Unit 1: Sequence Models
Lectures 7-8: Stochastic String Transformations
(a.k.a. “channel-models”)

Professor Liang Huang
liang.huang.sh@gmail.com
String Transformations

- General Framework for many NLP problems

- Examples
  - Part-of-Speech Tagging
  - Spelling Correction (Edit Distance)
  - Word Segmentation
  - Transliteration, Sound/Spelling Conversion, Morphology
  - Chunking (Shallow Parsing)
  - Beyond Finite-State Models (i.e., tree transformations)
    - Summarization, Translation, Parsing, Information Retrieval, ...

- Algorithms: Viterbi (both max and sum)
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          | p($w|t$)                     |
| Speech recognition               | word sequences     | speech signal                 | prob of word sequences         | acoustic model                |

$WFSA \xrightarrow{t \ldots t} WFST \xrightarrow{w \ldots w}$
Example 1: Part-of-Speech Tagging

- Use tag bigram as a language model
- Channel model is context-independent.

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

Source: \[ t_1 / P(t_1) \]

Channel: \[ t_2 / P(t_2 | t_1) \]

New string: \[ t_4 : w_4 / P(w_4 | t_1) \]
\[ t_6 : w_9 / P(w_9 | t_6) \]

\[ \text{they can fish} \]

\[ I \text{ saw her earrings.} \]

\[ \text{PRO} \rightarrow V \rightarrow \text{PRO} \rightarrow N \]

\[ \text{NOTE: choice of tag is influenced by both left and right context} \]

\[ \text{NOTE: influences can theoretically be long ranging} \]
Work out the compositions

- if you want to implement Viterbi...
- case 1: language model is a tag unigram model
  - \( p(t\ldots t) = p(t_1)p(t_2) \ldots p(t_n) \)
  - how many states do you get?
- case 1: language model is a tag bigram model
  - \( p(t\ldots t) = p(t_1)p(t_2 | t_1) \ldots p(t_n | t_{n-1}) \)
  - how many states do you get?
- case 3: language model is a tag trigram model...
The case of bigram model

context-dependence (from LM) propagates left and right!
In general...

- bigram LM with context-independent CM
  - $O(n \cdot m)$ states after composition
- g-gram LM with context-independent CM
  - $O(n \cdot m^{g-1})$ states after composition
  - the g-gram LM itself has $O(m^{g-1})$ states
HMM Representation

- HMM representation is not explicit about the search
- “hidden states” have choices over “variables”
- in FST composition, paths/states are explicitly drawn
Viterbi for argmax

Viterbi search for argmax $p(t_{\ldots}t) \cdot p(w_{\ldots}w|t_{\ldots}t)$:

for $j = 1$ to $m$

$Q[i,j] = P(t_j) \cdot P(w_i|t_j)$

for $i = 2$ to $n$

for $j = 1$ to $m$

$Q[i,j] = 0$

best-pred $[i,j] = 0$

best-score $= -\infty$

for $k = 1$ to $m$

$r = P(t_j|t_k) \cdot P(w_i|t_j) \cdot Q[i-1,k]$

if $r >$ best-score

best-score $= r$

best-pred $[i,j] = k$

$Q[i,j] = r$

final-best $= 0$

final-score $= -\infty$

for $j = 1$ to $m$

if $Q[n,j] >$ final-score

final-score $= Q[n,j]$

final-best $= j$

print $t_{\text{final-best}}$

current $= \text{final-best}$

for $i = n-1$ down to $0$

current $= \text{best-pred}[i+1, \text{current}]$

print $t_{\text{current}}$

$Q[i,j]$ = cost of shortest path ending with word $i$ getting assigned tag $j$.

How about unigram?
Python implementation

Complete this Python code implementing the Viterbi algorithm for part-of-speech tagging. It should print a list of word/tag pairs, e.g. [('a', 'D'), ('can', 'N'), ('can', 'A'), ('can', 'V'), ('a', 'D'), ('can', 'N')].

```python
from collections import defaultdict

best = defaultdict(lambda : defaultdict(float))
best[0]["<s>"] = 1
back = defaultdict(dict)

words = "<s> a can can a can </s>".split()

tags = {"a": ["D"], "can": ["N", "A", "V"], "</s>": ["</s>"]}  # possible tags for each word
ptag = {"D": {"N": 1, "V": {"</s>": 0.5, "D":0.5}, ... }  # ptag[x][y] = p(y | x)
pword = {"D": {"a": 0.5, "N": {"can": 0.1}, ... }  # pword[x][w] = p(w | x)

for i, word in enumerate(words[1:], 1):
    for tag in tags[word]:
        for prev in best[i-1]:
            if tag in ptag[prev]:
                score = best[i-1][prev] * ptag[prev][tag] * pword[tag][word]
                if score > best[i][tag]:
                    best[i][tag] = score
                    back[i][tag] = prev

def backtrack(i, tag):
    if i == 0:
        return []
    return backtrack(i-1, back[i][tag]) + [(words[i], tag)]

print backtrack(len(words)-1, "</s>"))[::-1]
```

Q: what about top-down recursive + memoization?
Q1. Why is this table not normalized?

Q2. Is “fish” equally likely to be a V or N?

Q3: How to train $p(w|t)$?
Excel Demo
Trigram HMM

for \( j = 1 \) to \( m \)
\[ Q_1[1,j] = \ldots \]

for \( j = 1 \) to \( m \)
for \( j_2 = 1 \) to \( m \)
\[ Q[2,j,j_2] = \ldots \]

for \( i = 3 \) to \( n \)
for \( j = 1 \) to \( m \)
for \( j_2 = 1 \) to \( m \)
\[ Q[i,j,j_2] = 0 \]
\[ \text{best-pred}[i,j,j_2] = 0 \]
\[ \text{best-score} = -\infty \]
for \( k = 1 \) to \( m \)
\[ r = P(t_{j_2} \mid t_j) \cdot P(w_i \mid t_{j_2}) \cdot Q[i-1,k,j] \]
if \( r > \text{best-score} \) …

Time complexity: \( O(nT^3) \)
in general: \( O(nT^g) \) for \( g \)-gram
A Side Note on Normalization

NOTE

final-best gives $P(t\cdots t) \cdot P(w\cdots w | t\cdots t)$

but this is not the same as $P(t\cdots t | w\cdots w)$

e.g. suppose there is only one $t\cdots t$ (all words unambiguous)

then $P(t\cdots t | w\cdots w) = 1$

need to divide

$$P(t\cdots t | w\cdots w) = \frac{P(t\cdots t) \cdot P(w\cdots w | t\cdots t)}{P(w\cdots w)} = \frac{P(t\cdots t) \cdot P(w\cdots w | t\cdots t)}{\sum_{t\cdots t} P(t\cdots t) \cdot P(w\cdots w | t\cdots t)}$$

how to compute the normalization factor?
Forward (sum instead of max)

Forward search: \[ \sum_{t \in T} P(t) \cdot P(w \mid t) = P(w) \]

\[ \alpha[1, j] = P(t_j \mid \text{START}) \cdot P(w_i \mid t_j) \]

\[ \alpha[i, j] = \sum_k \alpha[i-1, k] \cdot P(t_j \mid t_k) \cdot P(w_i \mid t_j) \]

no back pointer

\[ P(w) = \sum_k \alpha[n, k] \]

"Forward" procedure for \( P(w \ldots w) \)

for \( j = 1 \) to \( m \)

\[ \alpha[1, j] = P(t_j) \cdot P(w_i \mid t_j) \]

for \( i = 2 \) to \( n \)

for \( j = 1 \) to \( m \)

\[ \alpha[i, j] = 0 \]

for \( k = 1 \) to \( m \)

\[ \alpha[i, j] += P(t_j \mid t_k) \cdot P(w_i \mid t_j) \cdot \alpha[i-1, k] \]

\[ P(w \ldots w) = 0 \]

for \( j = 1 \) to \( m \)

\[ P(w \ldots w) += \alpha[n, j] \]

\[ \alpha[i, j] \text{ = costs of all paths ending w/ word w; getting to } t_j \text{ (costs summed)} \]
Forward vs. Argmax

- same complexity, different semirings (+, x) vs (max, x)
- for g-gram LM with context-indep. CM
  - time complexity $O(n m^g)$  space complexity $O(n m^{g-1})$

```plaintext
for j = 1 to m
    Q_1[1,j] = ...

for j = 1 to m
    for j2 = 1 to m
        Q[2,j,j2] = ...

for i = 3 to n
    for j = 1 to m
        for j2 = 1 to m
            Q[i,j,j2] = 0
            best-pred[i,j,j2] = 0
            best-score = -∞
            for k = 1 to m
                r = P(t_{j2} | t_j^k) \cdot P(w_i | t_{j2}) \cdot Q[i-1, k, j]
                if r > best-score ...

O(nm^3) complexity
```
Viterbi for DAGs with Semiring

1. topological sort

2. visit each vertex v in sorted order and do updates
   - for each incoming edge \((u, v)\) in E
   - use \(d(u)\) to update \(d(v)\):
     \[
     d(v) \oplus = d(u) \otimes w(u, v)
     \]
   - key observation: \(d(u)\) is fixed to optimal at this time

- time complexity: \(O(V + E)\)

\[ (A, \oplus, \otimes, 0, 1) \]

See tutorial on DP from course page
(hw3) From Spelling to Sound

- word-based or char-based

```
F -> O -> Z ?
|          |
|          |
|          |
AY         

P(θ) → English sounds sequence → Spell → P(e|s) → English letter sequence

data: AE R UH N S UH N
a a r o n s o n
p(a a | AE) = 0.04
```
Pronunciation Dictionary

- (hw3: eword-epron.data)  
  http://www.speech.cs.cmu.edu/cgi-bin/cmudict
  from CMU Pronunciation Dictionary
  39 phonemes (15 vowels + 24 consonants)

- AARON           EH R AH N
- AARONSON        AA R AH N S AH N

  echo 'W H A L E B O N E S' |
  carmel -sriIEQk 5 epron.wfsa epron-espell.wfst

- PEOPLE           P IY P AH L
- VIDEO            V IH D IY OW

- you can train p(s..s|w) from this, but what about unseen words?
- also need alignment to train the channel model p(s|e) & p(e|s)
CMU Dict: 39 Ame. Eng. Phonemes

WRONG! missing the SCHWA ə (merged with the STRUT ʌ “AH”!)

<table>
<thead>
<tr>
<th>CMU/IPA</th>
<th>Example</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>/ɑ/</td>
<td>odd</td>
</tr>
<tr>
<td>AE</td>
<td>/æ/</td>
<td>at</td>
</tr>
<tr>
<td>AH</td>
<td>/ʌ/</td>
<td>hut</td>
</tr>
<tr>
<td>AO</td>
<td>/ɔː/</td>
<td>ought</td>
</tr>
<tr>
<td>AW</td>
<td>/aʊ/</td>
<td>cow</td>
</tr>
<tr>
<td>AY</td>
<td>/aɪ/</td>
<td>hide</td>
</tr>
<tr>
<td>B</td>
<td>/b/</td>
<td>be</td>
</tr>
<tr>
<td>CH</td>
<td>/tʃ/</td>
<td>cheese</td>
</tr>
<tr>
<td>D</td>
<td>/d/</td>
<td>dee</td>
</tr>
<tr>
<td>DH</td>
<td>/ð/</td>
<td>thee</td>
</tr>
<tr>
<td>EH</td>
<td>/ɛ/</td>
<td>Ed</td>
</tr>
<tr>
<td>ER</td>
<td>/ɜ/</td>
<td>hurt</td>
</tr>
<tr>
<td>EY</td>
<td>/ɛɪ/</td>
<td>ate</td>
</tr>
<tr>
<td>F</td>
<td>/f/</td>
<td>fee</td>
</tr>
<tr>
<td>G</td>
<td>/g/</td>
<td>green</td>
</tr>
<tr>
<td>HH</td>
<td>/h/</td>
<td>he</td>
</tr>
<tr>
<td>IH</td>
<td>/ɪ/</td>
<td>it</td>
</tr>
<tr>
<td>IY</td>
<td>/iː/</td>
<td>eat</td>
</tr>
<tr>
<td>JH</td>
<td>/dʒ/</td>
<td>gee</td>
</tr>
<tr>
<td>K</td>
<td>/k/</td>
<td>key</td>
</tr>
<tr>
<td>L</td>
<td>/l/</td>
<td>lee</td>
</tr>
<tr>
<td>M</td>
<td>/m/</td>
<td>me</td>
</tr>
<tr>
<td>N</td>
<td>/n/</td>
<td>knee</td>
</tr>
<tr>
<td>NG</td>
<td>/ŋ/</td>
<td>ping</td>
</tr>
<tr>
<td>OW</td>
<td>/oʊ/</td>
<td>oat</td>
</tr>
<tr>
<td>OY</td>
<td>/ɔɪ/</td>
<td>toy</td>
</tr>
<tr>
<td>P</td>
<td>/p/</td>
<td>pee</td>
</tr>
<tr>
<td>R</td>
<td>/ɹ/</td>
<td>read</td>
</tr>
<tr>
<td>S</td>
<td>/s/</td>
<td>sea</td>
</tr>
<tr>
<td>SH</td>
<td>/ʃ/</td>
<td>she</td>
</tr>
<tr>
<td>T</td>
<td>/t/</td>
<td>tea</td>
</tr>
<tr>
<td>TH</td>
<td>/ð/</td>
<td>theta</td>
</tr>
<tr>
<td>UH</td>
<td>/u/</td>
<td>hood</td>
</tr>
<tr>
<td>UW</td>
<td>/u/</td>
<td>too</td>
</tr>
<tr>
<td>V</td>
<td>/v/</td>
<td>vee</td>
</tr>
<tr>
<td>W</td>
<td>/w/</td>
<td>we</td>
</tr>
<tr>
<td>Y</td>
<td>/j/</td>
<td>yield</td>
</tr>
<tr>
<td>Z</td>
<td>/z/</td>
<td>zee</td>
</tr>
<tr>
<td>ZH</td>
<td>/ʒ/</td>
<td>usual</td>
</tr>
</tbody>
</table>

CS 562 - Lec 11-13: String Transformations
WRONG! missing the SCHWA ə (merged with the STRUTʌ “AH”!
DOES NOT ANNOTATE STRESSES

A          AH
A          EY
AAA        T R IH P AH L EY
AABERG     AA B ER G
AACHEN     AA K AH N
...        
ABOUT      AH B AW T
...
ABRAMOVITZ AH B R AA M AH V IH T S
ABRAMOWICZ AH B R AA M AH V IH CH
ABRAMOWITZ AH B R AA M AH W IH T S
...
FATHER     F AA DH ER
...
ZYDECO     Z AY D EH K OW
ZYDECO     Z IH D AH K OW
ZYDECO     Z AY D AH K OW
...
ZZZZZ      Z IY Z
Linguistics Background: IPA

CONSONANTS (PULMONIC)

<table>
<thead>
<tr>
<th></th>
<th>Bilabial</th>
<th>Labiodental</th>
<th>Dental</th>
<th>Alveolar</th>
<th>Post alveolar</th>
<th>Retroflex</th>
<th>Palatal</th>
<th>Velar</th>
<th>Uvular</th>
<th>Pharyngeal</th>
<th>Glottal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plosive</td>
<td>p b</td>
<td>t d</td>
<td></td>
<td></td>
<td></td>
<td>t q</td>
<td>c j</td>
<td>k g</td>
<td>q g</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>Nasal</td>
<td>m n</td>
<td></td>
<td>n</td>
<td></td>
<td>n</td>
<td></td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trill</td>
<td>B r</td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tap or Flap</td>
<td>v R</td>
<td>v</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fricative</td>
<td>$ $</td>
<td>$ $</td>
<td>$ $</td>
<td>$ $</td>
<td>$ $</td>
<td>$ $</td>
<td>$ $</td>
<td>$ $</td>
<td>$ $</td>
<td>$ $</td>
<td>$ $</td>
</tr>
<tr>
<td>Lateral fricative</td>
<td>ѵ</td>
<td>ѵ</td>
<td>ѵ</td>
<td></td>
<td>ѵ</td>
<td></td>
<td></td>
<td>ѵ</td>
<td>ѵ</td>
<td>ѵ</td>
<td>ѵ</td>
</tr>
<tr>
<td>Approximant</td>
<td>ѵ</td>
<td>ѵ</td>
<td>ѵ</td>
<td></td>
<td>ѵ</td>
<td></td>
<td></td>
<td>ѵ</td>
<td>ѵ</td>
<td>ѵ</td>
<td>ѵ</td>
</tr>
<tr>
<td>Lateral approximant</td>
<td>ѵ</td>
<td>ѵ</td>
<td>ѵ</td>
<td></td>
<td>ѵ</td>
<td></td>
<td></td>
<td>ѵ</td>
<td>ѵ</td>
<td>ѵ</td>
<td>ѵ</td>
</tr>
</tbody>
</table>

Where symbols appear in pairs, the one to the right represents a voiced consonant. Shaded areas denote articulations judged impossible.

VOWELS

Where symbols appear in pairs, the one to the right represents a rounded vowel.
Linguistics Background: emic-etic
(hw3) From Sound to Spelling

- **input:** HH EH L OW B EH R
- **output:** H E L L O B E A R or H E L O B A R E ?

- \( p(e) \Rightarrow e \Rightarrow p(s|e) \Rightarrow s \)
- \( p(w) \Rightarrow w \Rightarrow p(e|w) \Rightarrow e \Rightarrow p(s|e) \Rightarrow s \)
- \( p(w) \Rightarrow w \Rightarrow p(s|w) \Rightarrow s \)
- \( p(w) \Rightarrow w \Rightarrow p(e|w) \Rightarrow e \Rightarrow p(s|e) \Rightarrow s \Rightarrow p(s) \)
- \( p(w) \Rightarrow w \Rightarrow p(w|e) \Rightarrow e \Rightarrow p(e|s) \Rightarrow s \Rightarrow p(s) \)
- \( w \Rightarrow p(w|s) \Rightarrow s \Rightarrow p(s) \)

- **what else?**
  
  echo 'HH EH L OW' | carmel -sliOEQk 50 epron-espell.wfst espell-eword.wfst eword.wfsa
Example: Transliteration

- \( V \Rightarrow B: \text{phoneme inventory mismatch} \)
- \( T \Rightarrow TO: \text{phonotactic constraint} \)

- KEVIN KNIGHT \( => \) KH EH VH IH N N AY T

KEVIN KNIGHT

ケビン ナイト
Japanese 101 (writing systems)

- Japanese writing system has four components
  - Kanji (Chinese chars): nouns, verb/adj stems, CJKV names
    - 日本 “Japan” 东京 “Tokyo” 电车 “train” 食べる “eat [inf.]”
  - Syllabaries
    - Hiragana: function words (e.g. particles), suffices
      - で de ("at") か ka (question) 食べました “ate”
    - Katakana: transliterated foreign words/names
      - コーヒー koohii (“coffee”)
  - Romaji (Latin alphabet): auxiliary purposes
Why Japanese uses Syllabaries

• all syllables are: 
  [consonant] + vowel + [nasal n]
  \[\text{C}^* \text{V}^+ \text{C}^* \text{T}\]
  \[\text{C}^{10}_n \text{V}_5 n_1^2\]

• 10 C x 5 V = 50 syllables
• plus some variations
• also possible for Mandarin
• other languages have many more syllables: use alphabets
  • alphabet = 10+5; syllabary = 10x5
• read the Writing Systems tutorial from course page!

http://brng.jp/90459562

CS 562 - Lec 11-13: String Transformations
Japanese Phonemes *(too few sounds!)*

**CONSONANTS (PULMONIC)**

<table>
<thead>
<tr>
<th></th>
<th>Bilabial</th>
<th>Labiodental</th>
<th>Dental</th>
<th>Alveolar</th>
<th>Post alveolar</th>
<th>Retroflex</th>
<th>Palatal</th>
<th>Velar</th>
<th>Uvular</th>
<th>Pharyngeal</th>
<th>Glottal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plosive</td>
<td>p b</td>
<td></td>
<td>t d</td>
<td>t d</td>
<td>c j</td>
<td>k g</td>
<td>q g</td>
<td></td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Nasal</td>
<td>m</td>
<td>m j</td>
<td>n</td>
<td>n</td>
<td>η</td>
<td>η</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trill</td>
<td>B</td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tap or Flap</td>
<td>v</td>
<td></td>
<td>r</td>
<td></td>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fricative</td>
<td>φ β f v θ ō s z</td>
<td></td>
<td>ħ ḫ</td>
<td>s z</td>
<td>c j</td>
<td>x y χ b</td>
<td>h f</td>
<td>h f</td>
<td>h f</td>
<td>h f</td>
<td></td>
</tr>
<tr>
<td>Lateral fricative</td>
<td>u</td>
<td></td>
<td>l</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approximant</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lateral approximant</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Where symbols appear in pairs, the one to the right represents a voiced consonant. Shaded areas denote articulations judged impossible.

**VOWELS**

Where symbols appear in pairs, the one to the right represents a rounded vowel.
Aside: Is Korean a Syllabary?

- A: Hangul is not a syllabary, but a “featural alphabet”
- a special alphabet where shapes encode phonological features
- the inventor of Hangul (c. 1440s) was the first real linguist

Q: 강남 스타일 = ?

- 14 consonants: 꾨 g, 꿔 n, 꼽 d, 꼿 l/r, 꼬 m, 꼷 b, 꼹 s, 꼺 null/ng, 꼻 j, 꼼 ch, 꼽 k, 꼾 t, 꼿 p, 꼼 h
- 5 double consonants: 꼻 kk, 꼻 tt, 꼼 pp, 꼼 ss, 꼼 jj
- 11 consonant clusters: 꼻 gs, 꼻 nj, 꼼 nh, 꼽 lg, 꼽 lm, 꼼 lb, 꼼 ls, 꼽 lt, 꼼 lp, 꼼 lh, 꼻 bs
- 6 vowel letters: 꼽 a, 꼽 eo, 꼽 o, 꼽 u, 꼽 eu, 꼽 i
- 4 iotized vowels (with a y): 꼽 ya, 꼽 yeo, 꼽 yo, 꼽 yu
- 5 (iotized) diphthongs: 꼽 ae, 꼽 yae, 꼽 e, 꼽 ye, 꼽 ui
- 6 vowels and diphthongs with a w: 꼽 wa, 꼽 wae, 꼽 oe, 꼽 wo, 꼽 we, 꼽 wi

CS 562 - Lec 5-6: Probs & WFSTs
Katakana Transliteration Examples

• コンピューター
  • ko n py u - ta -
  • kompyuutaa (uu=û)
  • computer
• アイスクリーム
  • a i su ku ri - mu
  • aisukuriimu
  • ice cream
• アンドリュー・ビタビ
  • andoryuubitabi
• ヤーグルト
  • yo - gu ru to
  • yogurt
Japanese just transliterates almost everything (even though its syllable inventory is really small...) but... it is quite easy for English speakers to decode .... if you have a good language model!

koohiikoonaa      coffee corner
saabisu               service
bulendokooohii     blend coffee
sutoreetokooohii  straight coffee
juusu                  juice
aisukuriimu         ice cream
toosuto              toast
More Japanese Transliterations

- rapputoppu  ラブトプ
- bideoteepu  ビデオテープ
- shoppingusentaa  ショッピングセンター
- shiitoberuto  シートベルト
- chairudoshiito  チャイルドシート
- andoryuubitabi  アンドリュー・ビタビ
- bitabiarugorizumu  ビタビリアルゴリズム
- laptop
- video tape
- shopping center
- seat belt
- child seat
- Andrew Viterbi
- Viterbi Algorithm
(hw3) Katakana => English

- your job in HW3: decode Japanese Katakana words (transcribed in Romaji) back to English words

- koohiikoonaa => coffee corner

[Knight & Graehl 98]
Decoding
  • really decipherment!

what about duplicate strings?
  • from different paths in WFST!

n-best cruching, or...

weighted determinisation
  • see extra reading on course website for Mohri+Riley paper

[Knight & Graehl 98]
Example: Word Segmentation

- you noticed that Japanese (e.g., Katakana) is written without spaces between words
- in order to guess the English you also do segmentation
  - e.g. アイスクリーム => アイス クリーム => ice cream
- how about “gaaruhurendo” and “shingururuumu”? 
- this is an even more important issue in Chinese
  - 南京市长江大桥
- also in other East Asian Languages
- also in English: sounds => words (speech recognition)
What if English were written as Chinese...

- thisisasoursetaughtinthefallsemesterofthisyearatusc

- actually, Latin used to be written exactly like this!
  - “scripta continua” => “interpuncts” (center dots) =>

- this might be a final project topic (on the easier side)
Chinese Word Segmentation

min-zhu people-dominate this was 5 years ago. now Google is good at segmentation!

“democracy”

jiang-ze-min zhu-xi "President Jiang Zemin"

xia yu tian di mian ji shui

graph search tagging problem
Word Segmentation Cascades

• a good idea for final project (Chinese/Japanese)
Machine Translation

- simplest model: word-substitution and permutation
- does it really work??
how would you model permutation in FSTs?
Phrase-based Decoding

yu Shalong held a talk with Sharon

与沙龙举行了会谈

yu Shalong juxing le huitan

held a talk with Sharon
Phrase-based Decoding

yu Shalong held a talk with Sharon

juxing le huitan
Phrase-based Decoding

yu Shalong juxing le huitan

held a talk with Sharon

source-side: coverage vector

held a talk
target-side: grow hypotheses
strictly left-to-right

space: $O(2^n)$, time: $O(2^n n^2)$ -- cf. traveling salesman problem
Phrase-based Cascades

- English LM \(\Rightarrow\) (English) \(\Rightarrow\) phrase substitutions \(n^2\) \(\Rightarrow\) (foreign phrases in English word order) \(\Rightarrow\) permutations \(2^n\) \(\Rightarrow\) (foreign)

- A good idea for final project (on the harder end)

- Wait, where does the phrase table come from?
  - \(\Rightarrow\) word-aligned English-foreign sentence pairs
Traveling Salesman Problem & MT

- a classical NP-hard problem
  - goal: visit each city once and only once
- exponential-time dynamic programming
  - state: cities visited so far (bit-vector)
  - search in this $O(2^n)$ transformed graph
- MT: each city is a source-language word
  - restrictions in reordering can reduce complexity => distortion limit
  - => syntax-based MT

(Held and Karp, 1962; Knight, 1999)
Example: Edit Distance

courtesy of Jason Eisner

- a) given x, y, what is \( p(y|x) \);
- b) what is the most likely seq. of operations?
- c) given x, what is the most likely output y?
- d) given y, what is the most likely input x (with LM)?
Edit Distance can model...

- part-of-speech tagging
- transliteration
- sound-spelling conversion
- word-segmentation
Given $x$ and $y$...

- given $x, y$: a) what is $p(y \mid x)$? (sum of all paths)
  b) what is the most likely conversion path?
Example: General Tagging

Source

Channel

Composition

State

Position in observed string

Store $Q[i,j]$ best score to here

$\Psi[i,j]$ backpointer to best pred

$\alpha[i,j]$ sum of scores to here

“a capital crime”
c) given correct English $x$, what’s the corrupted $y$ with the highest score?

\[
\begin{align*}
\text{X = "ab"} & \\
\begin{array}{c}
a:a \\
b:b \\
\end{array} & \quad + \quad \begin{array}{c}
a:a/0.2 \\
b:b/0.2 \\
\end{array} \\
\text{remove input symbols} & \\
\text{find best path} & \\
\begin{array}{c}
a:A/0.8 \\
b:B/0.8 \\
\end{array}
\end{align*}
\]

so, $\text{argmax } P(y|x) = AB$
DP for “most likely corrupted”

\[ x = "ab" \]

\[ \text{so, } \arg \max_y P(y|x=ab) = AB \]
d) Most Likely “Original Input”

- using an LM $p(e)$ as source model for *spelling correction*
- case 1: letter-based language model $p_L(e)$
- case 2: word-based language model $p_W(e)$

How would dynamic programming work for cases 1/2?
Dynamic Programming for d)

- given $y$, what is the most likely $x$ with $\max p(x) p(y|x)$
Beyond Finite-State Models

- sentence summarization

```
grammar
P(s) -> ...

add optional
P(l | s) -> ...

argmax
S
P(s) * P(l | s)

original short sentence (encourage grammaticality)

particular "lengthening" (encourage addition of optional material only; e.g., P(he didn't go | he did go) = LOW
P(the big dog | the dog) = MED
P(off of the couch | off the couch = HIGH)
```
Beyond Finite-State Models

- headline generation

\[ \text{argmax} \quad h \quad P(h) \cdot P(d|h) \]

looks like a proper headline

if this were a headline, \( d \) would be a reasonable document to go with it (i.e., \( d \) fleshes out \( h \)).
Beyond Finite-State Models

- information retrieval

\[ P(d) \rightarrow \text{document} \rightarrow P(q|d) \rightarrow \text{query} \]

- used to rank documents, not construct new ones!
- query may contain words not in document.
Beyond Finite-State Models

- parsing