CE4045 CZ4045 SC4002
Natural Language Processing

Parts of Speech and Named Entities

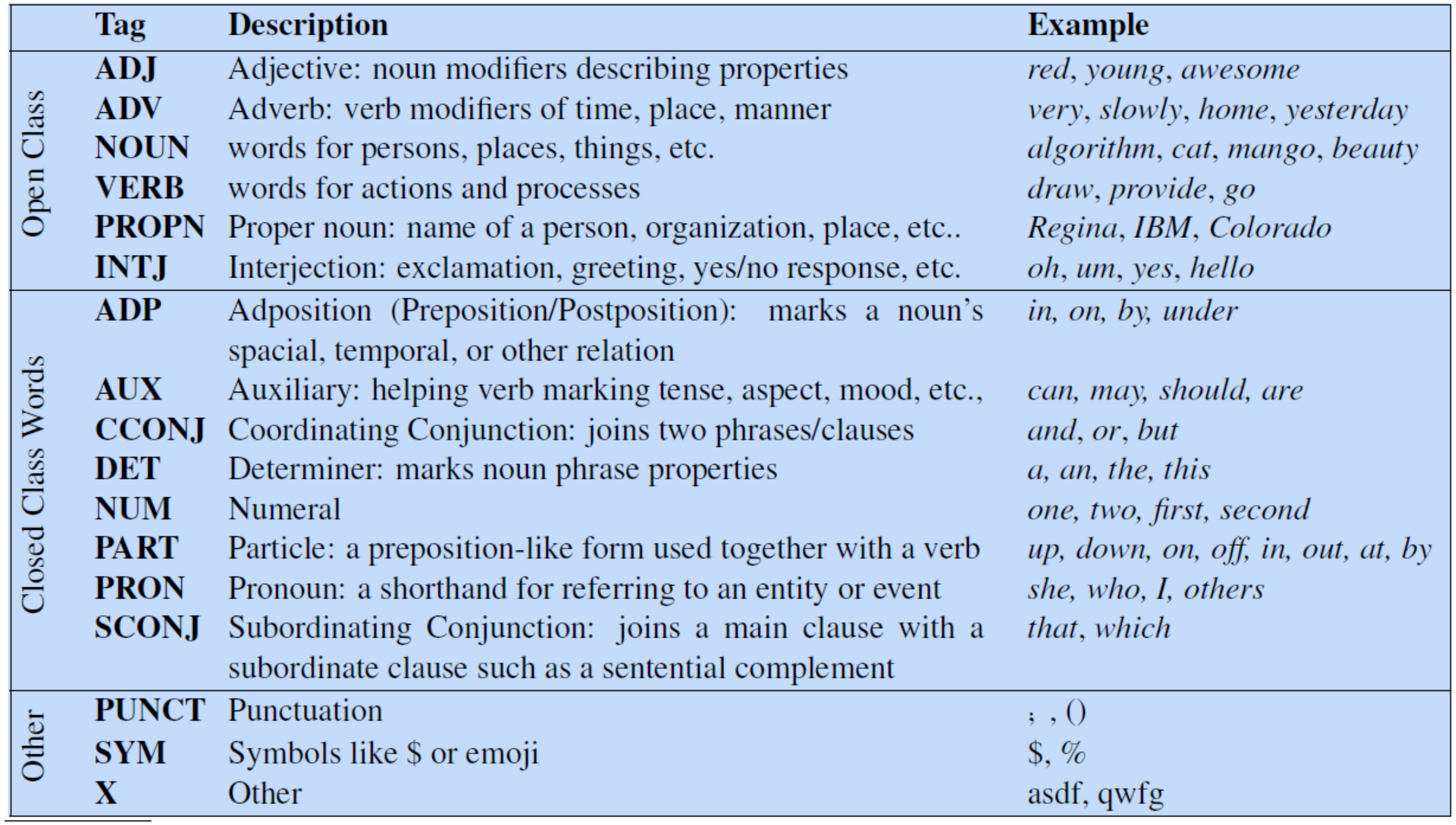
Dr. Sun Aixin

POS and NER → Sequence Labeling

- **Parts of speech** (POS) refers to **word classes** such as Noun, Verb, Adjective, etc.
  - Also known as lexical categories, word classes, morphological classes, lexical tags
  - Knowing word class tells us more about neighboring words and syntactic structure
    - E.g., nouns in English are often preceded by determiners and adjectives
    - Verbs have dependency links to nouns
  - POS tagging is a key aspect of parsing sentence structure.
- **Named entity** is proper name for person, location, organization, etc.
  - NEs are useful clues to sentence structure and meaning understanding.
  - Knowing if a named entity like *Washington* is a name of a person, a place, or a university is important to tasks like question answering and information extraction.
- Sequence labeling
  - **POS tagging**: takes a sequence of words and assigns each word a POS like NOUN or VERB
  - **Named Entity Recognition** (NER): assigns words or phrases tags like PERSON, LOCATION, or ORGANIZATION.

POS Tagging

- There are multiple POS tagsets defined (what POS tags can be assigned)
- Example: 17 parts of speech in the Universal Dependencies tagset



	Tag	Description	Example
Open Class	ADJ	Adjective: noun modifiers describing properties	*red, young, awesome*
	ADV	Adverb: verb modifiers of time, place, manner	*very, slowly, home, yesterday*
	NOUN	words for persons, places, things, etc.	*algorithm, cat, mango, beauty*
	VERB	words for actions and processes	*draw, provide, go*
	PROPN	Proper noun: name of a person, organization, place, etc..	*Regina, IBM, Colorado*
	INTJ	Interjection: exclamation, greeting, yes/no response, etc.	*oh, um, yes, hello*
Closed Class Words	ADP	Adposition (Preposition/Postposition): marks a noun's spacial, temporal, or other relation	*in, on, by, under*
	AUX	Auxiliary: helping verb marking tense, aspect, mood, etc.,	*can, may, should, are*
	CCONJ	Coordinating Conjunction: joins two phrases/clauses	*and, or, but*
	DET	Determiner: marks noun phrase properties	*a, an, the, this*
	NUM	Numeral	*one, two, first, second*
	PART	Particle: a preposition-like form used together with a verb	*up, down, on, off, in, out, at, by*
	PRON	Pronoun: a shorthand for referring to an entity or event	*she, who, I, others*
	SCONJ	Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement	*that, which*
Other	PUNCT	Punctuation	; , ()
	SYM	Symbols like $ or emoji	$, %
	X	Other	asdf, qwfg

More about word classes

- **Closed classes** are those with relatively **fixed membership**, such as prepositions; new prepositions are rarely coined
  - Closed class words are generally **function words** (e.g., *of, it, and)* which tend to be very short, occur frequently, and often have structuring uses in grammar

- Nouns and verbs are among **open classes**; new nouns and verbs like *iPhone* or *fax* are continually being created or borrowed
  - **Nouns** are words for people, places, or things, but include others as well. Common nouns include concrete terms like *cat* and *mango*, abstractions like *algorithm* and *beauty*, and verb-like terms like *pacing*.
    - **Proper nouns**, are names of specific persons or entities
  - **Verbs** refer to actions and processes, including main verbs like draw and provide.
    - English verbs have **inflections**: non-third-person-singular (eat), third-person singular (eats), progressive (eating), past participle (eaten).
  - **Adjectives** often describe properties or qualities of nouns.
  - **Adverbs** generally modify something (often verbs, hence the name "adverb").

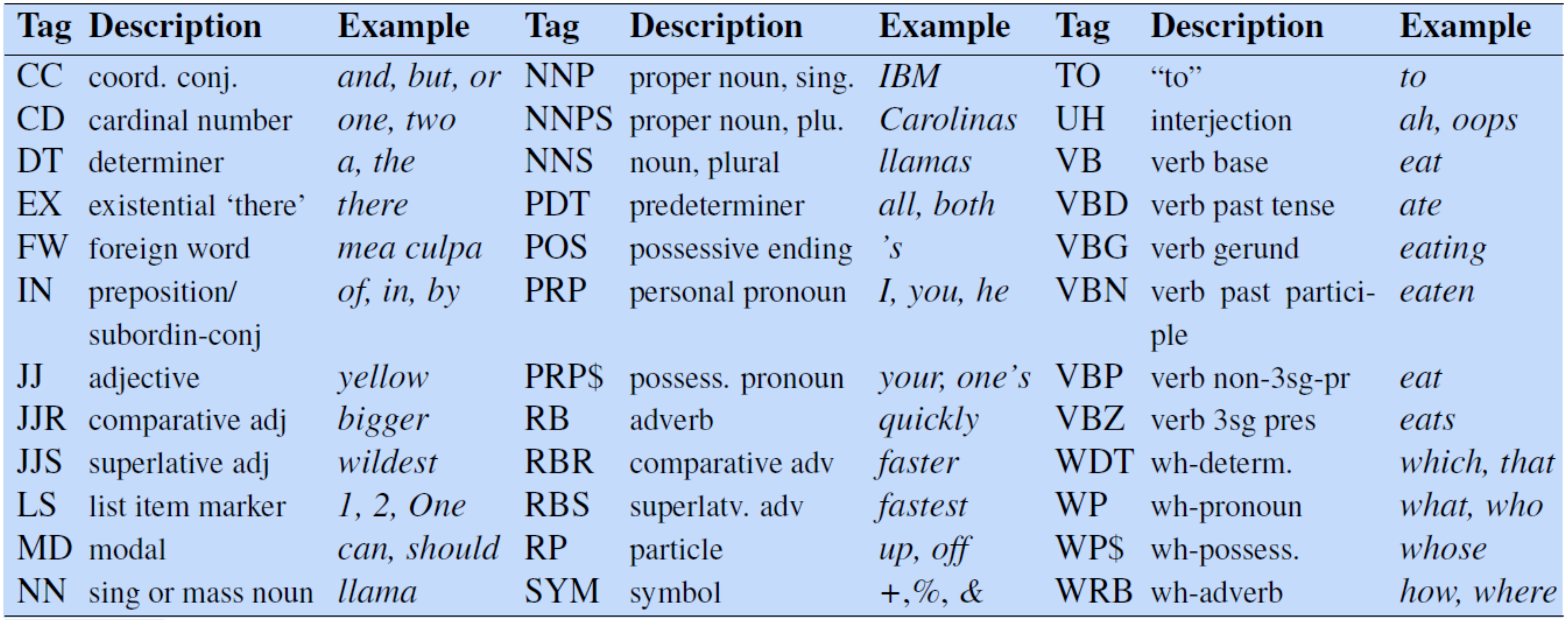
More about word classes

- **Prepositions** indicate spatial or temporal relations, and relations.
  - E.g., *on* it, *before* then, *by* the house; *on* time, *beside* herself; assignment *by* me.

- **Determiners** like *this* and *that* (this chapter, that page) can mark the start of an article English noun phrase.
  - **Articles** like a, an, and the, are a type of determiner that mark discourse properties of the noun and are quite frequent.

- **Pronouns** act as a shorthand for referring to an entity or event.
  - ***Personal pronouns*** refer to persons or entities (you, 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 act as complementizers (Frida, *who* married Diego. . .).

The 45-tag Penn Treebank tagset: another tagset example



Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	"to"	*to*
CD	cardinal number	*one, two*	NNPS	proper noun, plu.	*Carolinas*	UH	interjection	*ah, oops*
DT	determiner	*a, the*	NNS	noun, plural	*llamas*	VB	verb base	*eat*
EX	existential 'there'	*there*	PDT	predeterminer	*all, both*	VBD	verb past tense	*ate*
FW	foreign word	*mea culpa*	POS	possessive ending	*'s*	VBG	verb gerund	*eating*
IN	preposition/ subordin-conj	*of, in, by*	PRP	personal pronoun	*I, you, he*	VBN	verb past partici- ple	*eaten*
JJ	adjective	*yellow*	PRP$	possess. pronoun	*your, one's*	VBP	verb non-3sg-pr	*eat*
JJR	comparative adj	*bigger*	RB	adverb	*quickly*	VBZ	verb 3sg pres	*eats*
JJS	superlative adj	*wildest*	RBR	comparative adv	*faster*	WDT	wh-determ.	*which, that*
LS	list item marker	*1, 2, One*	RBS	superlatv. adv	*fastest*	WP	wh-pronoun	*what, who*
MD	modal	*can, should*	RP	particle	*up, off*	WP$	wh-possess.	*whose*
NN	sing or mass noun	*llama*	SYM	symbol	*+,%, &*	WRB	wh-adverb	*how, where*

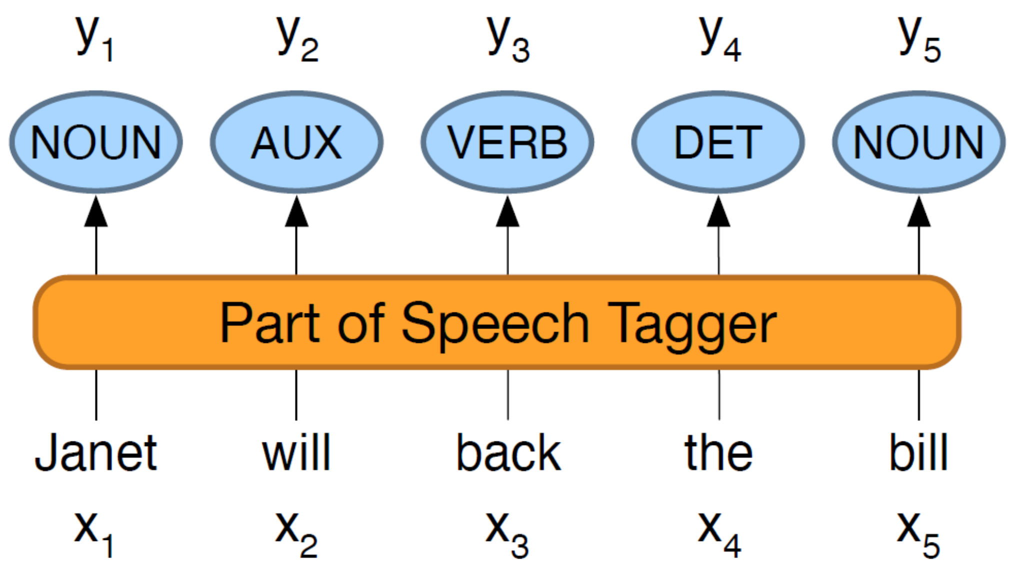
There/PRO/EX are/VERB/VBP 70/NUM/CD children/NOUN/NNS there/ADV/RB ./PUNC/.

Preliminary/ADJ/JJ findings/NOUN/NNS were/AUX/VBD reported/VERB/VBN in/ADP/IN today/NOUN/NN 's/PART/POS New/PROPN/NNP England/PROPN/NNP Journal/PROPN/NNP of/ADP/IN Medicine/PROPN/NNP

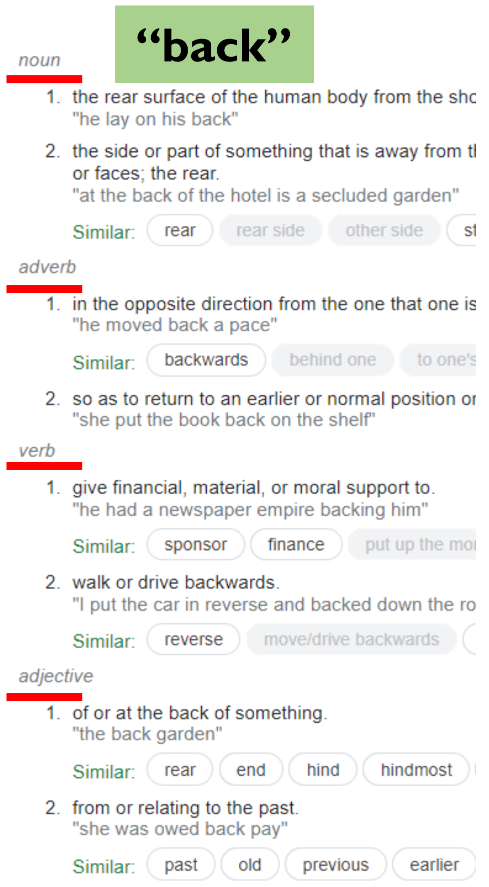
Example tagsets:
Penn Treebank tagset
Universal Dependencies tagset

Part-of-Speech Tagging

➢ Part-of-speech tagging is the process of assigning a part-of-speech to each word in a text.



- x_1 to x_5 are words (or tokens) in a sentence (or sequence)
- y_1 to y_5 are POS tags from a predefined tagset.



"back"

noun

1. the rear surface of the human body from the shc
   "he lay on his back"
2. the side or part of something that is away from tl
   or faces; the rear.
   "at the back of the hotel is a secluded garden"

   Similar: rear | rear side | other side | st

adverb

1. in the opposite direction from the one that one is
   "he moved back a pace"

   Similar: backwards | behind one | to one's
2. so as to return to an earlier or normal position or
   "she put the book back on the shelf"

verb

1. give financial, material, or moral support to.
   "he had a newspaper empire backing him"

   Similar: sponsor | finance | put up the moi
2. walk or drive backwards.
   "I put the car in reverse and backed down the ro

   Similar: reverse | move/drive backwards

adjective

1. of or at the back of something.
   "the back garden"

   Similar: rear | end | hind | hindmost
2. from or relating to the past.
   "she was owed back pay"

   Similar: past | old | previous | earlier

Why POS tagging is challenging

- Words are **ambiguous**—have more than one possible part-of-speech
  - Tagging is a disambiguation task; the goal is to find the correct tag for ***the situation***.
- Example words with multiple parts-of-speech
  - **Book** that flight; Hand me that **book**.
  - The ***back*** door; On my ***back***; Win the voters ***back***; Promised to ***back*** the bill
- Tag ambiguity in the Brown and WSJ corpora (Treebank 45-tag tagset).

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%**)

- The accuracy of POS tagging algorithms is extremely high, about 97%.
  - But: The most-frequent-tag baseline has an accuracy of about 92%

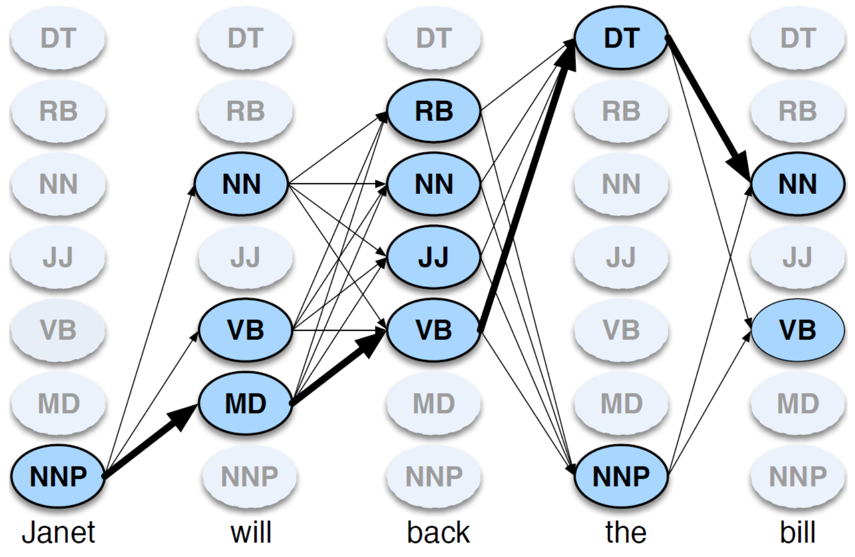
POS Tagging with Hidden Markov Model (HMM)

- A sequence labeler assigns a label to each unit (e.g., word) in a sequence (e.g., sentence), thus mapping a sequence of observations to a sequence of labels of the same length.
- The HMM is a classic probabilistic sequence model
  - Given a sequence of words, it computes a probability distribution **over possible sequences of labels** and chooses the **best label sequence**.
- POS Tagging in probabilistic view:
  - Consider all possible sequences of tags (each tag for one word)
  - Out of this universe of sequences, choose the tag sequence which is most probable given the observation sequence of n words $w_1 \dots w_n$.

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

POS Tagging with Hidden Markov Model (HMM)

- POS Tagging in probabilistic view:
  - Consider **all possible sequences of tags** (each tag for one word)
  - Out of this universe of sequences, choose the tag sequence which is **most probable** given the observation sequence of n words $w_1 \dots w_n$.

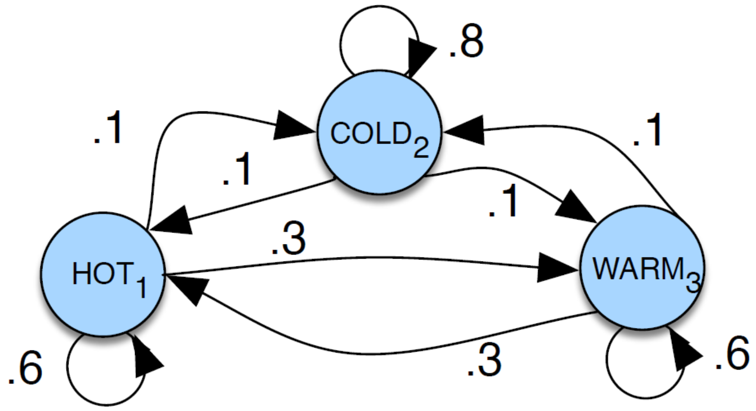


- Janet has **one** possible tag
- Will has **three** possible tags
- Back has **four** possible tags
- The has **two** possible tags
- Bill has **two** possible tags

There are **48** possible tag sequences

Markov Chains

- The HMM is based on augmenting the **Markov chain**.
- A Markov chain is a model on the probabilities of sequences of random variables (or states), each of which can take on values from some set.
  - For example: a set of possible weather states includes HOT, COLD, WARM
  - These sets can be tags, words, or symbols representing anything,
- Markov assumption: The probability of next state only depends on the current state, e.g., predict tomorrow's weather only based on todays' weather
  - $P(q_i = a|q_1 \dots q_{i-1}) = P(q_i = a|q_{i-1})$
  - $q_1 \dots q_{i-1}$ is a sequence of states
  - The probability of next state $q_i = a$ only depends on the state q_{i-1}.
  - Value a is from the set of possible states, or vocabulary e.g., {HOT, COLD, WARM}

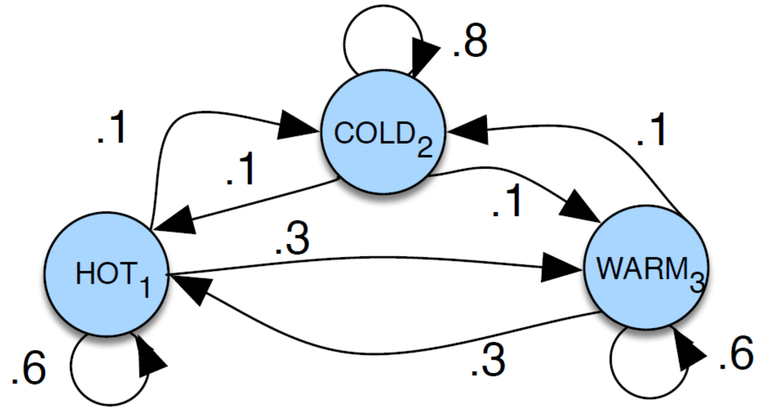


Markov Chains

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} a_{12} \ldots a_{N1} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	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, $\sum_{i=1}^{n} \pi_i = 1$

- Given π and the model on the right
- We can compute the probability of weather sequence:

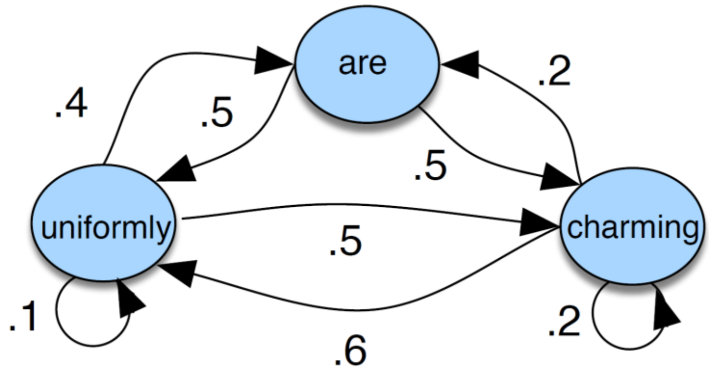
  HOT HOT COLD HOT



Markov Chains

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} a_{12} \ldots a_{N1} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	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, $\sum_{i=1}^{n} \pi_i = 1$

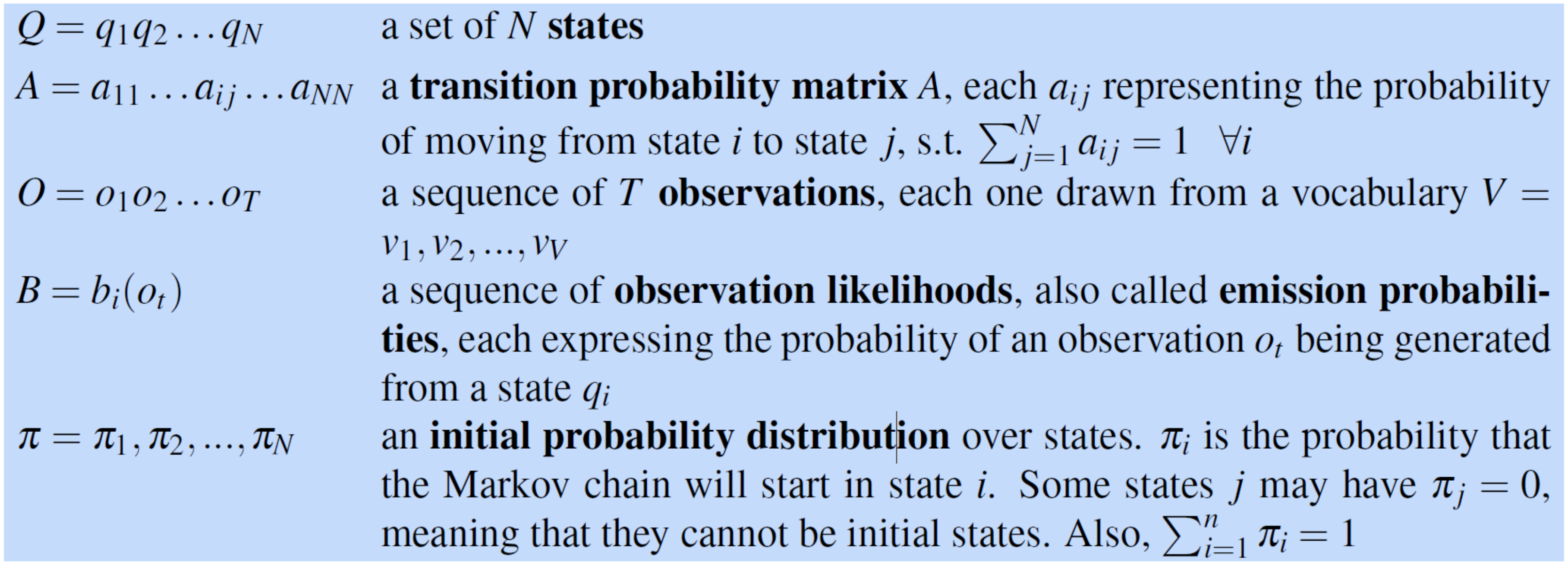
- Another example: a Markov chain for assigning a probability to a sequence of words $w_1 \ldots w_t$
- The bigram model, where edge is $P(w_j | w_i)$.



Hidden Markov Model

- A Markov chain is useful when we need to compute a probability for a sequence of observable events, e.g.,
  - based on today's weather to predict tomorrow's weather
  - based on current word to predict next word (as in bigram model)
- In many cases, however, the events we are interested in are **hidden**.
  - For example, in POS tagging, we can only observe words, but not their tags.
  - We cannot use the current tag to predict the next tag for a 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** and **hidden events**. For POS tagging:
  - Observed events are the words in the input sentence
  - Hidden events are the part-of-speech tags for these words
  - The observed events are considered as causal factors in this probabilistic model

Hidden Markov Model



$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state q_i
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	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, $\sum_{i=1}^{n} \pi_i = 1$

- **States** are the set of possible tags in tagset
- **Transition probability** is the probability of moving from one tag to another e.g., P(Noun|Adj), P(Noun|DT), P(Verb|Adj)
- **Observation** is a word; observations are the given sentence for POS tagging
- **Observation likelihood** is the likelihood of seeing a word for a tag e.g., P(table|Noun)

Hidden Markov Model

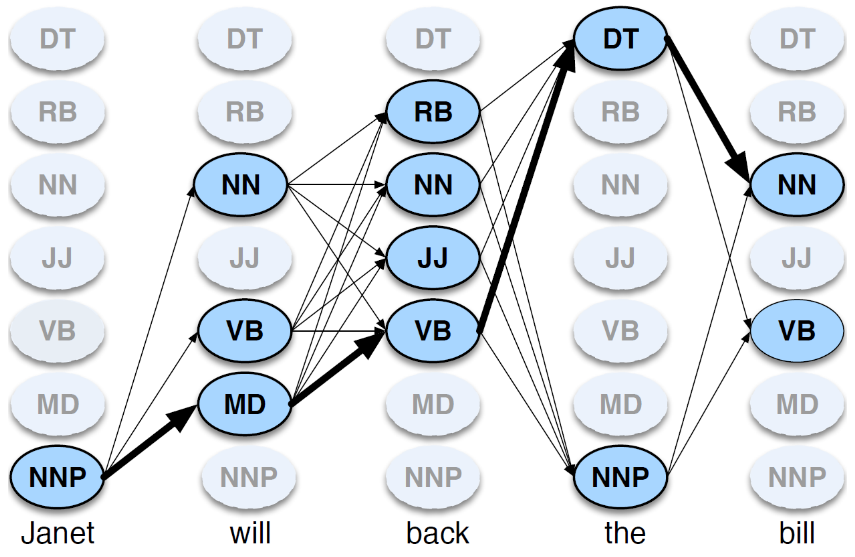
- A first-order hidden Markov model instantiates two simplifying assumptions.
- **Markov Assumption**: the probability of a particular state depends only on the previous state
    - $P(q_i = a | q_1 \dots q_{i-1}) \ = \boldsymbol{P(q_i = a | q_{i-1})}$
- **Output Independence Assumption**: the probability of an output observation o_i depends only on the state q_i that produced the observation and not on any other states or any other observations
    - $P(o_i | q_1, \dots, q_i, \dots, q_T \,; o_1, \dots, o_i, \dots, o_T) \ = \ \boldsymbol{P(o_i | q_i)}$
- **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.

POS tagging with HMM

- Out of all possible sequences of n tags $t_1 \dots t_n$ the single tag sequence such that $P(t_1 \dots t_n | w_1 \dots w_n)$ is highest.

$$\hat{t}_{1:n} = arg \max_{t_{1:n}} P(t_{1:n} | w_{1:n})$$

  - Hat ^ means "**our estimate** of the best one"
  - $arg \max_x f(x)$ means "**the x such that $f(x)$ is maximized**"



- Janet has **one** possible tag
- Will has **three** possible tags
- Back has **four** possible tags
- The has **two** possible tags
- Bill has **two** possible tags

There are **48** possible tag sequences

POS tagging with HMM

- Out of all possible sequences of n tags $t_1 \dots t_n$ the single tag sequence such that $P(t_1 \dots t_n | w_1 \dots w_n)$ is highest.

$$\hat{t}_{1:n} = arg \max_{t_{1:n}} P(t_{1:n} | w_{1:n})$$

  - Hat ^ means "**our estimate** of the best one"
  - $arg \max_{x} f(x)$ means "**the x such that $f(x)$ is maximized**"

- Use Bayes rule to transform this equation into a set of other probabilities that are easier to compute
  - $$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$
  - $$\hat{t}_{1:n} = arg \max_{t_{1:n}} P(t_{1:n} | w_{1:n}) = arg \max_{t_{1:n}} \frac{P(w_{1:n}|t_{1:n})P(t_{1:n})}{P(w_{1:n})}$$

$$\boxed{\approx \boldsymbol{arg \max_{t_{1:n}} P(w_{1:n}|t_{1:n})P(t_{1:n})}}$$

POS tagging with HMM

- $\underset{t_{1:n}}{arg\,max}\, \boldsymbol{P(w_{1:n}|t_{1:n})P(t_{1:n})}$

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

Output Independence Assumption: the probability of an output observation o_i depends only on the state q_i that produced the observation.

The probability of a word appearing depends only on **its own POS tag**, e.g., $P(table|Noun)$

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

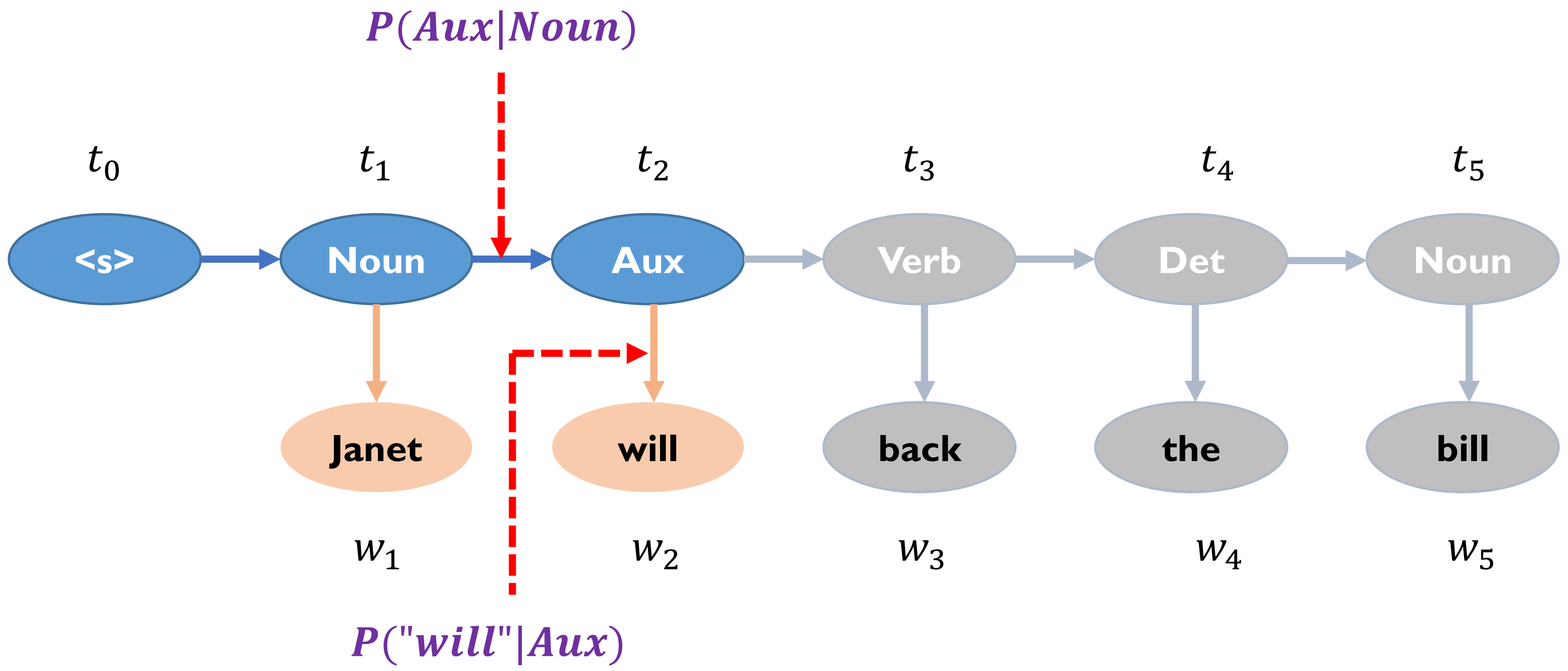
Markov Assumption: the probability of a particular state depends only on the previous state

The probability of a tag appearing depends only on **the previous tag**, e.g., $P(Noun|Adj)$

- $\boldsymbol{P(w_{1:n}|t_{1:n})P(t_{1:n}) \approx \underline{\prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})}}$

HMM is a generative model

- $\underset{t_{1:n}}{arg\ max}\ \boldsymbol{P(w_{1:n}|t_{1:n})P(t_{1:n})} \approx \prod_{i=1}^{n} \boldsymbol{P(w_i|t_i)P(t_i|t_{i-1})}$



Computing the two kinds of probabilities

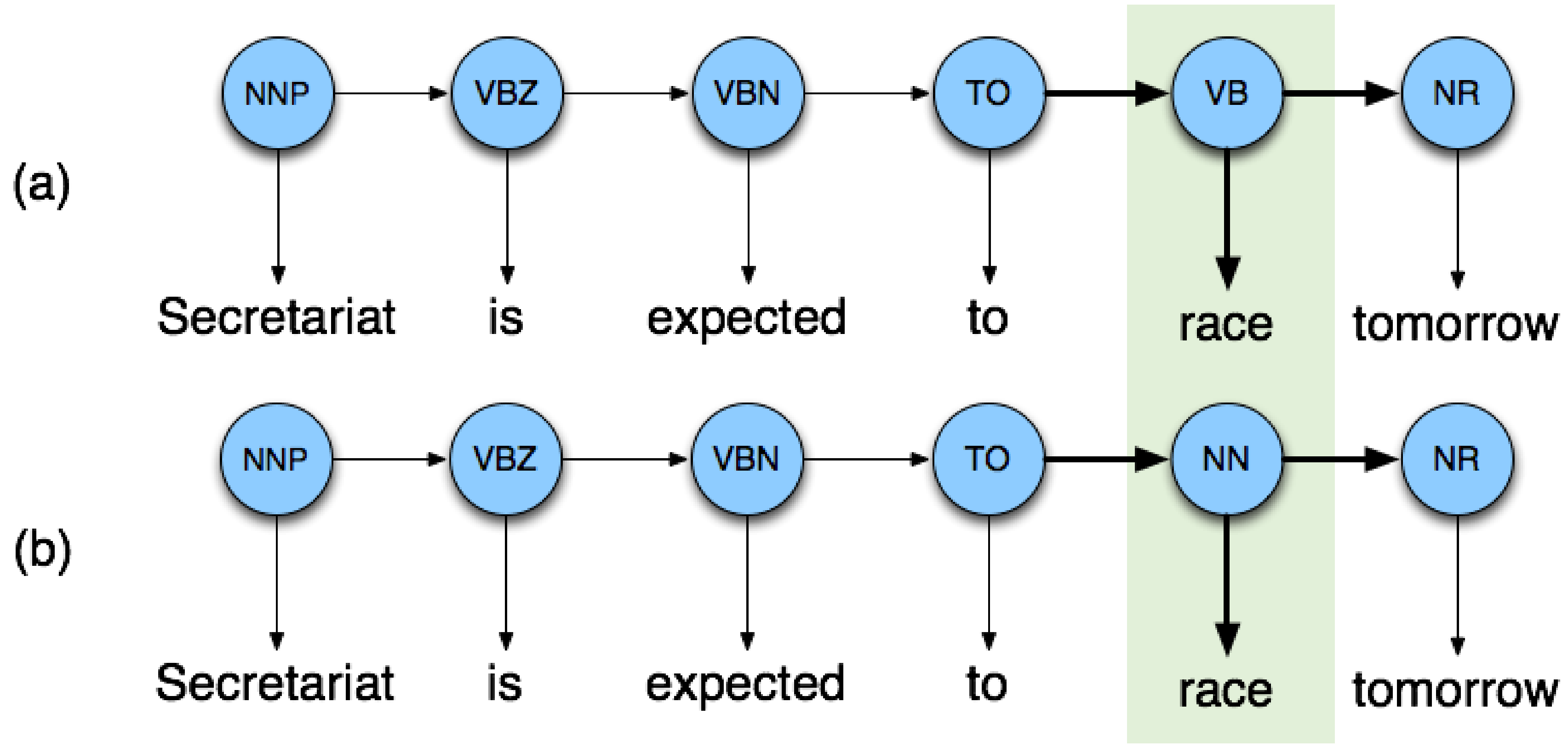
- Given a fully annotated dataset, where every token in a sentence is annotated with its POS tag.
  - An example sentence: The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.
  - We will have tag transitions: DT→JJ, JJ→NN, NN→VBD, …
  - We also have words and their frequencies for each tag

- **Tag transition** probabilities $p(t_i\,|t_{i-1})$
  - $P(t_i|t_{i-1}) = \frac{Count(t_{i-1},t_i)}{Count(t_{i-1})}$ e.g., relative frequency of JJ following DT.

- **Word likelihood** probabilities $p(w_i|t_i)$
  - $P(w_i|t_i) = \frac{Count(t_i,w_i)}{Count(t_i)}$ e.g., among all words tagged to DT, how many are "The"

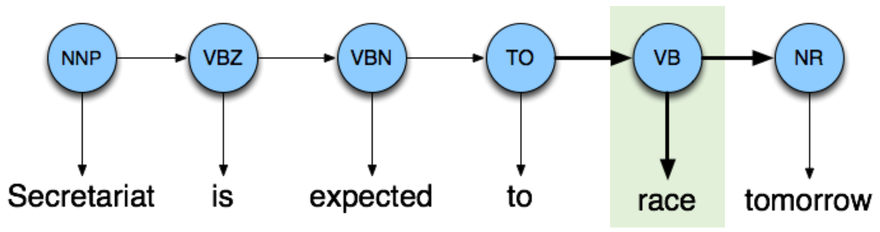
Example: to disambiguate "race"

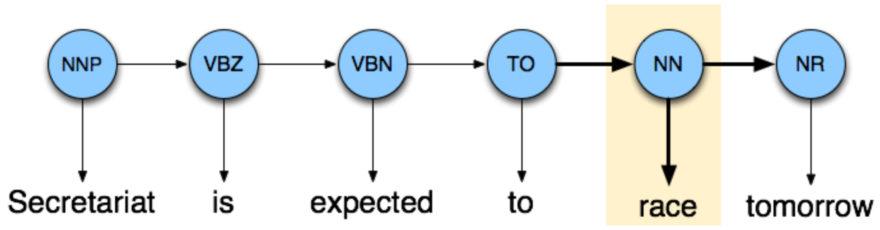
- Assuming tags of all other words are known



Computing probabilities of the two tag sequences

$$\underset{t_{1:n}}{arg\,max}\, \boldsymbol{P(w_{1:n}|t_{1:n})P(t_{1:n})} \approx \prod_{i=1}^{n} \boldsymbol{P(w_i|t_i)P(t_i|t_{i-1})}$$



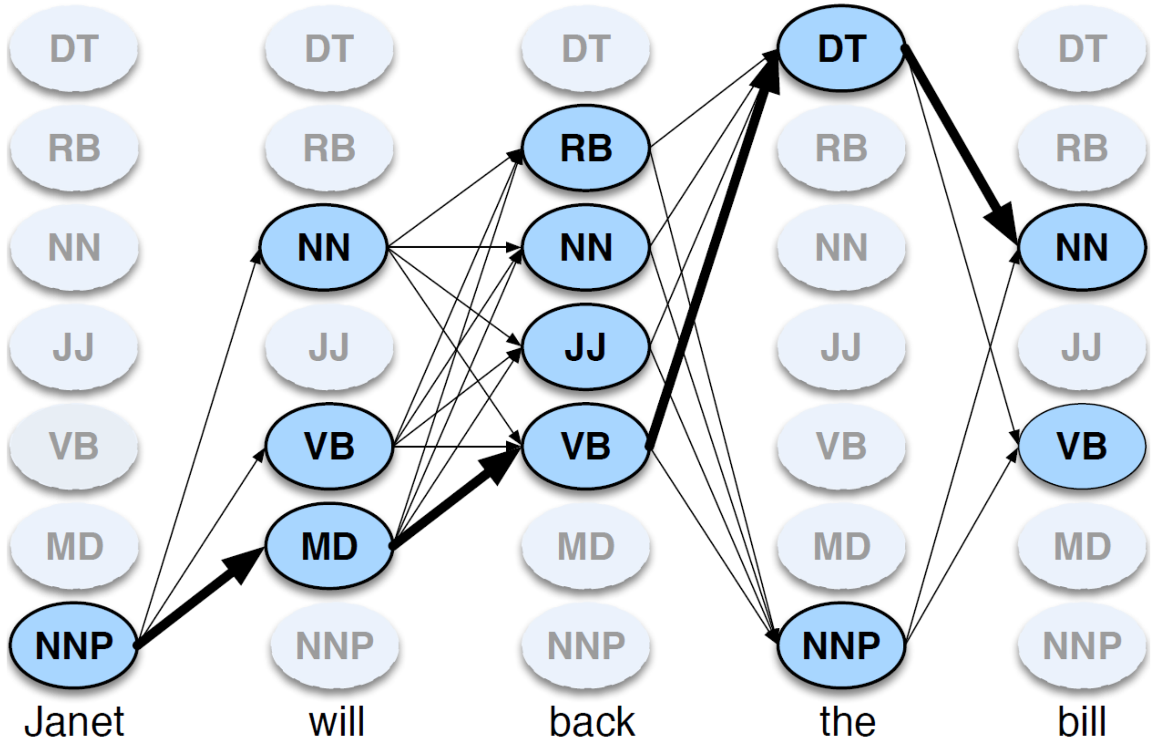


$$\begin{aligned}&p(Secretariat|NNP) * p(NNP|Start)\\ &* p(is|VBZ) * p(VBZ|NNP)\\ &* p(expected|VBN) * p(VBN|VBZ)\\ &* p(to|TO) * p(TO|VBN)\\ &* p(race|VB) * p(VB|TO)\\ &* p(tomorrow|NR) * p(NR|VB)\end{aligned}$$

$$\begin{aligned}&p(Secretariat|NNP) * p(NNP|Start)\\ &* p(is|VBZ) * p(VBZ|NNP)\\ &* p(expected|VBN) * p(VBN|VBZ)\\ &* p(to|TO) * p(TO|VBN)\\ &* p(race|NN) * p(NN|TO)\\ &* p(tomorrow|NR) * p(NR|NN)\end{aligned}$$

HMM Decoding

- We have the two probabilities $p(t_i \mid t_{i-1})$ and $p(w_i \mid t_i)$
  - Given a sentence, for each word, we know which tag can generate this word
  - We now have a full picture of all possible tag sequences

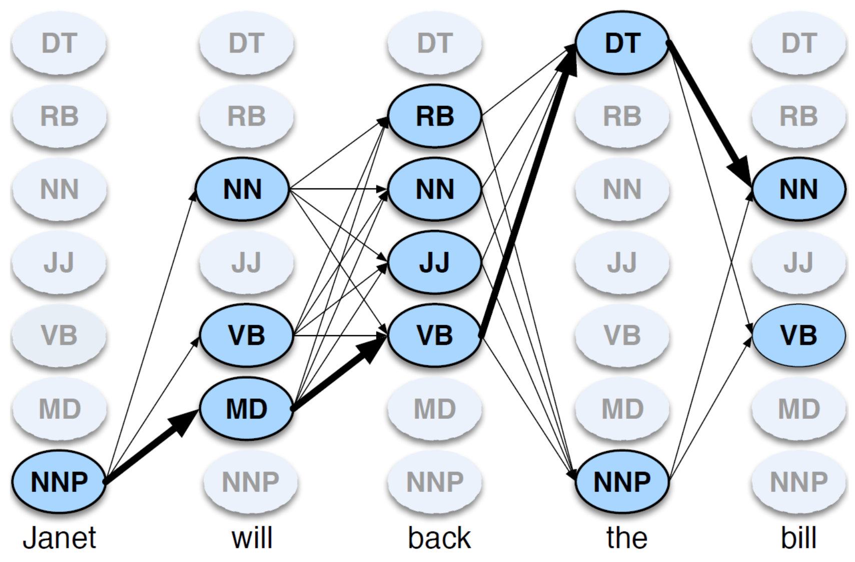


The tags in gray color cannot generate the words

There are many possible tag sequences; which is the best one!

HMM Decoding

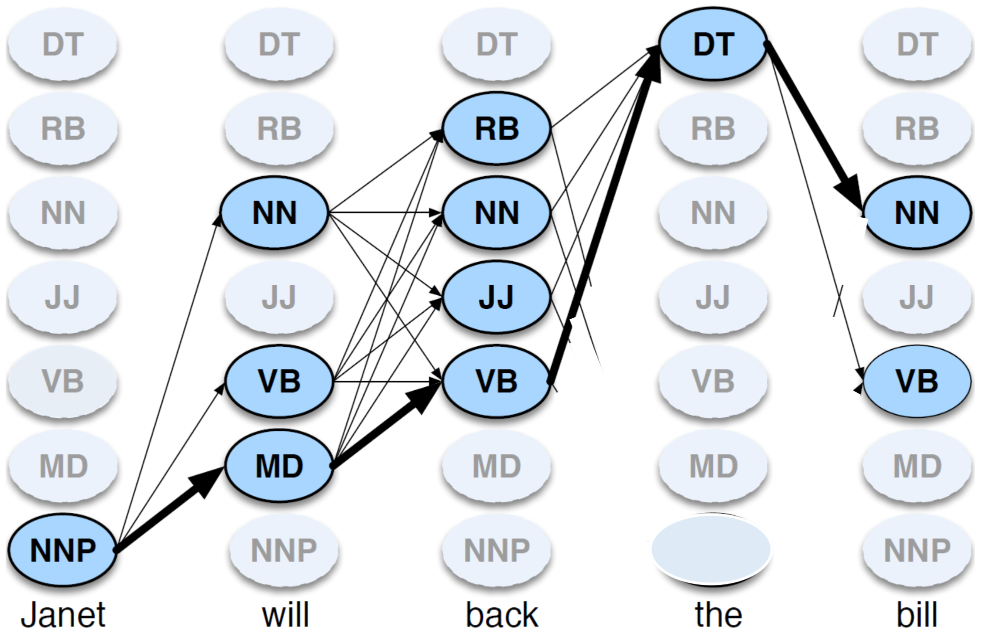
- There are many possible sequences
  - NNP, NN, RB, DT, NN
  - NNP, NN, NN, DT, NN
  - NNP, MD, VB, DT, NN
  - …
- We need a "clever" algorithm to minimize our computation cost
  - Many computations are reparative
  - We can save the computed results
  - → dynamic computing



$p(Secretariat|NNP)$
$* p(NNP|Start)$
$* p(is|VBZ)$
$* p(VBZ|NNP)$
$* p(expected|VBN)$
$* p(VBN|VBZ)$
$* p(to|TO) * p(TO|VBN)$
$* p(race|VB) * p(VB|TO)$
$* p(tomorrow|NR)$
$* p(NR|VB)$

$p(Secretariat|NNP)$
$* p(NNP|Start)$
$* p(is|VBZ)$
$* p(VBZ|NNP)$
$* p(expected|VBN)$
$* p(VBN|VBZ)$
$* p(to|TO) * p(TO|VBN)$
$* p(race|NN)$
$* p(NN|TO)$
$* p(tomorrow|NR)$
$* p(NR|NN)$

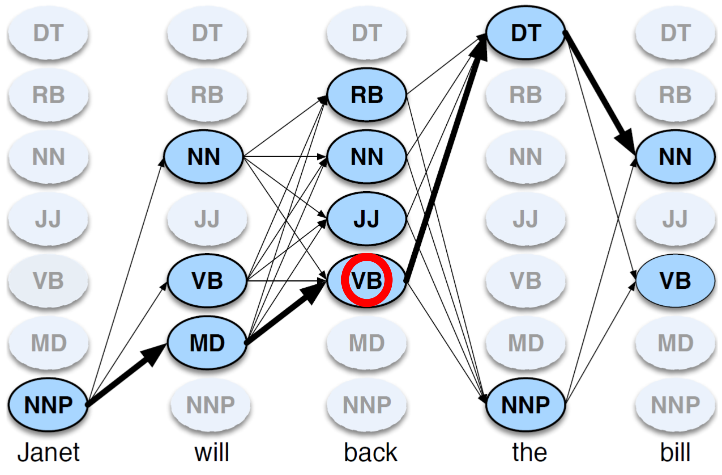
Example illustration



- Assume "the" can only take DT tag (simplify this problem)

- The best path to "bill" taking NN tag is:
  - $P(best\ path\ to\ DT) * P(NN|DT) * P(bill|NN)$
- The best path to "bill" taking VB tag is:
  - $P(best\ path\ to\ DT) * P(VB|DT) * P(bill|VB)$

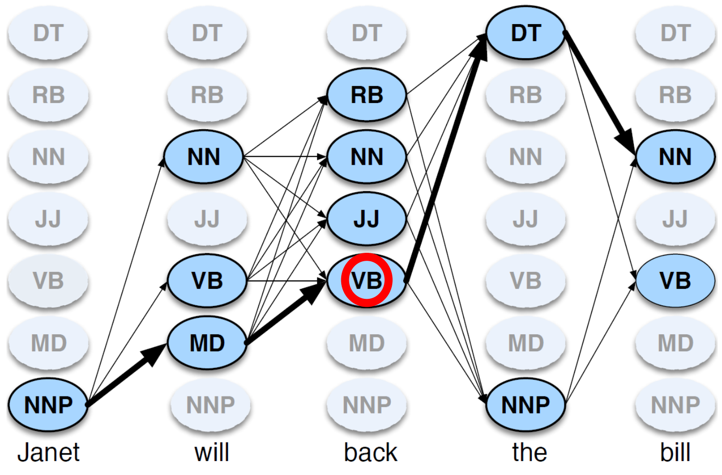
The Viterbi algorithm



- The Viterbi algorithm first sets up a **probability matrix** or lattice
  - One column for **each observation** o_t (a word)
  - One row for **each state** q_t (tag) in the state graph
  - Each cell of the lattice, $v_t(j)$, 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 model parameters.

- Take the cell with tag **VB** for word "**back**" as an example
  - There are three paths from starting NNP to reach VB for "back", each with a different probability along the path
    - NNP→ NN→ VB
    - NNP→ VB→ VB
    - NNP→ MD→ VB
  - $v_{"back"}(VB)$ is the probability computed based on **the most probable path** among the three, **till this observation** "back" for tag "VB".

The Viterbi algorithm

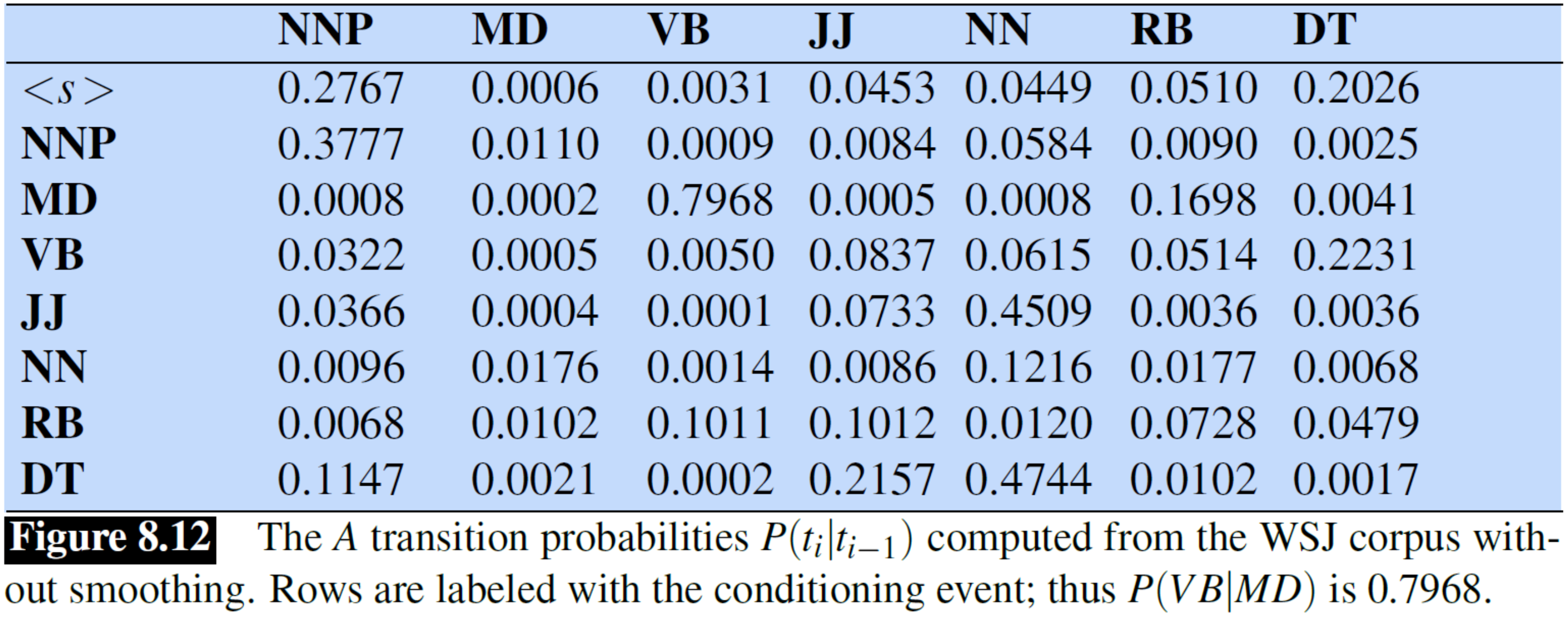


- $v_{"back"}(VB)$ is the probability computed based on **the most probable path till this observation** "back" for tag VB
- Similarly
  - $v_{"back"}(RB)$ is the probability computed based on **the most probable path till this observation** "back" for tag RB:
    - NNP→NN→RB, NNP→VB→RB, NNP→MD→RB
  - $v_{"back"}(NN)$, $v_{"back"}(JJ)$ are the probabilities computed based on **the most probable path till this observation** "back" for tags NN and JJ respectively.
- **How to compute $v_{"back"}(VB)$?**
  - $v_{"back"}(VB) = \max [\, v_{"will"}(NN) \times P(VB|NN),\ v_{"will"}(VB) \times P(VB|VB),\ v_{"will"}(MD) \times P(VB|MD)] \times \mathrm{P}("back"|\mathrm{VB})$
  - $v_t(j)$ iscomputed by **recursively** taking the most probable path to this cell

The Viterbi algorithm: an example

➢ Tag a sentence: Janet will back the bill

Tag transition probabilities $p(t_i \mid t_{i-1})$



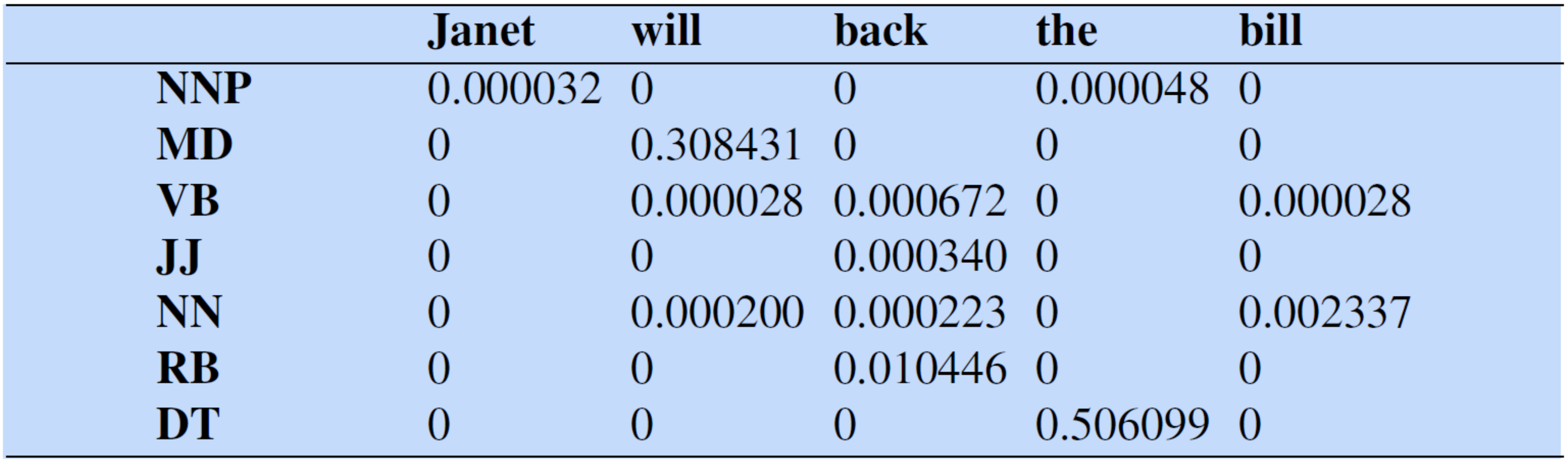
	NNP	MD	VB	JJ	NN	RB	DT
$<s>$	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	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 8.12 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.

The Viterbi algorithm: an example

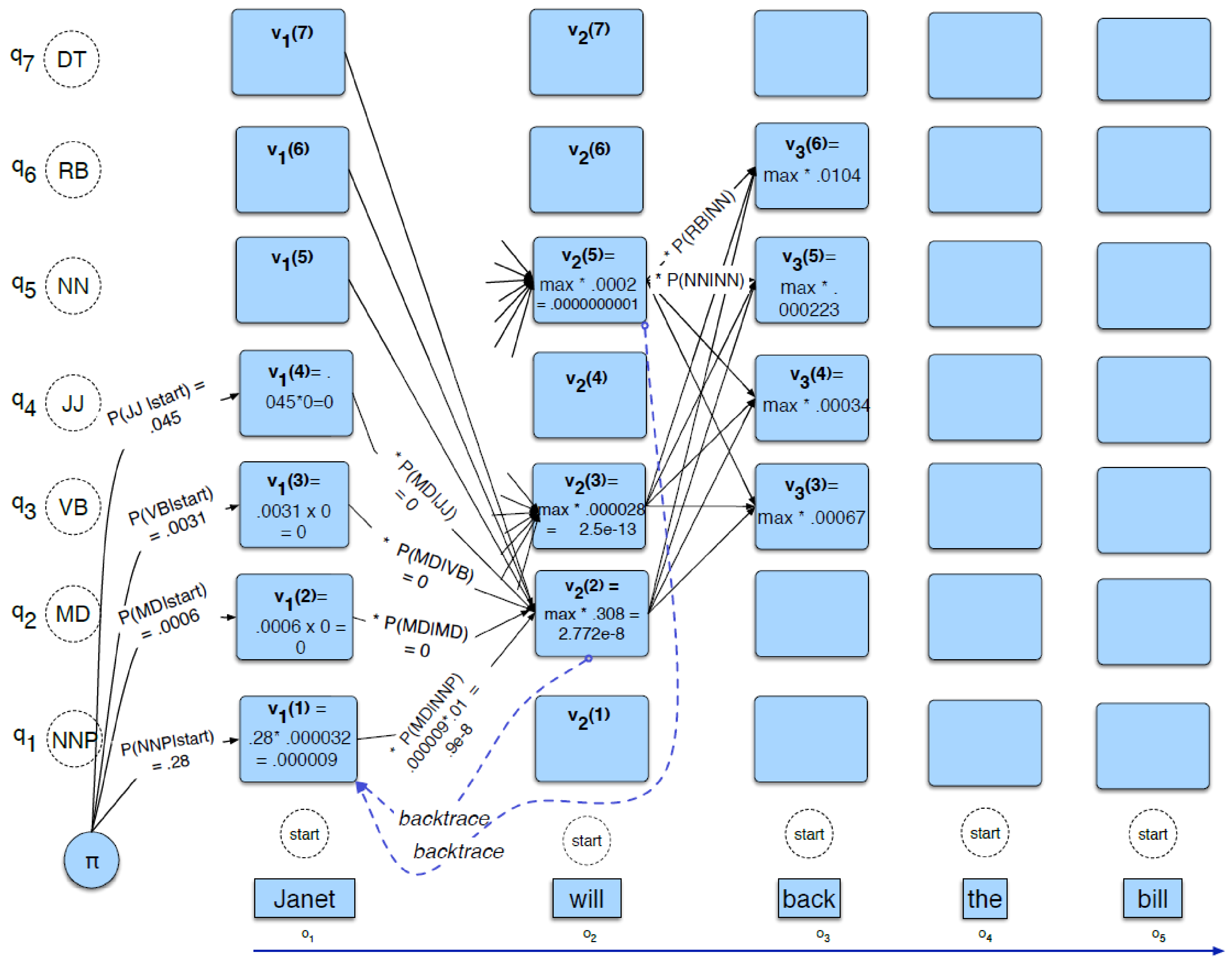
- Tag a sentence: Janet will back the bill

Word likelihood probabilities $p(w_i|t_i)$



	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

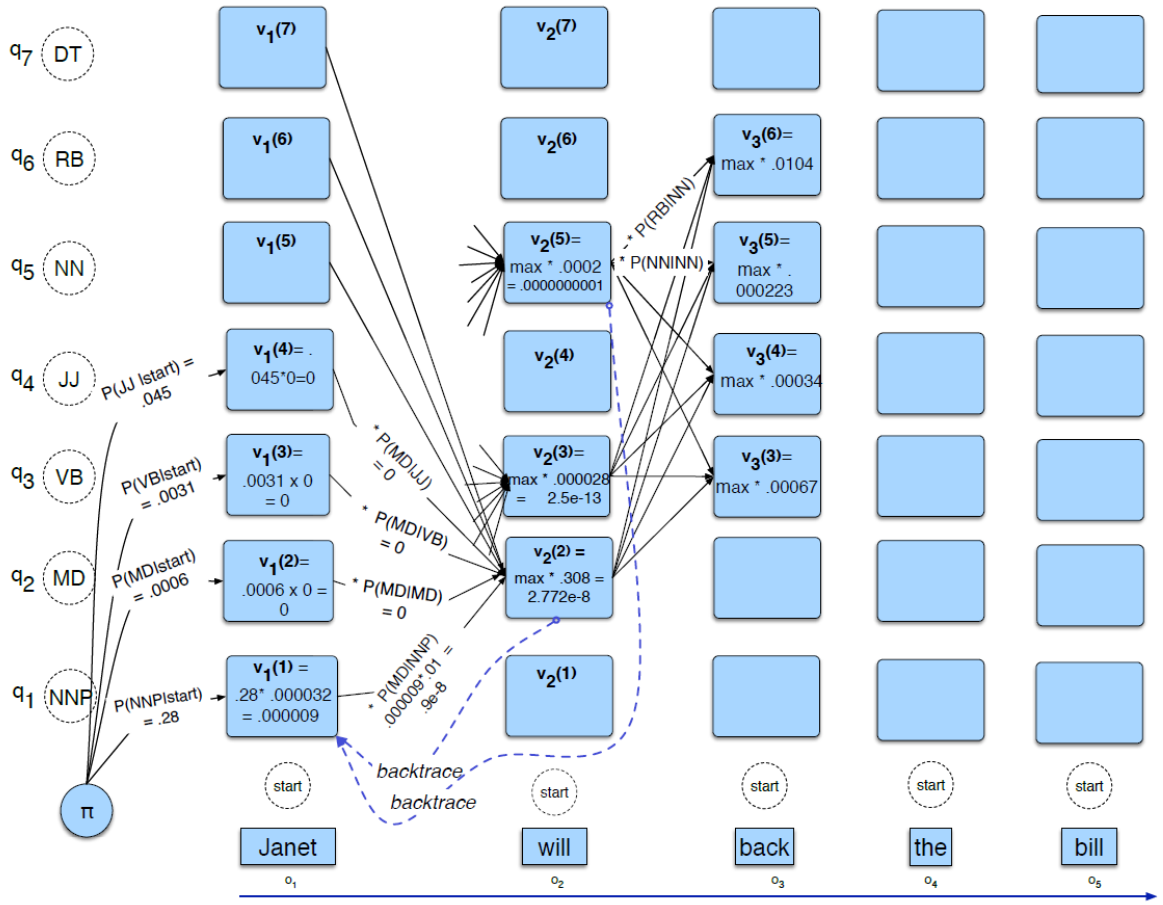
Figure 8.13 Observation likelihoods *B* computed from the WSJ corpus without smoothing, simplified slightly.



q7 DT
q6 RB
q5 NN
q4 JJ
q3 VB
q2 MD
q1 NNP
π
P(JJ |start) = .045
P(VB|start) = .0031
P(MD|start) = .0006
P(NNP|start) = .28
v1(7)
v1(6)
v1(5)
v1(4)= . 045*0=0
v1(3)= .0031 x 0 = 0
v1(2)= .0006 x 0 = 0
v1(1) = .28* .000032 = .000009
* P(MD|JJ) = 0
* P(MD|VB) = 0
* P(MD|MD) = 0
* P(MD|NNP) .000009*.01 = .9e-8
v2(7)
v2(6)
v2(5)= max * .0002 = .0000000001
v2(4)
v2(3)= max * .000028 = 2.5e-13
v2(2) = max * .308 = 2.772e-8
v2(1)
* P(RB|NN)
* P(NN|NN)
v3(6)= max * .0104
v3(5)= max * . 000223
v3(4)= max * .00034
v3(3)= max * .00067
backtrace
backtrace
start
start
start
start
start
Janet
will
back
the
bill
o1
o2
o3
o4
o5

Viterbi algorithm

- From let to right, column by column, compute Viterbi path probability for each cell $v_t(j)$
- Maintain a ***backtrace*** pointer for each cell, to indicate from which cell, the $v_t(j)$ is obtained.
- When $v_t(j)$'s for the last observation are computed, select the max value, and **traceback the sequence**.



	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 8.12 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 8.13 Observation likelihoods B computed from the WSJ corpus without smoothing, simplified slightly.

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
```

Conditional Random Fields (CRFs)

For Information Only

- HMM is a useful and powerful model, but needs some augmentations
  - It is not straightforward to handle unknown words like proper names and acronyms
  - It is hard for generative models to add arbitrary features directly
    - e.g., capitalization can be indicative for proper nouns
    - words ending with "-ed" tend to be past tense (VBD or VBN)
- CRF is a **discriminative** sequence model based on log-linear models
  - We briefly describe the **linear chain CRF**
  - Given we have a sequence of input words $X = x_1, \dots, x_n$ and want to compute a sequence of output tags $Y = y_1, \dots, y_n$
  - In a CRF, we compute $p(Y|X)$ directly, training the CRF to discriminate among the possible tag sequences

$$\hat{Y} = \arg\max_{Y \in \mathcal{Y}} P(Y|X)$$

CRF

- A CRF is a log-linear model that assigns a probability to **an entire output** (tag) sequence Y, out of all possible sequences $\mathcal{Y}$, given **the entire input** (word) sequence X.
  - CRF does not compute a probability for each tag at each time step
  - At each time step, CRF computes log-linear functions over a set of relevant features
  - These local features are aggregated and normalized to produce a global probability for the whole sequence
- In a CRF, the feature function F maps an entire input sequence X and an entire output sequence Y to a feature vector.
  - Assume we have K features, with a weight w_k for each feature F_k

$$p(Y|X) = \frac{\exp\left(\sum_{k=1}^{K} w_k F_k(X,Y)\right)}{\sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y')\right)}$$

CRF

- Re-writing the equation

$$p(Y|X) = \frac{1}{Z(X)} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y)\right)$$

$$Z(X) = \sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y')\right)$$

- These K functions $F_k(X,Y)$ are known as global features; each one is a property of the entire input sequence X and output sequence Y
  - $F_k(X,Y)$ is computed as a sum of local features for each position i in Y
  - Each local feature f_k in a **linear-chain CRF** uses: current output token y_i, the previous output token y_{i-1}, the entire input string X (or any subpart of it), and the current position i.

$$F_k(X,Y) = \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i)$$

  - A general (not linear-chain) CRF allows a feature to make use of any output token.

Features in a CRF POS Tagger

- In a linear-chain CRF, each local feature f_k at position i can depend on any information from: (y_{i-1}, y_i, X, i);
  - Example features:
    - $\mathbb{I}\{x_i = the, y_i = Det\}$
    - $\mathbb{I}\{y_i = PropN, x_{i+1} = street, y_{i-1} = NUM\}$
    - $\mathbb{I}\{y_i = Verb, y_{i-1} = Aux\}$
  - $\mathbb{I}\{x\}$ means "one if x is true, zero otherwise". In NLP, typically all CRF features take on the value one or zero.

- Feature template: automatically populate the set of features from every instance in the training and test set.
  - Example template: $< y_i, x_i >, < y_i, y_{i-1} >, < y_i, x_{i-1}, x_{i+2} >$
  - Example sentence: "Janet/NNP will/MD back/VB the/DT bill/NN", when x_i is the word "back", we will have the following features (with random feature numbers)
    - f_{123}: $y_i = VB$ and $x_i = back$ f_{456}: $y_i = VB$ and $y_{i-1} = MD$
    - f_{789}: $y_i = VB$ and $x_{i-1} = will$ and $x_{i+2} = bill$

Features in a CRF POS Tagger

- Features can be manually crafted
  - Word shape features: an abstract letter pattern of the word, by mapping lower-case letters to 'x', upper-case to 'X', numbers to 'd', and retaining punctuation.
    - Word "I.M.F." is mapped to X.X.X
    - Word "DC10-30" is mapped to XXdd-dd
  - Features based on prefix or suffix: a word contains a certain prefix like 'un-', or specific suffix like '-ing' or '-ed'

- CRF does not learn weights for each of these local features f_k.
  - We first sum the values of each local feature (for example feature f_{123}) over the entire sentence, to create each global feature (for example F_{123}).
  - The global features will then be multiplied by weight w_{123}.

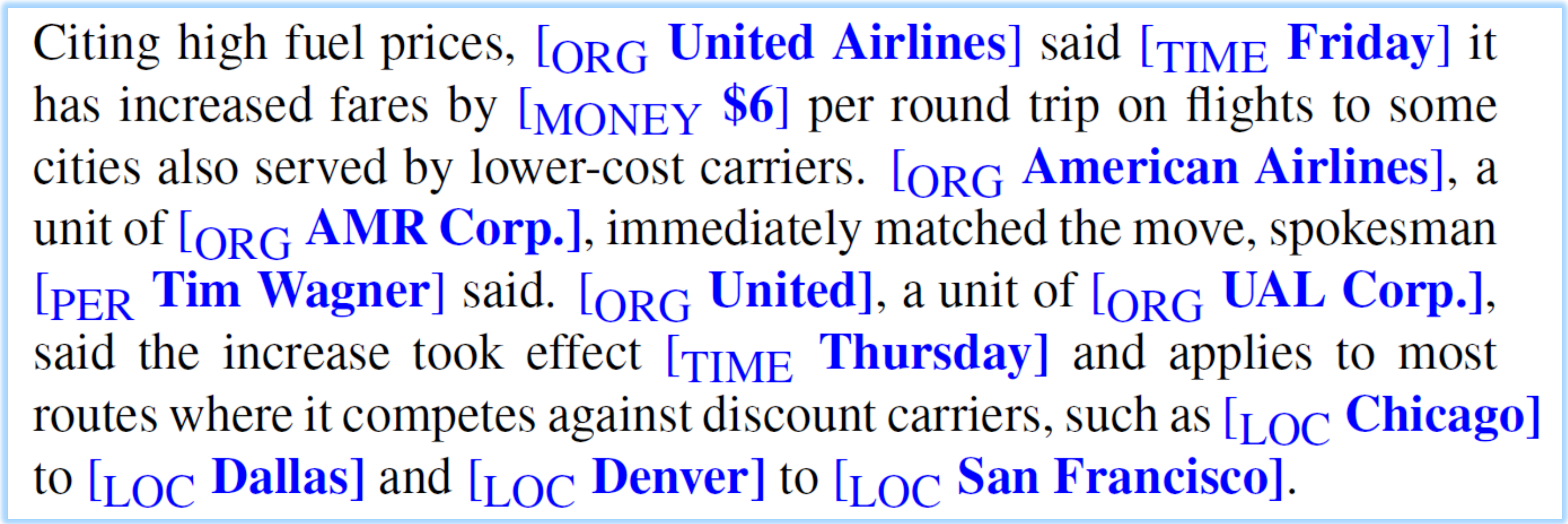
- For training and inference, there is always a fixed set of K features with K weights, even though the length of each sentence is different.
  - Refer to textbook for Inference and Training for CRFs

POS and NER → Sequence Labeling

- **Parts of speech** (POS) refers to **word classes** such as Noun, Verb, Adjective, etc.
  - Also known as lexical categories, word classes, morphological classes, lexical tags
  - Knowing word class tells us more about neighboring words and syntactic structure
    - E.g., nouns in English are often preceded by determiners and adjectives
    - Verbs have dependency links to nouns
  - POS tagging is a key aspect of parsing sentence structure.
- **Named entity** is proper name for person, location, organization, etc.
  - NEs are useful clues to sentence structure and meaning understanding.
  - Knowing if a named entity like ***Washington*** is a name of a person, a place, or a university is important to tasks like question answering and information extraction.
- Sequence labeling
  - **POS tagging**: takes a sequence of words and assigns each word a POS like NOUN or VERB
  - **Named Entity Recognition** (NER): assigns words or phrases tags like PERSON, LOCATION, or ORGANIZATION.

Named Entity Recognition

- A named entity is roughly anything that can be referred to with **a proper name**: a person, a location, an organization; or a concept, a treatment; or date and time; or a brand, a product; depends on the domain
- The task of **named entity recognition** (NER) is to find **spans of text** that constitute proper names and tag the type of the entity.
  - Four entity tags are most common: **PER** (person), **LOC** (location), **ORG** (organization), or **GPE** (geo-political entity).


Citing high fuel prices, [$_{\text{ORG}}$ **United Airlines**] said [$_{\text{TIME}}$ **Friday**] it has increased fares by [$_{\text{MONEY}}$ **$6**] per round trip on flights to some cities also served by lower-cost carriers. [$_{\text{ORG}}$ **American Airlines**], a unit of [$_{\text{ORG}}$ **AMR Corp.**], immediately matched the move, spokesman [$_{\text{PER}}$ **Tim Wagner**] said. [$_{\text{ORG}}$ **United**], a unit of [$_{\text{ORG}}$ **UAL Corp.**], said the increase took effect [$_{\text{TIME}}$ **Thursday**] and applies to most routes where it competes against discount carriers, such as [$_{\text{LOC}}$ **Chicago**] to [$_{\text{LOC}}$ **Dallas**] and [$_{\text{LOC}}$ **Denver**] to [$_{\text{LOC}}$ **San Francisco**].

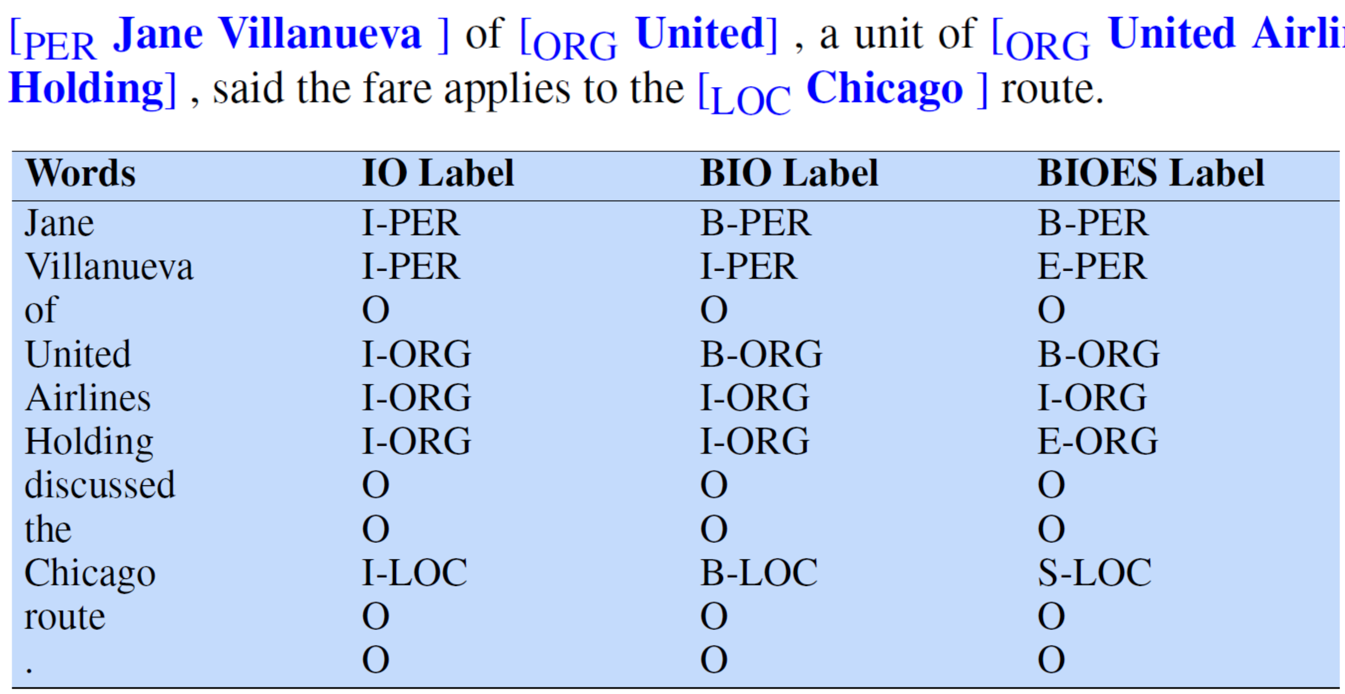
POS Tagging vs NER

- Differences between part-of-speech tagging and NER
  - In POS tagging, each word gets one tag,
  - In NER, we do not know the boundary of names, before we can label them

- Similarities between POS tagging and NER
  - The same word may have different POS tags, like adj and adv for "Back"
  - The same text span may have different NE types, like Victoria, Washington
  - Both POS tagging and NER require surrounding words as context to make the tagging
  - Both POS tagging and NER work at sentence level → sequence labeling

Sequence labeling for NER

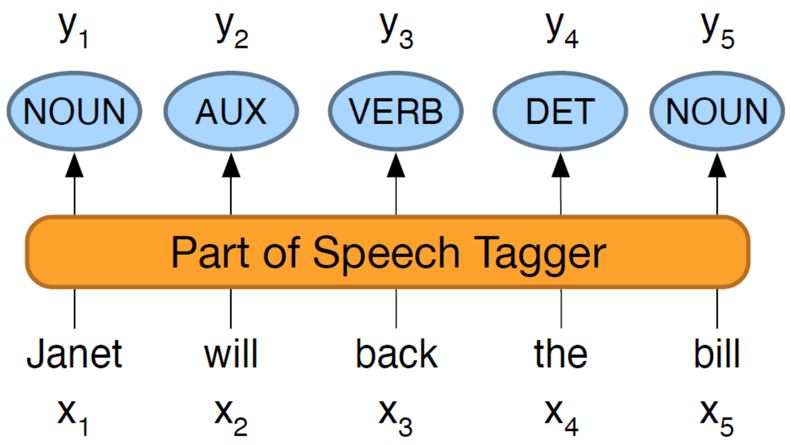
- A standard approach to sequence labeling for a **span-recognition problem** like NER is **BIO** tagging.
  - B: begin, I: inside, O: outside
- Variants: **IO**, **BIOES** (**E** for ending, **S** for single), **BIOEU** (**U** for unit)



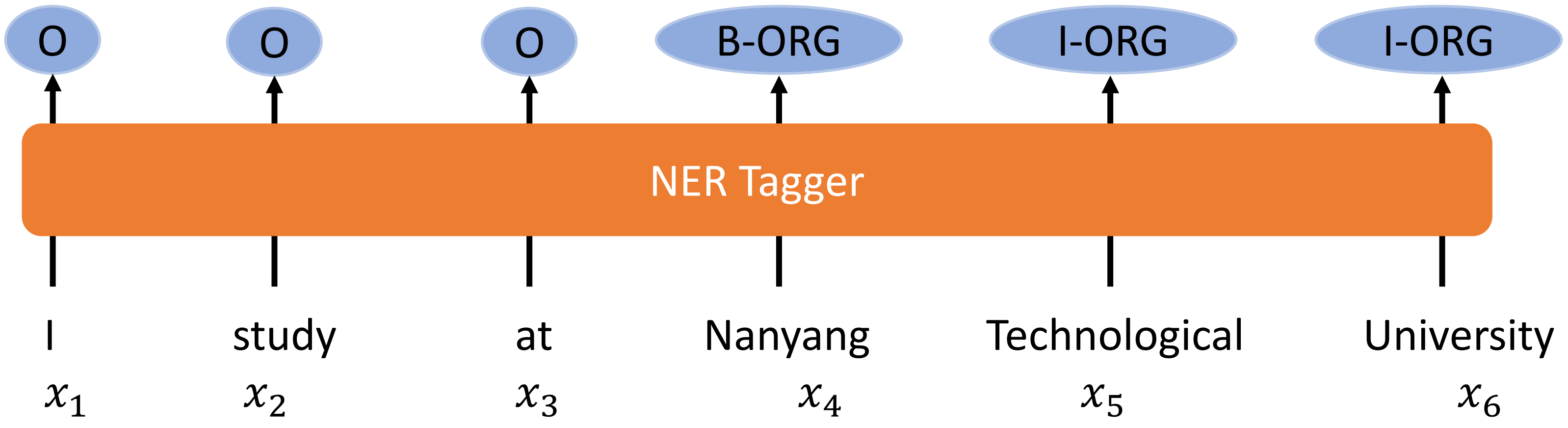
[$_{\text{PER}}$ **Jane Villanueva**] of [$_{\text{ORG}}$ **United**] , a unit of [$_{\text{ORG}}$ **United Airlines Holding**] , said the fare applies to the [$_{\text{LOC}}$ **Chicago**] route.

Words	IO Label	BIO Label	BIOES Label
Jane	I-PER	B-PER	B-PER
Villanueva	I-PER	I-PER	E-PER
of	O	O	O
United	I-ORG	B-ORG	B-ORG
Airlines	I-ORG	I-ORG	I-ORG
Holding	I-ORG	I-ORG	E-ORG
discussed	O	O	O
the	O	O	O
Chicago	I-LOC	B-LOC	S-LOC
route	O	O	O
.	O	O	O

POS Tagging vs NER



- Models for POS tagging can be applied to NER
  - HMM
  - CRF
- And other sequence labelling tasks



Features for NER for CRF models

- Example features
  - Identity of w_i (or the word itself), identity of neighboring words
  - Embeddings for w_i, embeddings for neighboring words
  - Part of speech of w_i, part of speech of neighboring words
  - Presence of w_i in a gazetteer
  - w_i contains a particular prefix (from all prefixes of length$\leq$4)
  - w_i contains a particular suffix (from all suffixes of length$\leq$4)
  - Word shape of w_i (e.g., upper/lower case letters, digits), word shape of neighboring words, e.g., XXxxdd
  - Short word shape of w_i, short word shape of neighboring words
    - For short word shape, consecutive character types are removed, so XXxxdd becomes Xxd.
- Gazetteer: a list of place names as an external resource. This can be replaced with a list of names in domain-specific settings.

Features for NER for CRF models

- Example features for a sentence
  - Not a full listing of features
  - Not including neighboring words
  - Assuming gazetteer includes Villanueva and Chicago

Words	POS	Short shape	Gazetteer	BIO Label
Jane	NNP	Xx	0	B-PER
Villanueva	NNP	Xx	1	I-PER
of	IN	x	0	O
United	NNP	Xx	0	B-ORG
Airlines	NNP	Xx	0	I-ORG
Holding	NNP	Xx	0	I-ORG
discussed	VBD	x	0	O
the	DT	x	0	O
Chicago	NNP	Xx	1	B-LOC
route	NN	x	0	O
.	.	.	0	O

Summary

- POS tag: word types
  - POS tagging with HMM
  - The Viterbi algorithm
  - Conditional Random Fields
- Named entity
  - NER as a sequence labelling task
- Reference:
  - Chapter 8 https://web.stanford.edu/~jurafsky/slp3/

What can we do?

- Given a sentence, we can select POS taggers to tag the words in the sentence with their correct word categories
  - This would immediately enable us to select the words in certain categories
  - We can also combine with RegEx to find word sequences by patterns
  - For example, a noun phrase may have this pattern: an optional determiner, zero, one or more adjectives, then a noun.
- Given a sentence, we can also find the named entities from the sentence with a NER model.
  - This offers many more ways to understand the document, like linking the entities to Wikipedia to understand the background information for each entities
- We may also formulate other related problems to a sequence labelling task, by using the BIO tagging scheme.