NLP Programming Tutorial 6 - Kana-Kanji Conversion

Graham Neubig
Nara Institute of Science and Technology (NAIST)
Formal Model for Kana-Kanji Conversion (KKC)

- In Japanese input, users type in phonetic Hiragana, but proper Japanese is written in logographic Kanji.

- **Kana-Kanji Conversion**: Given an unsegmented Hiragana string $X$, predict its Kanji string $Y$.

  かなかんじへんかんはにほんごにゅうりょくのいちぶ
  
  かな漢字変換は日本語入力の一部

- Also a type of structured prediction, like HMMs or word segmentation.
There are Many Choices!

・ How does the computer tell between good and bad?

Probability model! \[ \text{argmax} \ Y \mid X \]
Remember (from the HMM): Generative Sequence Model

- Decompose probability using Bayes' law

\[
\underset{Y}{\text{argmax}} \ P(Y|X) = \underset{Y}{\text{argmax}} \ \frac{P(X|Y)P(Y)}{P(X)} \\
= \underset{Y}{\text{argmax}} \ P(X|Y) P(Y)
\]

Model of Kana/Kanji interactions
“かんじ” is probably “感じ”

Model of Kanji-Kanji interactions
“漢字” comes after “かな”
Sequence Model for Kana-Kanji Conversion

- Kanji → Kanji language model probabilities
- Bigram model
- Kanji → Kana translation model probabilities

\[ P(Y) \approx \prod_{i=1}^{l+1} P_{LM}(y_i|y_{i-1}) \]

\[ P(X|Y) \approx \prod_{i=1}^{l} P_{TM}(x_i|y_i) \]
Wait! I heard this last week!!!
Differences between POS and Kana-Kanji Conversion

1. Sparsity of $P(y_i|y_{i-1})$:
   - **HMM**: POS $\rightarrow$ POS is not sparse $\rightarrow$ no smoothing
   - **KKC**: Word $\rightarrow$ Word is sparse $\rightarrow$ need smoothing

2. Emission possibilities
   - **HMM**: Considers all word-POS combinations
   - **KKC**: Considers only previously seen combinations

3. Word segmentation:
   - **HMM**: 1 word, 1 POS tag
   - **KKC**: Multiple Hiragana, multiple Kanji
1. Handling Sparsity

- Simple! Just use a smoothed bi-gram model

**Bigram:** \[ P(y_i|y_{i-1}) = \lambda_2 P_{ML}(y_i|y_{i-1}) + (1 - \lambda_2) P(y_i) \]

**Unigram:** \[ P(y_i) = \lambda_1 P_{ML}(y_i) + (1 - \lambda_1) \frac{1}{N} \]

- Re-use your code from Tutorial 2
2. Translation possibilities

- For translation probabilities, use maximum likelihood

\[ P_{TM}(x_i|y_i) = \frac{c(y_i \rightarrow x_i)}{c(y_i)} \]

- Re-use your code from Tutorial 5
- **Implication:** We only need to consider some words

\[
\begin{align*}
  c(\text{感じ} \rightarrow \text{かんじ}) &= 5 \\
  c(\text{漢字} \rightarrow \text{かんじ}) &= 3 \\
  c(\text{幹事} \rightarrow \text{かんじ}) &= 2 \\
  c(\text{トマト} \rightarrow \text{かんじ}) &= 0 \\
  c(\text{奈良} \rightarrow \text{かんじ}) &= 0 \\
  c(\text{監事} \rightarrow \text{かんじ}) &= 0 \\
\end{align*}
\]

\[ \rightarrow \text{Efficient search is possible} \]
3. Words and Kana-Kanji Conversion

- Easier to think of Kana-Kanji conversion using words

We need to do two things:

- Separate Hiragana into words
- Convert Hiragana words into Kanji

We will do these at the same time with the Viterbi algorithm
Search for Kana-Kanji Conversion

I'm back!
Search for Kana-Kanji Conversion

- Use the Viterbi Algorithm
- What does our graph look like?
Search for Kana-Kanji Conversion

- Use the Viterbi Algorithm

```
   1: 化  2: な  3: 化  5: 時  6: 減  8: 化
   1: か  2: 名  3: か  6: 経  8: か
   1: 下  2: 成  3: 下  8: 下
   2: かな  4: 管
   2: 仮名  4: 感
   3: 中
   5: 感じ
   5: 漢字
   8: 変化
   9: 変換
```
Search for Kana-Kanji Conversion

- Use the Viterbi Algorithm

```
0: <S>
1: 書
2: 無
3: 書
4: ん
5: じ
6: へ
7: ん
8: 書
9: ん
1: 化
2: な
3: 化
4: 時
5: 減
6: 経
8: 化
1: か
2: 名
3: か
6: 経
8: か
1: 下
2: 成
3: 下
8: 下
2: 仮名
4: 感
3: 中
5: 感じ
5: 漢字
7: 変
9: 管
8: 変
9: 変換
```
Steps for Viterbi Algorithm

• First, start at 0:<S>
  か な か ん じ へ ん か ん

0:<S> S["0:<S>"] = 0
Search for Kana-Kanji Conversion

- Expand 0 → 1, with all previous states ending at 0

\[
\begin{align*}
S["1: 書"] &= -\log (P_{TM}(か | 書) \cdot P_{LM}(書 |<S>)) + S["0:<S>"]) \\
S["1: 化"] &= -\log (P_{TM}(か | 化) \cdot P_{LM}(化 |<S>)) + S["0:<S>"]) \\
S["1: か"] &= -\log (P_{TM}(か | か) \cdot P_{LM}(か |<S>)) + S["0:<S>"]) \\
S["1: 下"] &= -\log (P_{TM}(か | 下) \cdot P_{LM}(下 |<S>)) + S["0:<S>"]) \\
\end{align*}
\]
Search for Kana-Kanji Conversion

- Expand 0 → 2, with all previous states ending at 0

\[
S["0:<S>"] = -\log (P_E(\text{かな}|\text{かな}) \times P_{LM}(\text{かな}|<S>)) + S["0:<S>"]
\]

\[
S["1: \text{かな}"] = -\log (P_E(\text{かな}|\text{かな}) \times P_{LM}(\text{かな}|<S>)) + S["0:<S>"]
\]

\[
S["1: \text{仮名}"] = -\log (P_E(\text{かな}|\text{仮名}) \times P_{LM}(\text{仮名}|<S>)) + S["0:<S>"]
\]
Search for Kana-Kanji Conversion

• Expand 1 → 2, with all previous states ending at 1

\[
\begin{align*}
S[“2: 無”] &= \min \\
&= -\log(P_E(な | 無) * P_{LM}(無 | 書)) + S[“1: 書”], \\
&\quad -\log(P_E(な | 無) * P_{LM}(無 | 化)) + S[“1: 化”], \\
&\quad -\log(P_E(な | 無) * P_{LM}(無 | か)) + S[“1: か”], \\
&\quad -\log(P_E(な | 無) * P_{LM}(無 | 下)) + S[“1: 下”] \\
\end{align*}
\]

\[
\begin{align*}
S[“2: な”] &= \min \\
&= -\log(P_E(な | な) * P_{LM}(な | 書)) + S[“1: 書”], \\
&\quad -\log(P_E(な | な) * P_{LM}(な | 化)) + S[“1: 化”], \\
&\quad -\log(P_E(な | な) * P_{LM}(な | か)) + S[“1: か”], \\
&\quad -\log(P_E(な | な) * P_{LM}(な | 下)) + S[“1: 下”] \\
\end{align*}
\]
Algorithm
Overall Algorithm

```
load lm                      # Same as tutorials 2
load tm                      # Similar to tutorial 5
# Structure is tm[pron][word] = prob

for each line in file
    do forward step          # Same as tutorial 5
    do backward step
print results                # Same as tutorial 5
```
Implementation: Forward Step

```python
edge[0]["<s>"] = NULL, score[0]["<s>"] = 0
for end in 1 .. len(line)
    create map my_edges
    for begin in 0 .. end - 1
        pron = substring of line from begin to end
        my_tm = tm_probs[pron]
        if there are no candidates and len(pron) == 1
            my_tm = (pron, 0)
        for curr_word, tm_prob in my_tm
            for prev_word, prev_score in score[begin]
                # Find the current score
                curr_score = prev_score + -log(tm_prob * P_LM(curr_word | prev_word))
                if curr_score is better than score[end][curr_word]
                    score[end][curr_word] = curr_score
                    edge[end][curr_word] = (begin, prev_word)
```

# For each ending point
# For each beginning point
# Find the hiragana
# Find words/TM probs for pron
# Map hiragana as-is
# For possible current words
# For all previous words/probs
Exercise
Exercise

- **Write** `kkc.py` and **re-use** `train-bigram.py`, `train-hmm.py`

- **Test** the program
  
  - `train-bigram.py test/06-word.txt > lm.txt`
  - `train-hmm.py test/06-pronword.txt > tm.txt`
  - `kkc.py lm.txt tm.txt test/06-pron.txt > output.txt`
  
  - **Answer:** `test/06-pronword.txt`
Exercise

• **Run** the program
  - `train-bigram.py data/wiki-ja-train.word > lm.txt`
  - `train-hmm.py data/wiki-ja-train.pronword > tm.txt`
  - `kkc.py lm.txt tm.txt data/wiki-ja-test.pron > output.txt`

• **Measure** the accuracy of your tagging with
  `06-kkc/gradekkc.pl data/wiki-ja-test.word output.txt`

• **Report** the accuracy (F-meas)

• **Challenge:**
  - Find a larger corpus or dictionary, run KyTea to get the pronunciations, and train a better model
Thank You!