

自然言語処理プログラミング勉強会 5 - 隠れマルコフモデルによる品詞推定

Graham Neubig
奈良先端科学技術大学院大学 (NAIST)

品詞推定

- 文 X が与えられた時の品詞列 Y を予測する

Natural language processing (NLP) is a field of computer science

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

JJ NN NN -LRB- NN -RRB- VBZ DT NN IN NN NN

- 予測をどうやって行うか？

タグ付けの確率モデル

- 文が与えられた場合、最も確率の高いタグ列を計算

Natural language processing (NLP) is a field of computer science

JJ NN NN LRB NN RRB VBZ DT NN IN NN NN

$$\operatorname{argmax}_Y P(Y|X)$$

- これをどうやってモデル化？

系列に対する生成モデル

- ベイズ則で確率を分解

$$\begin{aligned}\operatorname{argmax}_Y P(Y|X) &= \operatorname{argmax}_Y \frac{P(X|Y)P(Y)}{P(X)} \\ &= \operatorname{argmax}_Y P(X|Y)P(Y)\end{aligned}$$

単語と品詞の関係を考慮

「natural」はたぶん形容詞 (JJ) 名詞 (NN) が限定詞 (DET) に続く

前の品詞と次の品詞の関係を考慮

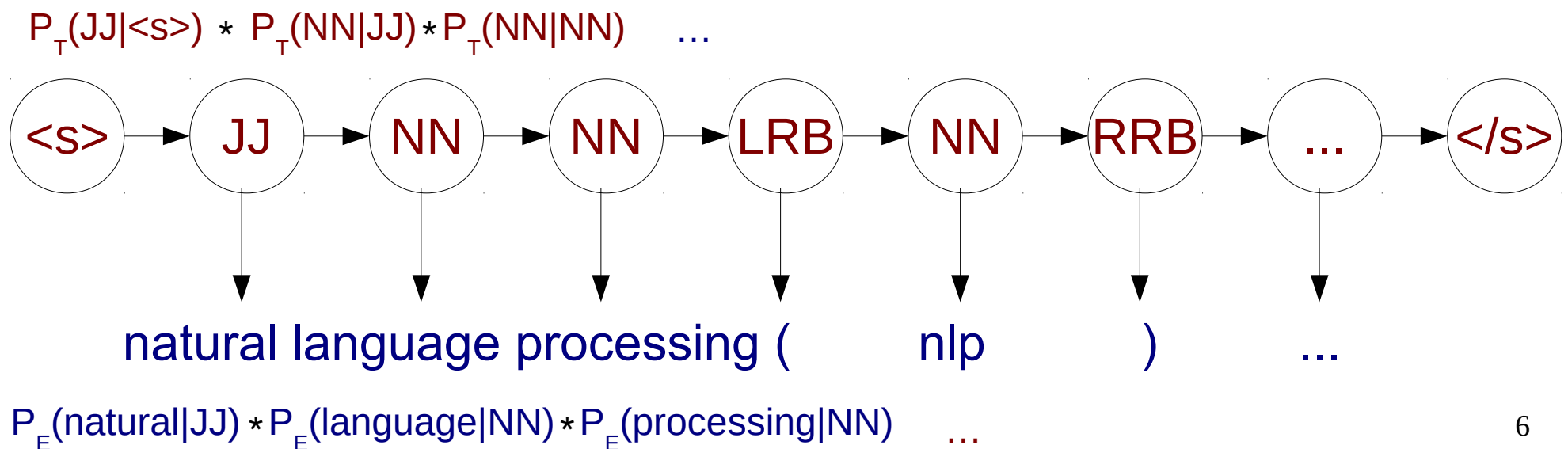
隠れマルコフモデル (HMM)

品詞推定のための (HMM)

- 品詞→品詞の遷移確率
 - 2-gram モデルとほぼ一緒
- 品詞→単語の生成確率

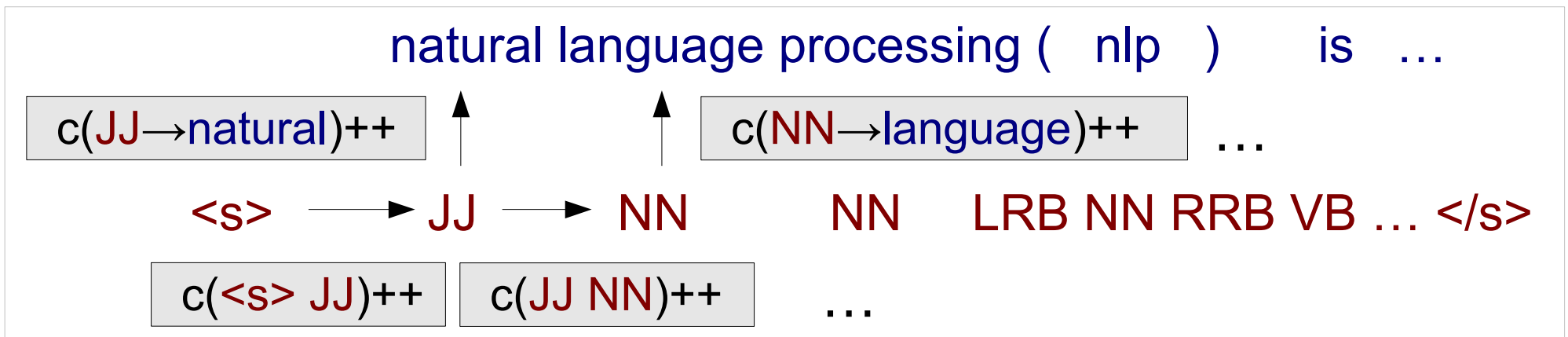
$$P(Y) \approx \prod_{i=1}^{l+1} P_T(y_i | y_{i-1})$$

$$P(X|Y) \approx \prod_1^l P_E(x_i | y_i)$$



タグ付きコーパスからの HMM 学習

- コーパス中の頻度を数え上げ、



- 文脈の頻度で割ることで確率を求める

$$P_T(\text{LRB}|\text{NN}) = c(\text{NN LRB})/c(\text{NN}) = 1/3$$

$$P_E(\text{language}|\text{NN}) = c(\text{NN} \rightarrow \text{language})/c(\text{NN}) = 1/3$$

学習アルゴリズム

```

# 入力データ形式は 「 natural_JJ language_NN ... 」
make a map emit, transition, context
for each line in file
    previous = "<s>" # 文頭記号
    context[previous]++
    split line into wordtags with " "
    for each wordtag in wordtags
        split wordtag into word, tag with "_"
        transition[previous+" "+tag]++ # 遷移を数え上げる
        context[tag]++ # 文脈を数え上げる
        emit[tag+" "+word]++ # 生成を数え上げる
        previous = tag
    transition[previous+" </s>"]++
# 遷移確率を出力
for each key, value in transition
    split key into previous, word with " "
    print "T", key, value/context[previous]
# 同じく生成確率を出力 ( 「T」 ではなく 「E」 を付与)

```


平滑化

- 2-gram モデルで平滑化を用いた：

$$P_{LM}(w_i|w_{i-1}) = \lambda P_{ML}(w_i|w_{i-1}) + (1-\lambda) P_{LM}(w_i)$$

- **HMM 遷移確率**：タグの数は少ないので平滑化は不要

$$P_T(y_i|y_{i-1}) = P_{ML}(y_i|y_{i-1})$$

- **HMM 生成確率**：未知語を扱うために平滑化が必要

$$P_E(x_i|y_i) = \lambda P_{ML}(x_i|y_i) + (1-\lambda) 1/N$$

品詞推定の探索

マルコフモデルを使った品詞推定

- やはりビタビアルゴリズムを利用

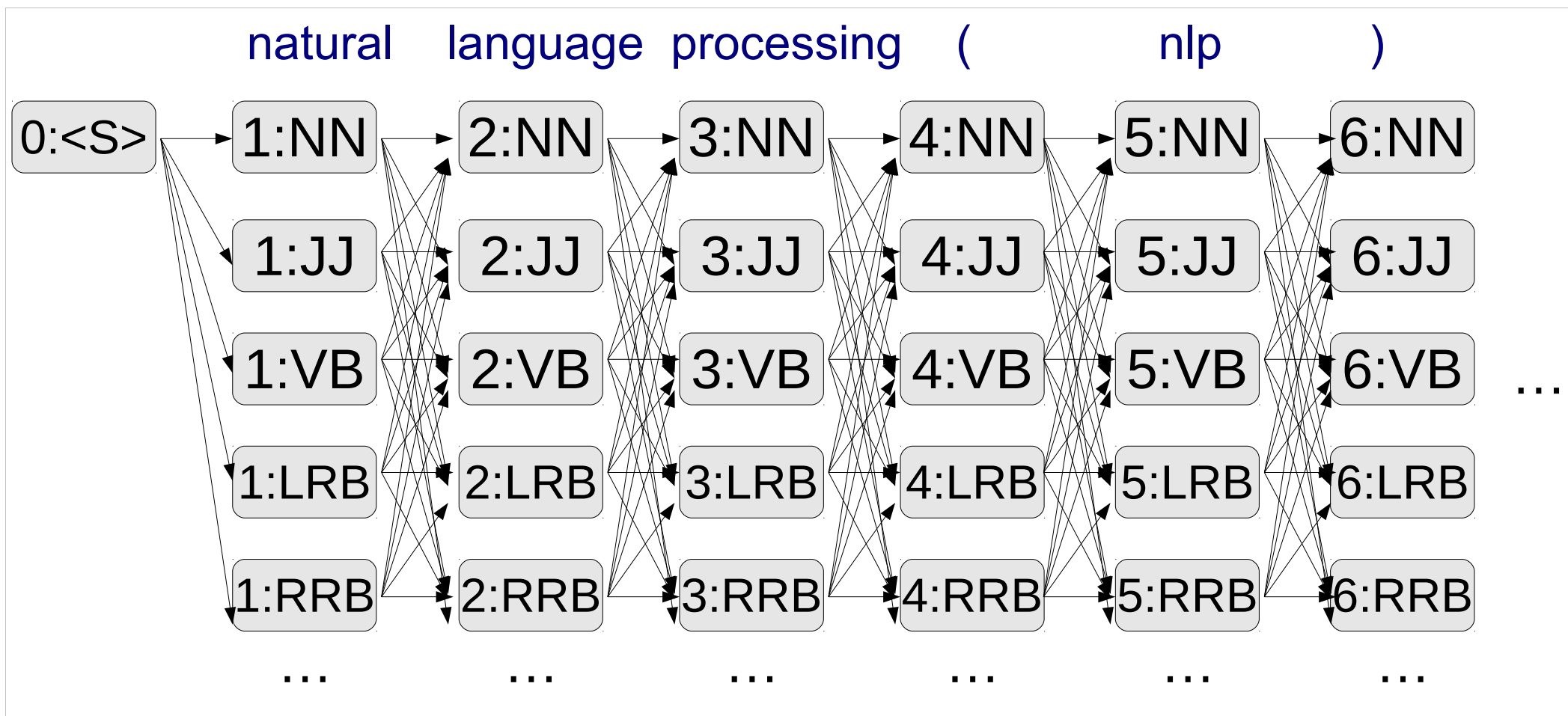
重要だと言っただろう！



- 品詞推定の探索グラフの形は？

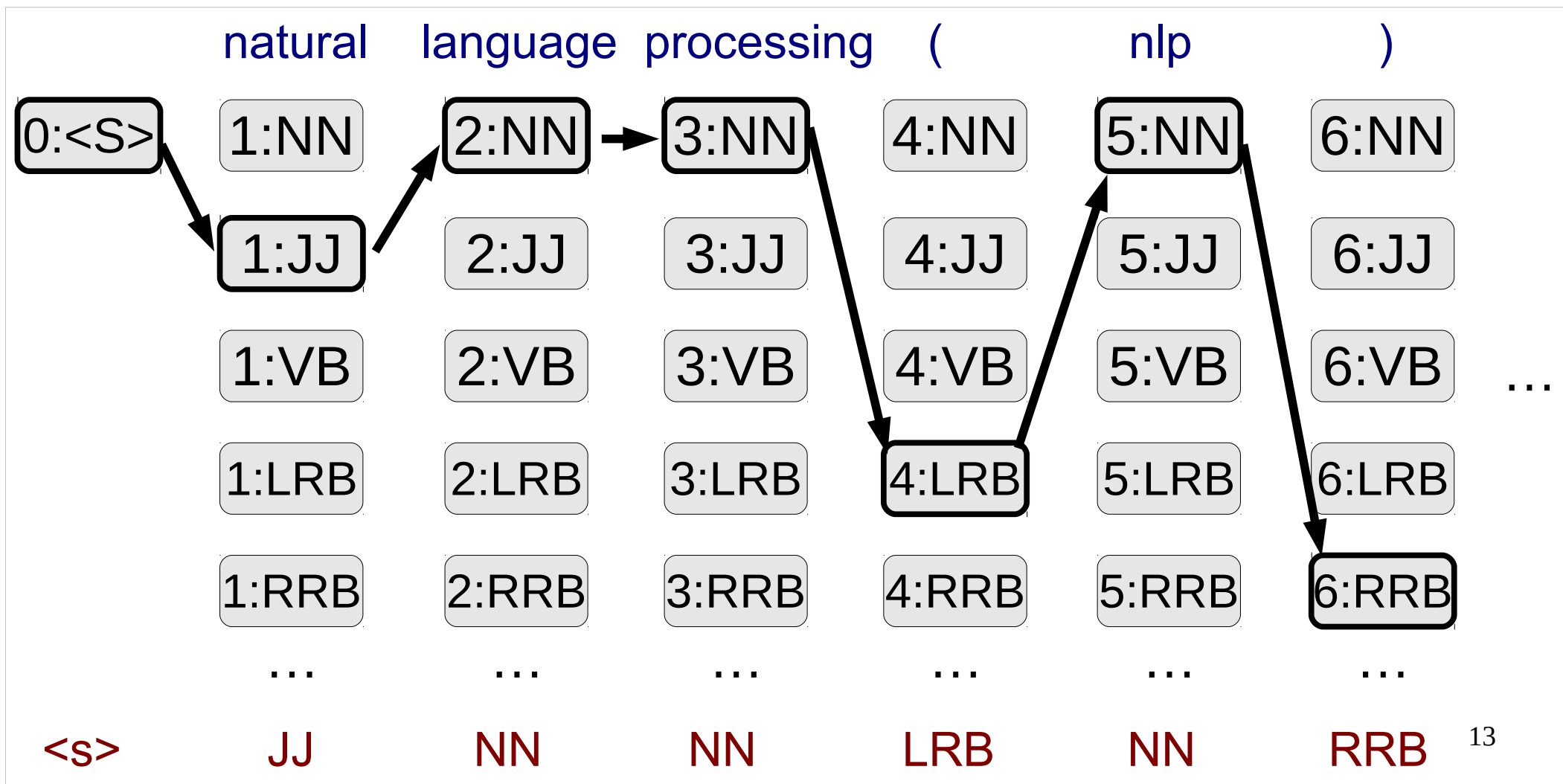
HMM 品詞推定のグラフ

- 品詞推定の探索グラフの形：



HMM 品詞推定のグラフ

- 各パスは品詞列を表す



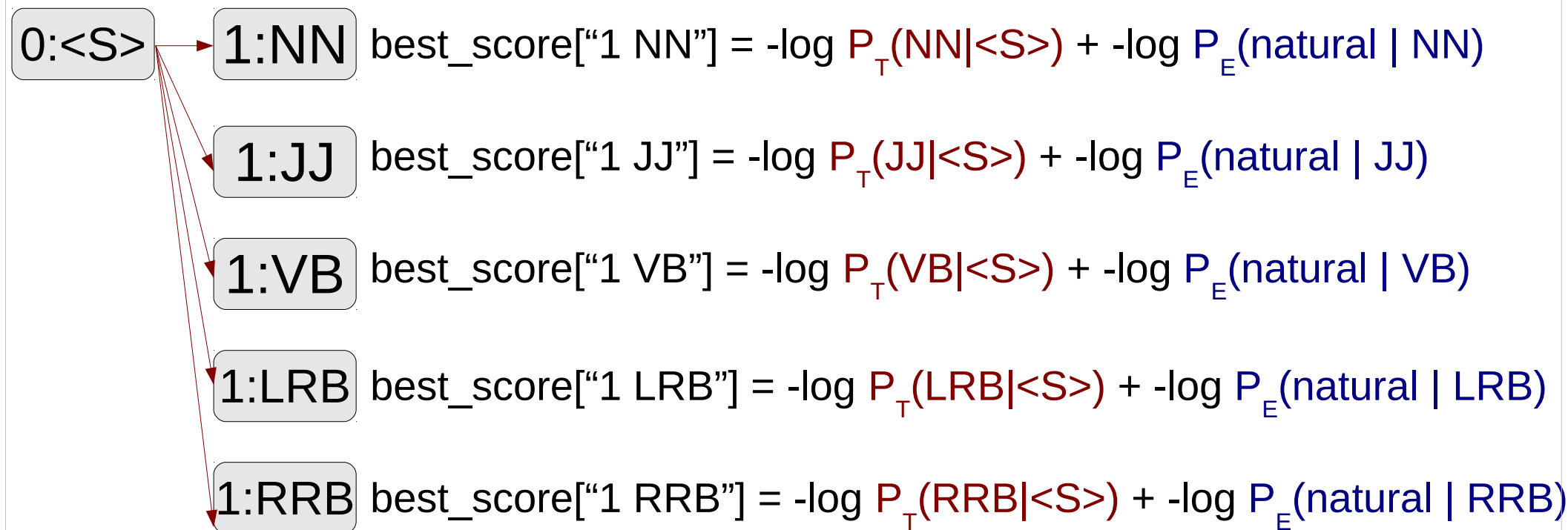
復習：ビタビアルゴリズムのステップ

- 前向きステップ：各ノードへたどる確率の計算
 - 負の対数尤度がもっとも低くなるパス
- 後ろ向きステップ：パスの復元
 - 単語分割とほとんど同じ

前向きステップ：文頭

- 文頭記号 <S> から 1 単語目への遷移と 1 単語目の生成の確率

natural



...

前向きステップ：中間

- 前の品詞を全部比べて、**これまでのパス**、**遷移**、**生成**を全て考慮した最短パスを利用

natural language

1:NN → 2:NN

1:JJ → 2:JJ

1:VB → 2:VB

1:LRB → 2:LRB

1:RRB → 2:RRB

...

...

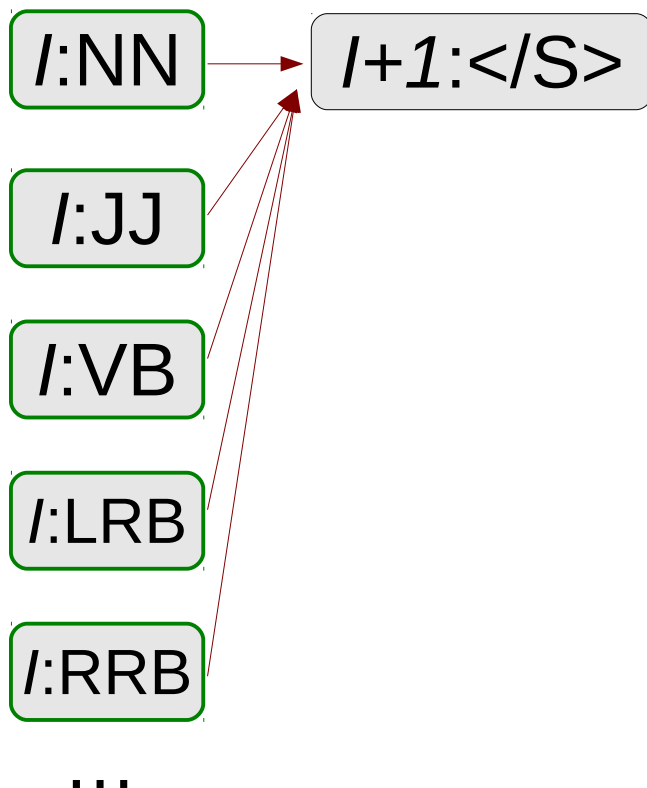
$$\begin{aligned} \text{best_score}["2 \text{ NN}"] = \min(\\ & \text{best_score}["1 \text{ NN}"] + -\log P_T(\text{NN}|\text{NN}) + -\log P_E(\text{language} | \text{NN}), \\ & \text{best_score}["1 \text{ JJ}"] + -\log P_T(\text{NN}|\text{JJ}) + -\log P_E(\text{language} | \text{NN}), \\ & \text{best_score}["1 \text{ VB}"] + -\log P_T(\text{NN}|\text{VB}) + -\log P_E(\text{language} | \text{NN}), \\ & \text{best_score}["1 \text{ LRB}"] + -\log P_T(\text{NN}|\text{LRB}) + -\log P_E(\text{language} | \text{NN}), \\ & \text{best_score}["1 \text{ RRB}"] + -\log P_T(\text{NN}|\text{RRB}) + -\log P_E(\text{language} | \text{NN}), \\ & \dots \\ &) \end{aligned}$$

$$\begin{aligned} \text{best_score}["2 \text{ JJ}"] = \min(\\ & \text{best_score}["1 \text{ NN}"] + -\log P_T(\text{JJ}|\text{NN}) + -\log P_E(\text{language} | \text{JJ}), \\ & \text{best_score}["1 \text{ JJ}"] + -\log P_T(\text{JJ}|\text{JJ}) + -\log P_E(\text{language} | \text{JJ}), \\ & \text{best_score}["1 \text{ VB}"] + -\log P_T(\text{JJ}|\text{VB}) + -\log P_E(\text{language} | \text{JJ}), \\ & \dots \\ &) \end{aligned}$$

前向きステップ：文末

- 文末記号への遷移を考慮して終わり

science



$$\begin{aligned} \text{best_score}["/ + 1 </S>"] = \min(\\ & \text{best_score}["/ NN"] + -\log P_T(</S>|NN), \\ & \text{best_score}["/ JJ"] + -\log P_T(</S>|JJ), \\ & \text{best_score}["/ VB"] + -\log P_T(</S>|VB), \\ & \text{best_score}["/ LRB"] + -\log P_T(</S>|LRB), \\ & \text{best_score}["/ NN"] + -\log P_T(</S>|RRB), \\ & \dots \\) \end{aligned}$$

実装：モデル読み込み

make a map for *transition*, *emission*, *possible_tags*

for each *line* in *model_file*

split *line* into *type*, *context*, *word*, *prob*

possible_tags[*context*] = 1 # 可能なタグとして保存

if *type* = "T"

transition["*context word*"] = *prob*

else

emission["*context word*"] = *prob*

実装：前向きステップ

split line into words

$l = \text{length}(\text{words})$

make maps best_score , best_edge

$\text{best_score}["0 \langle s \rangle"] = 0$ # $\langle s \rangle$ から始まる

$\text{best_edge}["0 \langle s \rangle"] = \text{NULL}$

for i in $0 \dots l-1$:

for each prev in keys of possible_tags

for each next in keys of possible_tags

if $\text{best_score}["i \text{ prev}"]$ **and** $\text{transition}["\text{prev next}"]$ **exist**

$\text{score} = \text{best_score}["i \text{ prev}] +$

$-\log P_T(\text{next}|\text{prev}) + -\log P_E(\text{word}[i]|\text{next})$

if $\text{best_score}["i+1 \text{ next}"]$ **is new or** $< \text{score}$

$\text{best_score}["i+1 \text{ next}] = \text{score}$

$\text{best_edge}["i+1 \text{ next}] = "i \text{ prev}"$

最後、 $\langle /s \rangle$ に対して同じ操作を行う

実装：後ろ向きステップ

```
tags = []
next_edge = best_edge[ "/+1 </s>" ]
while next_edge != "0 <s>"
    # このエッジの品詞を出力に追加
    split next_edge into position, tag
    append tag to tags
    next_edge = best_edge[ next_edge ]
tags.reverse()
join tags into a string and print
```

演習問題

演習問題

- train-hmm と test-hmm を実装
- テスト：
 - 入力： test/05-`{train,test}`-input.txt
 - 正解： test/05-`{train,test}`-answer.txt
- data/wiki-en-train.norm_pos を使ってモデルを学習し、 data/wiki-en-test.norm に対して品詞推定を行う
- 品詞推定の性能を評価して報告：
script/gradeupos.pl data/wiki-en-test.pos my_answer.pos
- 上級編：精度を向上させる方法を考える