CS4442/9542b
Artificial Intelligence II
prof. Olga Veksler

Lecture 10

Natural Language Processing

Part of Speech Tagging

Outline

- What is POS and POS tagging

- Why we need POS tagging

- Different Approaches to POS

  1. rule-based tagging
  2. statistical tagging

What is a Part of Speech ?

- Words that somehow 'behave' alike:
  - Appear in similar contexts
  - Perform similar functions in sentences
  - Undergo similar transformations
- Terminology
  - POS (part-of-speech tag) are also called grammatical tag, grammatical category, syntactic class, word class

Substitution Test

- Two words belong to the same part of speech if replacing one with another does not change the grammaticality of a sentence.
  - The {sad, big, fat, green, …} dog is barking.

Origin

- Perhaps started with Aristotle in the West (384–322 BCE)

- From Dionysius Thrax of Alexandria (c. 100 BCE) the idea that is still with us that there are 8 main parts of speech

- Those 8 are not exactly the ones taught today
    - Thrax: noun, verb, article, adverb, preposition, conjunction, participle, pronoun
    - School grammar: noun, verb, adjective, adverb, preposition, conjunction, pronoun, interjection

How many word classes are there?

- A basic set:
  - N(oun), V(erb), Adj(ective), Adv(erb), Prep(osition), Det(erminer), Aux(ilaries), Part(icle), Conj(unction)

- A simple division: open/content vs. closed/function
  - Open: N, V, Adj, Adv
    - New members are added frequently
  - Closed: Prep, Det, Aux, Part, Conj, Num
    - New members are added rarely

- Many subclasses, e.g.
  - eats/V $\Rightarrow$ eat/VB, eat/VBP, eats/VBZ, ate/VBD, eaten/VBN, eating/VBG, ...

POS tagging

- Goal: assign the right part of speech tag (noun, verb, …) to words in a text

  "The/AT representative/NN put/VBD chairs/NNS on/IN the/AT table/NN."

- What set of parts of speech do we use?

  - there are various standard tagsets to choose from; some have a lot more tags than others

  - the choice of tagset is based on the application

  - accurate tagging can be done with even large tagsets

What does Tagging do?

1. Collapses distinctions
   - E.g., all personal pronouns tagged as **PRP**
   - Lexical identity may be completely discarded

2. Introduces distinctions (by reducing ambiguity)
   - E.g., "deal" tagged with **NN** or **VB**

Why do POS Tagging?

- Word sense disambiguiaton (semantics)
  - limits the range of meanings, "deal" as noun vs. "deal" as verb
- Speech recognition and synthesis (better accuracy)
  - how to recognize/pronounce a word
  - *CONtent*/noun VS *conTENT*/adj
- Stemming
  - which morphological affixes the word can take
  - adverb - *ly* = noun
    - *friendly* - *ly* = friend
  - Cannot apply to adjectives
    - Example: sly
- Question answering
  - analyzing a query to understand what type of entity the user is looking for and how it is related to other noun phrases mentioned in the question
- Partial parsing/chunking
  - to find noun phrases/verb phrases
- Information extraction
  - tagging helps identify useful terms and relationships between them

Tag sets

- Different tag sets, depends on the purpose of the application
  - 45 tags in Penn Treebank
  - 62 tags in CLAWS with BNC corpus
  - 79 tags in Church (1991)
  - 87 tags in Brown corpus
  - 147 tags in C7 tagset
  - 258 tags in Tzoukermann and Radev (1995)

Penn Treebank

- First syntactically annotated corpus
- 1 million words from Wall Street Journal
- Part of speech tags and syntax trees

Important Penn Treebank tags

- 45 tags total

IN	preposition or subordinating conjunct.
JJ	adjective or numeral, ordinal
JJR	adjective, comparative
NN	noun, common, singular or mass
NNP	noun, proper, singular
NNS	noun, common, plural
TO	"to" as preposition or infinitive marker
VB	verb, base form
VBD	verb, past tense
VBG	verb, present participle or gerund
VBN	verb, past participle
VBP	verb, present tense, not 3rd p. singular
VBZ	verb, present tense, 3rd p. singular
…	

Verb inflection tags

VBP	base present	*take*
VB	infinitive	*take*
VBD	past	*took*
VBG	present participle	*taking*
VBN	past participle	*taken*
VBZ	present 3sg	*takes*
MD	modal	*can, would*

13

The entire Penn Treebank tagset

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	*+,%, &*
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	*#*
PDT	Predeterminer	*all, both*	"	Left quote	*(' or ")*
POS	Possessive ending	*'s*	"	Right quote	*(' or ")*
PP	Personal pronoun	*I, you, he*	(	Left parenthesis	*([, (, {, <)*
PP$	Possessive pronoun	*your, one's*	)	Right parenthesis	*(],), }, >)*
RB	Adverb	*quickly, never*	,	Comma	*,*
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	*(. ! ?)*
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	*(: ; ... – -)*
RP	Particle	*up, off*			

14

Terminology

- Given text:

"The cat decided to jump on the couch to play with another cat"

- Terminology
  - Word type
    - Distinct words in the text (vocabulary), the text above has 10 word types: "the cat decided to jump on couch play with another"
  - Word token
    - any word occurring in the text
    - The text above has 13 word tokens

Distribution of Tags

- Parts of speech follow the usual frequency-based distributional behavior
  - most word types have only one part of speech
  - of the rest, most have two
  - a small number of word types have lots of parts of speech
  - unfortunately, word types with lots of parts of speech occur with high frequency, and words that occur most frequently tend to have multiple tags

Most Word Types not Ambiguous but..

	num. word types	
Unambiguous (1 tag)	**35 340**	
Ambiguous (>1 tag)	**4 100**	
2 tags	3760	
3 tags	264	
4 tags	61	
5 tags	12	
6 tags	2	
7 tags	1	*"still"*

- but most word types are rare…
- Brown corpus (Francis&Kucera, 1982):
  - 11.5% word types are ambiguous (>1 tag)
  - 40% word tokens are ambiguous (>1 tag)

Why is Tagging Hard?

- Tagging is a type of disambiguation

- Examples:

    1. Book/**VB** that/**DT** flight/**NN**

        - "book" can also be **NN**

        - Can I read a **book** on this flight?

    2. Does/**VBZ** that/**DT** flight/**NN** serve/**VB** dinner/**NN** ?

        - "that" can also be a **complementizer**

        - My travel agent said **that** there would be a meal on this flight.

Potential Sources of Disambiguation

1. Lexical information:
   - look up all possible POS for a word in a dictionary
   - *"table"*: {noun, verb} but not a {adj, prep,…}
   - *"rose"*: {noun, adj, verb} but not {prep, …}

2. Syntagmatic information:
   - some tag sequences are more probable than others:
   - **DET** + **N** occur frequently but **DET**+**V** never occurs
   - **ART**+**ADJ**+**N**   is more probable than **ART**+**ADJ**+**VB**
   - Can find the syntagmatic information
     - by talking to the experts
     - or, better, from training corups

Syntagmatic Information from Corpus

- For a is a sequence of tags $t_1, t_2, .., t_k$ compute $P(t_1, t_2, .., t_k)$, which tells us how likely this tag sequence is
  - have done something similar before, i.e. we computed probability of a sequence of words
  - make similar approximations as before

$$P(t_n | t_1, t_2, .., t_{n-1}) = P(t_n | t_{n-k}...t_{n-1})$$

  - for computational efficiency, our assumption is

$$P(t_n | t_1, t_2, .., t_{n-1}) = P(t_n | t_{n-1})$$

POS Tagging Techniques

1. rule-based tagging

   - uses hand-written rules

2. statistical tagging

   - uses probabilities computed from training corpus

     - Charniak

     - Markov Model based

Rule-based POS Tagging

- Step 1: assign each word with all possible tags
  - use dictionary

- Step 2: use if-then rules to identify the correct tag in context (disambiguation rules)

Rule-based POS Tagging: Sample rules

- **N-IP rule**:

  A tag N (noun) cannot be followed by a tag IP (interrogative pronoun)

  … man who …
  - man: {N}
  - who: {RP, IP} --> {RP} relative pronoun

- **ART-V rule**:

  A tag ART (article) cannot be followed by a tag V (verb)

  …the book…
  - the: {ART}
  - book: {N, V} --> {N}

Rule-based Tagger

- using only syntagmatic patterns
  - Green & Rubin (1971)
  - accuracy of 77%
- In addition, it is very time consuming to come up with the rules and need an expert in English to come up with the rules

Statistical POSTagger: Charniak 1993

- Simplest statistical tagger

- Use corpus to calculate most probable tag for each word

- that is the one maximizing

$$\text{count}(word \text{ has tag } t)/\text{count}(word)$$

- Charniak tagging assigns POS tag to each word separately

- Given a word to tag,

  1. for each possible tag t for this word, compute

     $$\text{count}(word \text{ has tag } t)$$

  2. choose tag t that maximizes the above

Statistical POS Tagger: Charniak 1993

- Accuracy of 90%
  - contrast with 77% accuracy of the rule-based tagger!
  - evidence of power of statistical over rule-based methods
  - MUCH better than rule based, but not very good...
    - 1 mistake every 10 words
  - funny fact: every word will have only one POS assigned to it
    - book will always be assigned the noun tag
- This tagger is used mostly as baseline for evaluation
- How do we improve it?
  - tag of a word should depend on tags of other words around it, i.e. have to take "context" in the account
  - i.e. some sequence of tags are much more likely than others

Statistical Tagger: Markov Model Based

- Want to tag sentence of words $w_1 w_2 \ldots w_n$, shorthand $w_{1,n}$

- Let $t_1 t_2 \ldots t_n$ be possible sequence of tags, shorthand $t_{1,n}$

  - that is t_i is a tag for word w_i

- Want to find the "best" tagging $t_{1,n}$ out of all possible taggings

- Have 2 sources of information in our corpus:

1. given that previous word tag is t_i, can find how likely tag of the next word is t_{i+1}, namely $P(t_{i+1}|t_i)$

2. can find how likely is each word for each tag, namely $P(w_i|t_i)$

   - how likely part of speech t_i will give rise to word w_i

   - Example: if we know that tag of the word is t_i = **noun**, what is the probability of the word to be "book"

   - P(book|verb) > P(book|noun)

     - there are many more nouns than verbs

     - say 1,000 verbs and 10,000 nouns

Statistical Tagger: Markov Model Based

- Tag assignment $t_{1,n}$ for sentence $w_{1,n}$

- Using Bayes law: $P(t_{1,n} \mid w_{1,n}) = \dfrac{P(w_{1,n} \mid t_{1,n})P(t_{1,n})}{P(w_{1,n})}$

- Make two simplifying assumptions:

1. Given a POS tag, probability of a word is independent of the tags of other words in a sentence:

$$P(w_{1,n} \mid t_{1,n}) = \prod_{i=1}^{n} P(w_i \mid t_i)$$

Statistical Tagger: Markov Model Based

$$P\,(w_{1,n}\mid t_{1,n}) = \prod_{i=1}^{n} P\,(w_i\mid t_i)$$

$P(t_{1,n}$ give words $w_{1,n}) =$

$\qquad\qquad\qquad P(t_1$ gives word $w_1) \times P(t_2$ gives word $w_2) \times \ldots$

$\qquad\qquad\qquad\qquad \ldots \times P(t_n$ gives word $w_n)$

- $P(w_i \mid t_k)$ approximated from the tagged corpus as:

$$\frac{C(w_i \text{ has tag } t_k)}{C(t_k)}$$

- i.e. how many times word w_i has tag t_k divided by how often tag t_k occurs in the training corpus

Statistical Tagger: Markov Model Based

$$P(t_{1,n} \mid w_{1,n}) = \frac{P(w_{1,n} \mid t_{1,n})P(t_{1,n})}{P(w_{1,n})}$$

2. Each tag is only dependent only on one previous tag:

$$P(t_{1,n}) = \prod_{i=1}^{n} P(t_i \mid t_{i-1}) = P(t_1 \mid t_0) \times P(t_2 \mid t_1) \ldots \times P(t_n \mid t_{n-1})$$

- this is **Markov** assumption we saw in language modeling
- recall $P(t_2 \mid t_1) \approx C(t_1 t_2)/C(t_1)$
- Here $P(t_1 \mid t_0)$ stands for $P(t_1)$ and is approximated by $C(t_{12})/N$

Statistical Tagger: Markov Model Based

- Using these 2 simplifications, we get

$$P(t_{1,n} \mid w_{1,n}) = \frac{\prod_{i=1}^{n} P(w_i \mid t_i)P(t_i \mid t_{i-1})}{P(w_{1,n})}$$

- Given a possible sequence of tags $t_{1,n}$ for sentence $w_{1,n}$, we can evaluate how good this tag assignment is using $P(t_{1,n} \mid w_{1,n})$

- Algorithm: go over all possible tag assignments and choose the one which gives highest $P(t_{1,n} \mid w_{1,n})$
  - notice that $P(w_{1,n})$ does not effect maximization so we do not have to compute it

Statistical Tagger: Markov Model Based

- Algorithm: given sentence $\mathbf{w_{1,n}}$ go over all possible tag assignments $\mathbf{t_{1,n}}$ and compute

$$\prod_{i=1}^{n} \mathbf{P}(\mathbf{w_i} \mid \mathbf{t_i})\mathbf{P}(\mathbf{t_i} \mid \mathbf{t_{i-1}})$$

- Choose final tagging $\mathbf{t_{1,n}}$ which maximizes the above
- Efficiency
  - For each word $\mathbf{w_i}$, try only tags given by the dictionary
  - Example: for "fly", possible tags are NOUN, VERB and also ADJECTIVE (meaning "keen" or "artful", mainly in England)

Statistical Tagger: Markov Model Based

- Side note: Markov tagger becomes Charniak's tagger if tags are assumed independent, i.e. $P(t_i | t_{i-1}) = P(t_i)$

$$\prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-1}) = \prod_{i=1}^{n} P(w_i | t_i) P(t_i)$$

$$= \prod_{i=1}^{n} \frac{P(w_i, t_i)}{P(t_i)} P(t_i)$$

$$= \prod_{i=1}^{n} P(w_i, t_i)$$

Statistical Tagger: Markov Model Based

- Algorithm: given sentence $w_{1,n}$ go over all possible tag assignments $t_{1,n}$ and compute $\prod_{i=1}^{n} P(w_i \mid t_i) P(t_i \mid t_{i-1})$

- 40 % words have more than 1 tag
- Too many tag assignments to try
- If 2 tags per word, then 2^n possible assignments
- Exhaustive search is exponential

Statistical Tagger: Markov Model Based

$$\prod_{i=1}^{n} P(w_i \mid t_i) P(t_i \mid t_{i-1})$$
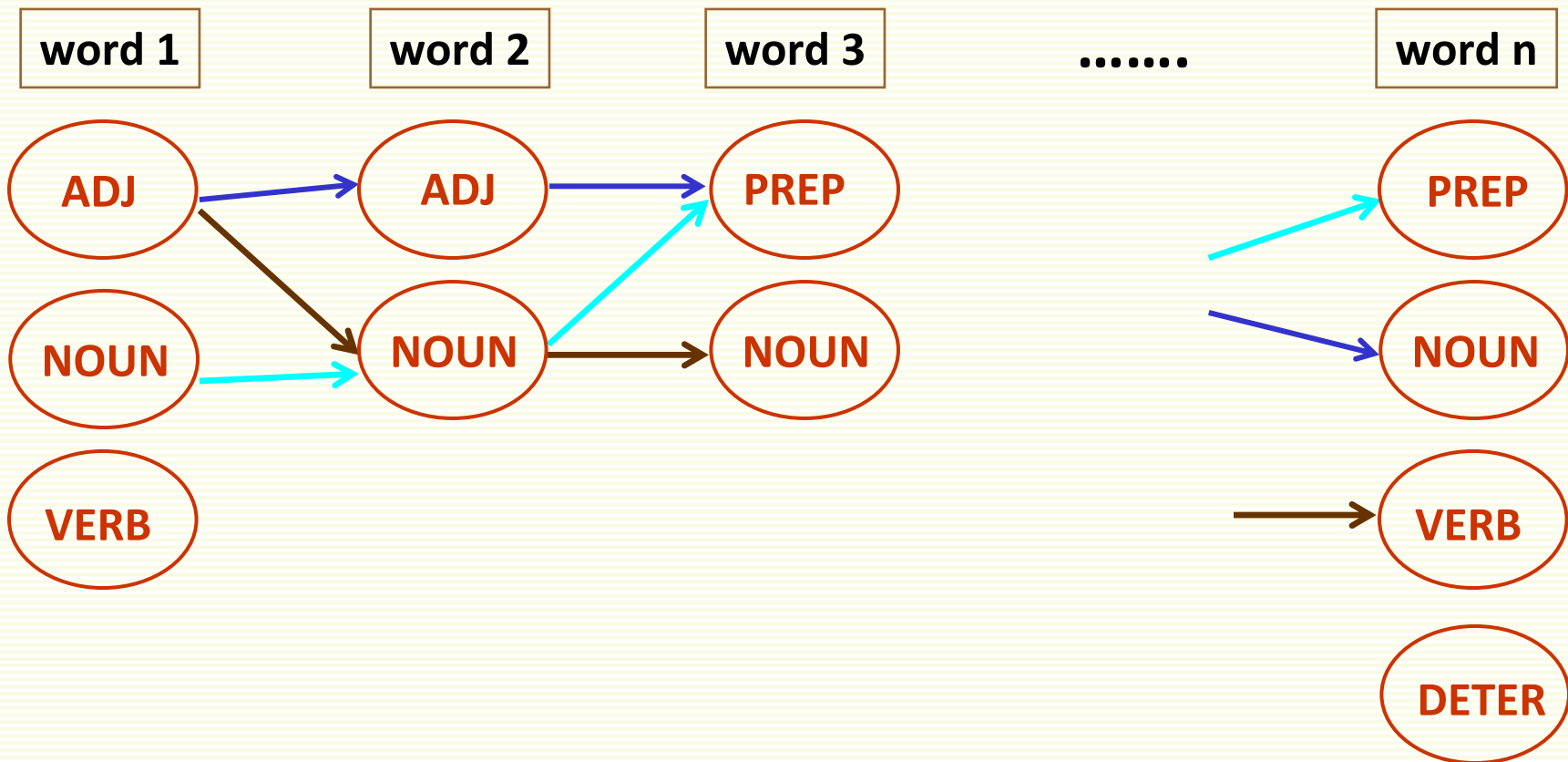
MM based Tagger: Viterbi Algorithm

- Fortunately there is a very useful algorithm (Viterbi)
- If **k** tags per word and **n** words, can find best tagging in time **k^2n**
  - There are **k^n** different tag assignments possible
- First, to avoid floating point underflows, take logarithm of

$$\prod_{i=1}^{n} P(w_i \mid t_i)P(t_i \mid t_{i-1})$$

$$\log\left[\prod_{i=1}^{n} P(w_i \mid t_i)P(t_i \mid t_{i-1})\right] = \sum_{i=1}^{n}(\log P(w_i \mid t_i) + \log P(t_i \mid t_{i-1}))$$

- Now we want to maximize:

$$\sum_{i=1}^{n}\left[\log P(w_i \mid t_i) + \log P(t_i \mid t_{i-1})\right] = \sum_{i=1}^{n}\log P(w_i \mid t_i) + \sum_{i=1}^{n}\log P(t_i \mid t_{i-1})$$

how likely word **w_i** is for tag **t_i**

how likely tag **t_i** to follow tag **t_{i-1}**

MM based Tagger: Viterbi Algorithm

- Turn maximizing:

$$\sum_{i=1}^{n} \log P(w_i \mid t_i) + \sum_{i=1}^{n} \log P(t_i \mid t_{i-1})$$

- Into equivalent minimizing

$$-\sum_{i=1}^{n} \log P(w_i \mid t_i) - \sum_{i=1}^{n} \log P(t_i \mid t_{i-1})$$

MM based Tagger: Viterbi Algorithm

- Need to find a sequence of tags **t₁**, **t₂**,...,**tₙ** to minimize:

$$-\sum_{i=1}^{n}\log P(w_i \mid t_i) - \sum_{i=1}^{n}\log P(t_i \mid t_{i-1})$$

- To simplify notation, will write
  - **L**(w_i|t_i)   instead of -log[**P**(w_i|t_i)]
  - **L**(t_i|t_{i-1}) instead of -log[**P**(t_i|t_{i-1})]

- In the new notation, need to find a sequence of tags **t₁, t₂,..., tₙ** to minimize:

$$\sum_{i=1}^{n}\left[L(w_i \mid t_i) + L(t_i \mid t_{i-1})\right]$$

MM based Tagger: Viterbi Algorithm

$$\sum_{i=1}^{n} \left[L(w_i \mid t_i) + L(t_i \mid t_{i-1}) \right]$$

word 1   word 2   word 3   ....

L(ADJ) → **ADJ**

L(w$_1$|ADJ)

L(NOUN|ADJ)

ADJ

L(w$_2$|ADJ)

PREP

L(w$_3$|PREP)

L(NOUN) → **NOUN**

L(w$_1$|NOUN)

NOUN

L(VERB|NOUN)

L(w$_2$|NOUN)

VERB

L(w$_3$|VERB)

L(VERB) → **VERB**

L(w$_1$|VERB)
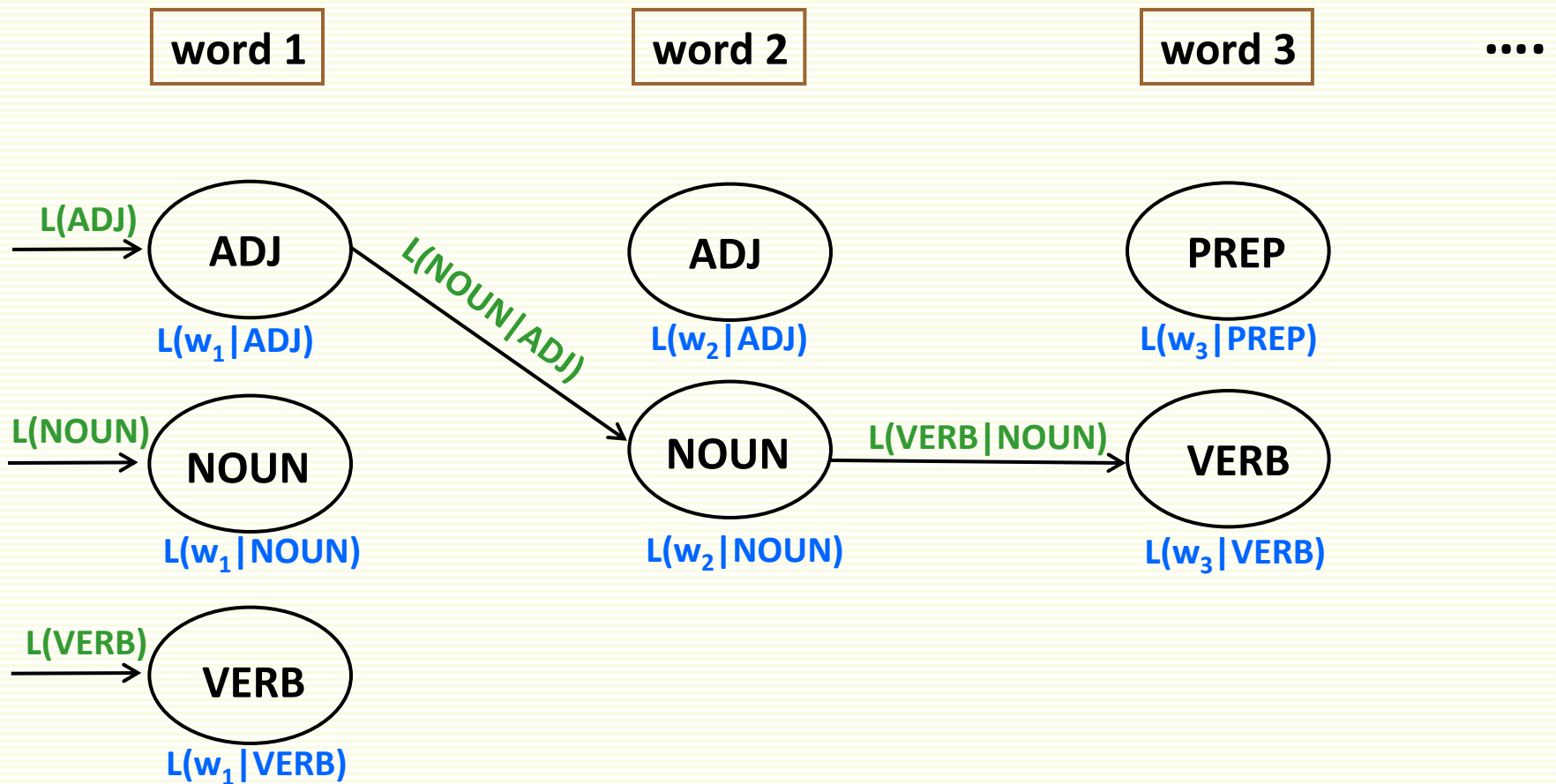
MM based Tagger: Viterbi Algorithm

- Change notation for the first word:

$L(w_1|t_1)$ will stand for $-\log[P(w_1|t_1)] - \log[P(t_1)]$

$$\sum_{i=1}^{n}\left[L(w_i|t_i)+L(t_i|t_{i-1})\right] \Rightarrow \sum_{i=1}^{n}\log P(w_i|t_i)+\sum_{i=2}^{n}\log P(t_i|t_{i-1})$$

instead of this: we will picture this:

| word 1 |

L(ADJ) → **ADJ**

$L(w_1|ADJ)$

L(NOUN) → **NOUN**

$L(w_1|NOUN)$

| word 1 |

ADJ

$L(w_1|ADJ)$

NOUN

$L(w_1|NOUN)$
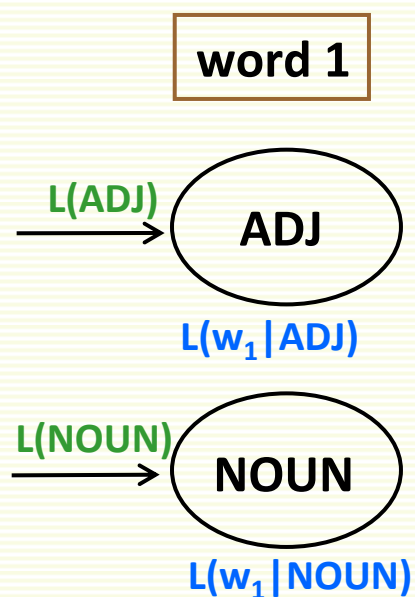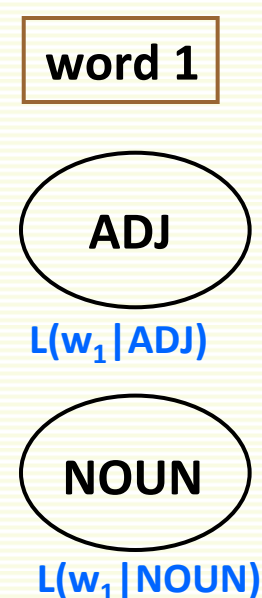
MM based Tagger: Viterbi Algorithm

- Each node has cost $L(w_i|t_i)$
- Each link between nodes has cost $L(t_i|t_{i-1})$
- Cost of path, summing up node costs and edge costs is:

$$\sum_{i=1}^{n} L(w_i\,|\,t_i) + \sum_{i=2}^{n} L(t_i\,|\,t_{i-1})$$
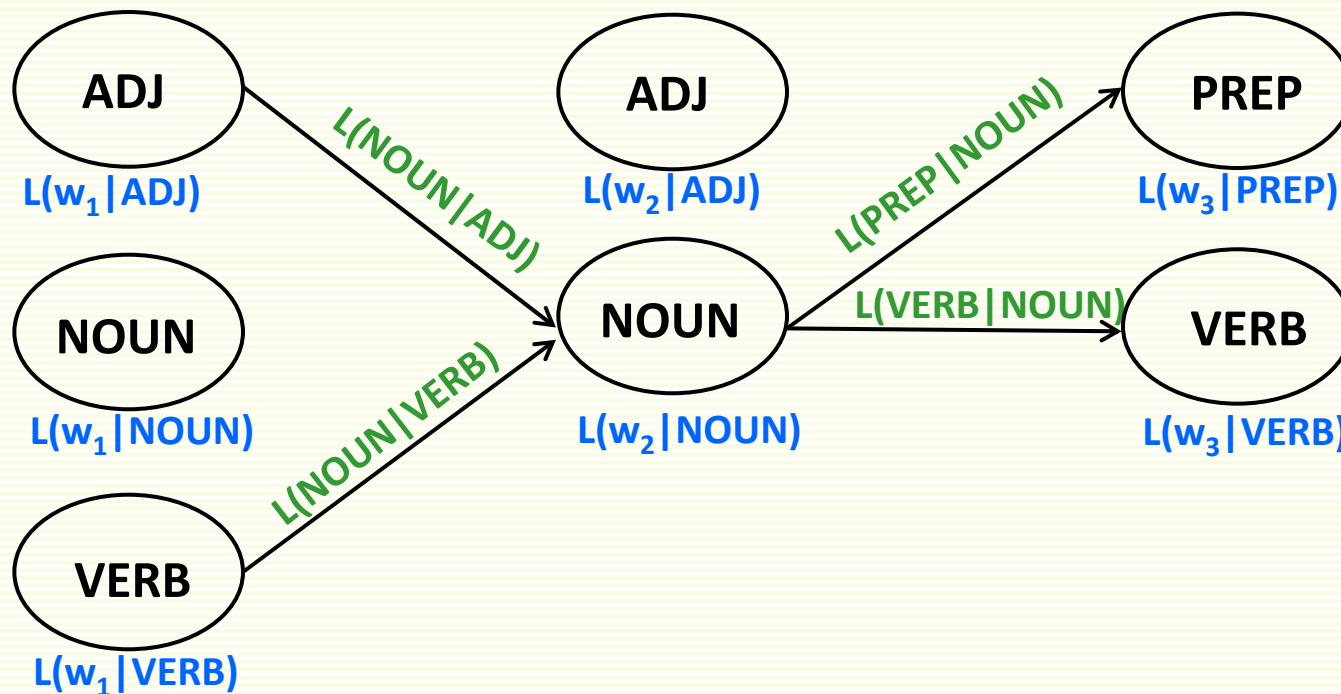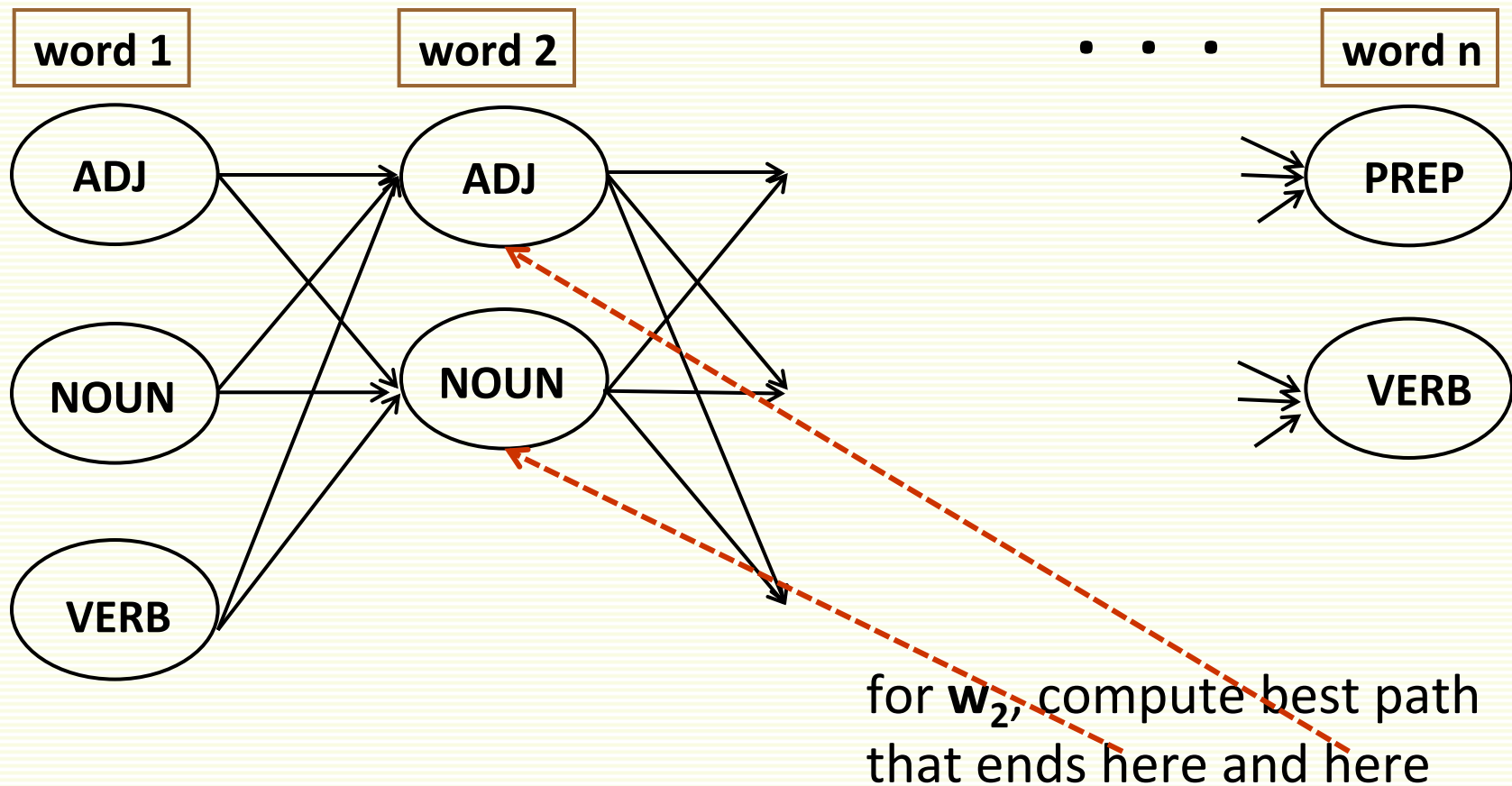
MM based Tagger: Viterbi Algorithm

- Find smallest cost path that starts at some node corresponding to **word 1** and ends at some node corresponding to **word n**

$$\sum_{i=1}^{n} L(w_i \mid t_i) + \sum_{i=2}^{n} L(t_i \mid t_{i-1})$$
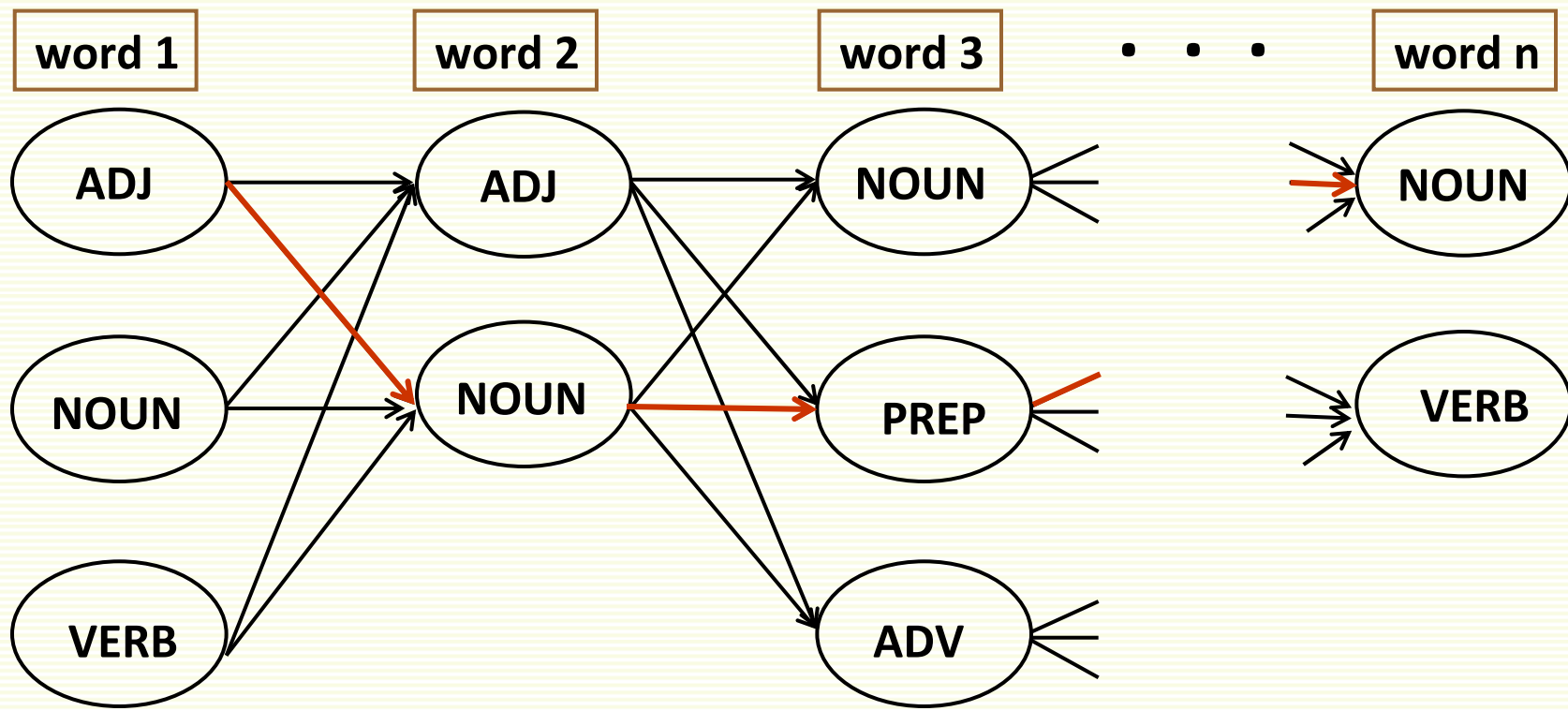
MM based Tagger: Viterbi Algorithm

- Step: for every node corresponding to w_i, efficiently find smallest cost path that ends at it, and starts at any node corresponding to w_1

word 1
word 2
· · ·
word n

ADJ

ADJ

PREP

NOUN

NOUN

VERB

VERB

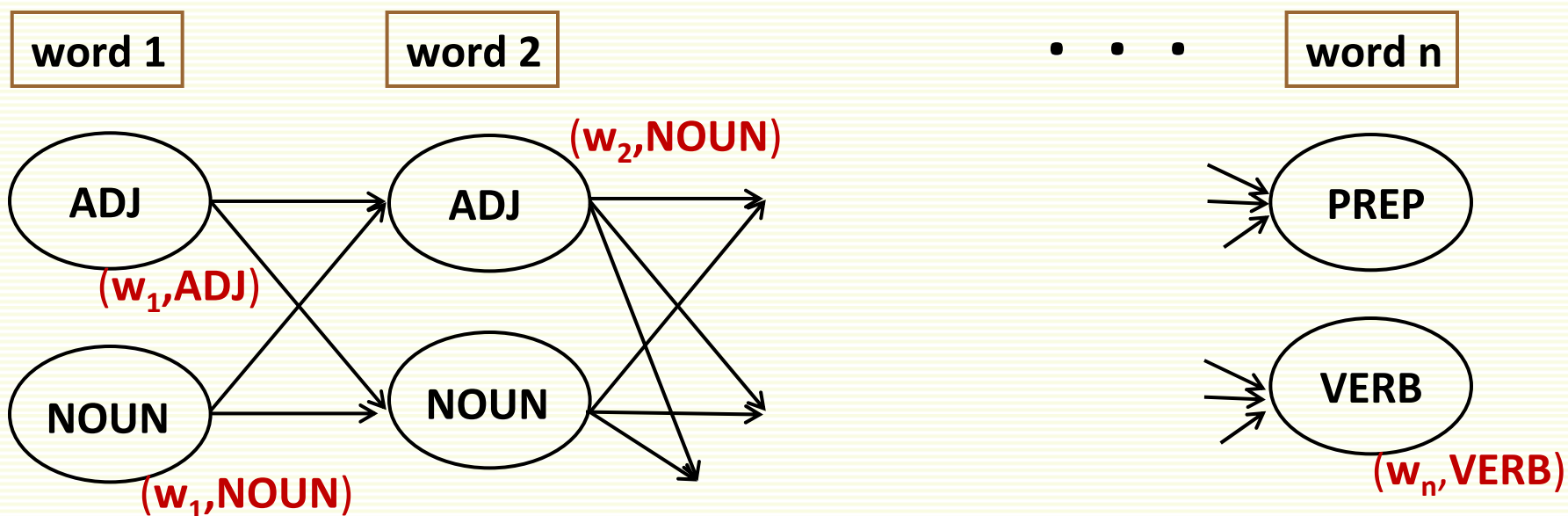for w_2, compute best path that ends here and here

MM based Tagger: Viterbi Algorithm

- First compute the best path that ends at any node for w_1
- Then compute the best path that ends at any node for w_2
- .....
- Finally compute the best path that ends at any node for w_n
- The best path overall is smallest cost path that end at w_n
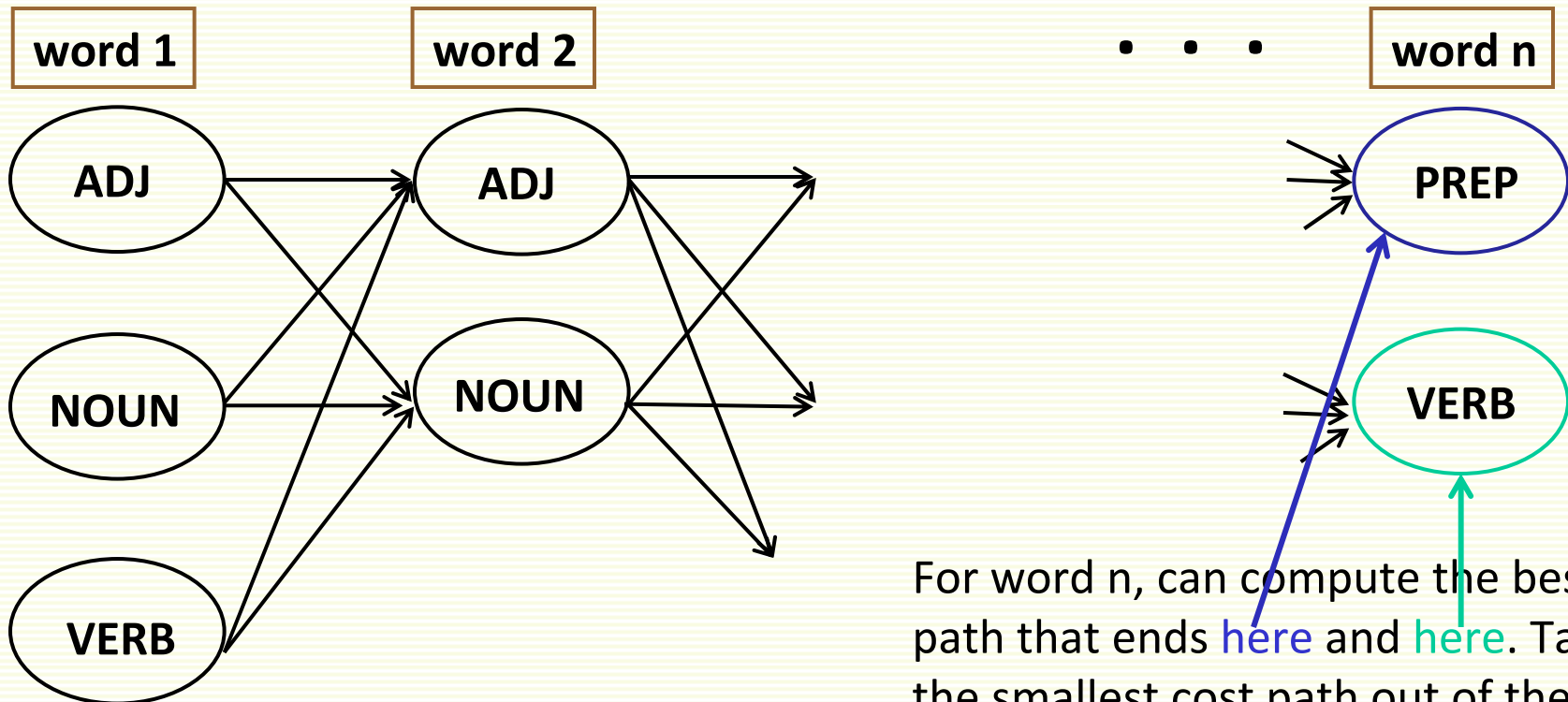
For word n, can compute the best path that ends here and here. Take the smallest cost path out of these two as the final best path

MM based Tagger: Viterbi Algorithm

- Node is specified by the word w_i and tag t, written as (w_i, t)

word 1   word 2   • • •   word n

$(w_2, NOUN)$

ADJ → ADJ → PREP

(w_1, ADJ)

NOUN → NOUN → VERB

$(w_1, NOUN)$

$(w_n, VERB)$

- Let $C(w_i, t)$ be the cost of the best path starting at any node for w_1 and ending at node (w_i, t)
- Let $P(w_i, t)$ be parent on best cost path starting at any node for w_1 and ending at node (w_i, t). Parent must be node (w_{i-1}, t')
- After all $C(w_i, t)$ are computed, the best cost path overall is given by the minimum of $C(w_n, t)$ over all t

MM based Tagger: Viterbi Algorithm

- First compute the best path that ends at any node for w_1
- Trivial, since the path has just one node
- For all tags of the first word t, $C(w_1, t) = L(w_1 | t)$
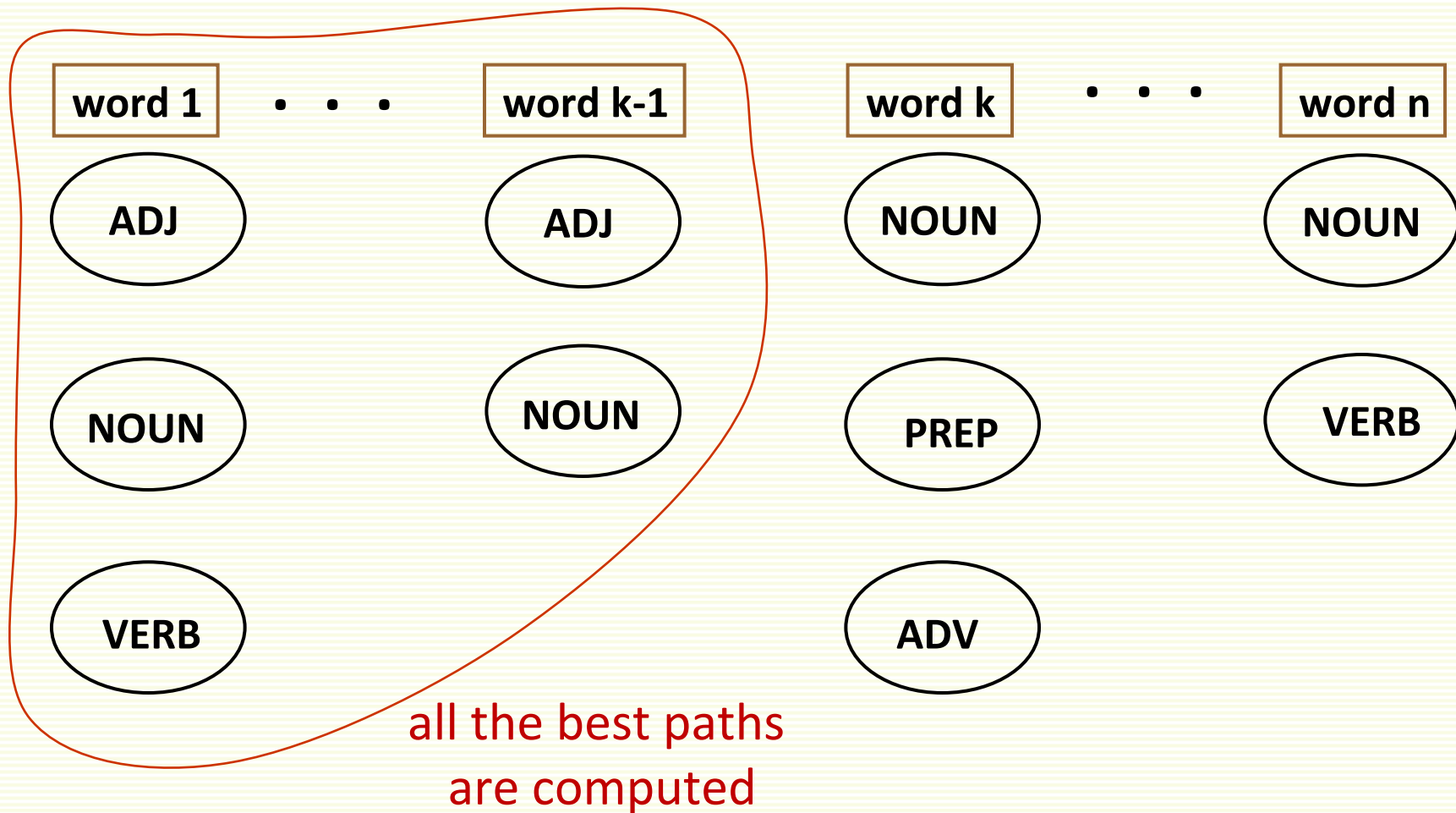- For all tags of the first word t, $P(w_1, t)$ = null

word 1

ADJ

$L(w_1 | ADJ)$

NOUN

$L(w_1 | NOUN)$

VERB

$L(w_1 | VERB)$

MM based Tagger: Viterbi Algorithm

- Now suppose computed $C(w_i, t)$ and $P(w_i, t)$ for all tags t and all $i < k$

word 1	· · ·	word k-1		word k	· · ·	word n
ADJ		ADJ		NOUN		NOUN
NOUN		NOUN		PREP		VERB
VERB				ADV		

all the best paths
are computed

MM based Tagger: Viterbi Algorithm

- Computed $C(w_i, t)$ for all tags t and all $i < k$
- Need to compute $C(w_k, t)$ and $P(w_k, t)$ for all possible tags t of word k
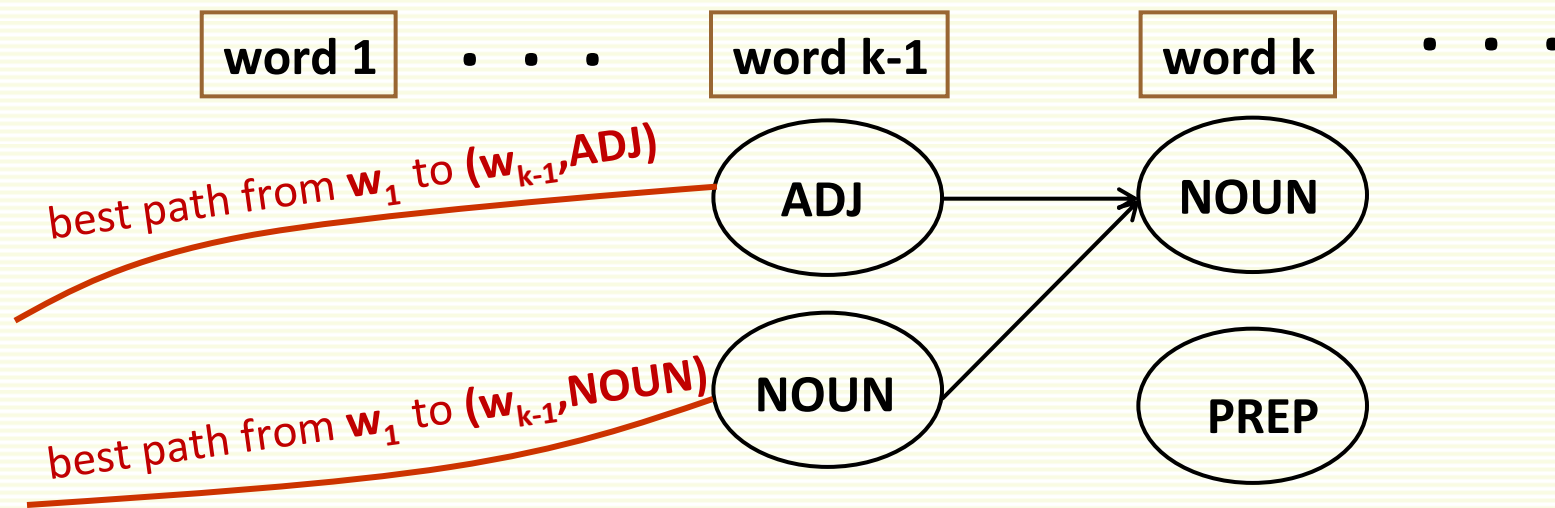
- Consider node $(w_k, \textbf{ADJ})$. Let P be the best path from the w_1 to node $(w_k, \textbf{ADJ})$
- Path P goes through either
    1) $(w_{k-1}, \textbf{ADJ})$, in which case P follows the best path from w_1 to $(w_{k-1}, \textbf{ADJ})$
    2) $(w_{k-1}, \textbf{NOUN})$, in which case P follows the best path from w_1 to $(w_{k-1}, \textbf{NOUN})$
    - because a sub-path of the best path is a best path itself

MM based Tagger: Viterbi Algorithm

- Therefore $C(w_k, ADJ)$ is the smaller of two quantities:
  1. $C(w_{k-1}, ADJ) + L(ADJ|ADJ) + L(w_k|ADJ)$
     - in this case, $P(w_k, ADJ) = (w_{k-1}, ADJ)$
  2. $C(w_{k-1}, NOUN) + L(ADJ|NOUN) + L(w_k|ADJ)$
     - in this case, $P(w_k, ADJ) = (w_{k-1}, NOUN)$
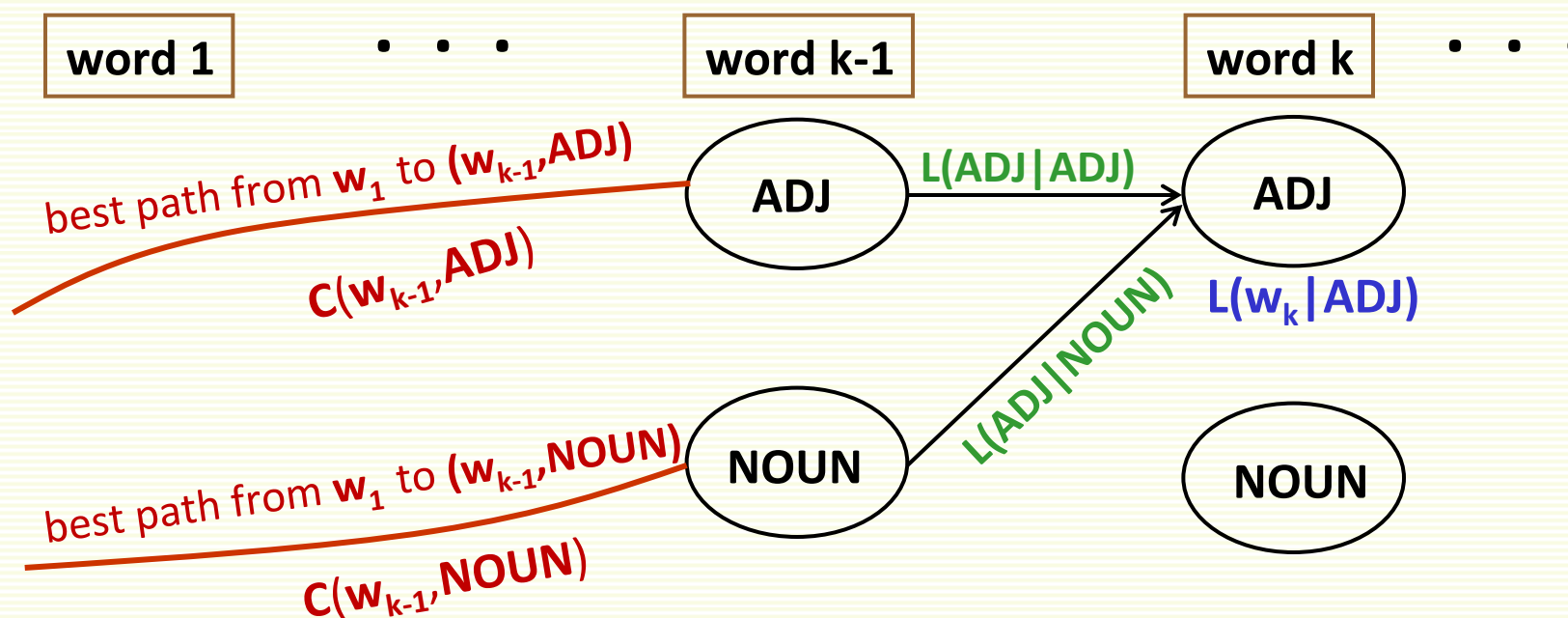
MM based Tagger: Viterbi Algorithm

- In general, $C(w_k, t)$ is computed as follows:

cost of best path from first word to node (word k-1, t)

cost of going through node (w_k, t)

$$C(w_k, t) = \min_{t' \in T(w_{k-1})} \{C(w_{k-1}, t') + L(t \mid t')\} + L(w_k \mid t)$$

search over all tags t' for word k-1

cost of going between nodes (w_{k-1}, t') and (w_k, t)

- $P(w_k, t) = (w_{k-1}, t^*)$ where t^* is the tag for word w_{k-1} minimizing the expression above

MM based Tagger: Viterbi Algorithm

- After computed $C(w_i, t)$ for all i and t, best cost path is found as the maximum of $C(w_n, t)$ over all tags t of word n
- Parents on the path traced back using $P(w_i, t)$

word 1	word 2	$\cdots$	word n-1	word n

ADJ   ADJ   ADJ   PREP

NOUN   NOUN   NOUN   VERB

$C(w_2, \textbf{NOUN})$ is smallest, $P(w_2, \textbf{NOUN}) = (w_1, \textbf{VERB})$

$C(w_n, \textbf{VERB})$ is smallest, $P(w_n, \textbf{VERB}) = (w_{n-1}, \textbf{ADJ})$

VERB   VERB
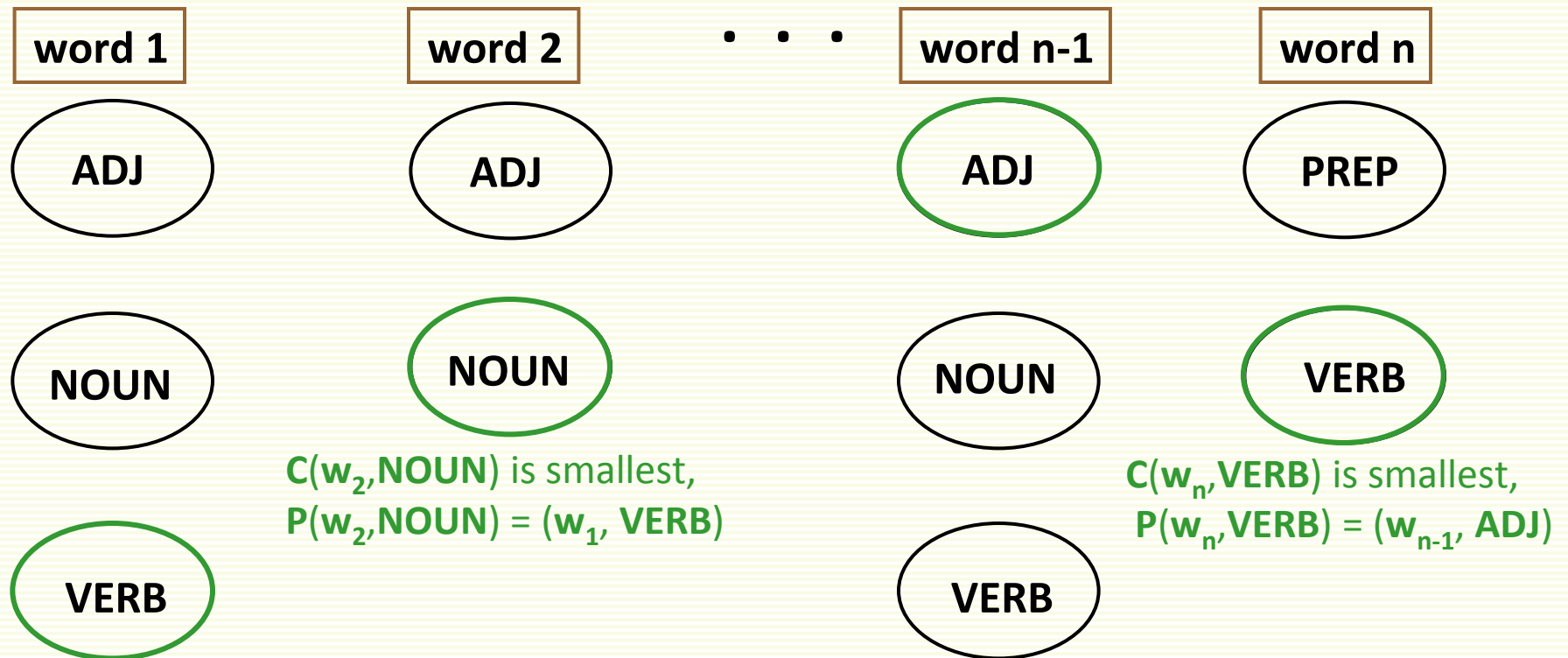
- Final tagging is: VERB   NOUN   …   ADJ   VERB

MM based Tagger: Small Example

L(BOOK|ADJ) = 10   L(THAT|PRON) = 2   L(FLIGHT|NOUN) = 2

L(BOOK|VERB) = 1   L(THAT|CONJ) = 4   L(FLIGHT|VERB) = 1

L(BOOK|NOUN) =2

| book |

ADJ

VERB

NOUN

L(PRON|VERB) = 3

L(CONJ|VERB)=4

L(PRON|NOUN) =2

L(CONJ|NOUN)= 1

L(PRON|ADJ) =1

L(CONJ|ADJ) =2

| that |

PRON

CONJ

L(NOUN|PRON) =1

L(VERB|PRON) =10

L(NOUN|CONJ) =4

L(VERB|CONJ) =2

| flight |

NOUN

VERB

MM based Tagger: Small Example

L(BOOK|ADJ) = 10

L(BOOK|VERB) = 1

L(BOOK|NOUN) = 2

book

ADJ

VERB

NOUN

- Iteration 1:
  - **C**(**book**,**ADJ**) = 10, **P**(**book**,**ADJ**) = null
  - **C**(**book**,**VERB**) = 1, **P**(**book**,**VERB**) = null
  - **C**(**book**,**NOUN**) = 2, **P**(**book**,**NOUN**) = null

MM based Tagger: Small Example

L(PRON|ADJ) =1

L(PRON|VERB) = 3

L(PRON|NOUN) =2

L(THAT|PRON) = 2

L(THAT|CONJ) = 4
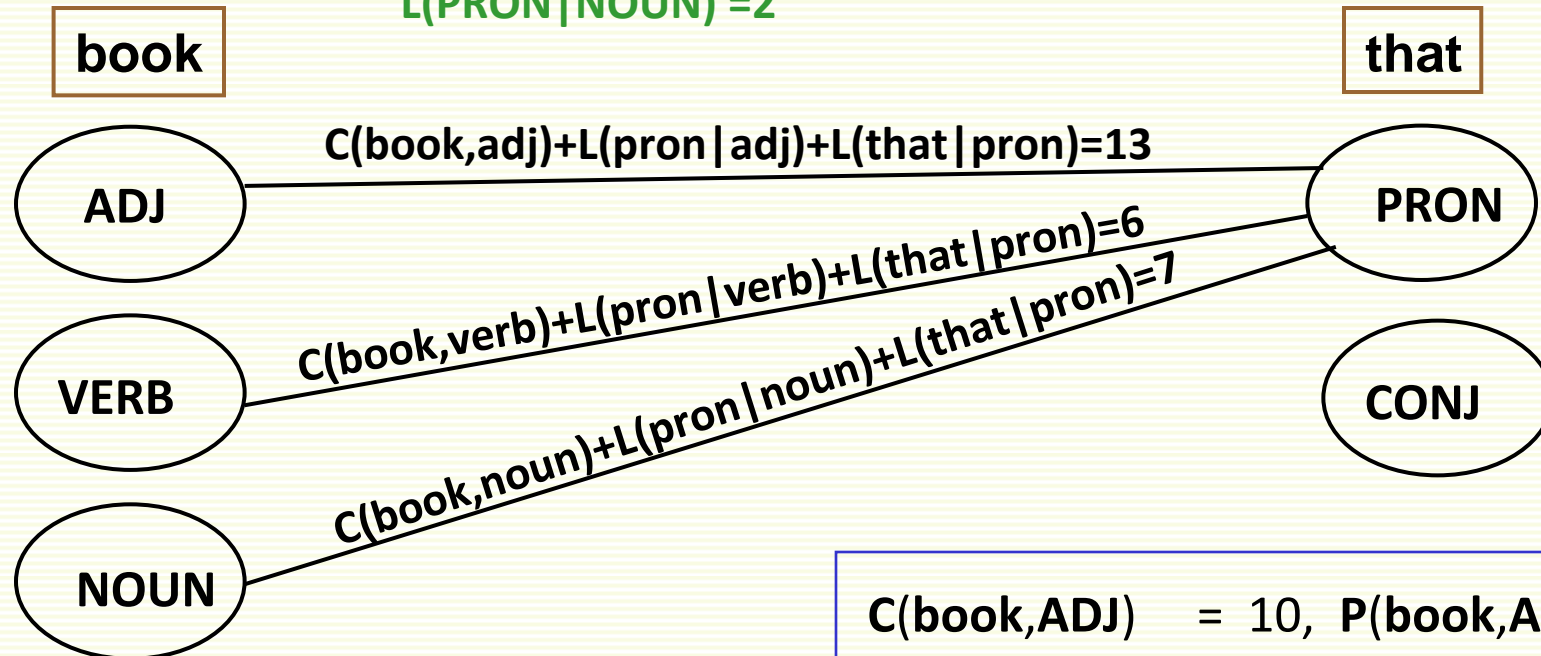
book

that

ADJ

PRON

C(book,adj)+L(pron|adj)+L(that|pron)=13

C(book,verb)+L(pron|verb)+L(that|pron)=6

C(book,noun)+L(pron|noun)+L(that|pron)=7

VERB

CONJ

NOUN

C(book,ADJ)   = 10, P(book,ADJ)   = null
C(book,VERB) = 1,   P(book,VERB) = null
C(book,NOUN) = 2,   P(book,NOUN) = null

- Iteration 2:
  - **C(that, PRON) = 6, P(that,PRON) = (book,VERB)**

MM based Tagger: Small Example

L(CONJ|VERB)=4   L(THAT|PRON) = 2

L(CONJ|NOUN)= 1   L(THAT|CONJ) = 4

L(CONJ|ADJ) =2

book   that

ADJ   PRON

C(book,adj)+L(conj|adj)+L(that|conj)=16

VERB   C(book,verb)+L(conj|verb)+L(that|conj)=9   CONJ

C(book,noun)+L(conj|noun)+L(that|conj)=8

NOUN

C(book,ADJ)   = 10, P(book,ADJ)   = null
C(book,VERB) = 1,   P(book,VERB) = null
C(book,NOUN) = 2,   P(book,NOUN) = null
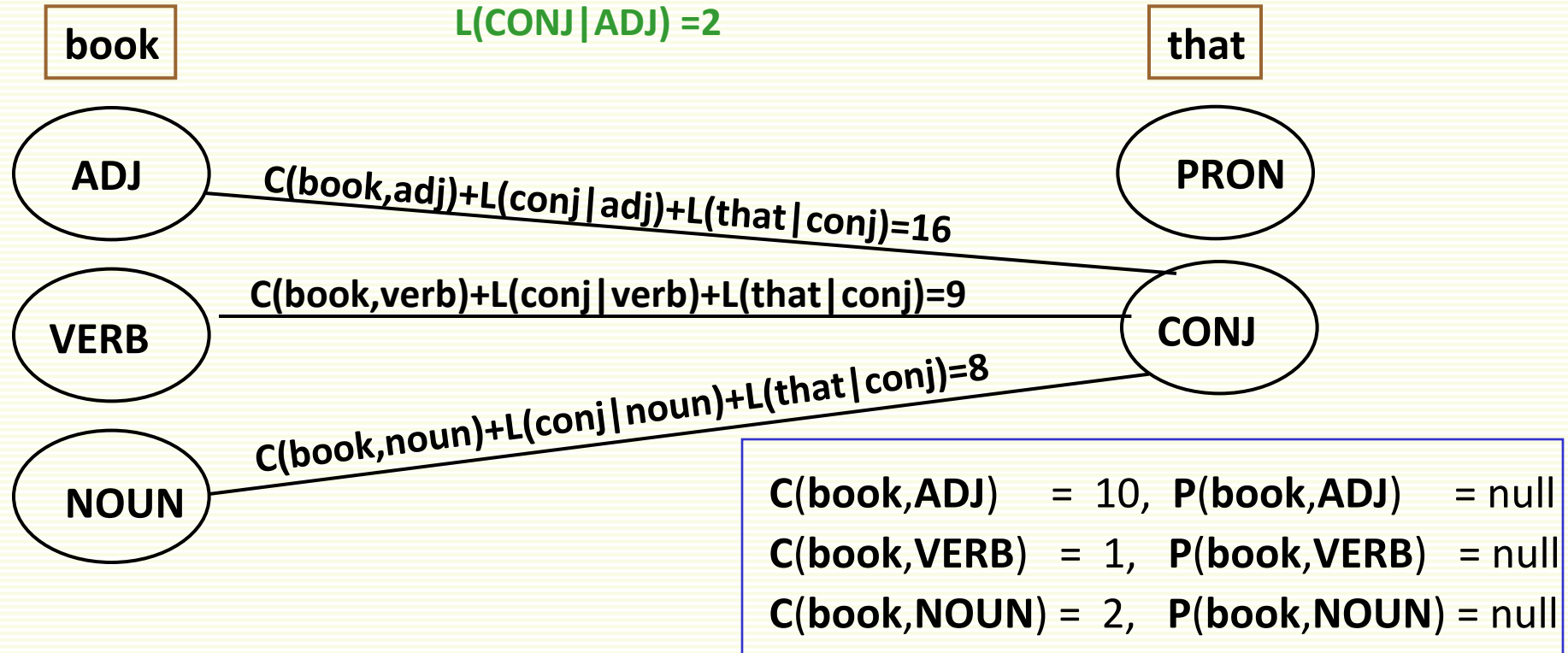
- Iteration 2:
  - **C(that, CONJ) = 8, P(that, CONJ) = (book,NOUN)**

MM based Tagger: Small Example

L(NOUN|PRON) =1
L(NOUN|CONJ) =4

L(FLIGHT|NOUN) = 2
L(FLIGHT|VERB) = 1
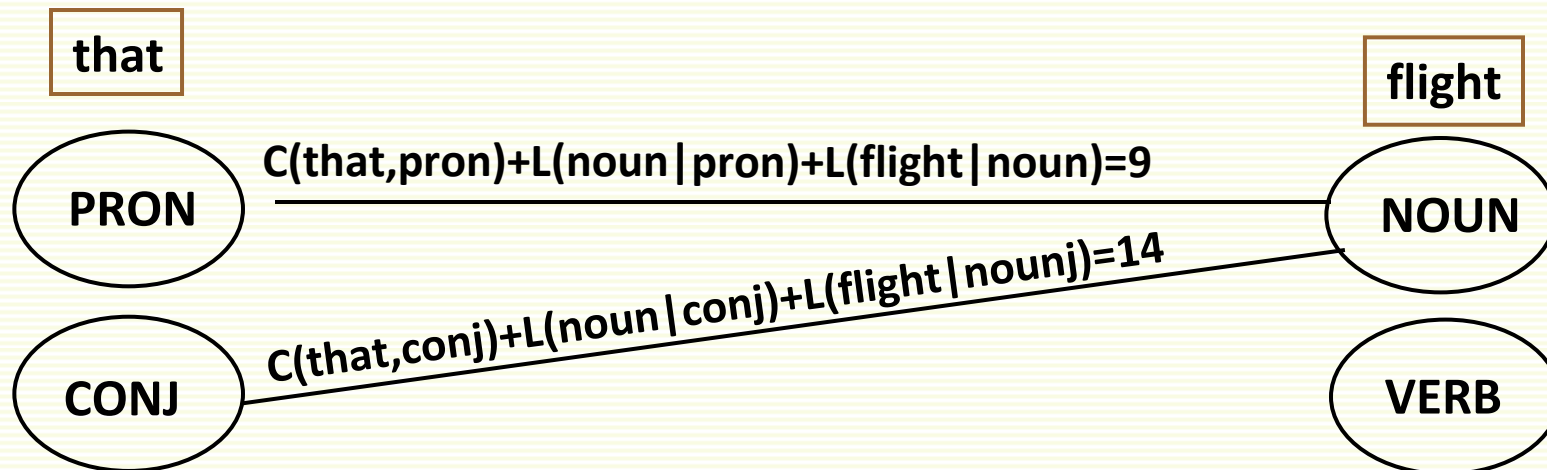
that

flight

PRON — C(that,pron)+L(noun|pron)+L(flight|noun)=9 — NOUN

CONJ — C(that,conj)+L(noun|conj)+L(flight|nounj)=14 — VERB

C(book,ADJ)   = 10, P(book,ADJ)   = null
C(book,VERB)   = 1,   P(book,VERB)   = null
C(book,NOUN) = 2,   P(book,NOUN) = null
C(that, PRON) = 6,   P(that, PRON)   = (book,VERB)
C(that, CONJ) = 8,   P(that, CONJ)   = (book,NOUN)

- Iteration 3:
  - **C(flight, NOUN)** = 9, **P(flight, NOUN)** = (that,PRON)

MM based Tagger: Small Example

L(VERB|PRON) =10

L(VERB|CONJ) =2

L(FLIGHT|NOUN) = 2

L(FLIGHT|VERB) = 1
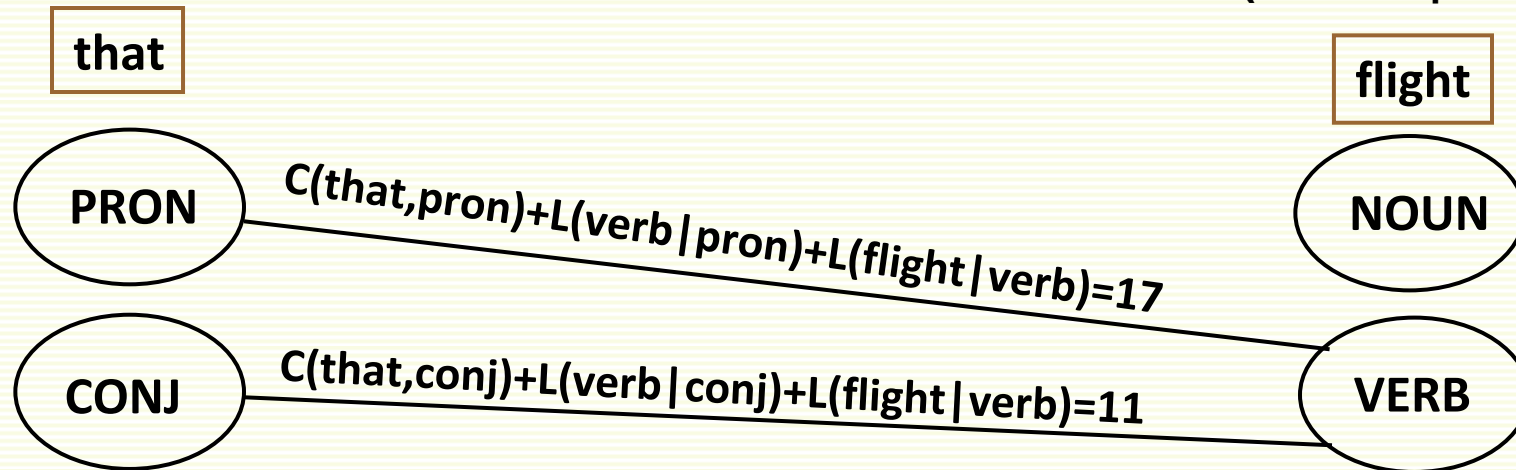
that

flight

PRON

C(that,pron)+L(verb|pron)+L(flight|verb)=17

NOUN

CONJ

C(that,conj)+L(verb|conj)+L(flight|verb)=11

VERB

C(**book,ADJ**)   = 10, **P**(**book,ADJ**)   = null

C(**book,VERB**) = 1,   **P**(**book,VERB**) = null

C(**book,NOUN**) = 2,   **P**(**book,NOUN**) = null

C(**that, PRON**) = 6,   **P**(**that, PRON**)   = (**book,VERB**)

C(**that, CONJ**) = 8,   **P**(**that, CONJ**)   = (**book,NOUN**)

- Iteration 3:
  - C(**flight**, **VERB**) = 11, **P**(**flight**, **VERB**) = (**that,CONJ**)

MM based Tagger: Small Example
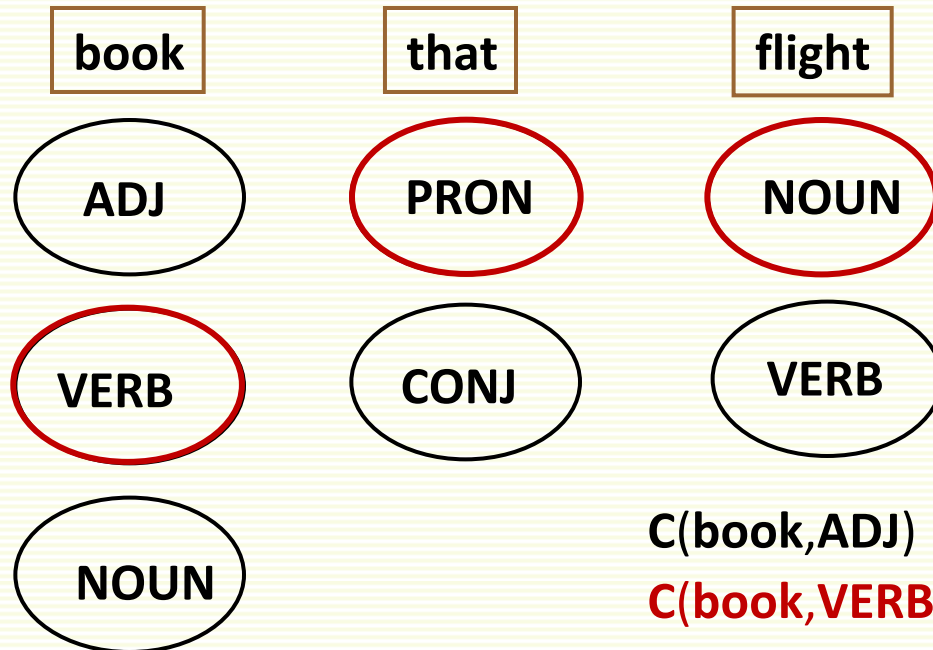
book   that   flight

ADJ   PRON   NOUN

VERB   CONJ   VERB

NOUN

C(book,ADJ)   = 10,   P(book,ADJ)   = null
C(book,VERB) = 1,   P(book,VERB) = null
C(book,NOUN) = 2,   P(book,NOUN) = null

C(that, PRON) = 6,   P(that, PRON)   = (book,VERB)
C(that, CONJ) = 8,   P(that, CONJ)   = (book,NOUN)

C(flight, NOUN) = 9,   P(flight, NOUN) = (that,PRON)
C(flight, VERB) = 11, P(flight,VERB)   = (that,PRON)

Final Tagging: Book<verb> that <pron> flight<noun>

Viterbi Algorithm

for each $t \in$ Tags(w_1) **do**

    C(w_1, t) = **L**($w_1 \mid$ t), **P**(w_1, t) = null

 for i $\leftarrow$ 2 t**o** n **do**

    for each t $\in$ Tag(w_i) **do**

        C(w_i, **t**) = -∞

        for each t' $\in$ Tag(w_{i-1}) **do**

            nextCost = **C**(w_{i-1},**t'**) + **L**(**t**$\mid$**t'**) + **L**($w_i \mid$**t**)

            if nextCost < cost(w_i, **t**) **do**

                C(w_i,**t**) = nextCost

                P(w_i,**t**) = **t'**

Note: Tags(w_i) is the set of all possible tags for w_i

Unknown Words

- Simplest method: assume an unknown word could belong to any tag; unknown words are assigned the distribution over POS over the whole lexicon
  - P(karumbula"|verb) = P("karumbula"|noun) = P("karumbula"|adjective) = …. etc
- Some tags are more common than others
  - for example a new word can be most likely a verb, a noun etc. but not a preposition or an article
- Use morphological and other cues
  - for example words ending in *–ed* are likely to be past tense forms or past participles

Tagging Accuracy

- Ranges from 96%-97%
- Depends on:
  - Amount of training data available
  - The tag set
  - Difference between training corpus and dictionary and the corpus of application
  - Unknown words in the corpus of application
- A change in any of these factors can have a dramatic effect on tagging accuracy – often much more stronger than the choice of tagging method