自然言語処理プログラミング勉強会 6 - かな漢字変換

Graham Neubig
奈良先端科学技術大学院大学 (NAIST)

かな漢字変換のモデル

- 日本語入力でひらがな列 X をかな漢字混じり文 Y へ変換

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

- HMM や単語分割と同じく、構造化予測の一部

選択肢が膨大！

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　良い！
仮名漢字変換は日本語入力の一部　良い？
かな漢字変換は二本後入力の一部　悪い
家中ん事変感歯に□　御乳力の胃治舞　?!?!

…

- 良い候補と悪い候補を区別するには？

確率モデル！　$\underset{Y}{\operatorname{argmax}} P(Y|X)$

系列生成モデル（HMM と同じ！）

- 確率をベイズ則で分解

$$\underset{Y}{\operatorname{argmax}}\, P(Y|X) = \underset{Y}{\operatorname{argmax}} \frac{P(X|Y)P(Y)}{P(X)}$$
$$= \underset{Y}{\operatorname{argmax}}\, P(X|Y)P(Y)$$

かなと漢字の関係を記述
「かんじ」は「感じ」になりやすい

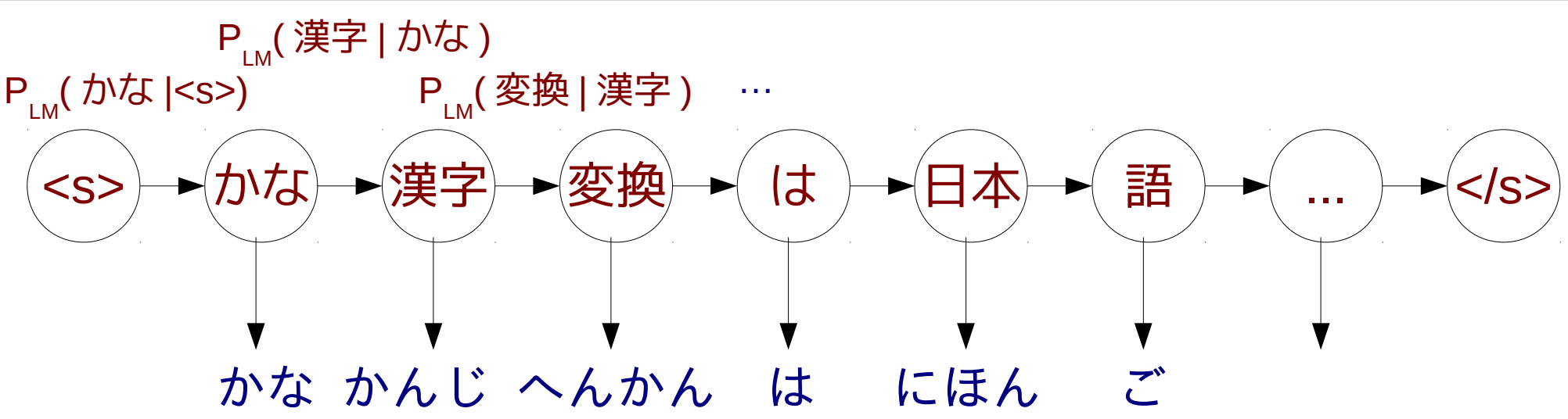
前の漢字と次の漢字の関係を記述
「漢字」は「仮名」に続きやすい

かな漢字変換の系列モデル

- 漢字→漢字の言語モデル確率
  - 2-gram モデル
- 漢字→かなの変換モデル確率

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

$$P(X|Y) \approx \prod_{1}^{I} P_{TM}(x_i | y_i)$$



P_{TM}(かな | かな) * P_{TM}(かんじ | 漢字) * P_{TM}(へんかん | 変換) …

系列生成モデル

生成・翻訳モデル確率

先週聞いた話と同じ！

遷移・言語モデル確率

構造化予測

品詞推定 (HMM) とかな漢字変換 (KKC)

- 1. $P(y_i|y_{i-1})$ の確率はスパース（疎）：
  - **HMM:** 品詞→品詞はスパースでない（平滑化なし）
  - **KKC:** 単語→単語はスパース（平滑化あり）
- 2. 生成確率
  - **HMM:** 全ての単語・品詞組み合わせを考慮
  - **KKC:** 学習データに現れる組み合わせのみを考慮
- 3. 単語分割
  - **HMM:** 1 単語→ 1 品詞
  - **KKC:** 複数のひらがな→複数の漢字

1. スパースな確率の扱い

- 扱いは簡単：平滑化された 2-gram モデルを利用

2-gram:
$$P(y_i|y_{i-1}) = \lambda_2 P_{ML}(y_i|y_{i-1}) + (1-\lambda_2) P(y_i)$$

1-gram:
$$P(y_i) = \lambda_1 P_{ML}(y_i) + (1-\lambda_1) \frac{1}{N}$$

- チュートリアル２の確率を再利用

2. 考慮する変換候補

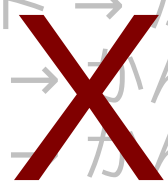
- 翻訳確率は最尤推定

$$P_{TM}(x_i|y_i) = c(y_i \rightarrow x_i)/c(y_i)$$

  - チュートリアル５のコードを再利用
- つまり、学習データに現れるもののみを考慮

c(感じ → かんじ) = 5
c(漢字 → かんじ) = 3
c(幹事 → かんじ) = 2

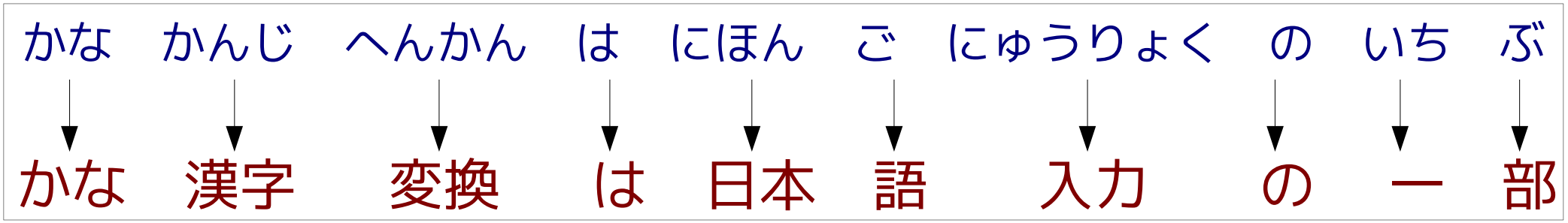
c(トマト → かんじ) = 0
c(奈良 → かんじ) = 0
c(監事 → かんじ) = 0
…



→ 効率的な探索が可能

3. かな漢字変換と単語

- かな漢字変換を「単語」で考えるのが直感的



- ２つの動作が必要：
  - ひらがなを単語へ分割
  - ひらがな単語を漢字へ変換
- この２つの動作をビタビアルゴリズムで同時に行う

かな漢字変換の探索

かな漢字変換の探索

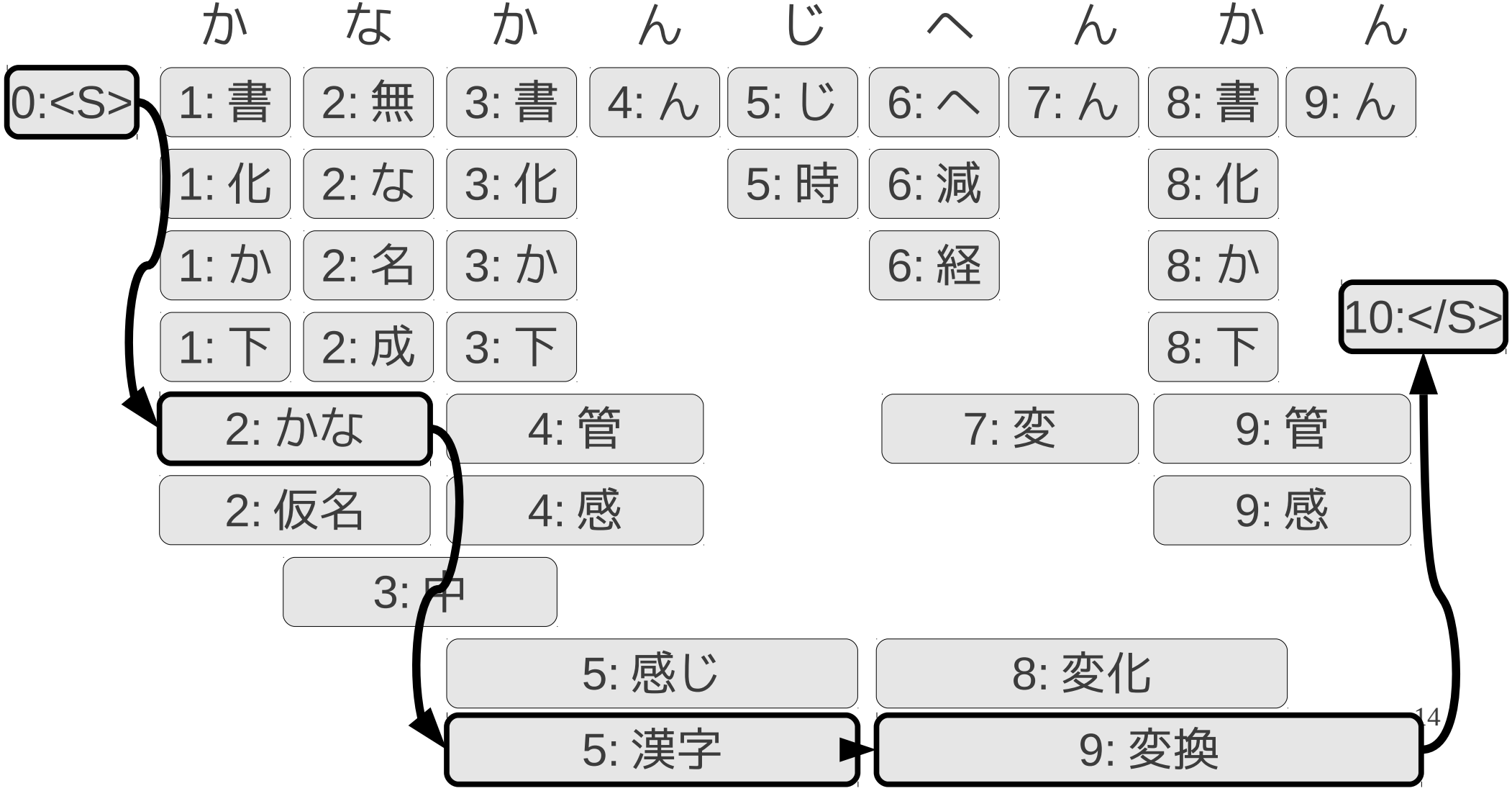
- ビタビアルゴリズムを利用
- グラフの形は？

かな漢字変換の探索

- ビタビアルゴリズムを利用

かな漢字変換の探索

- ビタビアルゴリズムを利用



かな漢字変換の探索

- 0:<S> で探索開始

か　な　か　ん　じ　へ　ん　か　ん

0:<S>　S[“0:<S>”] = 0

かな漢字変換の探索

- 0→1 のスパンをまたがる単語を全て展開

か　な　か　ん　じ　へ　ん　か　ん

0:<S>

1: 書　S[“1: 書”] = -log (P_{TM}(か | 書) * P_{LM}(書 |<S>)) + S[“0:<S>”]

1: 化　S[“1: 化”] = -log (P_{TM}(か | 化) * P_{LM}(化 |<S>)) + S[“0:<S>”]

1: か　S[“1: か”] = -log (P_{TM}(か | か) * P_{LM}(か |<S>)) + S[“0:<S>”]

1: 下　S[“1: 下”] = -log (P_{TM}(か | 下) * P_{LM}(下 |<S>)) + S[“0:<S>”]

かな漢字変換の探索

- 0→2 のスパンをまたがる単語を全て展開

か　な　か　ん　じ　へ　ん　か　ん

0:<S>

1: 書

1: 化

1: か

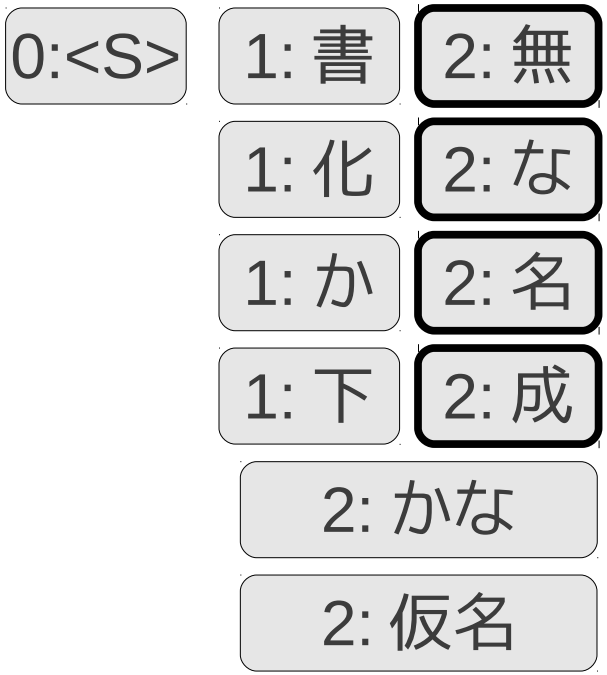
1: 下

2: かな　S[“1: かな”] = -log (P_E(かな | かな) * P_{LM}(かな |<S>)) + S[“0:<S>”]

2: 仮名　S[“1: 仮名”] = -log (P_E(かな | 仮名) * P_{LM}(仮名 |<S>)) + S[“0:<S>”]

かな漢字変換の探索

- 1→2 のスパンをまたがる単語を全て展開



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

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

■ ■ ■

アルゴリズム

アルゴリズムの全体像

```
load lm                          #  チュートリアル 2 と同じ
load tm                          #  チュートリアル 5 と同じ
                                 # tm[pron][word] = prob の形で格納

for each line in file
    do forward step
    do backward step             #  チュートリアル 5 と同じ
    print results                #  チュートリアル 5 と同じ
```

実装：前向きステップ

```
edge[0]["<s>"] = NULL, score[0]["<s>"] = 0
for end in 1 .. len(line)                          # 単語の終了点
    score[end] = {}
    edge[end] = {}
    for begin in 0 .. end – 1                      # 単語の開始点
        pron = substring of line from begin to end # ひらがなの部分文字列を獲得
        my_tm = tm_probs[pron]                     # ひらがなの単語・変換確率を
        if there are no candidates and len(pron) == 1
            my_tm = (pron, 0)                      # 未知語ならひらがなをそのまま
        for curr_word, tm_prob in my_tm            # 可能な単語を列挙
            for prev_word, prev_score in score[begin] # 前の単語とその確率に対して
                # 次のスコアを計算
                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)
```

演習課題

演習課題

- かな漢字変換プログラム kkc.py を作成
  train-bigram.py と train-hmm.py を再利用
- テスト：
  - `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`
  - 正解：`test/06-pronword.txt`

演習課題

- プログラムを実行
  - `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`
- 変換精度を評価
  `script/gradekkc.pl data/wiki-ja-test.word output.txt`
- 精度を報告（F 値）
- 上級編：
  - 最も頻繁に起こる誤りを分析する
  - 大きなコーパスに対して KyTea を使って読みを推定し、大きなデータによる変換精度の向上を考察する