

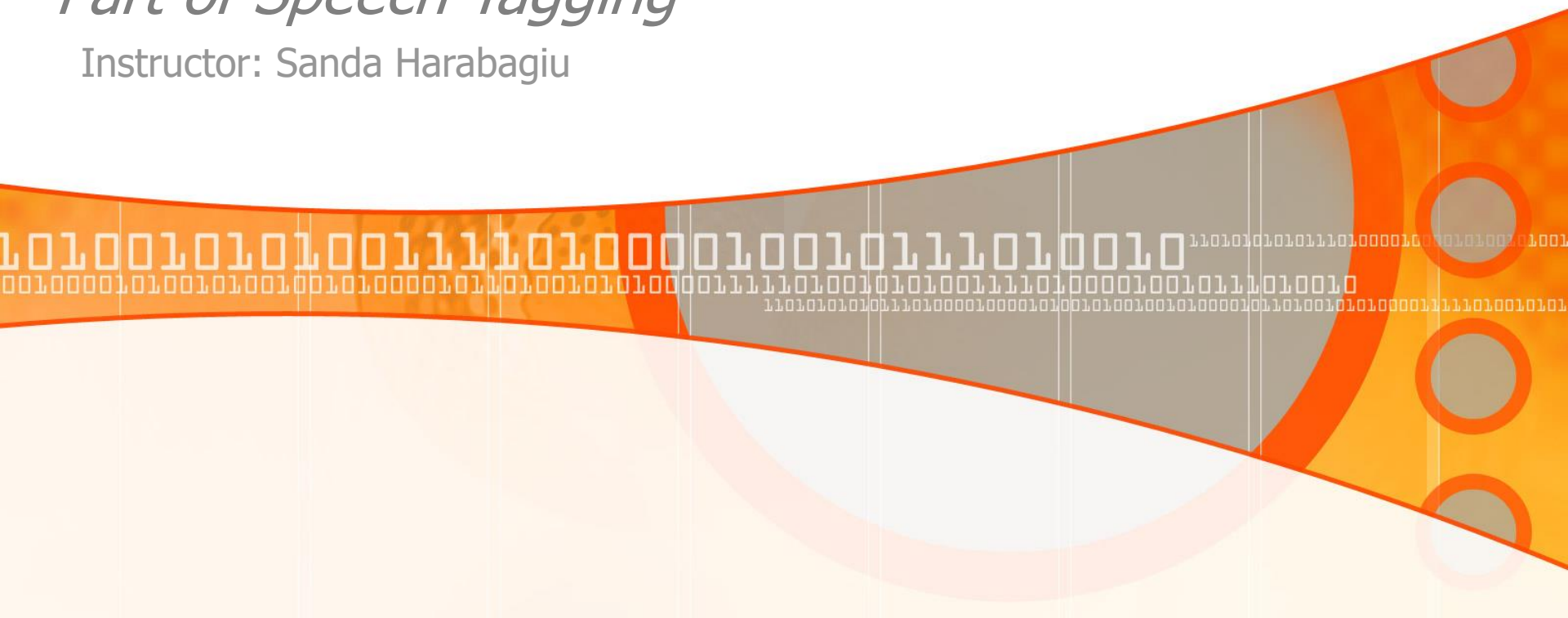
Natural Language Processing

CS 6320

Lecture 7

Part of Speech Tagging

Instructor: Sanda Harabagiu



Today

- *Define Parts of speech (POS)*
- *POS Tagsets*
- *HMM POS Tagging*
 - ❑ *Viterbi algorithm*
 - ❑ *Maximum Entropy Markov Models*
- *Bidirectionality on POS Taggers*
- *Conditional Random Fields*

What are Parts-of-speech?????

- ❑ *Parts-of-speech represent the word class, or lexical category of a word, e.g. **NOUNS** or **VERBS** or **ADJECTIVES** or **DETERMINERS** or **ADVERBS**, etc.*

- ❑ *Parts-of-speech can be divided into two broad super-categories: **closed class** types and **open class** types.*
 - *Closed classes are those with relatively fixed membership, such as prepositions—new prepositions are rarely coined.*
 - *By contrast, **nouns** and verbs are **open** classes—new nouns and verbs like iPhone or to fax are continually being created or borrowed.*

Parts of Speech

- Four major open classes occur in the languages of the world: **nouns**, **verbs**, **adjectives**, and **adverbs**. English has all four, although not every language has all of them!
- ❑ **NOUNS:** The syntactic class noun includes the words for most people, places, or things.
 - Nouns include concrete terms like ship and chair, abstractions like bandwidth and relationship, and verb-like terms like pacing as in:
His pacing to and from became quite annoying.
 - What defines a noun in English, is its ability to:
 1. occur with determiners (a goat, its bandwidth, Plato's Republic),
 2. to take possessives (IBM's annual revenue), and
 3. to occur in the plural form (goats, abaci)

More about Nouns

Open class nouns fall into two classes:

1. **Proper nouns**, like Regina, Colorado, and Columbia, are names of specific persons or entities.
 - In English, they generally aren't preceded by articles (e.g., "the book is upstairs", but "Regina is upstairs").
 - In written English, proper nouns are usually **capitalized**.
2. **Common nouns** are divided in two classes:
 - a. **count nouns**: - allow grammatical enumeration, occurring in both the singular and plural (goat/goats, relationship/relationships) and they can be counted (one goat, two goats).
 - b. **mass nouns**: used when something is conceptualized as a homogeneous group. So words like snow, salt, and communism are not counted (i.e., *two snows or *two communisms).
Mass nouns can also appear without articles, where singular count nouns cannot ("Snow is white" but not "*Goat is white").

Verbs and Adjectives

- **Verbs** refer to actions and processes, including main verbs like draw, provide, and go.
 - English verbs have **inflections** (non-third-person-sg (eat), third-person-sg (eats), progressive (eating), past participle (eaten)).
- **Adjectives**, include many terms for **properties** or **qualities**. Most languages have adjectives for the concepts of color (white, black), age (old, young), and value (good, bad), but there are languages without adjectives.

Adverbs

❑ Adverbs are a hodge-podge of words in both form and meaning. E.g.:

Actually, I ran home extremely quickly yesterday

- Adverbs are modifying something (often verbs, hence the name “adverb”)
 - **Directional adverbs** or **locative adverbs** (home, here, downhill) specify the direction or location of some action;
 - **Degree adverbs** (extremely, very, somewhat) specify the extent of some action, process, or property;
 - **Manner adverbs** (slowly, slinkily, delicately) describe the manner of some action or process;
 - **Temporal adverbs** describe the time that some action or event took place (yesterday).

Closed Classes

- *The closed classes differ more from language to language than do the open classes.*
- *Some of the important closed classes in English include:*

prepositions: on, under, over, near, by, at, from, to, with

particles: up, down, on, off, in, out, at, by

determiners: a, an, the

conjunctions: and, but, or, as, if, when

pronouns: she, who, I, others

auxiliary verbs: can, may, should, are

numerals: one, two, three, first, second, third

Prepositions

❑ **Prepositions** occur **before noun phrases**.

➤ Semantically, prepositions often indicate spatial or temporal relations, whether literal (on it, before then, by the house) or metaphorical (on time, with gusto, beside herself), but often indicate other relations as well, like marking the agent in:

Hamlet was written by Shakespeare

➤ **Particles**: A particle resembles a preposition or an adverb and **is used in combination with a verb**.

▪ Particles often have extended meanings that aren't quite the same as the prepositions they resemble, as in the particle over in:

she turned the paper over.

Prepositions from CELEX

of	540,085	through	14,964	worth	1,563	pace	12
in	331,235	after	13,670	toward	1,390	nigh	9
for	142,421	between	13,275	plus	750	re	4
to	125,691	under	9,525	till	686	mid	3
with	124,965	per	6,515	amongst	525	o'er	2
on	109,129	among	5,090	via	351	but	0
at	100,169	within	5,030	amid	222	ere	0
by	77,794	towards	4,700	underneath	164	less	0
from	74,843	above	3,056	versus	113	midst	0
about	38,428	near	2,026	amidst	67	o'	0
than	20,210	off	1,695	sans	20	thru	0
over	18,071	past	1,575	circa	14	vice	0

English Particles

aboard	aside	besides	forward(s)	opposite	through
about	astray	between	home	out	throughout
above	away	beyond	in	outside	together
across	back	by	inside	over	under
ahead	before	close	instead	overhead	underneath
alongside	behind	down	near	past	up
apart	below	east, etc.	off	round	within
around	beneath	eastward(s),etc.	on	since	without

Phrasal Verbs

- ❑ *A verb and a particle that act as a single syntactic and/or semantic unit are called **a phrasal verb**.*
- *The meaning of phrasal verbs is often problematically non compositional—not predictable from the distinct meanings of the verb and the particle.*
 - *Thus:*
 - *turn down means something like ‘reject’,*
 - *rule out means ‘eliminate’,*
 - *find out means ‘discover’, and*
 - *go on means ‘continue’.*

Determiners and Articles

- ❑ *A closed class that occurs with nouns, often marking the beginning of a noun phrase, is the **determiner**.*
- *One small subtype of determiners is **the article**.*

*English has three articles: **a**, **an**, and **the**.*

*A and an mark a noun phrase as **indefinite**, while the can mark it as definite; definiteness is a discourse property!*

- *Other determiners include **this** and **that** (this chapter, that page).*

Conjunctions

- ❑ *Conjunctions join two phrases, clauses, or sentences.*
- **Coordinating conjunctions** like and, or, and but join two elements of equal status.
- **Subordinating conjunctions** are used when one of the elements has some embedded status. For example, that in:

“I thought that you might like some milk”

is a subordinating conjunction that links the main clause “I thought” with the subordinate clause “you might like some milk”. This clause is called subordinate because this entire clause is the “content” of the main verb “thought”.

Subordinating conjunctions like that which link a verb to its complementizer argument in this way are also called complementizers.

Conjunctions

and	514,946	yet	5,040	considering	174	forasmuch as	0
that	134,773	since	4,843	lest	131	however	0
but	96,889	where	3,952	albeit	104	immediately	0
or	76,563	nor	3,078	providing	96	in as far as	0
as	54,608	once	2,826	whereupon	85	in so far as	0
if	53,917	unless	2,205	seeing	63	inasmuch as	0
when	37,975	why	1,333	directly	26	insomuch as	0
because	23,626	now	1,290	ere	12	insomuch that	0
so	12,933	neither	1,120	notwithstanding	3	like	0
before	10,720	whenever	913	according as	0	neither nor	0
though	10,329	whereas	867	as if	0	now that	0
than	9,511	except	864	as long as	0	only	0
while	8,144	till	686	as though	0	provided that	0
after	7,042	provided	594	both and	0	providing that	0
whether	5,978	whilst	351	but that	0	seeing as	0
for	5,935	suppose	281	but then	0	seeing as how	0
although	5,424	cos	188	but then again	0	seeing that	0
until	5,072	supposing	185	either or	0	without	0



Pronouns

- ❑ **Pronouns** are words that often act as a kind of shorthand for referring to some personal noun phrase or entity or event.
- **Personal pronouns** refer to persons or entities (you, possessive she, I, it, me, etc.).
- **Possessive pronouns** are forms of personal pronouns that indicate either actual possession or more often just an abstract relation between the person and some object (my, your, his, her, its, one's, our, their).
- **Wh-pronouns** (what, who, whom, whoever) are used in certain question forms, or may also act as complementizers (**Frida, who married Diego. . .**)

Auxiliary Verbs

- **Auxiliaries** mark semantic features of a main verb:

➤ whether an action takes place in the present, past, or future (**tense**),

She was gone all week.

➤ whether it is completed (**aspect**),

She has been gone all week.

➤ whether it is negated (**polarity**), and

She hasn't eaten the bagel.

➤ whether an action is necessary, possible, suggested, or copula desired (**mood**).

She would go to school.

Copula and Modals

- ❑ *English auxiliaries include the copula verb **be**, the two verbs **do** and modal **have**, along with their inflected forms, as well as a class of modal verbs.*
- *Be is called a copula because it connects subjects with certain kinds of predicate nominals and adjectives:*

He is a student.
- *The verb **have** can mark the perfect tenses: (I have gone, I had gone), and be is used as part of the passive (We were robbed) or progressive (We are leaving) constructions.*
- ***Modals** are used to mark the mood associated with the event depicted by the main verb: can indicates ability or possibility, may permission or possibility, must necessity. There is also a modal use of have (e.g., I have to go).*

Interjections and Negatives

English also has many words of more or less unique function:

- **interjection:** (oh, hey, alas, uh, um),
- **negatives** (no, not),
- **politeness markers** (please, thank you),
- **greetings** (hello, goodbye), and
- **the existential there** (there are two glasses on the table).

POS Tagging

- The process of assigning a part-of-speech or lexical class marker to each word in a collection.*

WORD

tag

the

DET

koala

N

put

V

the

DET

keys

N

on

P

the

DET

table

N



POS Tagging-- Choosing a Tagset

- *There are so many parts of speech, potential distinctions we can draw*
- *To do POS tagging, we need to choose a standard set of tags to work with*
- *Could pick very coarse tagsets*
 - *N, V, Adj, Adv.*
- *More commonly used set is finer grained, the “Penn TreeBank tagset”, 45 tags*
 - *PRP\$, WRB, WP\$, VBG*
- *Even more fine-grained tagsets exist*

Penn TreeBank POS Tagset

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	<i>and, but, or</i>	SYM	symbol	<i>+, %, &</i>
CD	cardinal number	<i>one, two, three</i>	TO	“to”	<i>to</i>
DT	determiner	<i>a, the</i>	UH	interjection	<i>ah, oops</i>
EX	existential ‘there’	<i>there</i>	VB	verb, base form	<i>eat</i>
FW	foreign word	<i>mea culpa</i>	VBD	verb, past tense	<i>ate</i>
IN	preposition/sub-conj	<i>of, in, by</i>	VBG	verb, gerund	<i>eating</i>
JJ	adjective	<i>yellow</i>	VCN	verb, past participle	<i>eaten</i>
JJR	adj., comparative	<i>bigger</i>	VBP	verb, non-3sg pres	<i>eat</i>
JJS	adj., superlative	<i>wildest</i>	VBZ	verb, 3sg pres	<i>eats</i>
LS	list item marker	<i>1, 2, One</i>	WDT	wh-determiner	<i>which, that</i>
MD	modal	<i>can, should</i>	WP	wh-pronoun	<i>what, who</i>
NN	noun, sing. or mass	<i>llama</i>	WP\$	possessive wh-	<i>whose</i>
NNS	noun, plural	<i>llamas</i>	WRB	wh-adverb	<i>how, where</i>
NNP	proper noun, singular	<i>IBM</i>	\$	dollar sign	<i>\$</i>
NNPS	proper noun, plural	<i>Carolinas</i>	#	pound sign	<i>#</i>
PDT	predeterminer	<i>all, both</i>	“	left quote	<i>‘ or “</i>
POS	possessive ending	<i>’s</i>	”	right quote	<i>’ or ”</i>
PRP	personal pronoun	<i>I, you, he</i>	(left parenthesis	<i>[, (, {, <</i>
PRP\$	possessive pronoun	<i>your, one’s</i>)	right parenthesis	<i>],), }, ></i>
RB	adverb	<i>quickly, never</i>	,	comma	<i>,</i>
RBR	adverb, comparative	<i>faster</i>	.	sentence-final punc	<i>. ! ?</i>
RBS	adverb, superlative	<i>fastest</i>	:	mid-sentence punc	<i>: ; ... - -</i>
RP	particle	<i>up, off</i>			

Using the Penn Tagset

EXAMPLE ANNOTATIONS:

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 ./.

Corpora labeled with POS tags

- ❑ Corpora labeled with parts-of-speech are crucial training (and testing) sets for statistical tagging algorithms.
 - Three main tagged corpora are consistently used for training and testing part-of-speech taggers for English:
 1. The **Brown corpus** is a million words of samples from 500 written texts from different genres published in the WSJ United States in 1961.
 2. The **WSJ corpus** contains a million words published in the Switchboard Wall Street Journal in 1989.
 3. The **Switchboard corpus** consists of 2 million words of telephone conversations collected in 1990-1991.
- The corpora were created by running an automatic part-of-speech tagger on the texts and then human annotators hand-corrected each tag.

POS Tagging

- *Part-of-speech tagging is the process of assigning a part-of-speech marker to each word in an input text.*

The input to a tagging algorithm is (1) a sequence of (tokenized) words and (2) a tagset, and the output is a sequence of tags, one per token.

*BUT: **lexical ambiguity!!!!!!!!!!!!!!!!!!!!!!***

❑ *Words often have more than one POS: back*

- *The **back** door = JJ*
 - *On my **back** = NN*
 - *Win the voters **back** = RB*
 - *Promised to **back** the bill = VB*
- *The POS tagging problem is to determine the POS tag for a particular instance of context for a word.*

Lexical ambiguity

- *POS Tagging is a disambiguation task because words are ambiguous —have more than one possible part-of-speech—and the goal is to find the correct tag for the context in which the word appears!*
- *How bad is the ambiguity????*

How Hard is POS Tagging? Measuring Ambiguity

- POS Tagging is a disambiguation task because words are ambiguous —have more than one possible part-of-speech—and the goal is to find the correct tag for the context in which the word appears!*

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%)

- Some of the most ambiguous frequent words are: that, back, down, put and set;*

Example: the lexical ambiguity of “back”

- *The word “back” may be tagged with 6 different parts-of-speech:*

earnings growth took a **back/JJ** seat
a small building in the **back/NN**
a clear majority of senators **back/VBP** the bill
Dave began to **back/VB** toward the door
enable the country to buy **back/RP** about debt
I was twenty-one **back/RB** then

Rule-Based Tagging: Some good news!!!

- ❑ *Many words are easy to disambiguate, because their different tags aren't equally likely.*
- *a simplistic baseline algorithm for part-of-speech tagging: given an ambiguous word, choose the tag which is most frequent in the training corpus.*
- *This is a key concept:*

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 this baseline? A standard way to measure the performance of part-of-speech taggers is **accuracy**: the percentage of tags correctly labeled (matching human labels on a test set)*

Starting point

- *If we train on the WSJ training corpus and test on sections 22-24 of the same corpus the most-frequent-tag baseline achieves an accuracy of **92.34%**.*
- *The state of the art in part-of-speech tagging on this dataset is around **97%** tag accuracy, a performance that is achievable by most algorithms (HMMs, MEMMs, neural networks, rule-based algorithms)*

Hidden Markov Model for POS Tagging

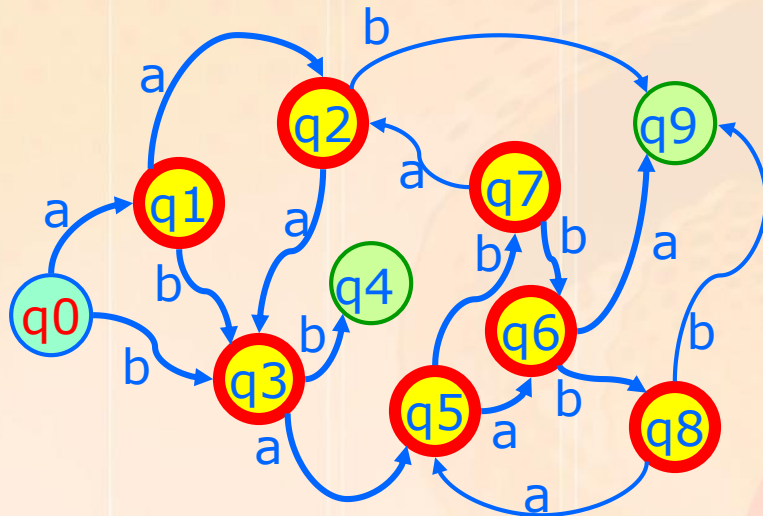
- ❑ *The Hidden Markov Model is a probabilistic sequence model:*
 - *given a sequence of units (words, sentences, whatever), it computes **a probability distribution** over possible sequences of labels and chooses the best label sequence.*
- *The HMM is based on augmenting the **Markov chain**.*

Definitions

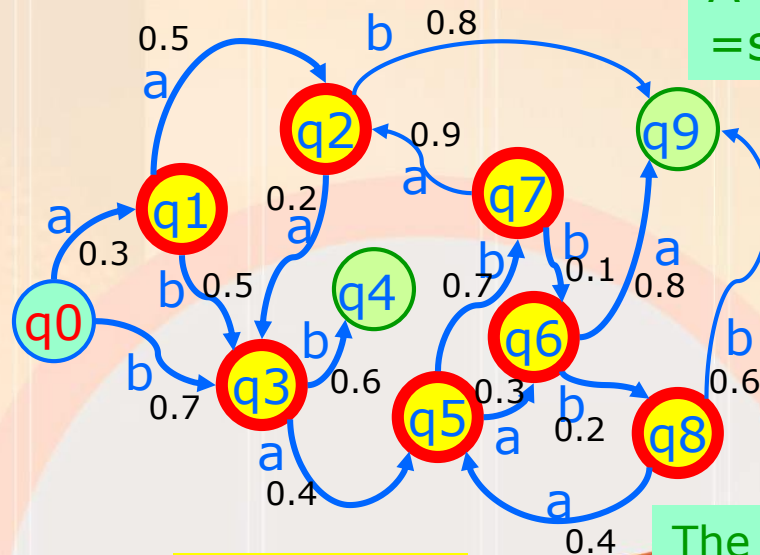
- A *weighted finite-state automaton* adds probabilities to the arcs
 - The sum of the probabilities leaving any arc must sum to one
- A *Markov chain* is a special case of a WFST in which the input sequence uniquely determines which states the automaton will go through
- ❑ *Markov chains can't represent inherently ambiguous problems*
 - Useful for assigning probabilities to unambiguous sequences

Formalizing Hidden Markov Model taggers

- HMM is an extension of finite automata
 - A FSA is defined by a *set of states* + a *set of transitions between states* that are taken based on observations.
- A *weighted finite-state automaton* is an augmentation of the FSA in which the arcs are associated with a *probability* – indicating how likely that path is to be taken.



FSA
Input: aab



WFSA
Input: aab

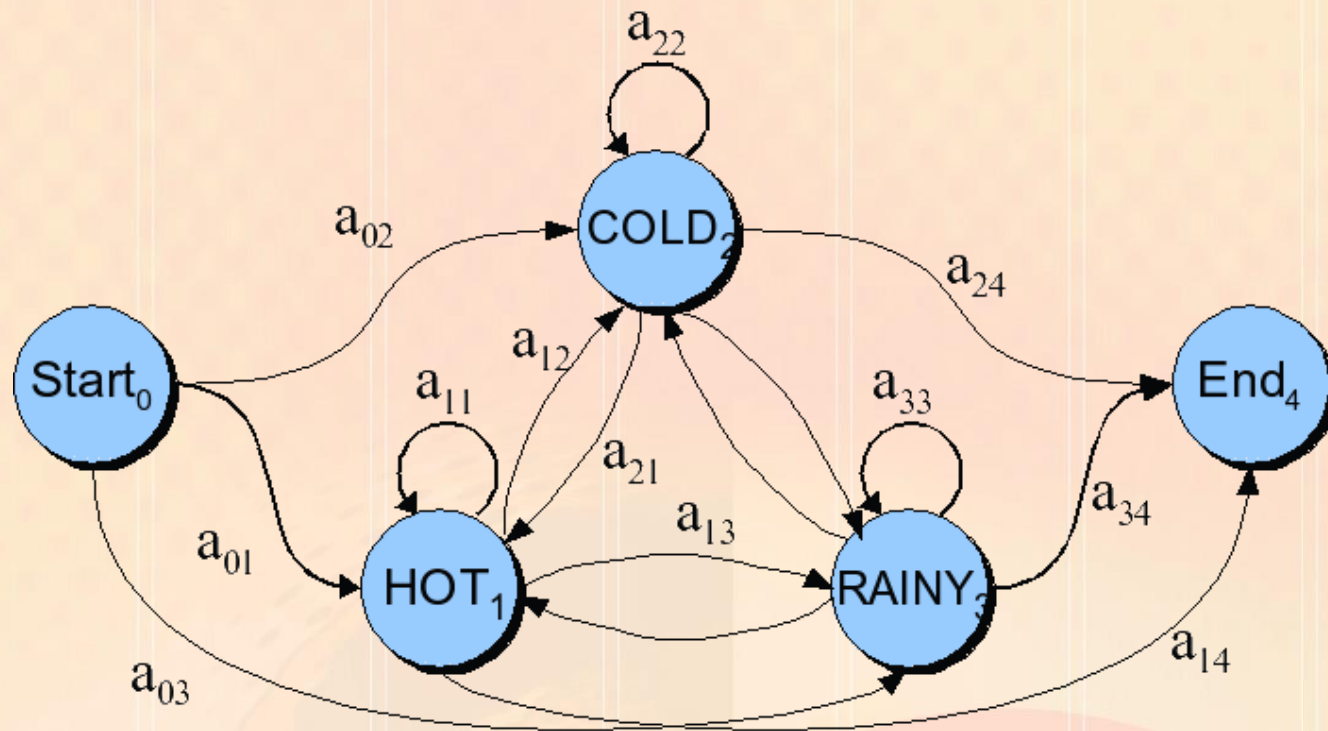
A Markov Chain
= special WFSA

The input sequence
Determines which
States the WFSA will
Go through.

Markov Chains

- *A Markov chain is a model that tells us something about the probabilities of sequences of random variables, (which represent the **states**), each of which can take on values from some value set.*
- *The value sets can be words, or tags, or symbols representing anything. For example **the weather**.*
- *A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is **the current state**. All the states before the current state have no impact on the future except via the current state. It's as if to predict tomorrow's weather you could examine today's weather but you weren't allowed to look at yesterday's weather.*

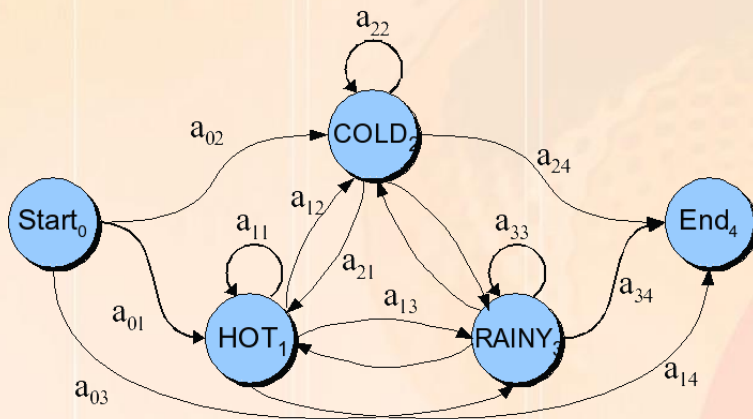
Markov Chain for Weather



A start distribution π is required; setting $\pi = [0.1, 0.7, 0.2]$

Markov Chain for Weather

- *What is the probability of 4 consecutive rainy days?*
- Sequence is *rainy-rainy-rainy-rainy*
- I.e., state sequence is 3-3-3-3
- $P(3,3,3,3) =$
 - $\pi_3 a_{33} a_{33} a_{33} a_{33} = 0.2 \times (0.6)^3 = 0.0432$



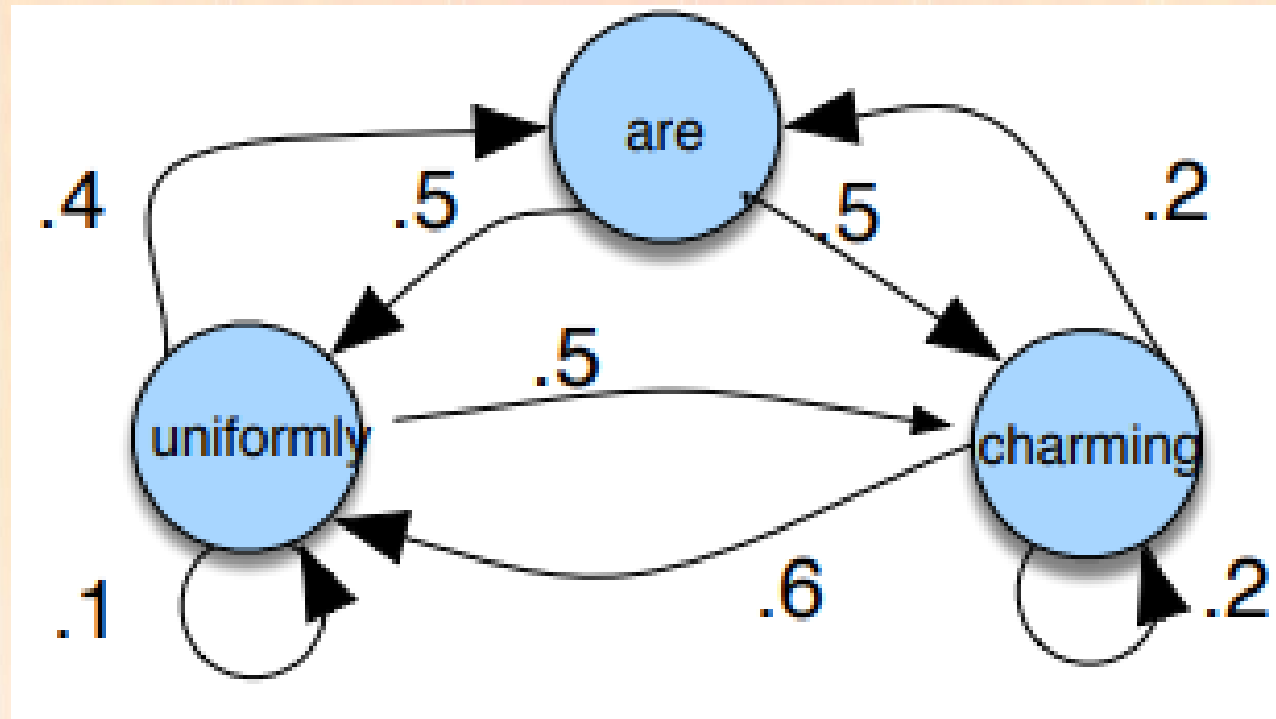
State 1	State 2	State 3
0.5	0.3	0.2



	State 1	State 2	State 3	State 4
State 0	0.5	0.3	0.2	
State 1	0.6	0.1	0.2	0.1
State 2	0.2	0.4	0.3	0.1
State 3	0.1	0.2	0.6	0.1



Markov Chain for Words



A bigram language model !!!



Markov Chain: “First-order observable Markov Model”

- Consider a sequence of state variables $q_1, q_2 \dots q_N$;
- A **Markov model** embodies the Markov assumption on the probabilities of this sequence: that Markov assumption when predicting the future, the past doesn't matter, only the present.
- Current state only depends on previous state

$$P(q_i \mid q_1 \dots q_{i-1}) = P(q_i \mid q_{i-1}) \quad \leftarrow \text{Markov Assumption}$$

Markov Chain Definition

- *A Markov chain is specified by :*

$$Q = q_1 q_2 \dots q_N$$

a set of N states

$$A = a_{11} a_{12} \dots a_{n1} \dots a_{nn}$$

a **transition probability matrix** A , each a_{ij} representing the probability of moving from state i to state j , s.t.
 $\sum_{j=1}^n a_{ij} = 1 \quad \forall i$

$$\pi = \pi_1, \pi_2, \dots, \pi_N$$

an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i . Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^n \pi_i = 1$

Back to the **Hidden Markov Model**

- *A Markov chain 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.*
 - *For example 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!*
- ❑ *A **hidden Markov model (HMM)** allows us to talk about both observed events (like words that we see in the input) and hidden events (like part-of-speech tags) that we think of as causal factors in a probabilistic model.*

Definition of HMM

□ *It has five components:*

$Q = q_1 q_2 \dots q_N$	a set of N states
$A = a_{11} \dots a_{ij} \dots a_{NN}$	a transition probability matrix A , each a_{ij} representing the probability of moving from state i to state j , s.t. $\sum_{j=1}^N a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \dots o_T$	a sequence of T observations , each one drawn from a vocabulary $V = v_1, v_2, \dots, v_V$
$B = b_i(o_t)$	a sequence of observation likelihoods , also called emission probabilities , each expressing the probability of an observation o_t being generated from a state i
$\pi = \pi_1, \pi_2, \dots, \pi_N$	an initial probability distribution over states. π_i is the probability that the Markov chain will start in state i . Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^n \pi_i = 1$

The components of an HMM tagger-1

❑ An HMM has two components, the **A** and **B** probabilities

➤ The **A matrix** contains the tag transition probabilities:

$P(t_i | t_{i-1})$ which represent the probability of a tag occurring given the previous tag.

For example, modal verbs like “will” are very likely to be followed by a verb in the base form, a **VB**, like “race”, so we expect this probability to be high. We compute the maximum likelihood estimate of this transition probability by counting, out of the times we see the first tag in a labeled corpus, how often the first tag is followed by the second:

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

Examples

- In the WSJ corpus, for example, the MD tag occurs 13124 times of which it is followed by the VB tag 10471, giving for an MLE estimate of:

$$P(VB|MD) = \frac{C(MD, VB)}{C(MD)} = \frac{10471}{13124} = .80$$

The components of an HMM tagger-2

- ❑ The **B emission probabilities**, $P(w_i | t_i)$, represent the probability, given a tag (say **MD**), that it will be associated with a given word (say **will**).
- The MLE of the emission probability is:

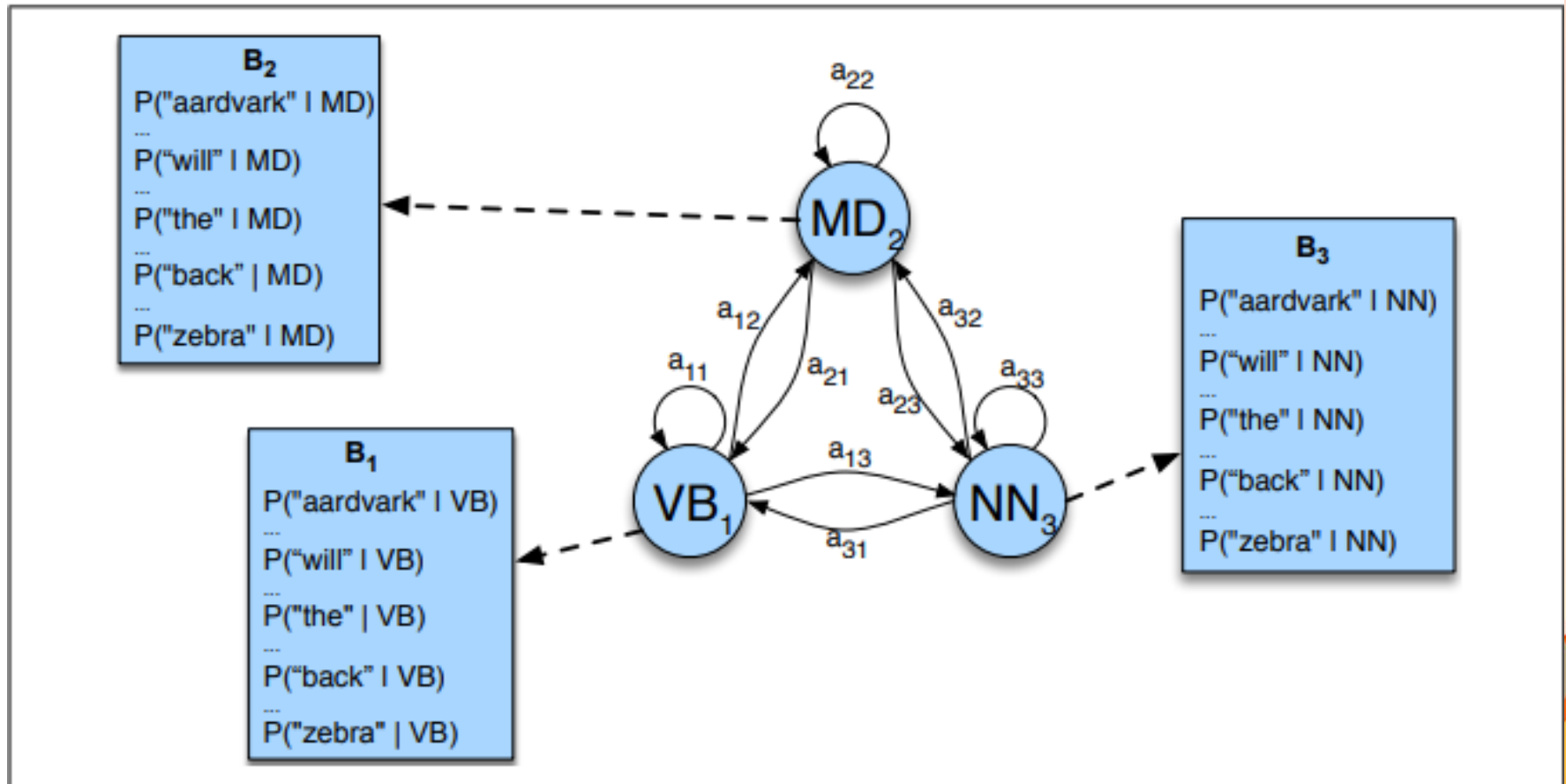
$$P(w_i | t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

- Example: Of the 13124 occurrences of **MD** in the WSJ corpus, it is associated with **will** 4046 times:

$$P(\text{will} | \text{MD}) = \frac{C(\text{MD}, \text{will})}{C(\text{MD})} = \frac{4046}{13124} = .31$$

HMM for POS Tagging

- A different view of HMMs, focused on the **A** transition probabilities used to compute the prior probability and the word likelihoods **B**.



HMM tagging as decoding

- **Decoding**: Given as input an HMM $\lambda = (\mathbf{A}, \mathbf{B})$ and a sequence of observations $O = o_1, o_2, \dots, o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \dots q_T$.
- For part-of-speech tagging, the goal of HMM decoding is to choose **the tag sequence** t_1^n that is most probable given the observation sequence of n words w_1^n :

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

Using Bayes Rule

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

&

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

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)}$$

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(w_1^n | t_1^n) P(t_1^n)$$

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} \overbrace{P(w_1^n | t_1^n)}^{\text{likelihood}} \overbrace{P(t_1^n)}^{\text{prior}}$$

Likelihood and Prior

- Assumption 1: the probability of a word appearing depends only on its own tag and is independent of neighboring words and tags:

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

- the **bigram** assumption: the probability of a tag is dependent only on the previous tag, rather than the entire tag sequence:

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

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n) \approx \operatorname{argmax}_{t_1^n} \prod_{i=1}^n \overbrace{P(w_i | t_i)}^{\text{emission transition}} \overbrace{P(t_i | t_{i-1})}$$

The Viterbi Algorithm for HMMs 1/6

- The most common decoding algorithm for HMMs is the **Viterbi algorithm**.

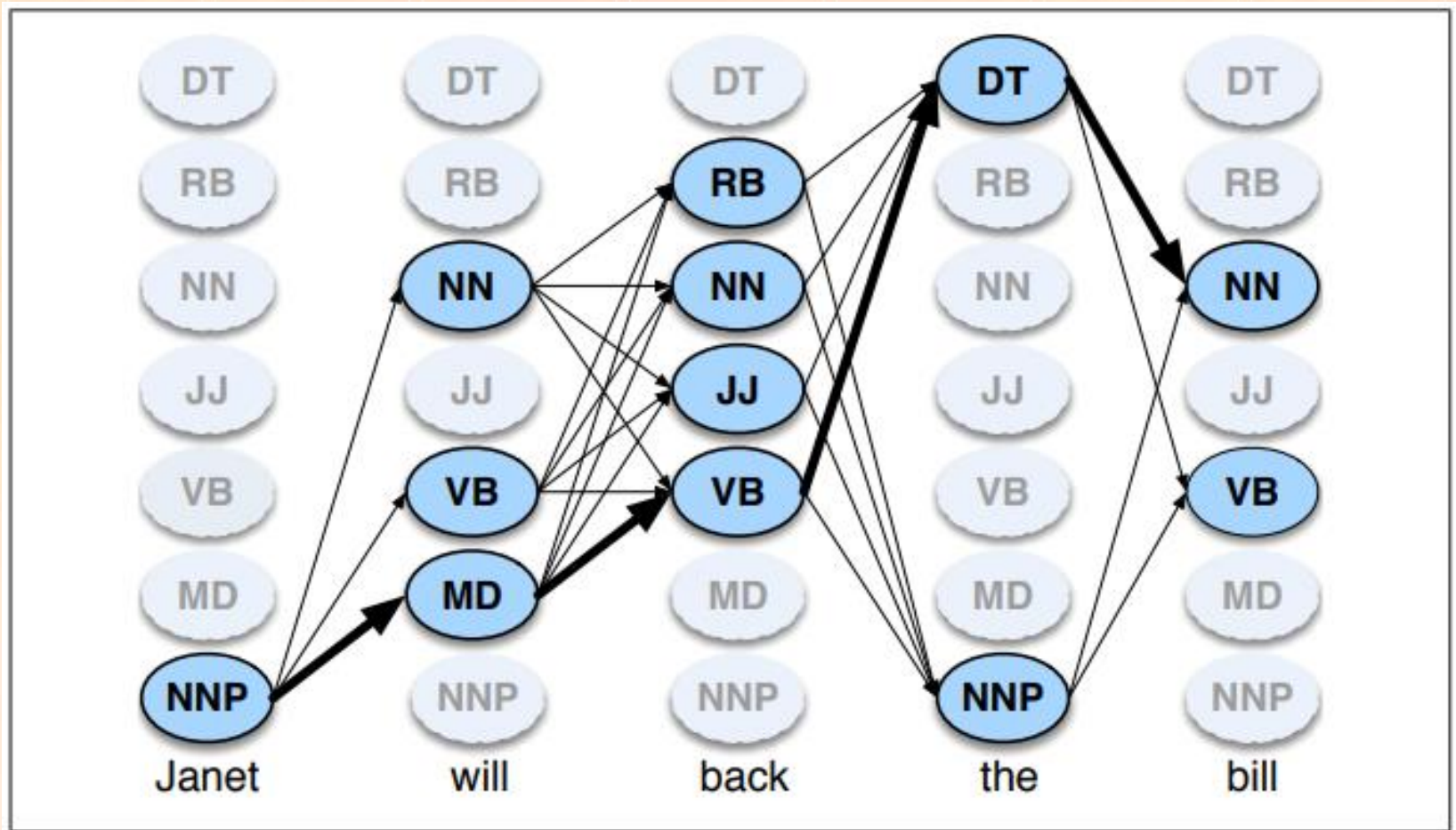
□ THE VITERBI ALGORITHM

- **INPUT:** (1) a single HMM and (2) a set of observed words $\mathbf{o}=(o_1, o_2, \dots o_t)$
- **OUTPUT:** (1) the most probable state/tag sequence $\mathbf{q}=(q_1, q_2 \dots q_t)$ together with (2) its probability.

*The Viterbi algorithm first sets up **a probability matrix** or **lattice**, with one column for each observation (word) and one row for each state (possible POS tag) in the state graph.*

Example of lattice

- the lattice for **Janet will back the bill**, showing the possible tags (q_i) for each word :



The 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 $bestpath$, $bestpathprob$



How does it work???

- Each cell of the trellis, $v_t(j)$, represents the probability that the HMM is in state j after seeing the first t observations and passing through the most probable state sequence q_1, \dots, q_{t-1} , given the HMM λ .
- We compute $v_t(j)$ by recursively taking the most probable path that could lead us to this cell:

$$v_t(j) = \max_{q_1, \dots, q_{t-1}} P(q_1 \dots q_{t-1}, o_1, o_2 \dots o_t, q_t = j | \lambda)$$

- Given that we had already computed the probability of being in every state at time $t - 1$, we compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell:

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t)$$

Important

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t)$$

- *The three factors that are multiplied:*

$v_{t-1}(i)$	the previous Viterbi path probability from the previous time step
a_{ij}	the transition probability from previous state q_i to current state q_j
$b_j(o_t)$	the state observation likelihood of the observation symbol o_t given the current state j

Walking through an example

- Let us tag:

Janet will back the bill

To define the HMM, we need the values of the matrix **A**:

	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

π

Transition probabilities

As well as the values of matrix **B**:

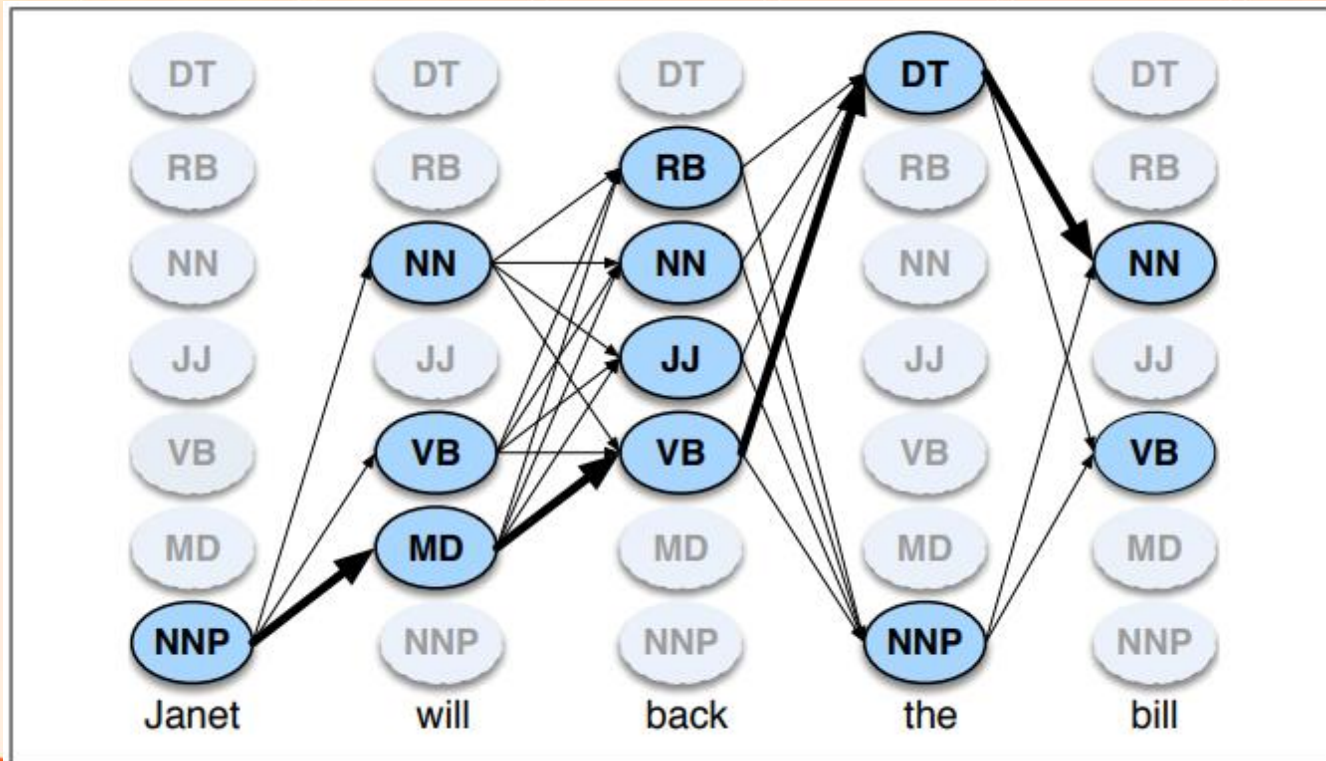
	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

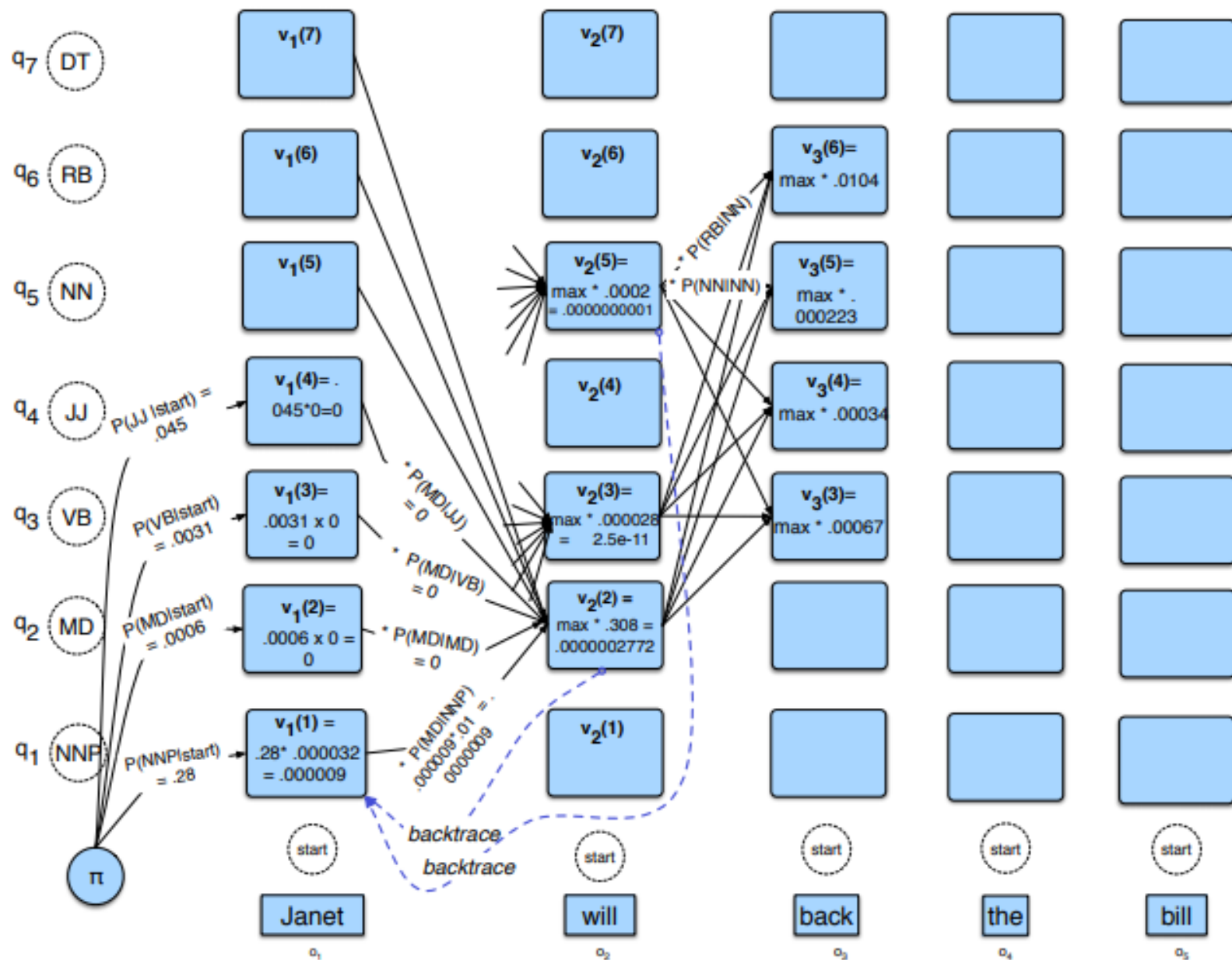
Observation likelihoods

Example of lattice

- $v_1(NNP) = 0.2767 \times 0.000032$ *INITIALIZATION STEP*
- For $t=2$: we compute $v_2(MD)$, $v_2(VB)$, and $v_2(NN)$ and chose the maximum

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t)$$





2nd example for the Viterbi Algorithm

□ How it works? It builds a probability matrix:

5	end						
4	NN		$0.41 \times 1.0 \times 0 = 0$	$.0045 \times 0.25 \times 0.000054$	$.0012 \times .00051 \times 0 = 0$		
3	TO		$0.042 \times 1.0 \times 0 = 0$	$.00079 \times .25 \times 0 = 0$	$.83 \times .000051 \times .99$	exercise	exercise
2	VB		$0.019 \times 1.0 \times 0 = 0$	$0.23 \times 0.25 \times 0.093 = .000051$	$.0038 \times .00051 \times 0 = 0$	exercise	exercise
1	PPSS		$0.067 \times 1.0 \times 0.37 = 0.25$	$.00014 \times .25 \times 0 = 0$	$.007 \times .00051 \times 0 = 0$		
0	start	1.0					
		#	I	want	to	race	#
		0	1	2	3	4	5

The Viterbi value=

The product of:

1. The transition probability $A[I,j]$
2. Previous path probability $Viterbi[s',t-1]$
3. Observation likelihood $B_s[o_t]$

One row for each state

One column for each observation

Tag Transition Probabilities (Brown corpus)

	VB	TO	NN	PPSS
<s>	.019	.0043	.041	.067
VB	.0038	.035	.047	.0070
TO	.83	0	.00047	0
NN	.0040	.016	.087	.0045
PPSS	.23	.00079	.0012	.00014

Observations likelihoods (Brown corpus)

	I	want	to	race
VB	0	.0093	0	.00012
TO	0	0	.99	0
NN	0	.000054	0	.00057
PPSS	.37	0	0	0

Viterbi Summary

- *Create a matrix*
 - *With columns corresponding to inputs*
 - *Rows corresponding to possible states*
- *Sweep through the array in one pass filling the columns left to right using our transition probabilities and observations probabilities*
- *Dynamic programming key is that we need only store the MAX probability path to each cell, (not all paths).*



Extending HMMs to trigrams 1/2

- Previous assumption: the tag appearing is dependent only on the previous tag:

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

- Most modern HMM taggers use a little more history – the previous 2 tags

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

- *o let the tagger know the location of the end of the sentence by adding dependence on an end-of-sequence marker for t_{n+1} . POS tagging is done by:*

$$\hat{t}_1^n = \arg \max_{t_1^n} P(t_1^n | w_1^n) \approx \arg \max_{t_1^n} \left[\prod_{i=1}^n P(w_i | t_i) P(t_i | t_{i-1}, t_{i-2}) \right] P(t_{n+1} | t_n)$$

One problem: data sparsity

Deleted interpolation 1/2

- Compute the trigram probability with MLE from counts:

$$P(t_i | t_{i-1}, t_{i-2}) = \frac{C(t_{i-2}, t_{i-1}, t_i)}{C(t_{i-2}, t_{i-1})}.$$

Many of these counts will be 0!

- *Solution: estimate the probability by combining more robust but weaker estimators. E.g if we never seen PRP VB TO, we cannot compute $P(\text{TO}|\text{PRP}, \text{VB})$ but we can compute $P(\text{TO}|\text{VB})$ or even $P(\text{TO})$.*

Trigrams $\hat{P}(t_i | t_{i-1}, t_{i-2}) = \frac{C(t_{i-2}, t_{i-1}, t_i)}{C(t_{i-2}, t_{i-1})}$

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

Unigrams $\hat{P}(t_i) = \frac{C(t_i)}{N}$

Maximum Likelihood Estimation

How should they be combined???

$$P(t_i | t_{i-1} t_{i-2}) = \lambda_1 \hat{P}(t_i | t_{i-1} t_{i-2}) + \lambda_2 \hat{P}(t_i | t_{i-1}) + \lambda_3 \hat{P}(t_i)$$

Deleted Interpolation 2/2

function DELETED-INTERPOLATION(*corpus*) **returns** $\lambda_1, \lambda_2, \lambda_3$

$\lambda_1 \leftarrow 0$

$\lambda_2 \leftarrow 0$

$\lambda_3 \leftarrow 0$

foreach trigram t_1, t_2, t_3 with $f(t_1, t_2, t_3) > 0$

depending on the maximum of the following three values

case $\frac{C(t_1, t_2, t_3) - 1}{C(t_1, t_2) - 1}$: increment λ_3 by $C(t_1, t_2, t_3)$

case $\frac{C(t_2, t_3) - 1}{C(t_2) - 1}$: increment λ_2 by $C(t_1, t_2, t_3)$

case $\frac{C(t_3) - 1}{N - 1}$: increment λ_1 by $C(t_1, t_2, t_3)$

end

end

normalize $\lambda_1, \lambda_2, \lambda_3$

return $\lambda_1, \lambda_2, \lambda_3$

Beam Search

- *When the number of states grows very large, the vanilla Viterbi algorithm will be slow! The complexity of the algorithm is $O(N^2T)$; N (the number of states) can be large for trigram taggers, which have to consider every previous pair of the 45 tags, resulting in $45^3 = 91,125$ computations per column. N can be even larger for other applications of Viterbi, for example to decoding in neural networks.*

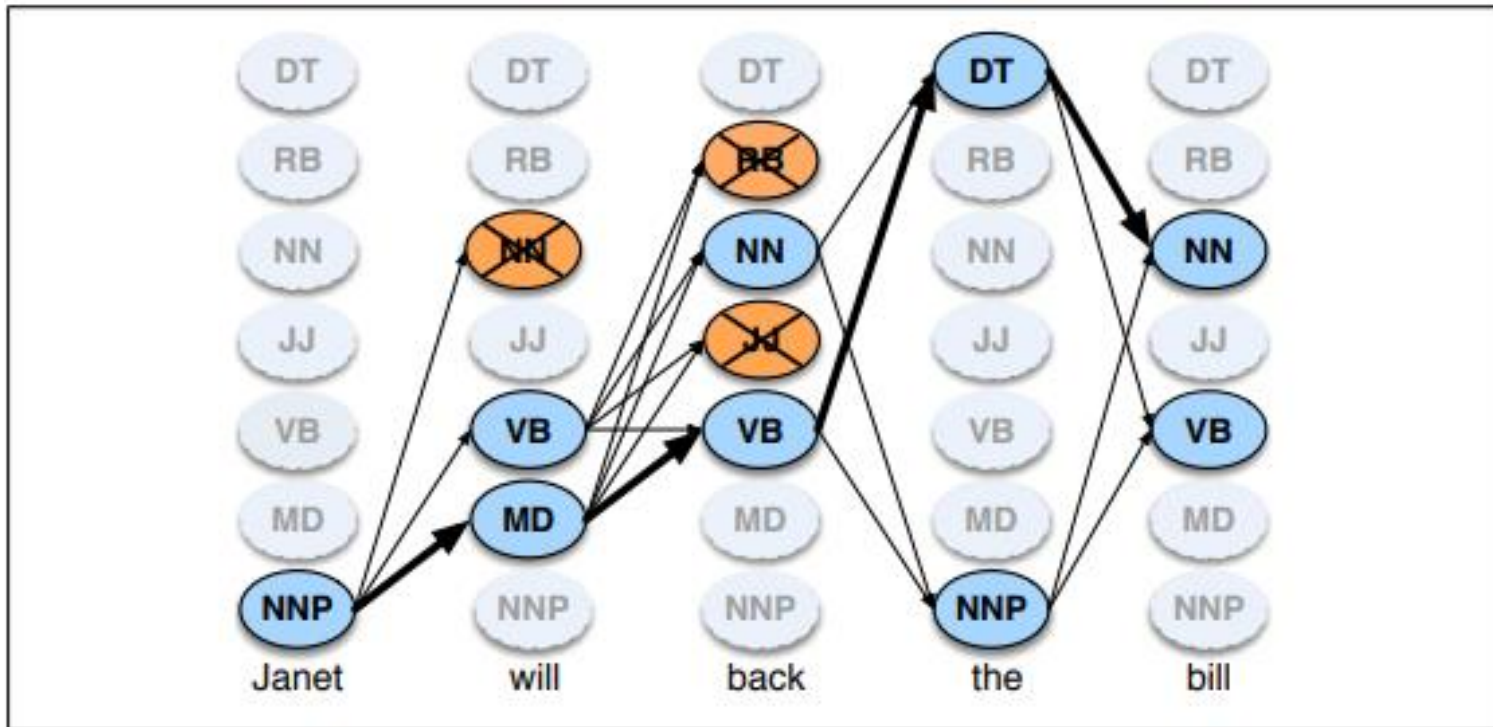
➤ Solution: **Beam Search**

*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.*

Beam Width

- One way to implement beam search is to keep a fixed number of states instead of all N current states. Here the beam width β is a fixed number of states. Alternatively β can be modeled as a fixed percentage of the N states, or as a probability threshold.

Example: $\beta=2$



At each time t , all (non-zero) states are computed, but then they are sorted and only the best 2 states are propagated forward and the rest are pruned, shown in orange.

Maximum Entropy Markov Models

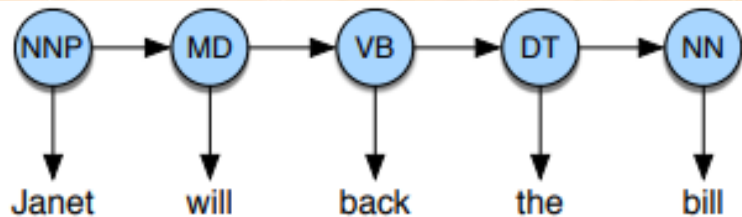
- ❑ Good News: an HMM can achieve very high accuracy!
- ❑ Bad News: it requires a number of architectural innovations to deal with: unknown words, backoff, suffixes, and so on.
- ❑ What we would like: easy way to add arbitrary features directly into the model, but that's hard for generative models like HMMs!
- Logistic regression could be a solution, **but**:
 1. it isn't a sequence model;
 2. it assigns a class to a single observation.
- **What can we do?????** 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 the **Maximum Entropy Markov Model** or **MEMM**

Difference HMM - MEMM

Let the sequence of words be $W = w_1^n$ and the sequence of tags $T = t_1^n$

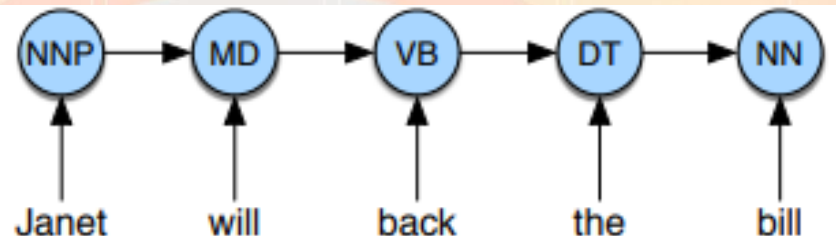
In an HMM to compute the best tag sequence that maximizes $P(T|W)$ we rely on Bayes' rule and the likelihood $P(W|T)$:

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T P(W|T)P(T) \\ &= \operatorname{argmax}_T \prod_i P(\text{word}_i | \text{tag}_i) \prod_i P(\text{tag}_i | \text{tag}_{i-1})\end{aligned}$$



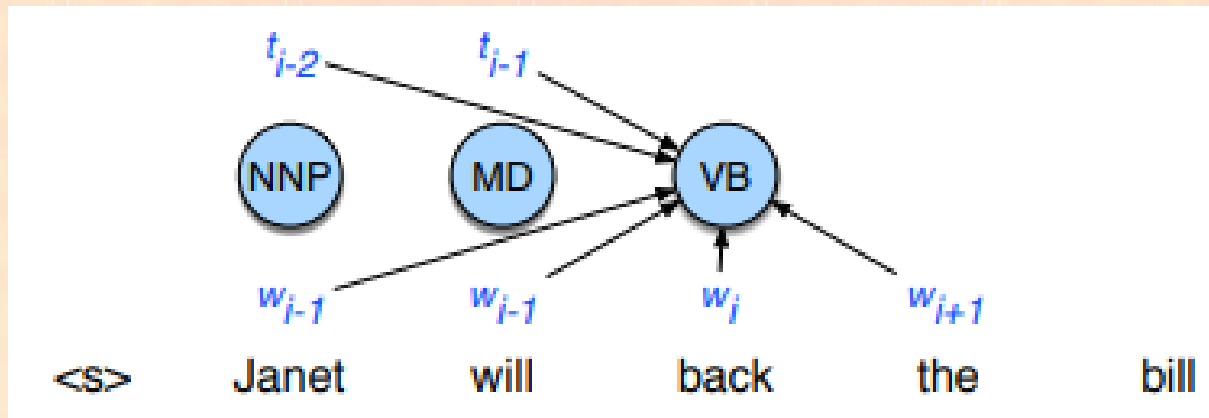
In an MEMM, we compute the posterior $P(T|W)$ directly, training it to discriminate among the possible tag sequences:

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



Features of an MEMM

Note: we don't build MEMMs that condition just on w_i and t_{i-1} . The reason to use a discriminative sequence model is that it's easier to incorporate a lots of features!



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

Abstract
Representation
Of features

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

Feature Templates

➤ *feature templates are used to automatically populate the set of features from every instance in the training and test set.*

➤ *Our example:*

Janet/NNP will/MD back/VB the/DT bill/NN,

-when w_i = “back”, we generate the following features:

$t_i = \text{VB}$ 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 $w_{i+1} = \text{the}$

More features: Word Shape

- ***Word shape features** are used to represent the abstract letter pattern of the word*
- ❑ *How???*
 - *Map lower-case letters to 'x',*
 - *Map upper-case to 'X',*
 - *Map numbers to 'd',*
 - *retaining punctuation. T*
- *Examples:*
 - *I.M.F. would map to X.X.X.*
 - *DC10-30 would map to XXdd-dd.*
 - *UTD maps into XXX*

Second class of word features

- A second class of shorter word shape features is also used.
- ❑ How???? In these features consecutive character types are removed!
- Example: DC10-30 (already mapped into XXdd-dd) would be mapped to Xd-d
- but I.M.F would still map to X.X.X.
- the word **well-dressed** would generate the following non-zero valued feature values:

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

Decoding and Training MEMMs

OUR GOAL: compute the most likely sequence of POS tags, given a sequence of words!

- How??? The most likely sequence of tags is computed by combining the features of the input word w_i , its neighbors within l words: w_{i-l}^{i+l} , and the previous k tags t_{i-k}^{i-1} (using θ to refer to feature weights instead of w to avoid the confusion with w meaning words):

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T \prod_i P(t_i | w_{i-l}^{i+l}, t_{i-k}^{i-1}) \\ &= \operatorname{argmax}_T \prod_i \frac{\exp \left(\sum_j \theta_j f_j(t_i, w_{i-l}^{i+l}, t_{i-k}^{i-1}) \right)}{\sum_{t' \in \text{tagset}} \exp \left(\sum_j \theta_j f_j(t', w_{i-l}^{i+l}, t_{i-k}^{i-1}) \right)}\end{aligned}$$

features

$t_i = \text{VB}$ 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

$\text{prefix}(w_i) = w$
 $\text{prefix}(w_i) = \text{we}$
 $\text{prefix}(w_i) = \text{well}$
 $\text{prefix}(w_i) = \text{well}$
 $\text{suffix}(w_i) = \text{ssed}$
 $\text{suffix}(w_i) = \text{sed}$
 $\text{suffix}(w_i) = \text{ed}$
 $\text{suffix}(w_i) = \text{d}$
 $\text{has-hyphen}(w_i)$
 $\text{word-shape}(w_i) = \text{xxxx-xxxxxxx}$
 $\text{short-word-shape}(w_i) = \text{x-x}$

Option 1: Greedy Decoding

- Build a local **classifier** that decides the POS tag for each word left to right, making a hard classification of the first word in the sentence, then a hard decision on the second word, and so on. This is called a greedy decoding algorithm, because we greedily choose the best tag for each word!

Features, θ



```
function GREEDY SEQUENCE DECODING(words W, model P) returns tag sequence T
```

```
for  $i = 1$  to  $\text{length}(W)$ 
```

```
   $\hat{t}_i = \underset{t' \in T}{\operatorname{argmax}} P(t' \mid w_{i-l}^{i+l}, t_{i-k}^{i-1})$ 
```

- ❑ The problem with the greedy algorithm is that by making a hard decision on each word before moving on to the next word, the classifier can't use evidence from future decisions. Although the greedy algorithm is very fast, and occasionally has sufficient accuracy to be useful, in general the hard decision causes too much a drop in performance, and we don't use it.

Decode an MEMM with the Viterbi algorithm

- *What do we need to change????*

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*

More details

- *How????*

In the recursive step:

- *the Viterbi equation computes the Viterbi value of time t for state j as:*

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T$$

which is the HMM implementation of:

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) P(s_j|s_i) P(o_t|s_j) \quad 1 \leq j \leq N, 1 < t \leq T$$

- *The MEMM requires only a slight change to this latter formula, replacing the **A** and **B** prior and likelihood probabilities with the direct posterior:*

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) P(s_j|s_i, o_t) \quad 1 \leq j \leq N, 1 < t \leq T$$

Learning in MEMMs

- *Learning in MEMMs relies on the supervised learning algorithms.*
 - *E.g. logistic regression.*
 - *Given:*
 - *a sequence of observations, f*
 - *feature functions, and*
 - *corresponding hidden states*
 - *use gradient descent to train the weights to maximize the log-likelihood of the training corpus.*

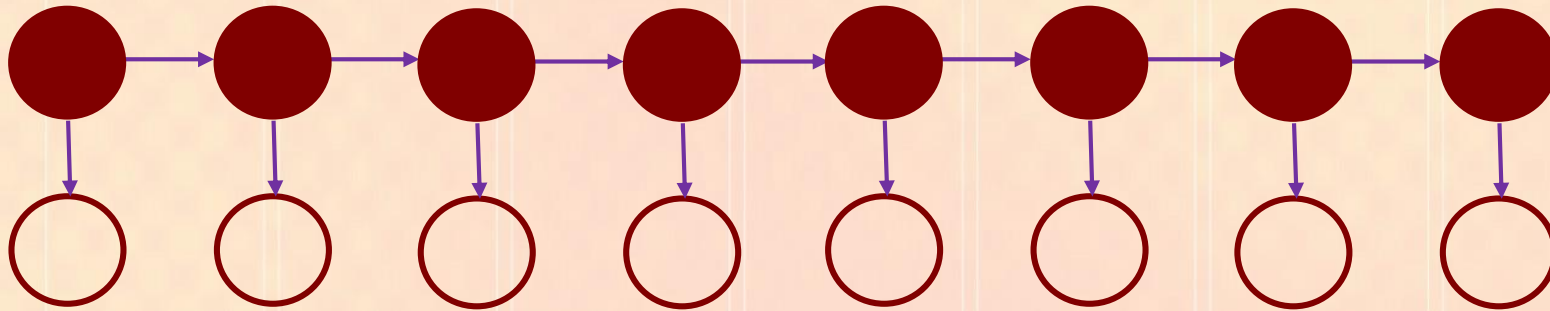
Bi-directionality

Problem: *the MEMM and HMM models exclusively run left-to-right.*

- *While the Viterbi algorithm still allows present decisions to be influenced indirectly by future decisions, it would help even more if a decision about word w_i could directly use information about future tags t_{i+1} and t_{i+2} .*
- ❑ *One way to implement bidirectionality is to switch to a more powerful model called a **conditional random field** or **CRF**.*
- ❑ *What are CRFs????*

From HHMs to MEMM to CRF -1

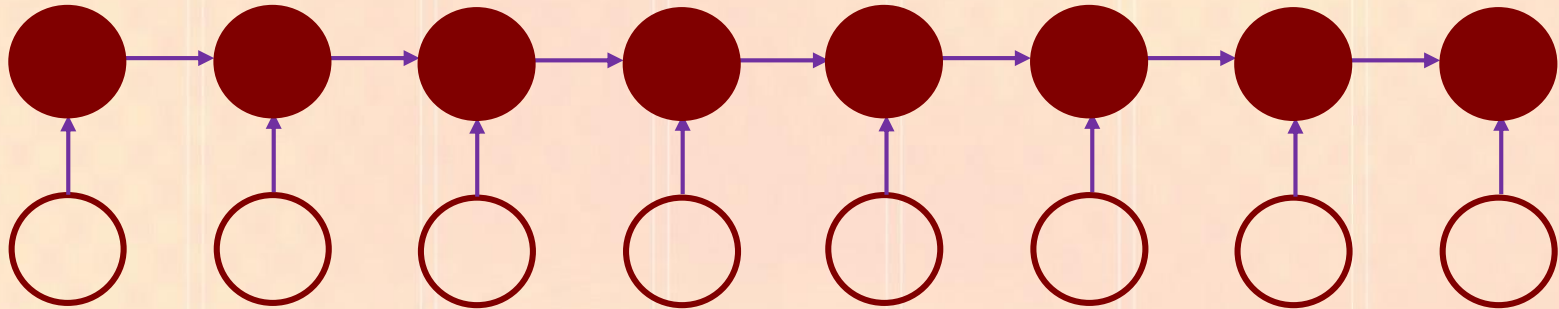
□ HMMS



- *Generative Models*
 - *Find parameters to maximize $P(X, Y)$*
- *Assumes features are independent !*
- *When labeling X_i future observations are taken into account (forward-backward)*

From HHMs to MEMM to CRF -2

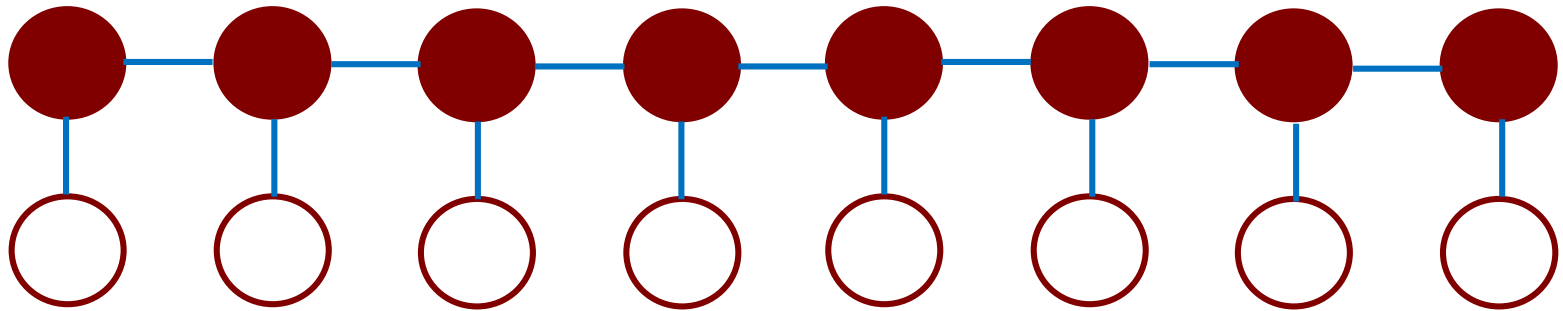
□ MEMM



- *Discriminative*
 - *Find parameters to maximize $P(Y|X)$*
- *No longer assume that features are independent !*
- *Do not take future observations into account*

From HHMs to MEMM to CRF -3

□ CRF



- *Discriminative*
 - *Doesn't assume that features are independent*
 - *When labeling Y_i future observations are taken into account*
- ➔ *The best of both worlds!*

TUTORIALS:

<https://homepages.inf.ed.ac.uk/csutton/publications/crftut-fnt.pdf>

<https://people.cs.umass.edu/~mccallum/papers/crf-tutorial.pdf>

CRFs

- for a word sequence $\mathbf{x} = (x_1, \dots, x_{|x|})$ and a POS sequence $\mathbf{y} = (y_1, \dots, y_{|x|})$ is defined as:

$$p(\mathbf{y} | \mathbf{x}; \mathbf{w}) \propto \prod_{i=n}^{|x|} \exp \left(\overset{\text{weights}}{\mathbf{w} \cdot \phi(y_{i-n}, \dots, y_i, x, i)} \right)$$

- where n denotes the model order, \mathbf{w} the model parameter vector, and ϕ the feature extraction function.
- at each time step the CRF computes log-linear functions over **a clique**, a set of relevant features.

More details: <http://www.aclweb.org/anthology/P14-2043>

- CRF implementations:

CRF++

<http://crfpp.sourceforge.net/>

MALLET

<http://mallet.cs.umass.edu/>

GRMM

<http://mallet.cs.umass.edu/grmm/>

CRFSuite

<http://www.chokkan.org/software/crfsuite/>

FACTORIE

<http://www.factorie.cc>

Invitation:

- *Go and play!!!*
- *Stanford POS tagger:*

<https://nlp.stanford.edu/static/software/tagger.shtml>

It has a version for twitter as well
English and multi-lingual

Summary

❑ *Introduced parts-of-speech and part-of-speech tagging:*

- *Two common approaches to sequence modeling are a generative approach, HMM tagging, and a discriminative approach, MEMM tagging.*
- *The probabilities in HMM taggers are estimated by maximum likelihood estimation on tag-labeled training corpora. The Viterbi algorithm is used for decoding, finding the most likely tag sequence.*
- *Beam search is a variant of Viterbi decoding that maintains only a fraction of high scoring states rather than all states during decoding.*
- *Maximum entropy Markov model or MEMM taggers train logistic regression models to pick the best tag given an observation word and its context and the previous tags, and then use Viterbi to choose the best sequence of tags.*
- *Modern taggers are generally run bidirectionally.*