NLP Programming Tutorial 5 - Part of Speech Tagging with Hidden Markov Models

Graham Neubig
Nara Institute of Science and Technology (NAIST)

Part of Speech (POS) Tagging

- Given a sentence X, predict its part of speech sequence Y

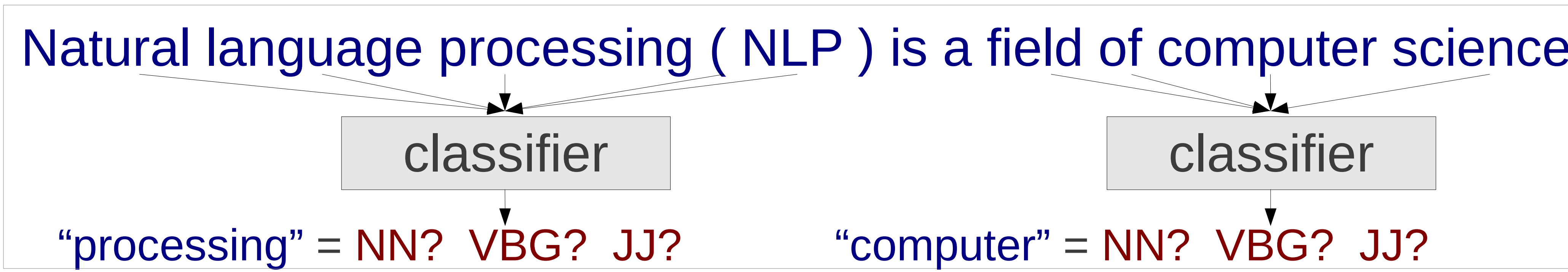
Natural language processing (NLP) is a field of computer science

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

- A type of "structured" prediction, from two weeks ago
- How can we do this? Any ideas?

Many Answers!

- Pointwise prediction: predict each word individually with a classifier (e.g. perceptron, tool: KyTea)



- Generative sequence models: todays topic! (e.g. Hidden Markov Model, tool: ChaSen)
- Discriminative sequence models: predict whole sequence with a classifier (e.g. CRF, structured perceptron, tool: MeCab, Stanford Tagger)

Probabilistic Model for Tagging

- “Find the most probable tag sequence, given the sentence”

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

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

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

- Any ideas?

Generative Sequence Model

- First decompose probability using Bayes' law

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

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

Model of word/POS interactions
"natural" is probably a JJ

Model of POS/POS interactions
NN comes after DET

- Also sometimes called the "noisy-channel model"

Hidden Markov Models

Hidden Markov Models (HMMs) for POS Tagging

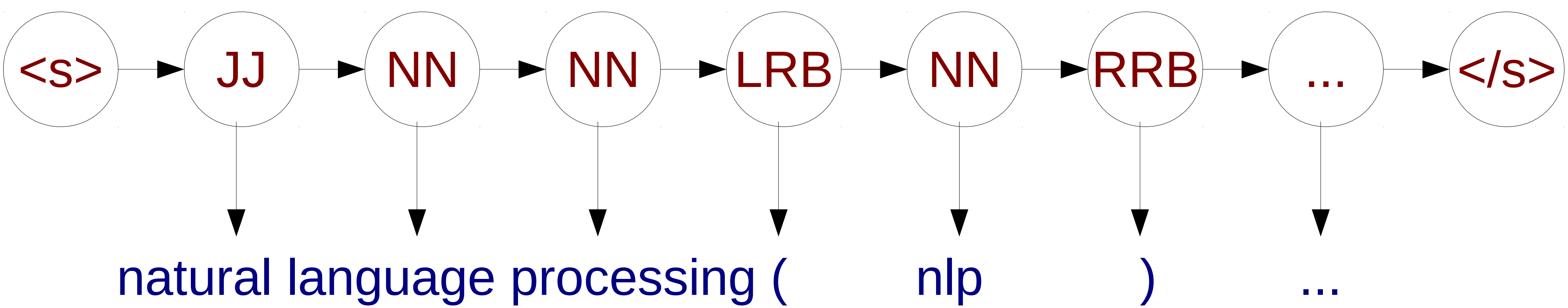
- POS→POS **transition** probabilities
  - Like a bigram model!

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

- POS→Word **emission** probabilities

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

$P_T(JJ|<s>) * P_T(NN|JJ) * P_T(NN|NN)$ …



$P_E(natural|JJ) * P_E(language|NN) * P_E(processing|NN)$ …

Learning Markov Models (with tags)

- Count the number of occurrences in the corpus and

natural language processing (nlp) is …

c(JJ→natural)++ c(NN→language)++ …

<s> → JJ → NN NN LRB NN RRB VB … </s>

c(<s> JJ)++ c(JJ NN)++ …

- Divide by context to get probability

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

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

Training Algorithm

```
# Input data format is "natural_JJ language_NN …"
make a map emit, transition, context
for each line in file
    previous = "<s>"                        # Make the sentence start
    context[previous]++
    split line into wordtags with " "
    for each wordtag in wordtags
        split wordtag into word, tag with "_"
        transition[previous+" "+tag]++      # Count the transition
        context[tag]++                      # Count the context
        emit[tag+" "+word]++                # Count the emission
        previous = tag
    transition[previous+" </s>"]++
# Print the transition probabilities
for each key, value in transition
    split key into previous, word with " "
    print "T", key, value/context[previous]
# Do the same thing for emission probabilities with "E"
```

Note: Smoothing

- In bigram model, we smoothed probabilities

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

- HMM transition prob.: there are not many tags, so smoothing is not necessary

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

- HMM emission prob.: smooth for unknown words

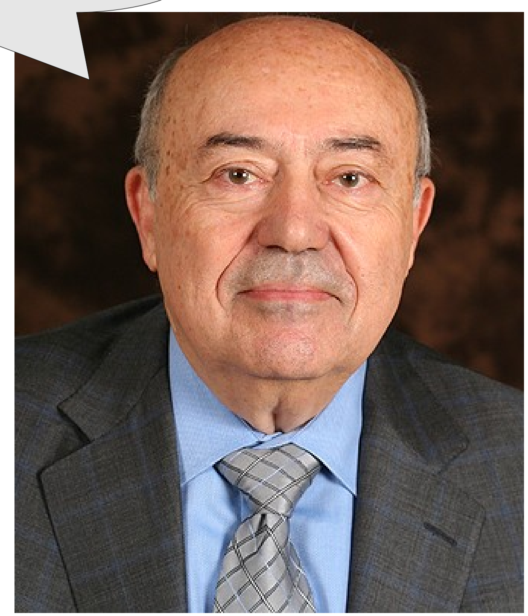
$$P_E(x_i|y_i) = \lambda\ P_{ML}(x_i|y_i) + (1-\lambda)\ 1/N$$

Finding POS Tags

Finding POS Tags with Markov Models

- Use the Viterbi algorithm again!!

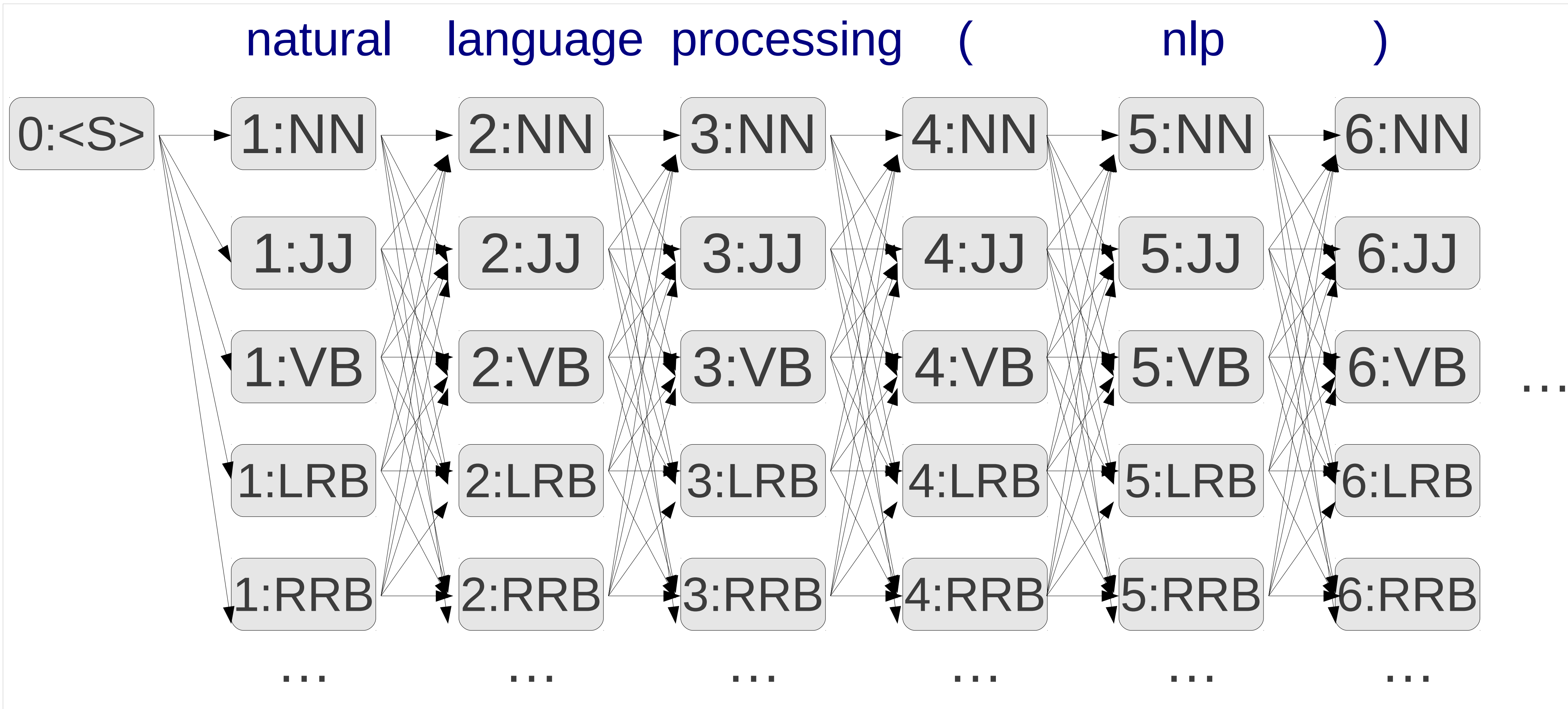
I told you I
was important!!



- What does our graph look like?

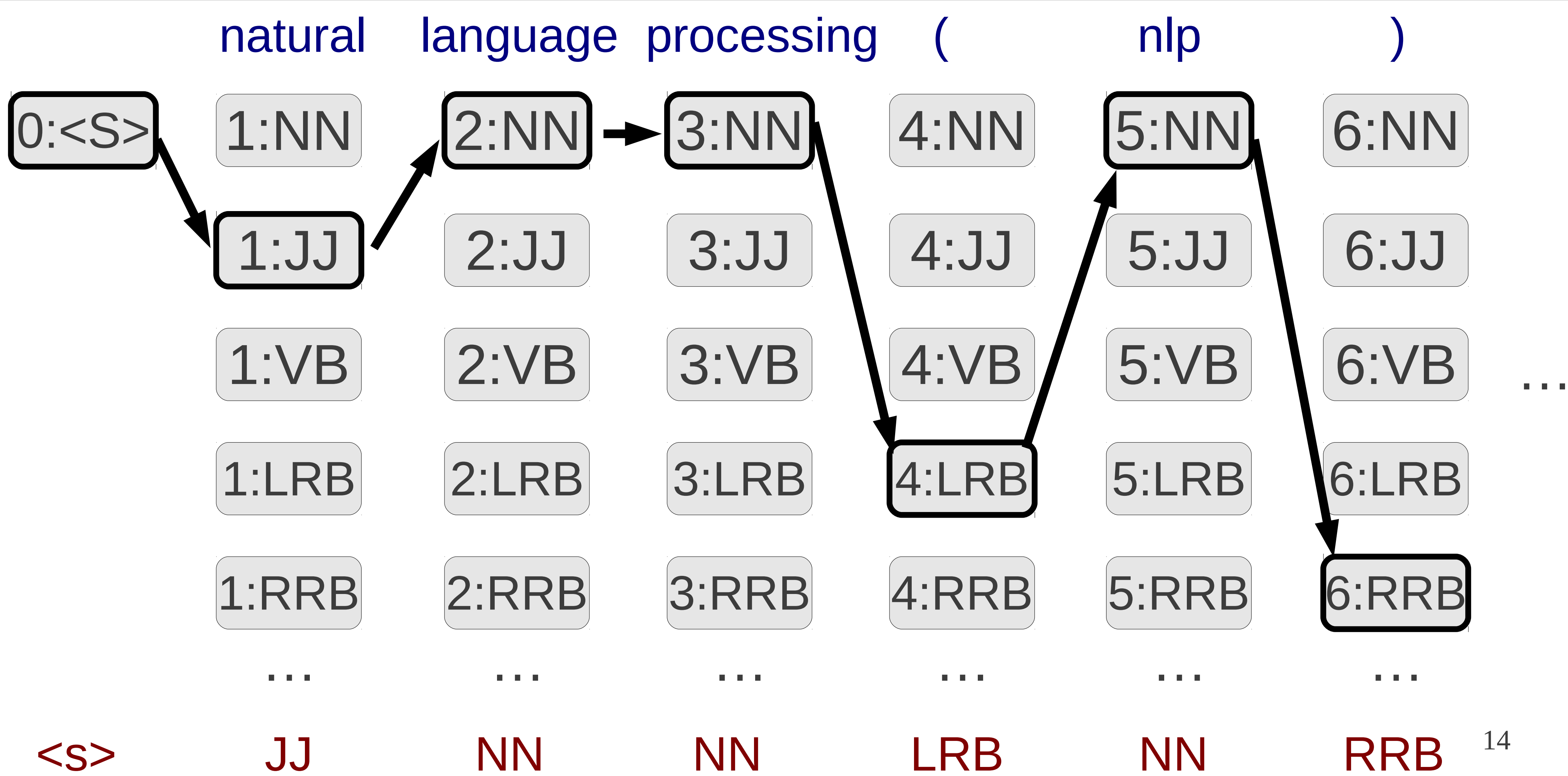
Finding POS Tags with Markov Models

- What does our graph look like? Answer:



Finding POS Tags with Markov Models

- The best path is our POS sequence



Remember: Viterbi Algorithm Steps

- Forward step, calculate the best path to a node
  - Find the path to each node with the lowest negative log probability
- Backward step, reproduce the path
  - This is easy, almost the same as word segmentation

Forward Step: Part 1

- First, calculate transition from <S> and emission of the first word for every POS

natural

0:<S>

1:NN best_score[“1 NN”] = -log $P_T(NN|<S>)$ + -log $P_E(natural | NN)$

1:JJ best_score[“1 JJ”] = -log $P_T(JJ|<S>)$ + -log $P_E(natural | JJ)$

1:VB best_score[“1 VB”] = -log $P_T(VB|<S>)$ + -log $P_E(natural | VB)$

1:LRB best_score[“1 LRB”] = -log $P_T(LRB|<S>)$ + -log $P_E(natural | LRB)$

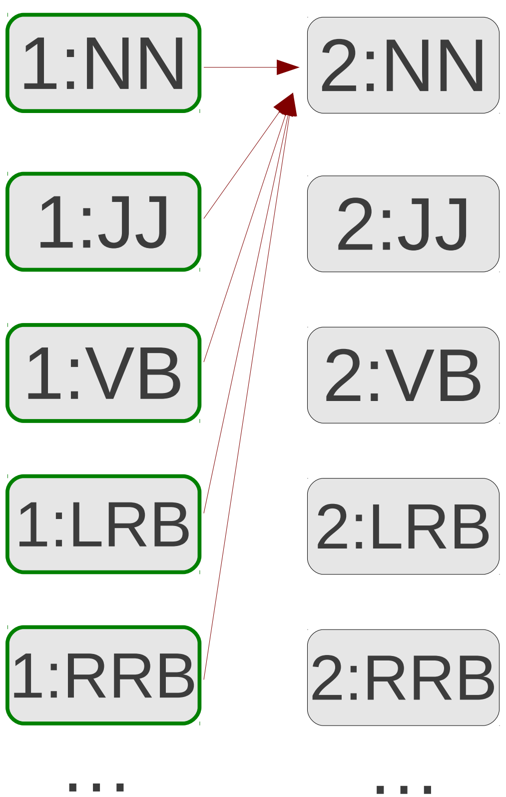
1:RRB best_score[“1 RRB”] = -log $P_T(RRB|<S>)$ + -log $P_E(natural | RRB)$

…

Forward Step: Middle Parts

- For middle words, calculate the minimum score for all possible previous POS tags

natural language



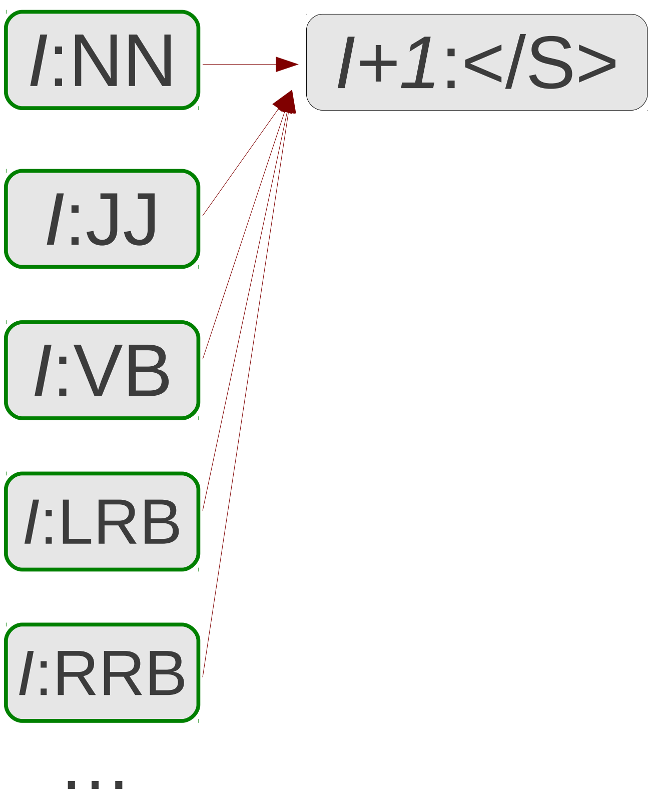
best_score["2 NN"] = min(
 best_score["1 NN"] + -log P_T(NN|NN) + -log P_E(language | NN),
 best_score["1 JJ"] + -log P_T(NN|JJ) + -log P_E(language | NN),
 best_score["1 VB"] + -log P_T(NN|VB) + -log P_E(language | NN),
 best_score["1 LRB"] + -log P_T(NN|LRB) + -log P_E(language | NN),
 best_score["1 RRB"] + -log P_T(NN|RRB) + -log P_E(language | NN),
 ...
)

best_score["2 JJ"] = min(
 best_score["1 NN"] + -log P_T(JJ|NN) + -log P_E(language | JJ),
 best_score["1 JJ"] + -log P_T(JJ|JJ) + -log P_E(language | JJ),
 best_score["1 VB"] + -log P_T(JJ|VB) + -log P_E(language | JJ),
 ...

Forward Step: Final Part

- Finish up the sentence with the sentence final symbol

science



best_score["*I+1* </S>"] = min(
  best_score["*I* NN"] + -log P_T(</S>|NN),
  best_score["*I* JJ"] + -log P_T(</S>|JJ),
  best_score["*I* VB"] + -log P_T(</S>|VB),
  best_score["*I* LRB"] + -log P_T(</S>|LRB),
  best_score["*I* NN"] + -log P_T(</S>|RRB),
  ...
)

Implementation: Model Loading

```
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  # We use this to
                                # enumerate all tags
    if type = "T"
        transition["context word"] = prob
    else
        emission["context word"] = prob
```

Implementation: Forward Step

```
split line into words
I = length(words)
make maps best_score, best_edge
best_score["0 <s>"] = 0   # Start with <s>
best_edge["0 <s>"] = NULL
for i in 0 … I-1:
    for each prev in keys of possible_tags
        for each next in keys of possible_tags
            if best_score["i prev"] and transition["prev next"] exist
                score = best_score["i prev"] +
                                -log P_T(next|prev) + -log P_E(word[i]|next)
                if best_score["i+1 next"] is new or > score
                    best_score["i+1 next"] = score
                    best_edge["i+1 next"] = "i prev"
# Finally, do the same for </s>
```

Implementation: Backward Step

tags = [ ]
next_edge = *best_edge*[*“I </s>”*]
while *next_edge* != “0 <s>”
    # Add the substring for this edge to the words
    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**

Exercise

Exercise

- Write train-hmm and test-hmm
- Test the program
  - Input: `test/05-{train,test}-input.txt`
  - Answer: `test/05-{train,test}-answer.txt`
- Train an HMM model on `data/wiki-en-train.norm_pos` and run the program on `data/wiki-en-test.norm`
- Measure the accuracy of your tagging with `script/gradepos.pl data/wiki-en-test.pos my_answer.pos`
- Report the accuracy
- Challenge: think of a way to improve accuracy

Thank You!