POS Tagging and Sequence Models

JURAFSKY AND MARTIN CHAPTERS 8

Corpora with manual POS tags

Brown corpus – 1 million words of 500 written English texts from different genres.

WSJ corpus – 1 million words from the Wall Street Journal

Switchboard corpus – 2 million words of telephone conversations

The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.

There/EX are/VBP 70/CD children/NNS **there/RB**

Preliminary/JJ findings/NNS were/VBD **reported/VBN** in/IN today/NN **'s/POS** New/NNP England/NNP Journal/NNP of/IN Medicine/NNP ./.

POS Tagging

Words are ambiguous, so tagging must resolve disambiguate.

Types:		WSJ	Brown
Unambiguous	(1 tag)	44,432 (**86%**)	45,799 (**85%**)
Ambiguous	(2+ tags)	7,025 (**14%**)	8,050 (**15%**)
Tokens:			
Unambiguous	(1 tag)	577,421 (**45%**)	384,349 (**33%**)
Ambiguous	(2+ tags)	711,780 (**55%**)	786,646 (**67%**)

Figure 10.2 The amount of tag ambiguity for word types in the Brown and WSJ corpora, from the Treebank-3 (45-tag) tagging. These statistics include punctuation as words, and assume words are kept in their original case.

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two*	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, sing.	*IBM*	$	dollar sign	*$*
NNPS	proper noun, plural	*Carolinas*	#	pound sign	*#*
PDT	predeterminer	*all, both*	"	left quote	' or "
POS	possessive ending	*'s*	"	right quote	' or "
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	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*			

Some words have up to 6 tags

	Sentence	Tag
1	Earnings took a **back** seat	
2	A small yard in the **back**	
3	Senators **back** the bill	
4	He started to **back** towards the door	
5	To **buy back** stock.	
6	I was young **back** then.	

Most frequent class baseline

Many words are easy to disambiguate, because their different tags aren't equally likely.

Simplistic baseline for POS tagging: given an ambiguous word, choose the tag which is most frequent in the training corpus.

Most Frequent Class Baseline: Always compare a classifier against a baseline at least as good as the most frequent class baseline (assigning each token to the class it occurred in most often in the training set).

How good is the baseline?

This lets us know how hard the task is (and how much room for improvement real models have).

Accuracy for POS taggers is measured as the percent of tags that are correctly labeled when compared to human labels on a test set.

Most Frequent Class Baseline:	92%
State of the art in POS tagging:	97%

(Much harder for other languages and other genres)

Sequence Models

A **sequence model** or **sequence classifier** is a model whose job is to assign a label or class to each unit in a sequence, thus mapping a sequence of observations to a sequence of labels.

A Hidden Markov Model (HMM) is a probabilistic sequence model: given a sequence of words, it computes a probability distribution over possible sequences of labels and chooses the best label sequence.

What is hidden?

We used a Markov model in n-gram LMs. This kind of model is sometimes called a Markov chain**.** It is useful when we need to compute a probability for a sequence of observable events.

In many cases, however, the events we are interested in are **hidden**: we don't observe them directly. We don't normally observe part-of-speech tags in a text. Rather, we see words, and must infer the tags from the word sequence.

We call the tags hidden because they are not observed.

HMMs for tagging

Basic equation for HMM tagging

$$\hat{t}_1^N = \arg\max_{t_1^N} P(t_1^N | w_1^N)$$

Find the best (hidden) tag sequence, given an (observed) word sequence

Use Bayes rule

$$= \arg\max_{t_1^N} \frac{P\left(w_1^N \middle| t_1^N\right) P(t_1^N)}{P(w_1^N)}$$

$$= \arg\max_{t_1^N} P\left(w_1^N \middle| t_1^N\right) P(t_1^N)$$

Simplifying Assumptions

1. Probability of a word only depends on its own tag, and it is intendent of neighboring word and tags

$$P(t_1^N | w_1^N) \approx \prod_{i=1}^{N} p\,(w_i | t_i)$$

2. Bigram assumption. The probability of a tag depends only on previous tag, not the whole tag sequence.

$$P(t_1^N | w_1^N) \approx \prod_{i=1}^{N} p\,(t_i | t_{i-1})$$

Simplifying Assumptions

1. Probability of a word only depends on its own tag, and it is intendent of neighboring word and tags

$$P(t_1^N | w_1^N) \approx \prod_{i=1}^{N} p\,(w_i | t_i)$$

Emission probability

2. Bigram assumption. The probability of a tag depends only on previous tag, not the whole tag sequence.

$$P(t_1^N | w_1^N) \approx \prod_{i=1}^{N} p\,(t_i | t_{i-1})$$

Transition probability

Combining everything

$$\hat{t}_1^N = \arg\max_{t_1^N} P(t_1^N | w_1^N) \approx \prod_{i=1}^{N} p\,(t_i | t_{i-1}) p(w_i | t_i)$$

HMM Tagger Components

Transition probability

$$P(t_i|t_{i-1}) = \frac{count(t_{i-1},t_i)}{count(t_{i-1})}$$

In the WSJ corpus, a modal verb occurs 13,124 times. 10,471 time the MD is followed by a verb. Therefore,

$$P(VB|MD) = \frac{10{,}471}{13{,}124} = .80$$

Transition probabilities are called the **A probabilities**.

HMM Tagger Components

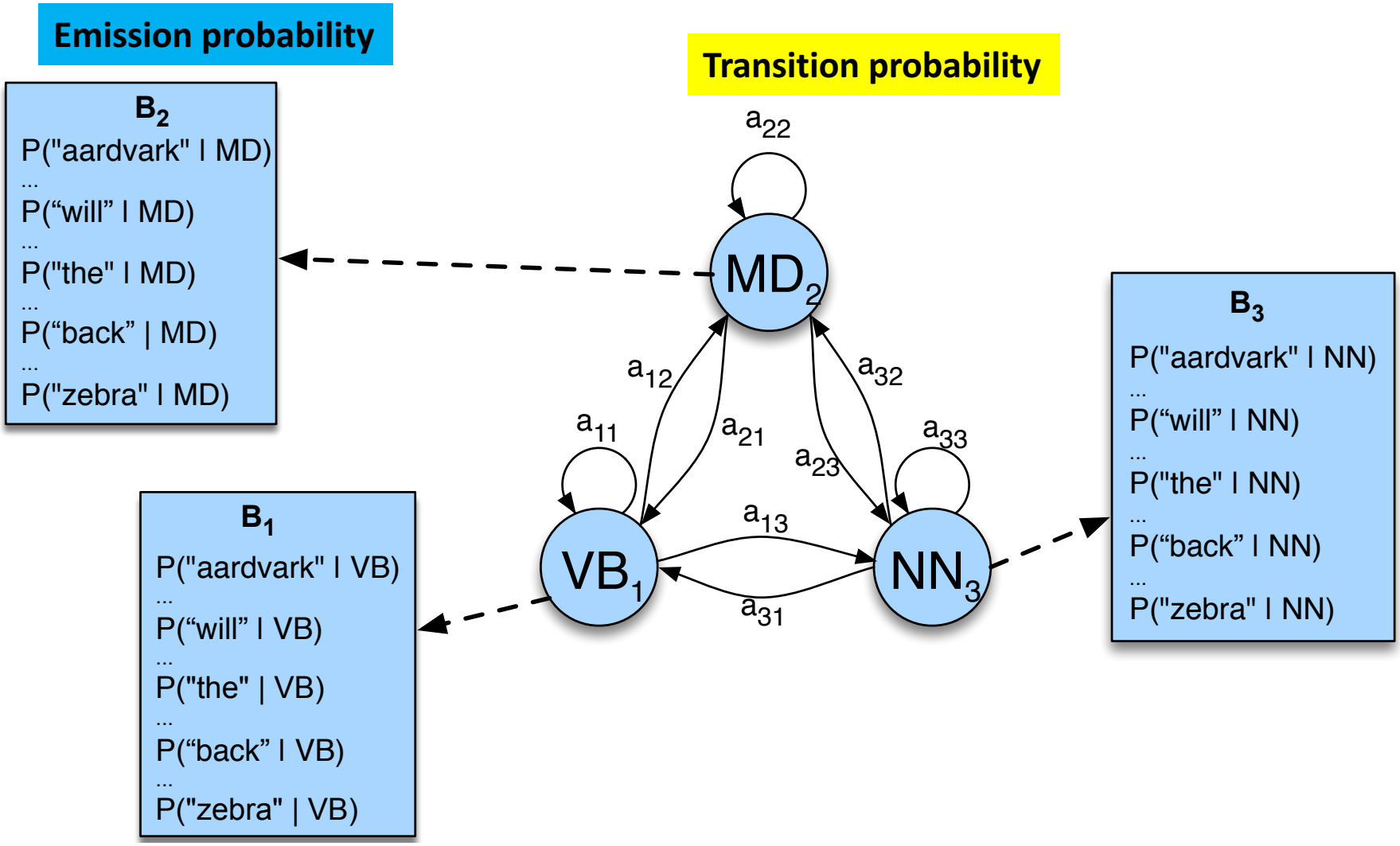
Emission probability

$$P(w_i|t_i) = \frac{count(w_i,t_i)}{count(t_i)}$$

Of the 13,124 occurrences of modal verbs in the WSJ corpus, the word ***will*** represents 4,046 of the words tagged as MD.

$$P(will|MD) = \frac{4{,}046}{13{,}124} = .31$$

Emission probabilities are called the **B probabilities**.


Emission probability
Transition probability
B2
P("aardvark" I MD)
...
P(“will” I MD)
...
P("the" I MD)
...
P(“back” | MD)
...
P("zebra" I MD)
B1
P("aardvark" I VB)
...
P(“will” I VB)
...
P("the" | VB)
...
P(“back” I VB)
...
P("zebra" | VB)
B3
P("aardvark" I NN)
...
P(“will” I NN)
...
P("the" I NN)
...
P(“back” I NN)
...
P("zebra" I NN)
MD2
VB1
NN3
a22
a12
a21
a32
a11
a33
a23
a13
a31

HMM decoding

For any model, such as an HMM, that contains hidden variables, the task of determining the hidden variables sequence corresponding to the sequence of observations is called **decoding**.

Decoding: Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, \ldots, o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \ldots q_T$.

$$\hat{t}_1^N = \arg\max_{t_1^N} P(w_1^N | t_1^N) P(t_1^N)$$

HMM decoding

Input:	Let	us	learn	about	HMMs
Output Best Labels:	VB	PRP	VB	IN	NNP

Compute probability for **all possible sequence of labels:**

Let	us	learn	about	HMMs	
VB	PRP	VB	IN	NNP	p=0.45
IN	VB	VB	NN	DT	p=0.03
		…			
PRP	.	NN	IN	WP	p=0.00006

How many label sequences?

Input:	Let	us	learn	about	HMMs
	CC	CC	CC	CC	CC
	CD	CD	CD	CD	CD
	DT	DT	DT	DT	DT
	EX	EX	EX	EX	EX
	FW	FW	FW	FW	FW
	IN	IN	IN	IN	IN
	JJ	JJ	JJ	JJ	JJ
	JJR	JJR	JJR	JJR	JJR
	JJS	JJS	JJS	JJS	JJS
	LS	LS	LS	LS	LS
	MD	MD	MD	MD	MD
	NN	NN	NN	NN	NN
	NNS	NNS	NNS	NNS	NNS
	NNP	NNP	NNP	NNP	NNP
	NNPS	NNPS	NNPS	NNPS	NNPS
	PDT	PDT	PDT	PDT	PDT

How many label sequences?

Input:	Let	us	learn	about	HMMs
	CC	CC	CC	CC	CC
	CD	CD	CD	CD	CD
	DT	DT	DT	DT	DT
	JJS	JJS	JJS	JJS	JJS
	LS	LS	LS	LS	LS
	MD	MD	MD	MD	MD
	NN	NN	NN	NN	NN
	NNS	NNS	NNS	NNS	NNS
	NNP	NNP	NNP	NNP	NNP
	NNPS	NNPS	NNPS	NNPS	NNPS
	PDT	PDT	PDT	PDT	PDT

For POS tagging a sentence of length T = 5, and number of states (tags) = 45

$N^T = 60{,}466{,}176$

Dynamic Programming

Coined by Richard Bellman in 1940s

- “My boss, Secretary of Defense, actually had a pathological fear and hatred of the word research *Dynamic* has a very interesting property as an adjective, and that it's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible!”

Method for solving complex problems by breaking them down into simpler sub-problems and storing their solutions

Technique of storing solutions to sub-problems instead of recomputing them is called ”**memoization**”

Dynamic Programming

Fibonacci Series

$$fib(n) = fib(n - 1) + fib(n - 2)$$

- fib(5)
  - fib(4) + fib(3)
  - (fib(3) + fib(2)) + (fib(2) + fib(1))
  - ((fib(2) + fib(1)) + (fib(1) + fib(0))) + ((fib(1) + fib(0)) + fib(1))
  - (((fib(1) + fib(0)) + fib(1)) + (fib(1) + fib(0))) + ((fib(1) + fib(0)) + fib(1))

Instead of calling fib(3) multiple times, we should store it and lookup instead of recomputing

Viterbi Algorithm

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
$viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
$backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
for each state s **from** 1 **to** N **do**
$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$; termination step
bestpath $\leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Viterbi Algorithm

function VITERBI(*observations* of len *T*,*state-graph* of len *N*) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
for each state *s* **from** 1 **to** *N* **do** ; initialization step
viterbi[*s*,1] $\leftarrow \pi_s * b_s(o_1)$

The complexity of the Viterbi algorithm for this HMM is $O(T * N^2)$.

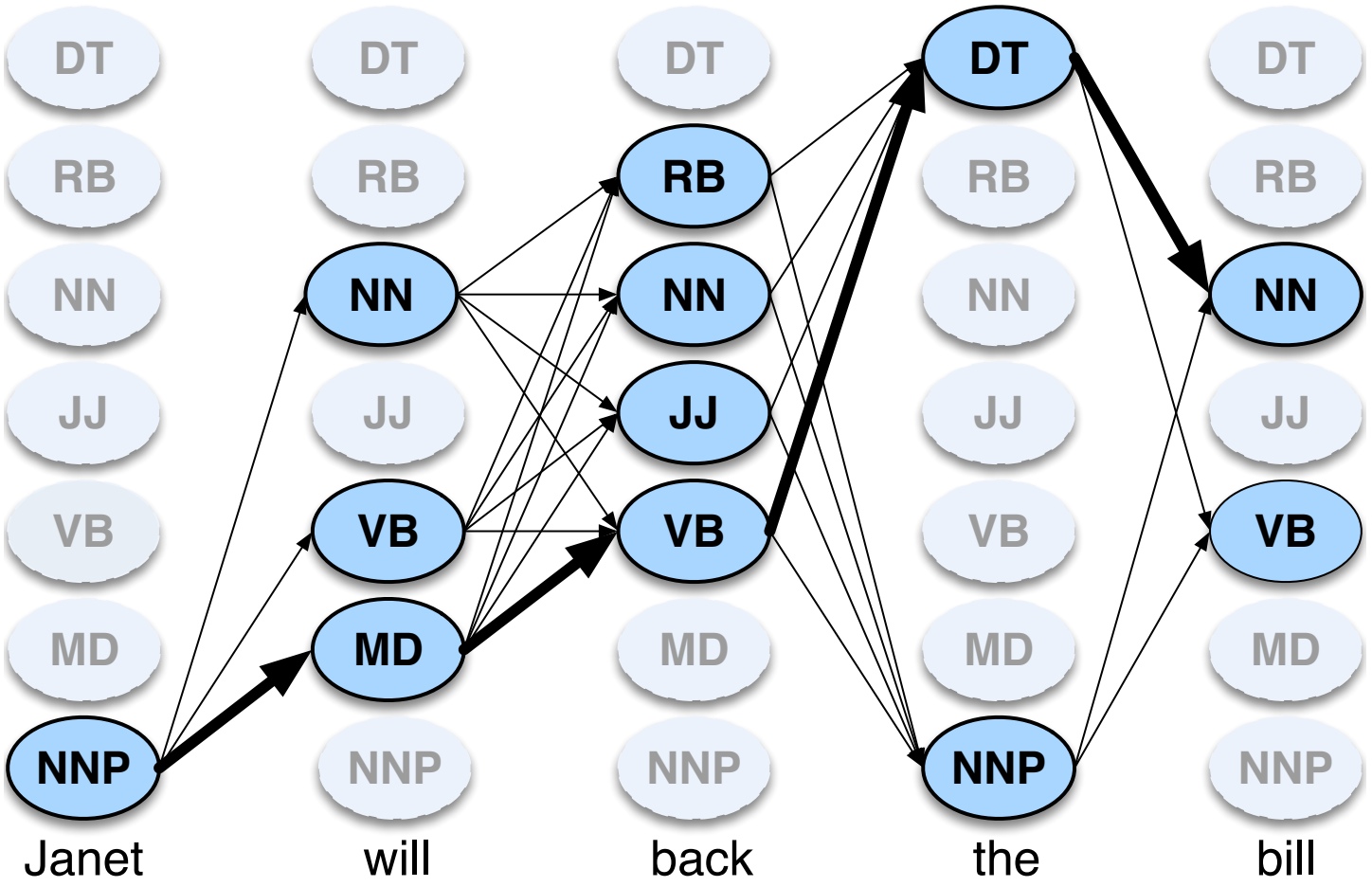
So POS tagging a sentence of length T = 5 with N = 45 states (tags) goes from:

$N^T = 60{,}466{,}176$

to computations

$T * N^2 = 10{,}125$ computations!

Viterbi Lattice



	NNP	MD	VB	JJ	NN	RB	DT
$<s>$	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

Figure 8.7 The A transition probabilities $P(t_i|t_{i-1})$ computed from the WSJ corpus without smoothing. Rows are labeled with the conditioning event; thus $P(VB|MD)$ is 0.7968.

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Figure 8.8 Observation likelihoods B computed from the WSJ corpus without smoothing, simplified slightly.

Trigram HMMs

So far, we had a bigram assumption. The probability of a tag depends only on previous tag, not the whole tag sequence.

$$P(t_1^N) \approx \prod_{i=1}^{N} p\,(t_i|t_{i-1})$$

We could extend it to a trigram model

$$P(t_1^N) \approx \prod_{i=1}^{N} p\,(t_i|t_{i-1}, t_{i-2})$$

Trigram HMMs

So far, we had a bigram assumption. The probability of a tag depends only on previous tag, not the whole tag sequence.

$$P(t_1^N) \approx \prod^{N} p\,(t_i|t_{i-1})$$

We co

The complexity of the trigram HMM increases from $O(N^2T)$ to $O(N^3T)$. The number of states (N) gets larger since we have to compare every pair of 45 tags, instead of just each tag, so we have 45^3 = **91,125 computations per column.**

Beam Search

One common solution to the complexity problem is the use of beam search decoding. In beam search, instead of keeping the entire column of states at each time point *t*, we just keep the best few hypothesis at that point.

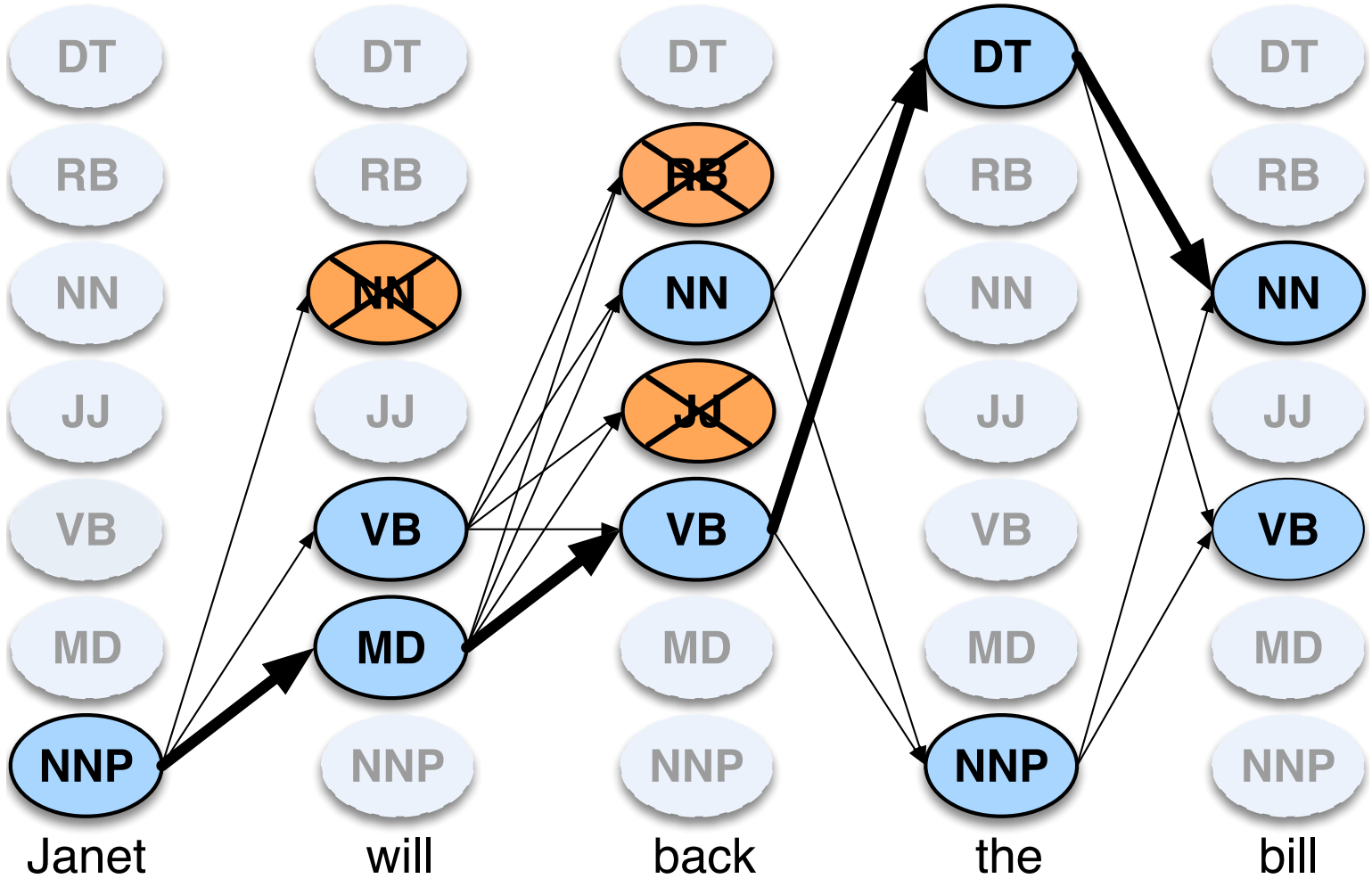
At time *t* this requires computing the Viterbi score for each of the *N* cells, sorting the scores, and keeping only the best-scoring states. The rest are pruned out and not continued forward to time *t*+1.

Unknown words

To achieve high accuracy with POS taggers, it is also important to have a good model for dealing with **unknown words**.

Proper names and acronyms are created very often, and even new common nouns and verbs enter the language at a surprising rate.

Beam Search



Unknown words

One useful feature for distinguishing parts of speech is **word shape** (proper nouns start with a capital).

The strongest feature is **morphology**.

Words that end in

- **-s** tend to be **plural nouns (NNS)**
- **-ed** tend to be **past participles (VBN)**
- **-able** tend to be **adjectives (JJ)**
- and so on

Learning suffix model

Store the final letter sequence (suffixes) for up to 10 letters.

For each such sequence, record the probability of the tag that it was associated with during training.

Use back-off to smooth these probabilities for. Successively shorter sequences.

Trigram HMM with unknown word handling:	96.7%
State of the art neural networkPOS tagging:	97%

Maximum Entropy Markov Models

Could we add features like word shape and suffixes directly into the model in a clean way? We had this for classification with **logistic regression**. But logistic regression isn't a sequence model, since it assigns a class to a single observation.

We can turn logistic regression into a discriminative sequence model simply by running it on successive words, using the class assigned to the prior word as a feature in the classification of the next word. This is called a **MEMM**.

MEMMs v HMMs

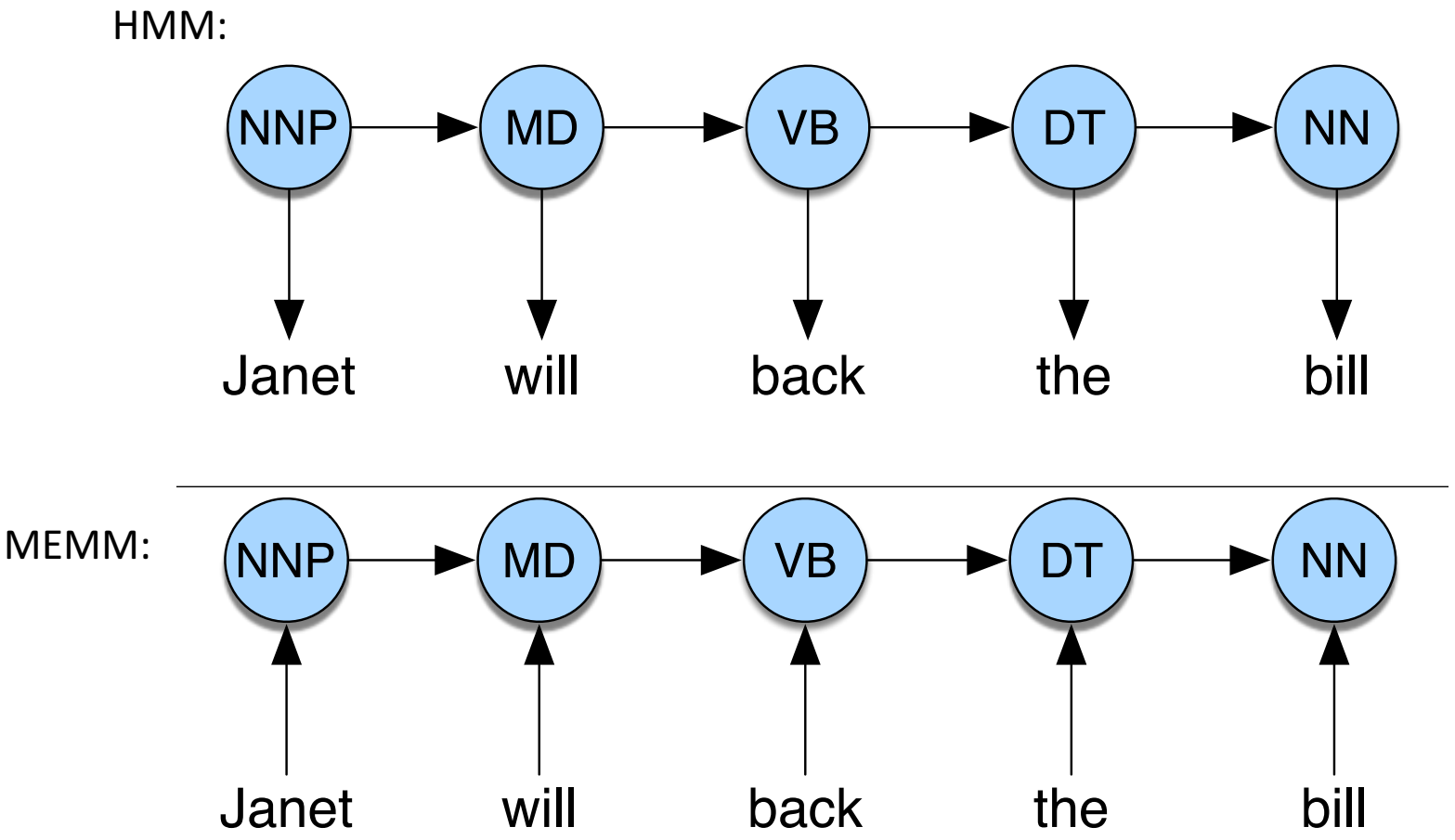
HMM:

$$\begin{aligned}\hat{T} &= \operatorname*{argmax}_T P(T|W) \\ &= \operatorname*{argmax}_T P(W|T)P(T) \\ &= \operatorname*{argmax}_T \prod_i P(word_i|tag_i) \prod_i P(tag_i|tag_{i-1})\end{aligned}$$

MEMM:

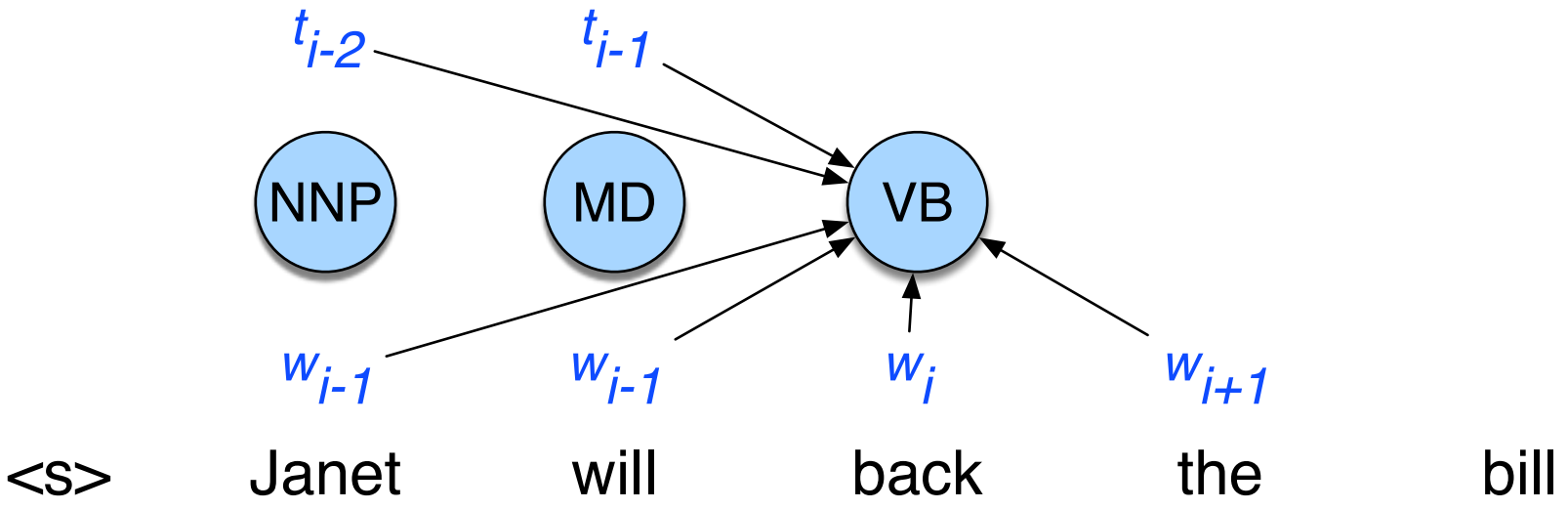
$$\begin{aligned}\hat{T} &= \operatorname*{argmax}_T P(T|W) \\ &= \operatorname*{argmax}_T \prod_i P(t_i|w_i, t_{i-1})\end{aligned}$$

MEMMs v HMMs


HMM:
NNP
MD
VB
DT
NN
Janet
will
back
the
bill
MEMM:
NNP
MD
VB
DT
NN
Janet
will
back
the
bill

Features in a MEMM

We can build MEMMs that don't just condition on w_i and t_{i-1}. It is easy to incorporate lots of features in a discriminative sequence model.



Feature templates

A basic MEMM part-of-speech tagger conditions on the observation word it- self, neighboring words, and previous tags, and various combinations, using feature templates like the following

$$\langle t_i, w_{i-2}\rangle, \langle t_i, w_{i-1}\rangle, \langle t_i, w_i\rangle, \langle t_i, w_{i+1}\rangle, \langle t_i, w_{i+2}\rangle$$
$$\langle t_i, t_{i-1}\rangle, \langle t_i, t_{i-2}, t_{i-1}\rangle,$$
$$\langle t_i, t_{i-1}, w_i\rangle, \langle t_i, w_{i-1}, w_i\rangle \langle t_i, w_i, w_{i+1}\rangle,$$

Janet/NNP will/MD back/VB the/DT bill/NN, when w_i is the word *back*

$$t_i = \text{VB and } w_{i-2} = \text{Janet}$$
$$t_i = \text{VB and } w_{i-1} = \text{will}$$
$$t_i = \text{VB and } w_i = \text{back}$$
$$t_i = \text{VB and } w_{i+1} = \text{the}$$
$$t_i = \text{VB and } w_{i+2} = \text{bill}$$
$$t_i = \text{VB and } t_{i-1} = \text{MD}$$
$$t_i = \text{VB and } t_{i-1} = \text{MD and } t_{i-2} = \text{NNP}$$
$$t_i = \text{VB and } w_i = \text{back and } w_{i+1} = \text{the}$$

Features for unknown words

w_i contains a particular prefix (from all prefixes of length ≤ 4)
w_i contains a particular suffix (from all suffixes of length ≤ 4)
w_i contains a number
w_i contains an upper-case letter
w_i contains a hyphen
w_i is all upper case
w_i's word shape
w_i's short word shape
w_i is upper case and has a digit and a dash (like *CFC-12*)
w_i is upper case and followed within 3 words by Co., Inc., etc.

Features for *well-dressed*

prefix(w_i) = `w`
prefix(w_i) = `we`
prefix(w_i) = `wel`
prefix(w_i) = `well`
suffix(w_i) = `ssed`
suffix(w_i) = `sed`
suffix(w_i) = `ed`
suffix(w_i) = `d`
has-hyphen(w_i)
word-shape(w_i) = `xxxx-xxxxxxx`
short-word-shape(w_i) = `x-x`

Guest speaker on Wednesday

Jacob Devlin
Google

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding