Unit 2: Natural Language Learning

Unsupervised Learning

(EM, forward-backward, inside-outside)

Liang Huang
liang.huang.sh@gmAYl.com
Review of Noisy-Channel Model

Application | Input | Output | $p(i)$ | $p(o|i)$
--- | --- | --- | --- | ---
Machine Translation | $L_1$ word sequences | $L_2$ word sequences | $p(L_1)$ in a language model | translation model
Optical Character Recognition (OCR) | actual text | text with mistakes | prob of language text | model of OCR errors
Part Of Speech (POS) tagging | POS tag sequences | English words | prob of POS sequences | acoustic model
Speech recognition | word sequences | speech signal | prob of word sequences |
Example 1: Part-of-Speech Tagging

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

- Use tag bigram as a language model
- Channel model is context-independent.
**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 $pho, \ldots, phon$
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 $1/n$!
## Ideal vs. Available Data

**HW2: ideal**
- EY B AH L
- A B E R U
- 1 2 3 4 4

- AH B AW T
- A B A U T O
- 1 2 3 3 4 4

- AH L ER T
- A R A A T O
- 1 2 3 3 4 4

- EY S
- E E S U
- 1 1 2 2

**HW4: realistic**
- EY B AH L
- A B E R U
- 1 2 3 4 4

- AH B AW T
- A B A U T O
- 1 2 3 3 4 4

- AH L ER T
- A R A A T O
- 1 2 3 3 4 4

- EY S
- E E S U
- 1 1 2 2
Incomplete Data / Model

- **Complete data** → ML training \( \arg\max_m P(d|m) \) → **Complete model** (parameters set)

- **Incomplete data** → **Data generation** (e.g., Viterbi) \( \arg\max_d P(d|m) \) → **Complete data**

- **Incomplete data** → **EM** → **Complete model & data**

**Idea:** \( \arg\max_m P(\text{incomplete-data} | m) \)
**EM: Expectation-Maximization**

Example: Cryptography. \[
\arg\max_m P(c_1, \ldots, c_n \mid m) = \arg\max_m \sum_{e_1 \ldots e_n} P(e_1 \ldots e_n) \cdot P(c_1, \ldots, c_n \mid e_1 \ldots e_n, m) \\
\arg\max_m \sum_{e_1 \ldots e_n} P(e_1 \ldots e_n) \cdot P(c_i \mid e_i, m) \ldots P(c_n \mid e_n, m)
\]

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

Which is best?

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

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

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

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

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

**Idea #1**

Values for all parameters, including hidden ones

ML training "count & normalize"

Suggests iterative procedure.

Initially:
\[
\begin{align*}
t(a|x) &= 0.5 \\
t(b|x) &= 0.5 \\
t(a|y) &= 0.5 \\
t(b|y) &= 0.5
\end{align*}
\]

Viterbi: $a \ a \ a \ b \ a \ a \ b \ a \ a$

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

\[
\begin{align*}
\text{viterbi:} & \quad a & a & a & b & a & a & b & a & a \\
& \quad y & x & x & x & x & x & y & x & x \\
\text{revised:} & \quad t(a|x) = 6/7 \\
& \quad t(b|x) = 1/7 \\
& \quad t(a|y) = 1/2 \\
& \quad t(b|y) = 1/2
\end{align*}
\]

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

2) Soft

**Idea #1**

- Values for all parameters, including hidden ones
- Viterbi
  - ML training
    - "Count & normalize"

**Idea #2**

- Values for all parameters
- Composition
  - ML training
    - "Fractional counts"
- All ways of completing data, individually weighted
- EM Training
Fractional Counts

- distribution over all possible hallucinated hidden variables

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>A</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>A</td>
<td>I</td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

hard-EM counts

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A Y</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

fractional counts

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A Y</td>
<td>0.333</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>0.667</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0.667</td>
<td></td>
</tr>
</tbody>
</table>

regenerate: $2/3 \times 1/3 \times 1/3$ $2/3 \times 1/3 \times 2/3$ $1/3 \times 1/3 \times 2/3$

fractional counts

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A Y</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0.25</td>
<td></td>
</tr>
</tbody>
</table>

eventually

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Is EM magic? well, sort of...

- how about

```
W E H T
```

```
W E T O
```

```
B I Y     B I Y
|   |\    |   |\  \\
B I I     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) pAYr):
      - for each hidden $z$: compute $p(x, z)$ from the current model
      - $p(x) = \text{sum}_z p(x, z)$; \[\text{[debug: corpus prob } p(\text{data}) *= p(x)\]\]
      - for each hidden $z = (z_1 \ z_2 \ ... \ z_n)$: for each $i$:
        - $\#(z_i) += p(x, z) / p(x)$; $\#(\text{LHS}(z_i)) += p(x, z) / p(x)$
  - M-step: count-n-divide on fraccounts => new model

\[
p(\text{RHS}(z_i) \mid \text{LHS}(z_i)) = \#(z_i) / \#(\text{LHS}(z_i))
\]
EM: slow version (non-DP)

- distribution over all possible hallucinated hidden variables

<table>
<thead>
<tr>
<th>W</th>
<th>A</th>
<th>Y</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>/</td>
<td>\</td>
</tr>
<tr>
<td>W</td>
<td>A</td>
<td>I</td>
<td>N</td>
</tr>
</tbody>
</table>

fractional counts: 1/3

- regenerate \( p(x, z) \):

\[
\frac{2}{3} * \frac{1}{3} * \frac{1}{3}
\]

- renormalize by \( p(x) = \frac{2}{27} + \frac{4}{27} + \frac{2}{27} = \frac{8}{27} \)

fractional counts: 1/4

<table>
<thead>
<tr>
<th>A</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.333</td>
<td>0.333</td>
</tr>
</tbody>
</table>

- regenerate \( p(x, z) \):

\[
\frac{3}{4} * \frac{1}{4} * \frac{1}{4}
\]

- renormalize by \( p(x) = \frac{3}{64} + \frac{18}{64} + \frac{3}{64} = \frac{3}{8} \)

fractional counts: 1/8

<table>
<thead>
<tr>
<th>A</th>
<th>I</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.333</td>
<td>0.333</td>
<td>0.333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W</th>
<th>A</th>
<th>Y</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>W</td>
<td>A</td>
<td>I</td>
<td>N</td>
</tr>
</tbody>
</table>

fractional counts: 1/3

- regenerate \( p(x, z) \):

\[
\frac{3}{4} * \frac{1}{2} * \frac{3}{4}
\]

- renormalize by \( p(x) = \frac{3}{64} + \frac{18}{64} + \frac{3}{64} = \frac{3}{8} \)

fractional counts: 3/4
**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) pAYr):
      - 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 => 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, ...

inside-outside: PCFG, SCFG, ...

alignment, edit-distance, ...
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]
```
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[s] = back[t] = 1.0
  - forw[t] = back[s] = p(x)
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 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 => new model

\( \sum_z: (u, v) \text{ in } z \ p(x, z) \)
**Example: Cryptanalysis**

- $X_1 \ldots X_n$: observed ciphertext
- $Z_1 \ldots Z_n$: hidden plaintext
- $b(z_j | z_k)$: source bijection probabilities
- $t(x_j | z_k)$: channel substitution ("encoding") probabilities

**Joint Probability**

$$P(X_1 \ldots X_n, Z_1 \ldots Z_n) = \prod_{i=1}^{n} b(z_i | z_{i-1}) \cdot t(x_i | z_i)$$

**Conditional Probability**

$$P(Z_1 \ldots Z_n | X_1 \ldots X_n) = \frac{P(X_1 \ldots X_n, Z_1 \ldots Z_n)}{P(X_1 \ldots X_n)}$$
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 $p(\text{data})$ iteratively?

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

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

convex auxiliary function

converge to local maxima

KL-divergence
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)

\[ \begin{array}{ccc}
5x & 3x & 2x
\end{array} \]