Unit 3: Natural Language Learning

Unsupervised Learning

(EM, forward-backward, inside-outside)
# 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

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

## $p(o|i)$
- translation model
- model of OCR errors
- acoustic model

\[ \text{WFSA} \rightarrow t \cdots t \rightarrow \text{WFST} \rightarrow w \cdots w \]
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-indep.
**Ideal vs. Available Data**

**CRYPTOGRAPHY**
1. generate $e_1, ..., e_n$ 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 $pho, ..., phon$
2. transform into $c_1, ..., c_m$ by WFST

**MT**
1. generate $e_1, ..., e_n$ 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**

\[ \text{argmax } P(\text{data} | m) \]

→ **Complete model**

\( (\text{parameters set}) \)

**Incomplete data** → **Data generation**

\( (\text{e.g., Viterbi}) \)

\[ \text{argmax } P(d | m) \]

→ **Complete data**

**Incomplete data** → **EM**

→ **Complete model & data**

**Idea:**

\[ \text{argmax } \frac{P(\text{incomplete-data} | m)}{m} \]

---

**B O W T**

**B O O T O**

**1 2 2 3 3**

**B O W T**

**B O O T O**

\[ p(B | B) = 1 \]

\[ p(0 0 | OW) = 0.8 \]

\[ p(T | T) = 0.3 \]

**B O W T**

**B O O T O**

**T E H S T**

**T E S U T O**
**EM: Expectation-Maximization**

**Example: Cryptography.**

\[
\begin{align*}
\arg\max_m & \quad P(c_1 \ldots c_n | m) \\
& \quad \arg\max_m \sum_{e_1 \ldots e_n} P(e_1 \ldots e_n) \cdot P(c_1 \ldots c_n | e_1 \ldots e_n, m) \\
& \quad \arg\max_m \sum_{e_1 \ldots e_n} P(e_1 \ldots e_n) \cdot P(c_1 | e_1, m) \ldots P(c_n | e_n, m)
\end{align*}
\]

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

Which is best?

Start with \( m \) such that \( P(c_i | e_j, m) = \frac{1}{27} \).

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

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

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

(\& 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

**viterbi:**

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

**NOTE:** other decodings are equally good.
(tie break)

**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 $m$? 2) Soft

Idea #1
- Values for all parameters, including hidden ones
- Viterbi
- Complete data
- ML training "count & normalize"

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

- **hard-EM counts**

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

- **fractional counts**

<table>
<thead>
<tr>
<th></th>
<th>0.333</th>
<th>0.333</th>
<th>0.333</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>0.333</th>
<th>A</th>
<th>0.333</th>
<th>I</th>
<th>0.333</th>
</tr>
</thead>
<tbody>
<tr>
<td>AY</td>
<td>--&gt;</td>
<td>A</td>
<td>0.333</td>
<td>A</td>
<td>0.333</td>
<td>I</td>
</tr>
<tr>
<td>W</td>
<td>--&gt;</td>
<td>W</td>
<td>0.667</td>
<td>W</td>
<td>0.333</td>
<td>I</td>
</tr>
<tr>
<td>N</td>
<td>--&gt;</td>
<td>N</td>
<td>0.667</td>
<td>I</td>
<td>0.333</td>
<td>N</td>
</tr>
</tbody>
</table>

- **fractional counts**

<table>
<thead>
<tr>
<th></th>
<th>0.25</th>
<th>0.5</th>
<th>0.25</th>
</tr>
</thead>
</table>

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

- **eventually**

<table>
<thead>
<tr>
<th></th>
<th>... 0</th>
<th>... 1</th>
<th>... 0</th>
</tr>
</thead>
</table>
Fractional Counts

• 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: \((e...e, j...j)\) 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 \( i \):
        - \( \text{fraccount}(z_i) += \frac{p(x, z)}{p(x)} \)
  - 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...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

\[
\sum_z: (u, v) \text{ in } z \quad p(x, z)
\]
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, ...

inside-outside: PCFG, SCFG, ...

CS 562 - EM
for HW5. this example shows forward only.

\[
\begin{align*}
n, m &= \text{len}(eprons), \text{len}(jprons) \\
\text{forward}[0][0] &= 1 \\

\text{for } i \text{ in } \text{xrange}(0, n): \\
& \quad \text{epron} = \text{eprons}[i] \\
& \quad \text{for } j \text{ in } \text{forward}[i]: \\
& \quad \quad \text{for } k \text{ in } \text{range}(1, \text{min}(m-j, 3)+1): \\
& \quad \quad \quad \text{jseg} = \text{tuple}(\text{jprons}[j:j+k]) \\
& \quad \quad \quad \text{score} = \text{forward}[i][j] \times \text{table}[\text{epron}][\text{jseg}] \\
& \quad \quad \quad \text{forward}[i+1][j+k] += \text{score} \\
\text{totalprob} &= \text{totalprob} \times \text{forward}[n][m]
\end{align*}
\]
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]
```
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 $\text{label}(u, v) = z_i$
        - $\text{fraccount}(z_i) += \text{forw}[u] \times \text{back}[v] \times \text{prob}(u, v) / p(x)$
  - M-step: count-n-divide on fraccouts $\Rightarrow$ new model

\[
\sum_z: (u, v) \in z \ p(x, z) = \text{forw}[t] = \text{back}[s] = p(x) = \sum_z p(x, z)
\]
**EM**

**Composition**

Observable parameters, hidden parameters, values for all

"fractional count" ML training

All completions of data, weighted

---

**Example: Cryptanalysis**

\[ X_1 \ldots X_n \] observed ciphertext

\[ Z_1 \ldots Z_n \] hidden plaintext

\[ b(z_j | z_k) \] source bijam probabilities

\[ t(x_j | z_k) \] channel substitution ("encoding") probs

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

\[
P(X_1 \ldots X_m) = \sum_{Z_1 \ldots Z_n} \prod_{i=1}^{n} b(z_i | z_{i-1}) \cdot t(x_i | z_i)
\]

\[
P(Z_1 \ldots Z_n | X_1 \ldots X_n) = \frac{P(X_1 \ldots X_n, Z_1 \ldots Z_n)}{P(X_1 \ldots X_n)}
\]

**Observed, fixed**

**Hidden**

**Generative story**

**Forward procedure**

**Condition. prob.**
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 $\theta = \theta_t$,

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

convex auxiliary function

converge to local maxima

$\text{KL}-\text{divergence}$
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)