair):
      - for each hidden $z$: compute $p(x, z)$ from the current model
      - $p(x) = \sum_z p(x, z)$; [debug: corpus prob $p(\text{data}) *= p(x)$]
      - for each hidden $z = (z_1 z_2 ... z_n)$: for each $i$:
        - $\text{fraccount}(z_i) += p(x, z)$ / $p(x)$
  - M-step: count-n-divide on fraccounts => new model
EM: fast version (DP)

- initialize the conditional prob. table to uniform

- repeat until converged:
  - E-step:
    - for each training example $x$ (here: $(e\ldots e, j\ldots j)$ pair):
      - forward from $s$ to $t$; note: $\text{forw}[t] = p(x) = \sum_z p(x, z)$
      - backward from $t$ to $s$; note: $\text{back}[t] = 1; \text{back}[s] = \text{forw}[t]$
      - for each edge $(u, v)$ in the DP graph with $\text{label}(u, v) = z_i$
        - $\text{fraccount}(z_i) += \frac{\text{forw}[u] \cdot \text{back}[v] \cdot \text{prob}(u, v)}{p(x)}$
  - M-step: count-n-divide on fraccounts => new model

$\sum_z: (u, v) \in z \quad p(x, z)$
How to avoid enumeration?

- dynamic programming: the forward-backward algorithm
- forward is just like Viterbi, replacing max by sum
- backward is like reverse Viterbi (also with sum)

POS tagging, crypto, ...

alignment, edit-distance, ...

CS 562 - EM

Monday, November 7, 2011
for HW5. this example shows forward only.

```python
n, m = len(eprons), len(jprons)
forward[0][0] = 1

for i in xrange(0, n):
    epron = eprons[i]
    for j in forward[i]:
        for k in range(1, min(m-j, 3)+1):
            jseg = tuple(jprons[j:j+k])
            score = forward[i][j] * table[epron][jseg]
            forward[i+1][j+k] += score

totalprob *= forward[n][m]
```

Total probability is the product of forward values at the end.

---

**Diagram:**

The diagram illustrates the forward algorithm with a matrix and arrows indicating the transitions. Each cell in the matrix represents a combination of emissions and states, with arrows showing how values are accumulated and propagated forward. The final probability is calculated at the bottom right cell of the matrix.
for HW5, this example shows forward only.

```
n, m = len(eprons), len(jprons)
forward[0][0] = 1

for i in xrange(0, n):
    epron = eprons[i]
    for j in forward[i]:
        for k in range(1, min(m-j, 3)+1):
            jseg = tuple(jprons[j:j+k])
            score = forward[i][j] * table[epron][jseg]
            forward[i+1][j+k] += score

totalprob *= forward[n][m]
```
Example Forward Code

- for HW5. this example shows forward only.

```python
n, m = len(eprons), len(jprons)
forward[0][0] = 1

for i in xrange(0, n):
    epron = eprons[i]
    for j in forward[i]:
        for k in range(1, min(m-j, 3)+1):
            jseg = tuple(jprons[j:j+k])
            score = forward[i][j] * table[epron][jseg]
            forward[i+1][j+k] += score

totalprob *= forward[n][m]
```

```
forw[u]  
\downarrow \ 
\downarrow \ 
\downarrow \  
|   |   |   |
\uparrow \  
\uparrow \  
\uparrow \  
|   |   |   |
  s  \rightarrow v \rightarrow t
\downarrow \  
\downarrow \  
\downarrow \  
|   |   |   |
  forw[s] = back[t] = 1.0
\downarrow \  
\downarrow \  
\downarrow \  
|   |   |   |
  forw[t] = back[s] = p(x)
```

CS 526 - EM

Monday, November 7, 2011
EM: fast version (DP)

- initialize the conditional prob. table to uniform
- repeat until converged:
  - E-step:
    - for each training example $x$ (here: (e...e, j...j) pair):
      - forward from $s$ to $t$; note: $\text{forw}[t] = p(x) = \sum_z p(x, z)$
      - backward from $t$ to $s$; note: $\text{back}[t] = 1; \text{back}[s] = \text{forw}[t]$
      - for each edge $(u, v)$ in the DP graph with $\text{label}(u, v) = z_i$
        - $\text{fraccount}(z_i) += \frac{\text{forw}[u] \times \text{back}[v] \times \text{prob}(u, v)}{p(x)}$
  - M-step: count-n-divide on fraccounts $\Rightarrow$ new model

$$\sum_z: (u, v) \in z \quad p(x, z)$$
EM

example: cryptanalysis

\[ x_1 \ldots x_n \quad \text{observed ciphertext} \]
\[ z_1 \ldots z_n \quad \text{hidden plaintext} \]
\[ b(z_j|z_k) \quad \text{source bijam probabilities} \]
\[ t(x_j|z_k) \quad \text{channel substitution ("encoding") probs} \]

\[ \Pr(x_1 \ldots x_n, z_1 \ldots z_n) = \prod_{i=1}^{n} b(z_i|z_{i-1}) \cdot t(x_i|z_i) \]

\[ \Pr(x_1 \ldots x_n) = \sum_{z_1 \ldots z_n} \prod_{i=1}^{n} b(z_i|z_{i-1}) \cdot t(x_i|z_i) \]

\[ \Pr(z_1 \ldots z_n | x_1 \ldots x_n) = \frac{\Pr(x_1 \ldots x_n, z_1 \ldots z_n)}{\Pr(x_1 \ldots x_n)} \]

Composition

observable parameters, hidden parameters, values for all

"fractional count"

ML training

all completions of data, weighted

CS 56_ ___
Why EM increases $p_{\text{data}}$ iteratively?

$$D = \log p(x; \theta) = \log \sum_z p(x, z; \theta) \frac{p(z|x; \theta_t)}{p(z|x; \theta_t)}.$$
Why EM increases $p(\text{data})$ iteratively?

$$D = \log p(x; \theta) = \log \sum_z p(x, z; \theta) \frac{p(z|x; \theta_t)}{p(z|x; \theta_t)}.$$ 

Note that $\sum_z p(z|x; \theta_t) = 1$ and $p(z|x; \theta_t) \geq 0$ for all $z$. Therefore $D$ is the logarithm of a weighted sum, so we can apply Jensen’s inequality, which says $\log \sum_j w_j v_j \geq \sum_j w_j \log v_j$, given $\sum_j w_j = 1$ and each $w_j \geq 0$. Here, we let the sum range over the values $z$ of $Z$, with the weight $w_j$ being $p(z|x; \theta_t)$. We get

$$D \geq E = \sum_z p(z|x; \theta_t) \log \frac{p(x, z; \theta)}{p(z|x; \theta_t)}.$$
Why EM increases $p(\text{data})$ iteratively?

\[ D = \log p(x; \theta) = \log \sum_z p(x, z; \theta) \frac{p(z|x; \theta_t)}{p(z|x; \theta_t)}. \]

Note that $\sum_z p(z|x; \theta_t) = 1$ and $p(z|x; \theta_t) \geq 0$ for all $z$. Therefore $D$ is the logarithm of a weighted sum, so we can apply Jensen’s inequality, which says $\log \sum_j w_j v_j \geq \sum_j w_j \log v_j$, given $\sum_j w_j = 1$ and each $w_j \geq 0$. Here, we let the sum range over the values $z$ of $Z$, with the weight $w_j$ being $p(z|x; \theta_t)$. We get

\[ D \geq E = \sum_z p(z|x; \theta_t) \log \frac{p(x, z; \theta)}{p(z|x; \theta_t)}. \]

Separating the fraction inside the logarithm to obtain two sums gives

\[ E = \left( \sum_z p(z|x; \theta_t) \log p(x, z; \theta) \right) - \left( \sum_z p(z|x; \theta_t) \log p(z|x; \theta_t) \right). \]

Since $E \leq D$ and we want to maximize $D$, consider maximizing $E$. The weights $p(z|x; \theta_t)$ do not depend on $\theta$, so we only need to maximize the first sum, which is

\[ \sum_z p(z|x; \theta_t) \log p(x, z; \theta). \]
Why EM increases $p(data)$ iteratively?

How do we know that maximizing $E$ actually leads to an improvement in the likelihood? With $\theta = \theta_t$,

$$E = \sum_z p(z|x; \theta_t) \log \frac{p(x, z; \theta_t)}{p(z|x; \theta_t)} = \sum_z p(z|x; \theta_t) \log p(x; \theta_t) = \log p(x; \theta_t)$$

 converge to local maxima

convex auxiliary function

KL-divergence

Jensen's inequality
How to maximize the auxiliary?

\[ \sum_z p(z|x; \theta_t) \log p(x, z; \theta). \]

In general, the E-step of an EM algorithm is to compute \( p(z|x; \theta_t) \) for all \( z \). The M-step is then to find \( \theta \) to maximize \( \sum_z p(z|x; \theta_t) \log p(x, z; \theta) \).
How to maximize the auxiliary?

\[ \sum_z p(z|x; \theta_t) \log p(x, z; \theta). \]

In general, the E-step of an EM algorithm is to compute \( p(z|x; \theta_t) \) for all \( z \). The M-step is then to find \( \theta \) to maximize \( \sum_z p(z|x; \theta_t) \log p(x, z; \theta) \).

\begin{align*}
& p(z|x) = 0.5 & p(z'|x) = 0.3 & p(z''|x) = 0.2
\end{align*}
How to maximize the auxiliary?

$$\sum_{z} p(z|x; \theta_t) \log p(x, z; \theta).$$

In general, the E-step of an EM algorithm is to compute $p(z|x; \theta_t)$ for all $z$. The M-step is then to find $\theta$ to maximize $\sum_{z} p(z|x; \theta_t) \log p(x, z; \theta)$.

$p(z|x)=0.5 \quad p(z'|x)=0.3 \quad p(z''|x)=0.2$

just count-n-divide on the fractional data!

(as if MLE on complete data)