Part-of-Speech Tagging

Part-of-Speech

Part-of-Speech

- **Each word belongs to a word class**.
- The **word class of a word** is known as **part-of-speech (POS)** of that word.
- Most POS tags implicitly encode fine-grained specializations of eight basic parts of speech:
  - *noun, verb, pronoun, preposition, adjective, adverb, conjunction, article*
- These categories are based on morphological and distributional similarities (not semantic similarities).
- Part of speech is also known as:
  - *word classes, morphological classes, lexical tags, syntactic categories*

Part-of-Speech

- A **POS tag of a word** describes **the major and minor word classes of that word**.
- A POS tag of a word gives a significant amount of information about that word and its neighbours.
  - For example, a possessive pronoun (my, your, her, its) most likely will be followed by a noun, and a personal pronoun (I, you, he, she) most likely will be followed by a verb.

- Most of words have a single POS tag, but some of them have more than one.
- For example, **book/noun** or **book/verb**
  - I bought a **<u>book</u>**.
  - Please **<u>book</u>** that flight.

English Word Classes

- Part-of-speech can be divided into two broad categories:
  - **closed class types** -- such as prepositions
  - **open class types** -- such as noun, verb
- **Closed class words** are generally also **function words**.
  - Function words play important role in grammar
  - Some function words are: *of*, *it*, *and*, *you*
  - Functions words are most of time very short and frequently occur.
- There are **four major open classes**.
  - noun, verb, adjective, adverb
  - a new word may easily enter into an open class.
- Word classes may change depending on the natural language, but all natural languages have at least two word classes: *noun* and *verb*.

Nouns

- Nouns can be divided as:
  - ***proper nouns*** -- names for specific entities such as Ankara, John, Ali
  - ***common nouns***
- Proper nouns do not take an article but common nouns may take.
- Common nouns can be divided as:
  - ***count nouns*** -- they can be singular or plural -- chair/chairs
  - ***mass nouns*** -- they are used when something is conceptualized as a homogenous group -- snow, salt
- Mass nouns cannot take articles ***a*** and ***an***, and they can not be plural.

Verbs

- Verb class includes the words referring actions and processes.
- Verbs can be divided as:
  - main verbs -- open class -- draw, bake
  - auxiliary verbs -- closed class -- can, should
- Auxiliary verbs can be divided as:
  - copula -- be, have
  - modal verbs -- may, can, must, should
- Verbs have different morphological forms:
  - non-3rd-person-sg eat
  - 3rd-person-sg - eats
  - progressive -- eating
  - past -- ate
  - past participle -- eaten

Adjectives

- Adjectives describe properties or qualities
  - for color -- black, white
  - for age -- young, old

- In Turkish, all adjectives can also be used as noun.

kırmızı kitap	*red book*
kırmızıyı	*the red one (ACC)*

Adverbs

- Adverbs normally modify verbs.
- Adverb categories:
  - locative adverbs -- home, here, downhill
  - degree adverbs -- very, extremely
  - manner adverbs -- slowly, delicately
  - temporal adverbs -- yesterday, Friday
- Because of the heterogeneous nature of adverbs, some adverbs such as Friday may be tagged as nouns.

Major Closed Classes

- Prepositions -- on, under, over, near, at, from, to, with
- Determiners -- a, an, the
- Pronouns -- I, you, he, she, who, others
- Conjunctions -- and, but, if, when
- Participles -- up, down, on, off, in, out
- Numerals -- one, two, first, second

Prepositions

- Occur before noun phrases
- indicate spatial or temporal relations
- Example:
  - <u>on</u> the table
  - <u>under</u> chair
- They occur so often. For example, some of the frequency counts in a 16 million word corpora (COBUILD).

of	540,085
in	331,235
for	142,421
to	125,691
with	124,965
on	109,129
at	100,169

Particles

- A particle combines with a verb to form a larger unit called **phrasal verb**.
  - go on
  - turn on
  - turn off
  - shut down

Articles

- A small closed class
- Only three words in the class: a an the
- Marks definite or indefinite
- They occur so often. For example, some of the frequency counts in a 16 million word corpora (COBUILD).

– the	1,071,676
– a	413,887
– an	59,359

- Almost 10% of words are articles in this corpus.

Conjunctions

- Conjunctions are used to combine or join two phrases, clauses or sentences.
- **Coordinating conjunctions --** and or but
  - join two elements of equal status
  - Example: you and me
- **Subordinating conjunctions --** that who
  - combines main clause with subordinate clause
  - Example:
    - <u>I thought</u> *that* <u>you might like milk</u>

Pronouns

- Shorthand for referring to some entity or event.
- Pronouns can be divided:

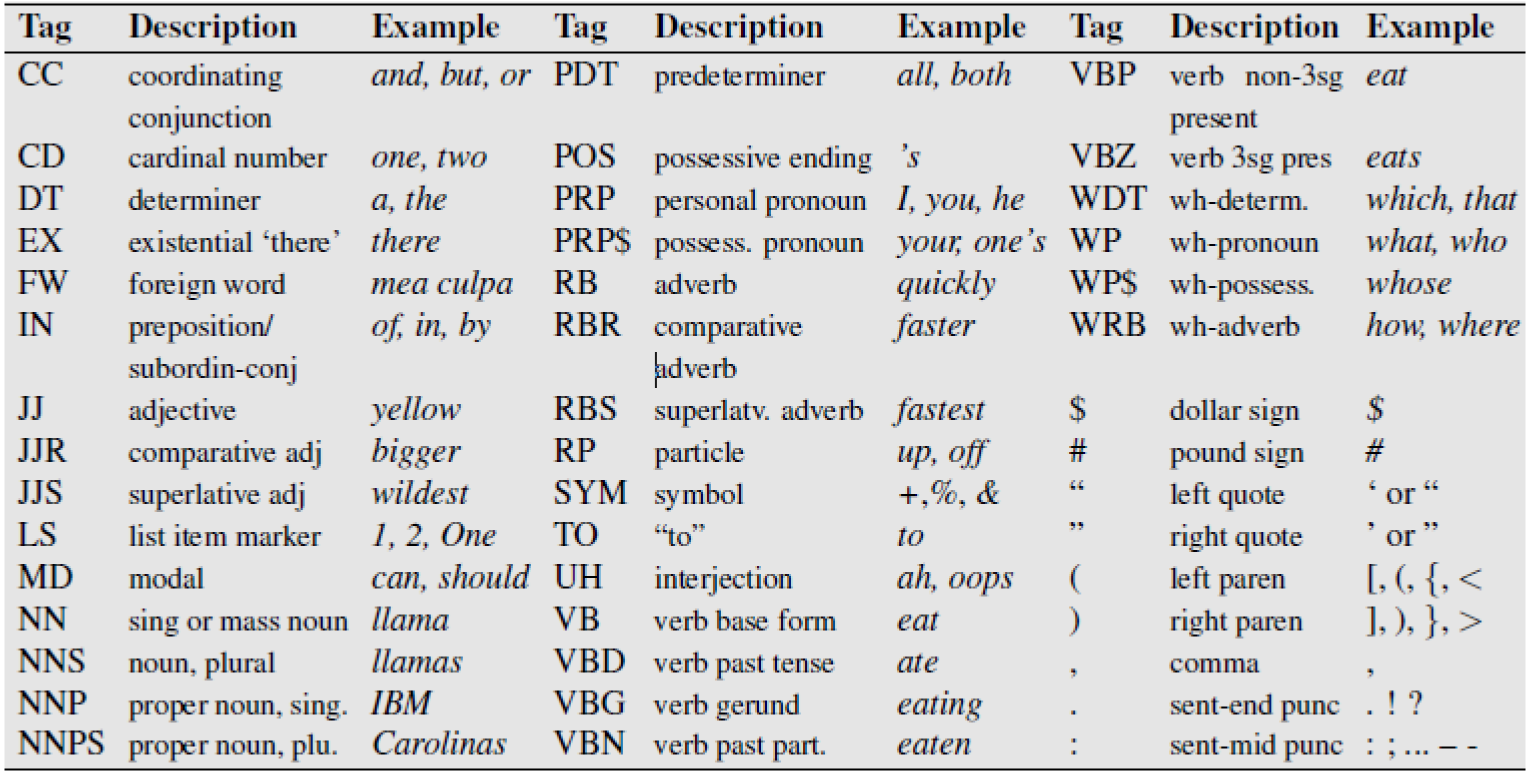
– **personal**	you she I	
– **possessive**	my your his	
– **wh-pronouns**	who what	-- who is the president?

Part-of-Speech Tagsets

- There are various tag sets to choose.
- The choice of the tag set depends on the nature of the application.
  - We may use small tag set (more general tags) or
  - large tag set (finer tags).
- Some of widely used part-of-speech tag sets:
  - Penn Treebank has 45 tags
  - Brown Corpus has 87 tags
  - C7 tag set has 146 tags
- In a tagged corpus, each word is associated with a tag from the used tag set.

Penn Treebank Part-of-Speech Tagset

- An important tagset for English is the 45-tag Penn Treebank tagset.
- A sentence from Brown corpus which is tagged using Penn Treebank tagset.
  - The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.



Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*'s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, that*
EX	existential 'there'	*there*	PRP$	possess. pronoun	*your, one's*	WP	wh-pronoun	*what, who*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, where*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	*$*
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	*#*
JJS	superlative adj	*wildest*	SYM	symbol	*+,%, &*	“	left quote	‘ or “
LS	list item marker	*1, 2, One*	TO	“to”	*to*	”	right quote	’ or ”
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

Part-of-Speech Tagging

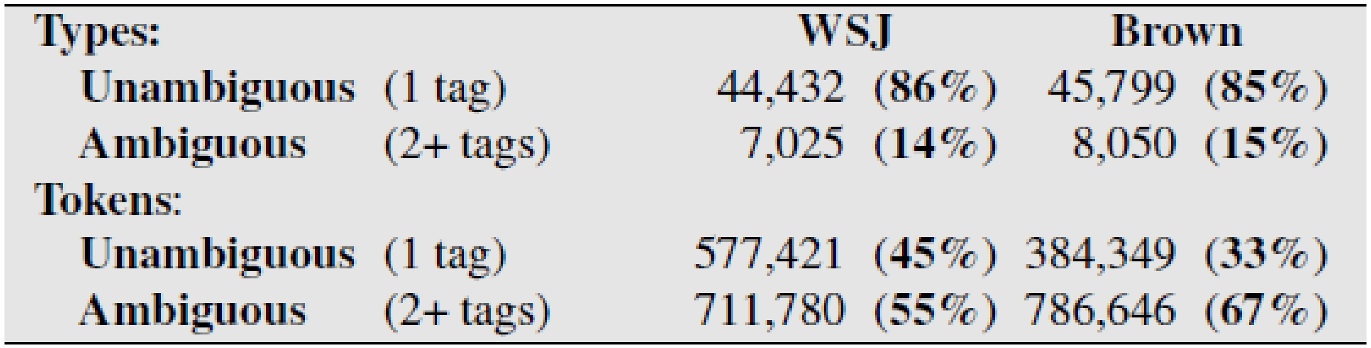
- **Rule-Based POS Tagging**
- **Transformation-Based Tagging**
- **HMM Part-of-Speech Tagging**

Part-of-Speech Tagging

- **Part of speech tagging is simply assigning the correct part of speech tag for each word in an input text.**
- Tagging is a **disambiguation task**; words are ambiguous—have more than one possible part-of-speech and the goal is to find the correct tag for the situation.
  - For example, **book** can be a **verb** (*book that flight*) or a **noun** (*hand me that book*).
- There are different algorithms for tagging.
  - Rule Based Tagging
  - Transformation Based Tagging
  - Statistical Tagging (HMM Part-of-Speech Tagging)

How hard is tagging?

- Most words in English are unambiguous. They have only a single tag.
- But many of most common words are ambiguous:
  - can/verb can/auxiliary can/noun



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%)

Tag ambiguity for word types in Brown and WSJ, Penn Treebank (45-tag) tagging.

How hard is tagging?

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

- Most Frequent Tag Baseline achieves an accuracy of 92% for WSJ corpus.

- The state of the art in part-of-speech tagging achieves an accuracy more than 97% for WSJ corpus.
- Some taggers can perform 99% percent.

Rule-Based Part-of-Speech Tagging

- The rule-based approach uses handcrafted sets of rules to tag input sentence.
- There are two stages in rule-based taggers:
  - **First Stage**: Uses a dictionary to assign each word a list of potential parts-of-speech.
  - **Second Stage**: Uses a large list of handcrafted rules to window down this list to a single part-of-speech for each word.
- The ENGTWOL is a rule-based tagger
  - In the first stage, uses a two-level lexicon transducer
  - In the second stage, uses hand-crafted rules (about 1100 rules).
    - Rule-1: if (the previous tag is an article)
      then eliminate all verb tags
    - Rule-2: if (the next tag is verb)
      then eliminate all verb tags

Rule-Based Part-of-Speech Tagging: Example

- Example: He had a fly.

- The first stage:

– he	**he/pronoun**	
– had	**have/verbpast**	have/auxliarypast
– a	**a/article**	
– fly	**fly/verb**	fly/noun

- The second stage:

apply rule: if (the previous tag is an article)
then eliminate all verb tags

– he	**he/pronoun**	
– had	**have/verbpast**	have/auxliarypast
– a	**a/article**	
– fly	fly/verb	**fly/noun**

Transformation-Based Tagging

- Transformation-based tagging is also known as **Brill Tagging**.
- **Brill Tagging uses transformation rules** and rules are learned from a tagged corpus.
- Then these learned rules are used in tagging.

- Before the rules are applied, the tagger labels every word with its most likely tag.
  - We get these most likely tags from a tagged corpus.

Transformation-Based Tagging: Example

- Example:
  - He is expected to race tomorrow
  - he/PRN is/VBZ expected/VBN to/TO race/NN tomorrow/NN
- After selecting most-likely tags, we apply transformation rules.
  - Change NN to VB when the previous tag is TO
  - This rule converts **race/NN** into **race/VB**
- This may not work for every case
  - ….. According to race

Transformation-Based Tagging

How Transformation Rules are Learned?

- We assume that we have a tagged corpus.

- Brill Tagger algorithm has three major steps.
  - **Tag the corpus with the most likely tag for each (unigram model)**
  - **Choose a transformation that deterministically replaces an existing tag with a new tag such that the resulting tagged training corpus has the lowest error rate out of all transformations.**
  - **Apply the transformation to the training corpus.**
- These steps are repeated until a stopping criterion is reached.
- The result (which will be our tagger) will be:
  - First tags using most-likely tags
  - Then apply the learned transformations in the learning order.

Transformation-Based Tagging

Transformation Rules

- **A transformation rule is selected from a small set of templates.**

Change **tag a** to **tag b** when

- The preceding (following) word is tagged z.
- The word two before (after) is tagged z.
- One of two preceding (following) words is tagged z.
- One of three preceding (following) words is tagged z.
- The preceding word is tagged z and the following word is tagged w.
- The preceding (following) word is tagged z and the word two before (after) is tagged w.

HMM Part-of-Speech Tagging

HMM Part-of-Speech Tagging

Markov Chains

- **A Markov chain is a model that tells us something about the probabilities of sequences of states** (random variables).
  - 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 (Markov assumption).
  - All states before the current state have no impact on the future except via the current state.

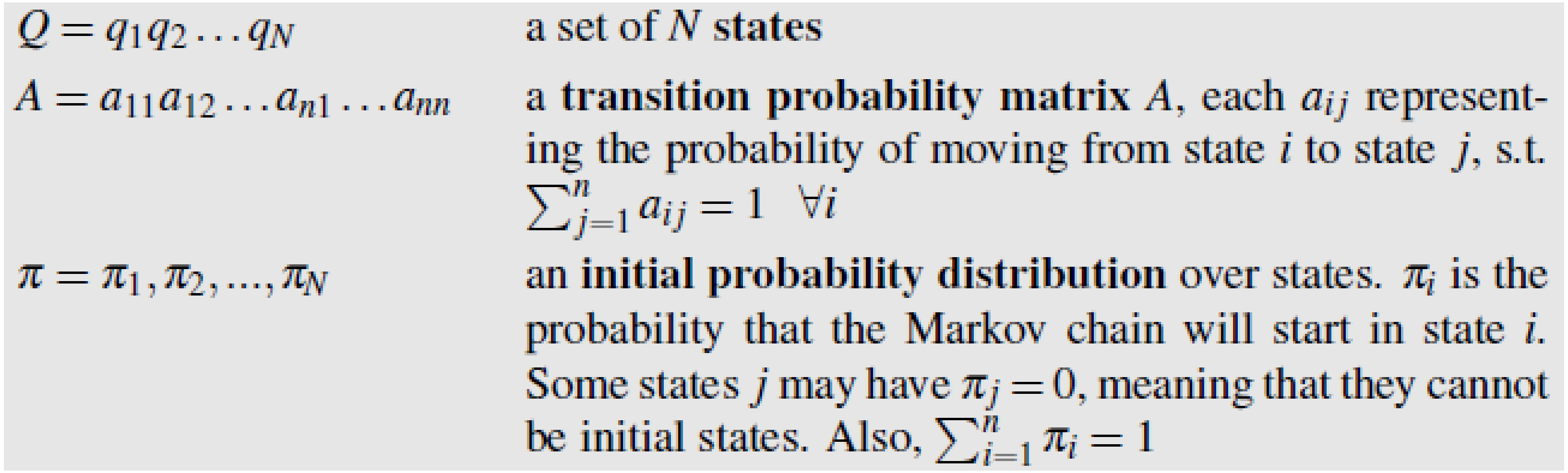
- A Markov model embodies **Markov assumption** on the probabilities of the sequence $q_1 \ldots q_{i-1}\, q_i$:

Markov Assumption: $$P(q_i \mid q_1 \ldots q_{i-1}) = P(q_i \mid q_{i-1})$$

HMM Part-of-Speech Tagging

Markov Chains

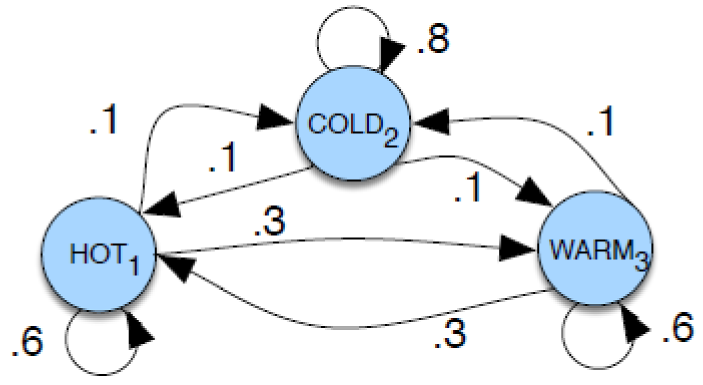
- A Markov chain is specified by the following components:



$Q = q_1 q_2 \ldots q_N$	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, $\sum_{i=1}^{n} \pi_i = 1$

HMM Part-of-Speech Tagging
Markov Chains

- ***A Markov chain for weather transitions***:



- ***A start distribution π is required***.
  - setting π = [0.1, 0.7, 0.2] would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

- Probability of the sequence: cold hot hot warm
  - P(cold hot hot warm) = π_2 * P(hot|cold) * P(hot|hot) * P(warm|hot)
    = 0.7 * 0.1 * 0.6 * 0.3

HMM Part-of-Speech Tagging

Hidden Markov Model

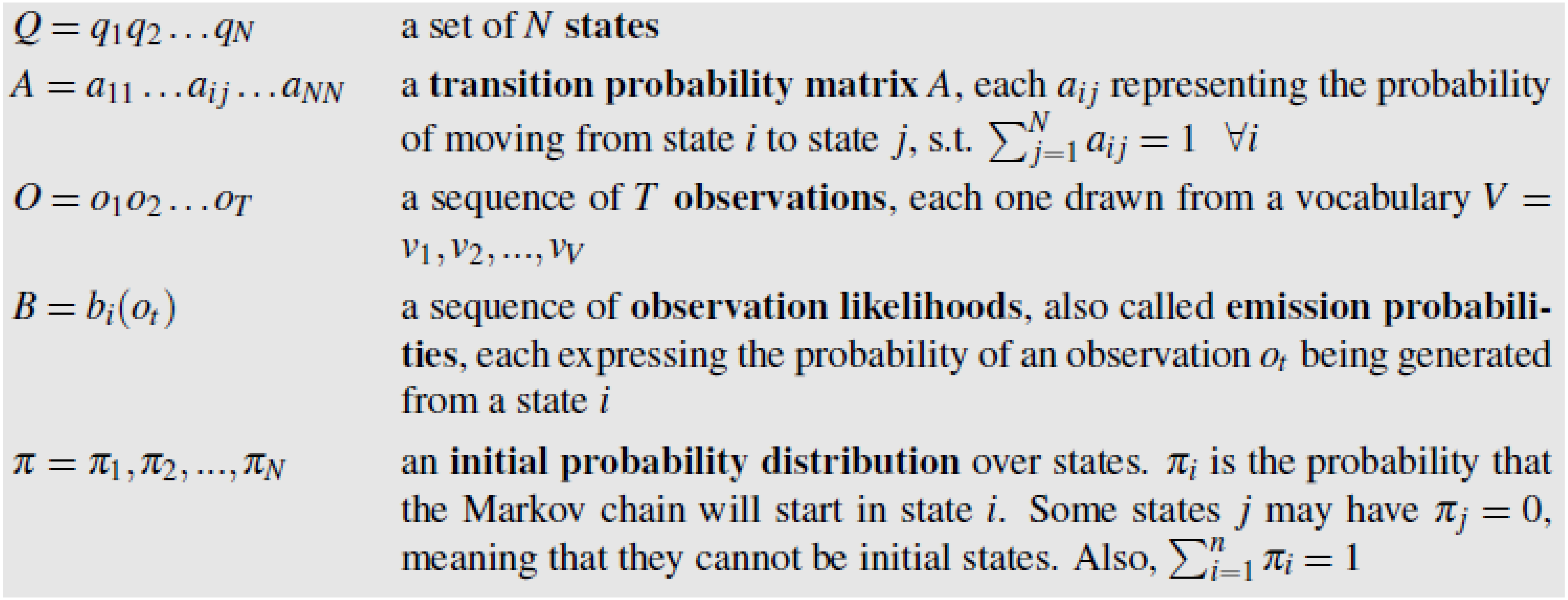
- **Markov chain** is useful to compute a probability for a sequence of **observable events**.
- In many cases, the events we are interested in are **hidden events**:
  - We don't observe **hidden events** directly.
  - For example we don't normally observe part-of-speech tags in a text. Rather, we see 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 our probabilistic model.

HMM Part-of-Speech Tagging

Hidden Markov Model

- A HMM is specified by the following components:



$Q = q_1 q_2 \ldots q_N$	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 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, $\sum_{i=1}^{n} \pi_i = 1$

HMM Part-of-Speech Tagging

First-Order Hidden Markov Model

- A first-order hidden Markov model uses two simplifying assumptions:

1. As with a first-order Markov chain, the probability of a particular state depends only on the previous state:
   - **Markov Assumption:** $\mathbf{P(q_i \mid q_1 \ldots q_{i-1}) = P(q_i \mid q_{i-1})}$

2. Probability of an output observation o_i depends only on the state that produced the observation q_i and not on any other states or any other observations:
   - **Output Independence:** $\mathbf{P(o_i \mid q_1 \ldots q_i \ldots q_n, o_1 \ldots o_i \ldots o_n) = P(o_i \mid q_i)}$

First-Order HMM for Part-of-Speech Tagging

States: Set of part-of-speech tags.

Transition Probabilities: Tag transition probabilities.

- A **tag transition probability** **$P(tag_b \mid tag_a)$** represents **the probability of a tag** **tag_b** **occurring given the previous tag** **tag_a**.
- **$P(tag_b \mid tag_a) = \frac{\text{count}(taga\ tagb)}{\text{count}(taga)}$** Example: $P(VB \mid MD) = \frac{\text{count}(MD\ VB)}{\text{count}(MD)}$

Observations: Words (Vocabulary)

Observation Likelihoods (Emission Probabilities):Emission Probabilities P(word|tag)

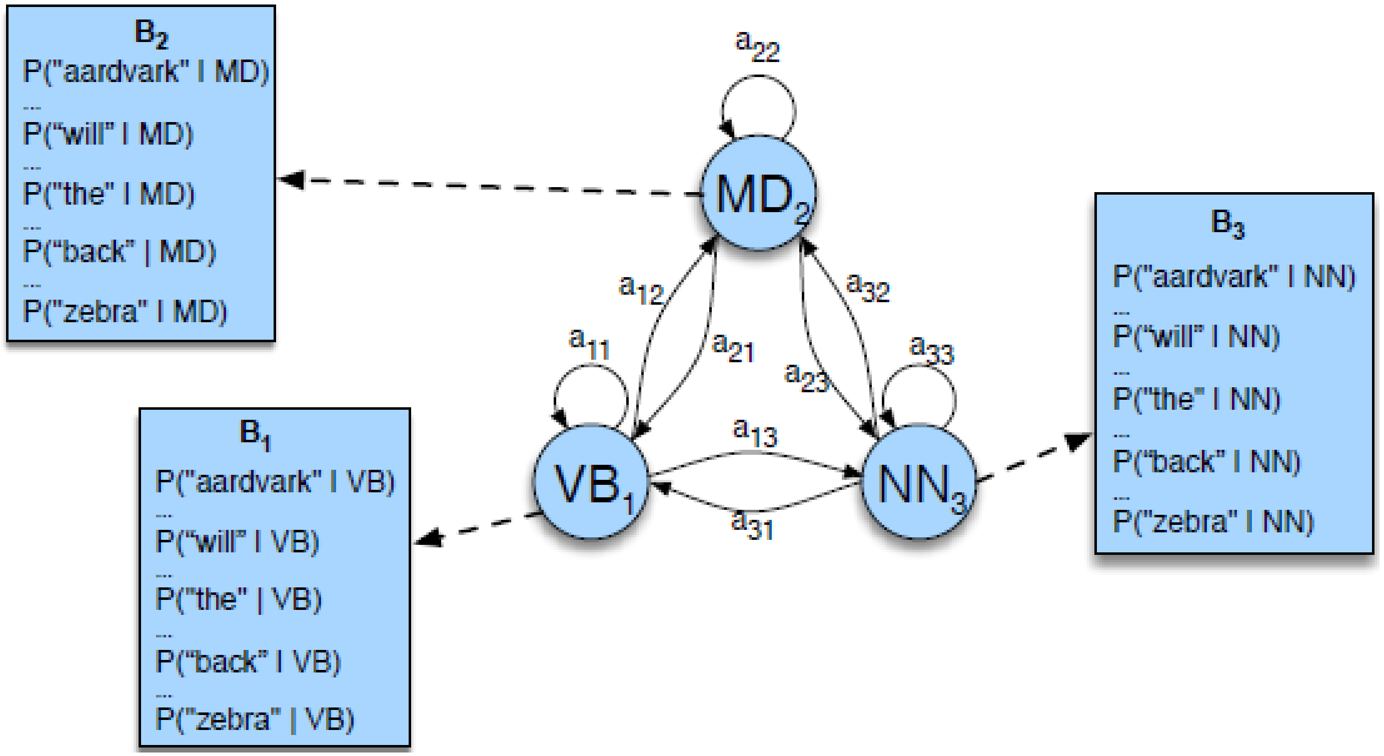
- A **emission probability** **P(word | tag)** represents **probability of** **tag** **producing** **word**.
- **$P(word \mid tag) = \frac{\text{count}(tag, word)}{\text{count}(tag)}$** Example: $P(will \mid MD) = \frac{\text{count}(MD, will)}{\text{count}(MD)}$

Initial Probability Distribution: First Tag Probabilities P(tag |<s>) in sentences.

- **$P(tag \mid <s>) = \frac{\text{count}(<s>\ tag)}{\text{count}(<s>)}$**

First-Order HMM for POS Tagging: Example

- The **A transition probabilities**, and **B observation likelihoods** (emission probabilities) of the HMM are illustrated for three states in an HMM part-of-speech tagger; the full tagger would have one state for each tag.



HMM Tagging as Decoding

- For an HMM that contains hidden variables, **task of determining hidden variables sequence corresponding to sequence of observations** is called **decoding**.

Decoding:

Given as input an HMM λ = (TransProbs, ObsLikelihoods) and a sequence of observations $O = o_1,\ldots,o_n$, find the most probable sequence of states $Q = q_1,\ldots,q_n$.

- **For part-of-speech tagging, we will find the most probable sequence of tags $t_1,\ldots,t_n$ (hidden variables) for a given sequence of words $w_1,\ldots,w_n$ (observations).**

HMM Tagging as Decoding

- For part-of-speech tagging, we will find most probable tag sequence **$T=t_1,\ldots,t_n$** for a given sequence of n words **$W=w_1,\ldots,w_n$** .

- **The most probable tag sequence $\hat{T}$ (among possible tag sequences τ) is:**

$$\hat{T} = \underset{T\in\tau}{\operatorname{argmax}}\ P(T|W)$$

- By Bayes rule:

$$\hat{T} = \underset{T\in\tau}{\operatorname{argmax}}\ \frac{P(W|T)\,P(T)}{P(W)}$$

- Since P(W) is same for all tag sequences.

$$\hat{T} = \underset{T\in\tau}{\operatorname{argmax}}\ P(W|T)\,P(T)$$

HMM Tagging as Decoding

- HMM taggers make two simplifying assumptions.

- **The first is that the probability of a word appearing depends only on its own tag and is independent of neighboring words and tags:**

$$\mathbf{P(W|T) = P(w_1 \ldots w_n | t_1 \ldots} \boldsymbol{t_n}\mathbf{)} \approx \prod_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{P(w_i|t_i)}$$

- **The second assumption, the bigram assumption (first-order HMM), is that the probability of a tag is dependent only on the previous tag, rather than the entire tag sequence:**

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

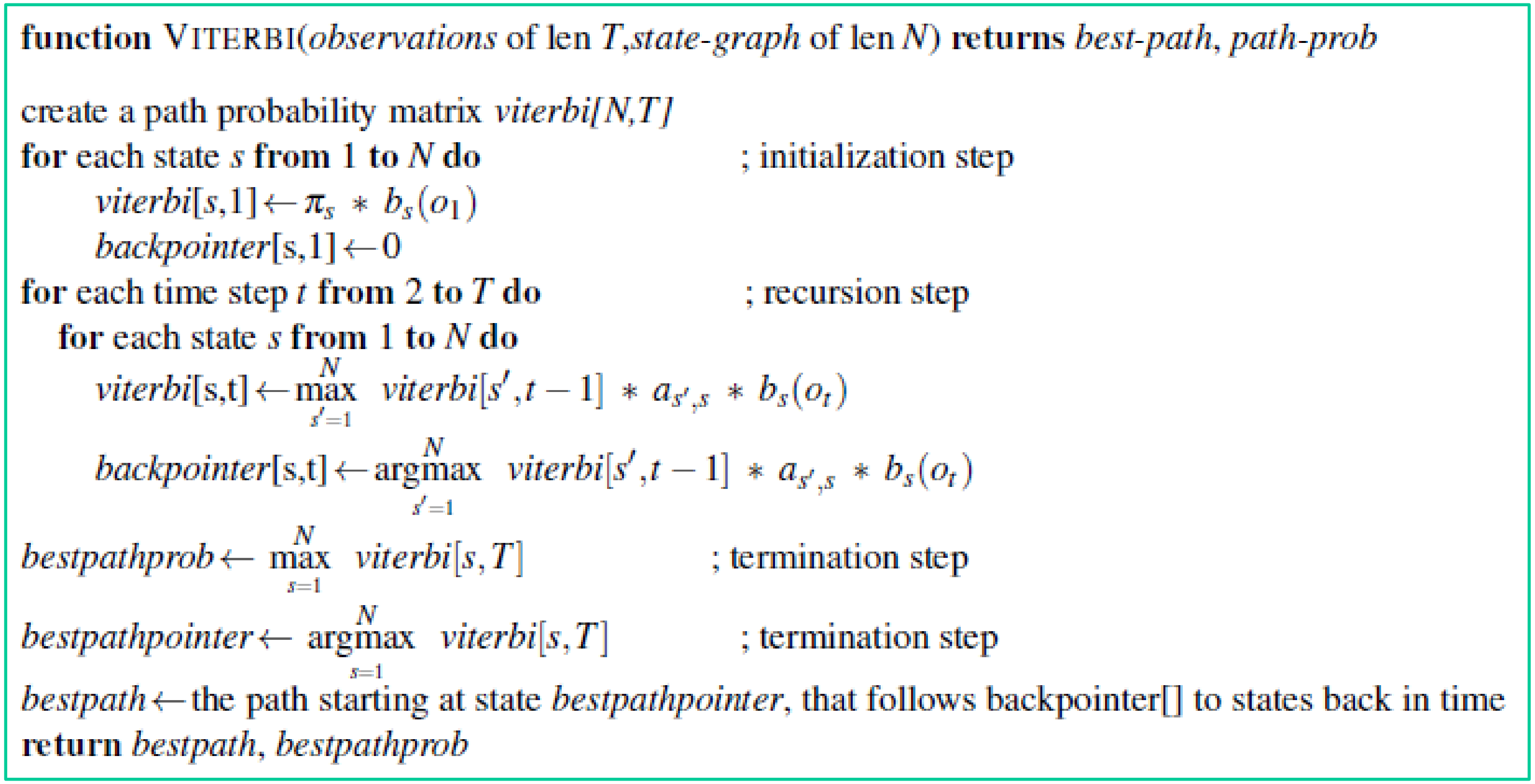
HMM Tagging as Decoding

- Plugging the simplifying assumptions results in the following equation for **the most probable tag sequence from a bigram tagger (first-order HMM)**:

$$\hat{\mathbf{T}} = \underset{\mathbf{T} \in \boldsymbol{\tau}}{\mathbf{argmax}}\ \mathbf{P(T|W)} = \underset{\mathbf{T} \in \boldsymbol{\tau}}{\mathbf{argmax}} \prod_{\mathbf{i=1}}^{\mathbf{n}} \underbrace{\mathbf{P(w_i|t_i)}}_{\text{emission probability}}\ \underbrace{\mathbf{P(t_i|t_{i-1})}}_{\text{transition probability}}$$

Viterbi Algorithm

- The decoding algorithm for HMMs is the Viterbi algorithm.
- Viterbi algorithm finds the optimal sequence of tags.
  - Given an observation sequence and an HMM λ = (A, B) the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.



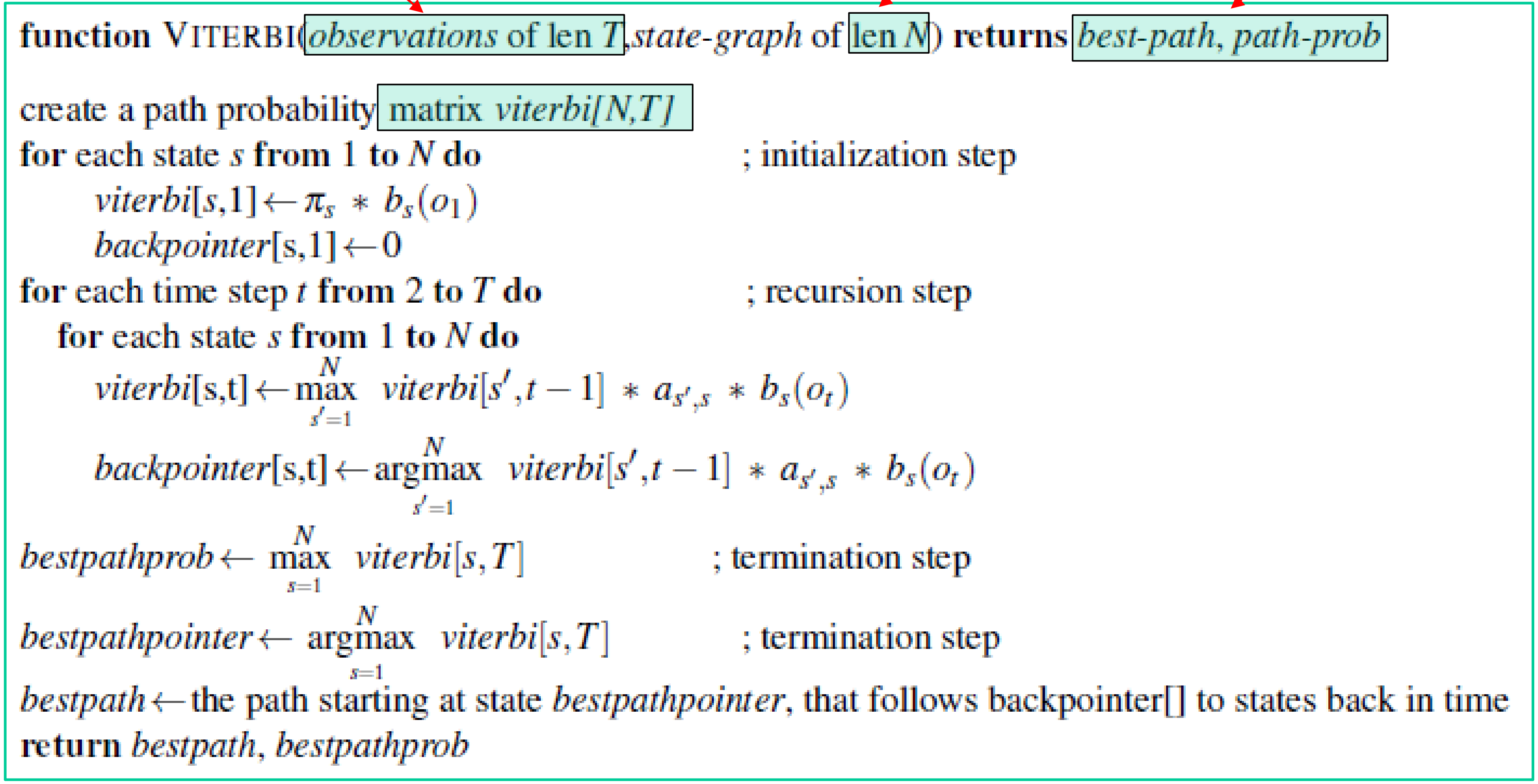
```
function VITERBI(observations of len T, state-graph of len N) returns best-path, path-prob

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 for POS Tagging

word sequence $o_1,\dots,o_T$

number of tags

most probable tag sequence



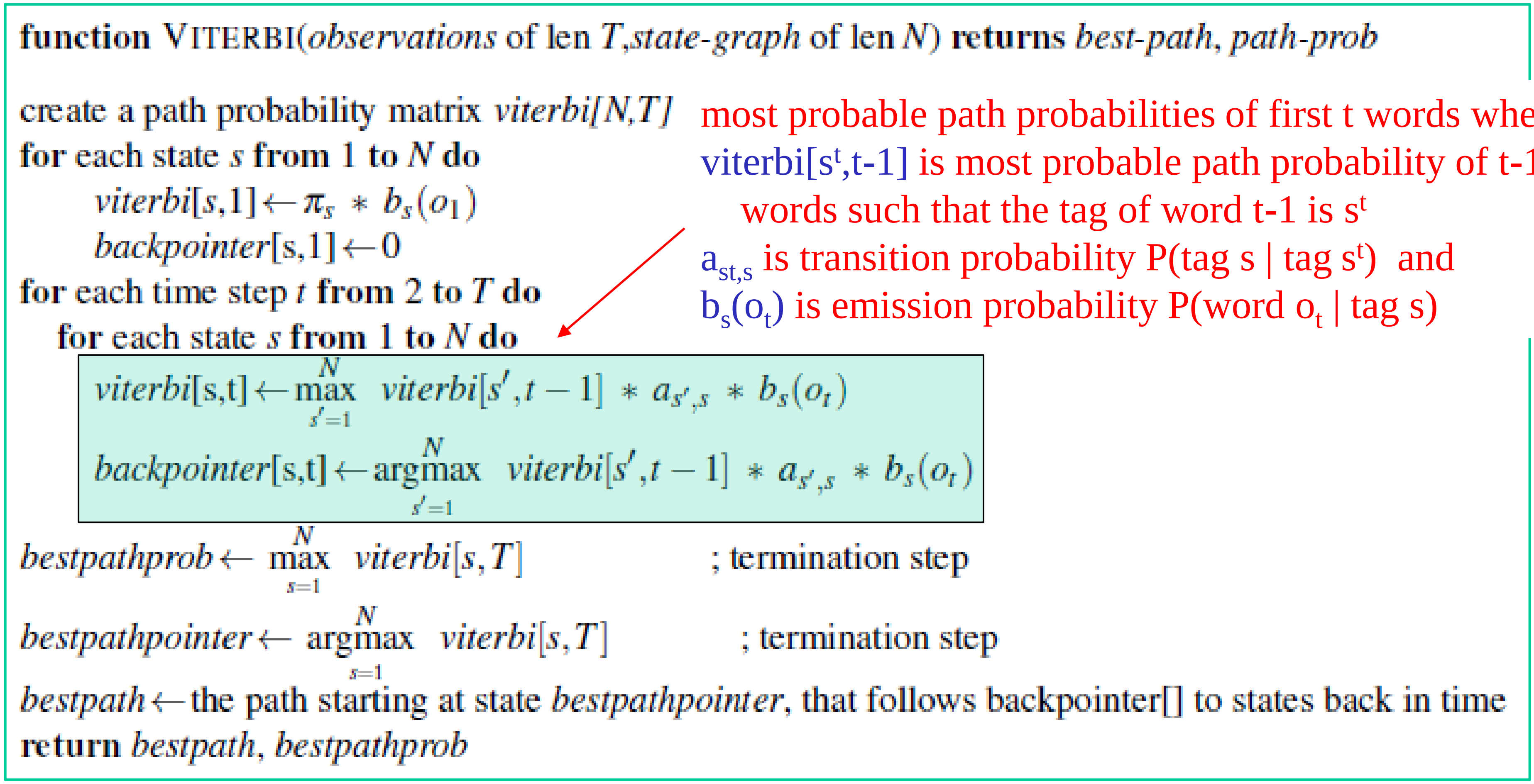
```
function VITERBI(observations of len T, state-graph of len N) returns best-path, path-prob

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 ← the path starting at state bestpathpointer, that follows backpointer[] to states back in time
return bestpath, bestpathprob
```

Viterbi Algorithm for POS Tagging

```
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
    $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
    $backpointer[s,1] \leftarrow 0$
for each time step t from 2 to T do
  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
```

most probable path probabilities of first word o_1 where
π_s is first tag probability of tag s and
$b_s(o_1)$ is emission probability P(word o_1 | tag s)

Viterbi Algorithm for POS Tagging



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

most probable path probabilities of first t words where

viterbi[s^t,t-1] is most probable path probability of t-1 words such that the tag of word t-1 is s^t

$a_{st,s}$ is transition probability P(tag s | tag s^t) and

$b_s(o_t)$ is emission probability P(word o_t | tag s)

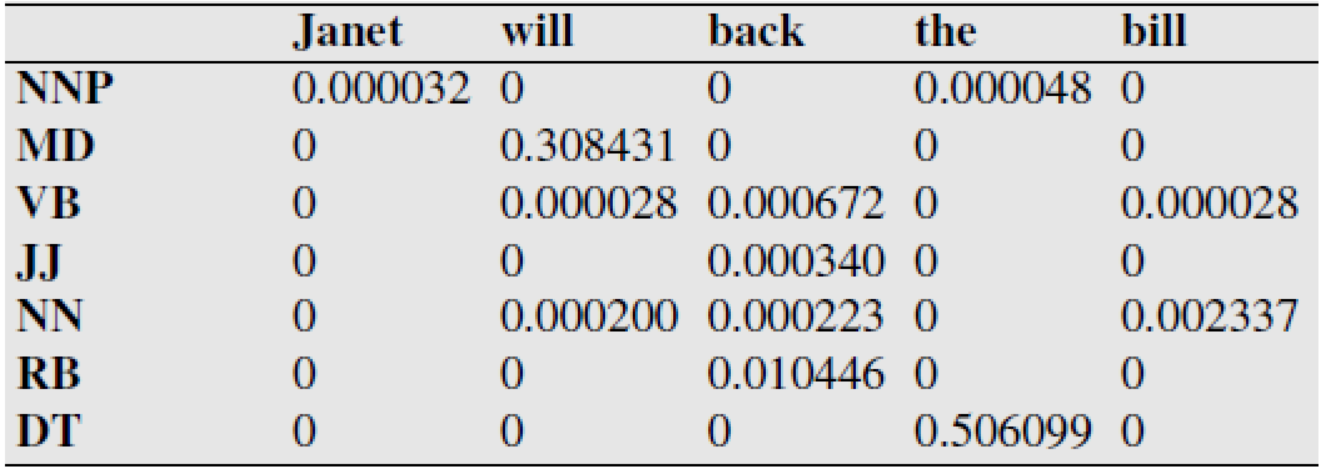
Viterbi Algorithm for POS Tagging

```
**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**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$
$bestpath \leftarrow$ the path starting at state $bestpathpointer$, that follows backpointer[] to states back in time
**return** bestpath, bestpathprob
```

most probable path probability of T words

Viterbi Algorithm for POS Tagging
Example: Janet will back the bill

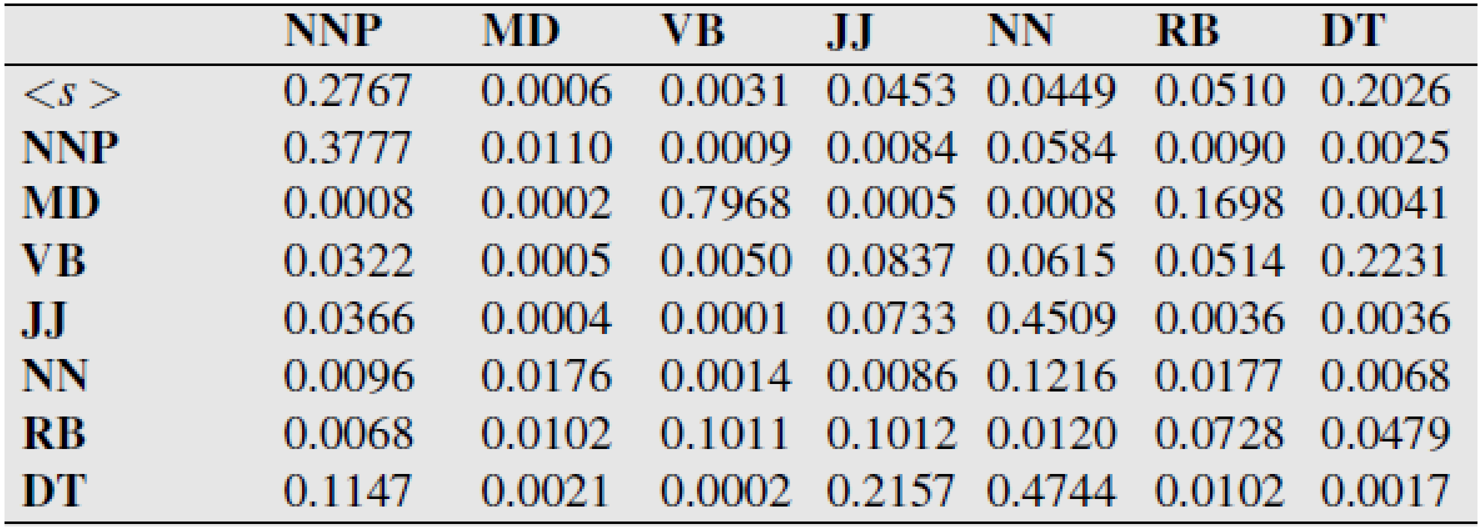
- **Observation likelihoods B** computed from the WSJ corpus without smoothing



	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

Viterbi Algorithm for POS Tagging
Example: Janet will back the bill

- The **A transition probabilities P($t_i|t_{i-1}$)** computed from the WSJ corpus without smoothing.

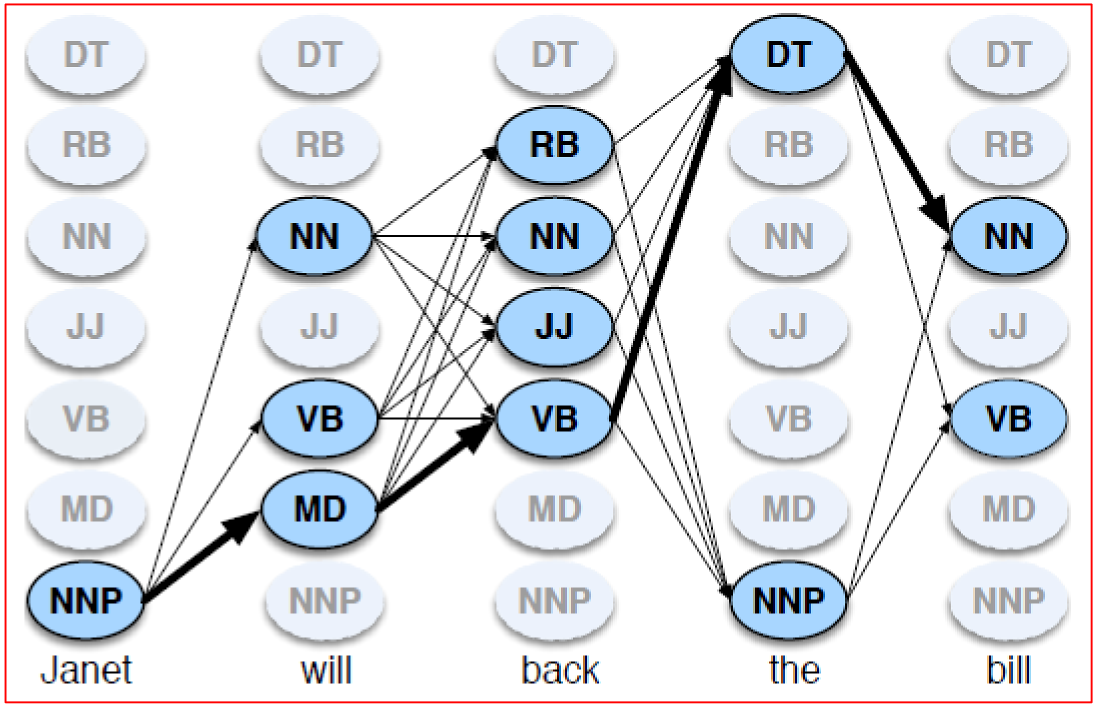


	NNP	MD	VB	JJ	NN	RB	DT
$<s>$	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176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

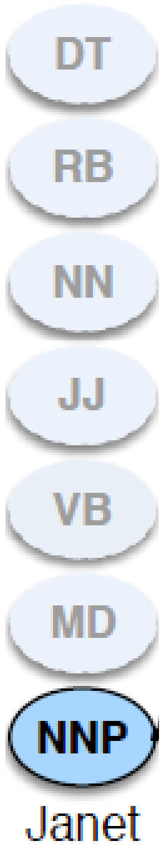
Viterbi Algorithm for POS Tagging Example: Janet will back the bill

Sketch of Viterbi matrix for **Janet will back the bill**,

- possible tags for each word and highlighting the path corresponding to the correct tag sequence through the hidden states. States (parts-of-speech) which have a zero probability of generating a particular word according to the B matrix (such as the probability that a determiner DT will be realized as Janet) are greyed out..



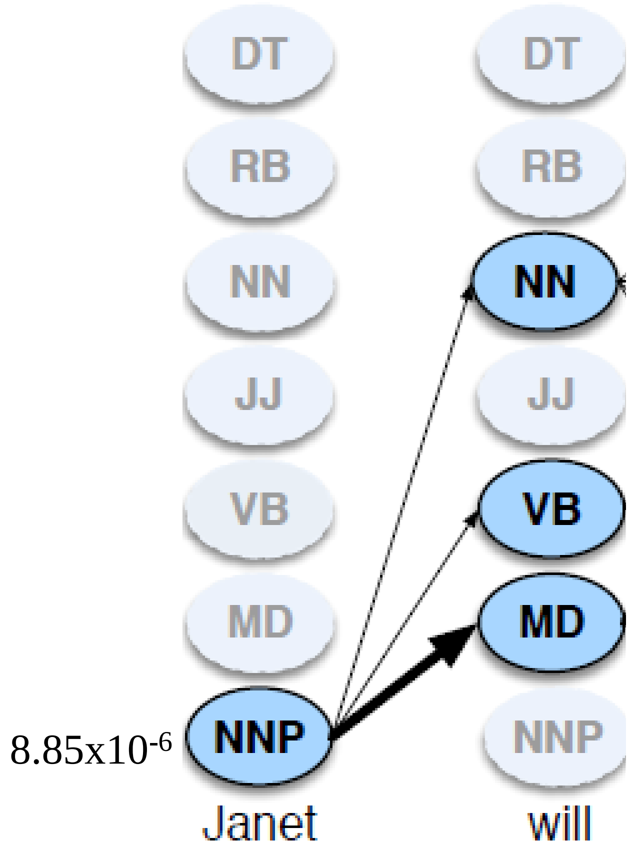
Viterbi Algorithm for POS Tagging
Example: Janet will back the bill



- All other values in the first column will be zero because observation likelihoods for all other tags (such as P(Janet|MD), P(Janet|VB), …) are zero.

Viterbi[NNP,Janet] =
P(NNP|<s>)*P(Janet|NNP) = 0.2767*0.000032 = 0.00000885 = $8.85\text{x}10^{-6}$

Viterbi Algorithm for POS Tagging
Example: Janet will back the bill

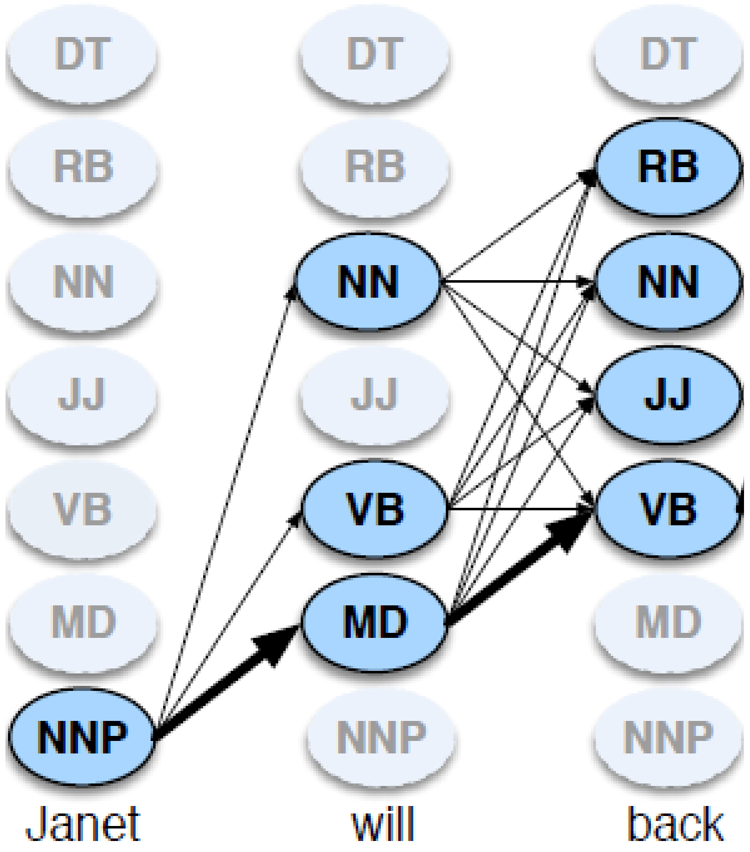


Viterbi[NN,will] = Viterbi[NNP,Janet]*P(NN|NNP)*P(will|NN)
$= 8.85\times10^{-6}*0.0584*0.0002 = 1.03368\times10^{-9}$

Viterbi[VB,will] = Viterbi[NNP,Janet]*P(VB|NNP)*P(will|VB)
$= 8.85\times10^{-6}*0.0009*0.000028 = 0.22302\times10^{-12}$

Viterbi[MD,will] = Viterbi[NNP,Janet]*P(MD|NNP)*P(will|MD)
$= 8.85\times10^{-6}*0.011*0.308431 = 0.203\times10^{-6}$

Viterbi Algorithm for POS Tagging
Example: Janet will back the bill



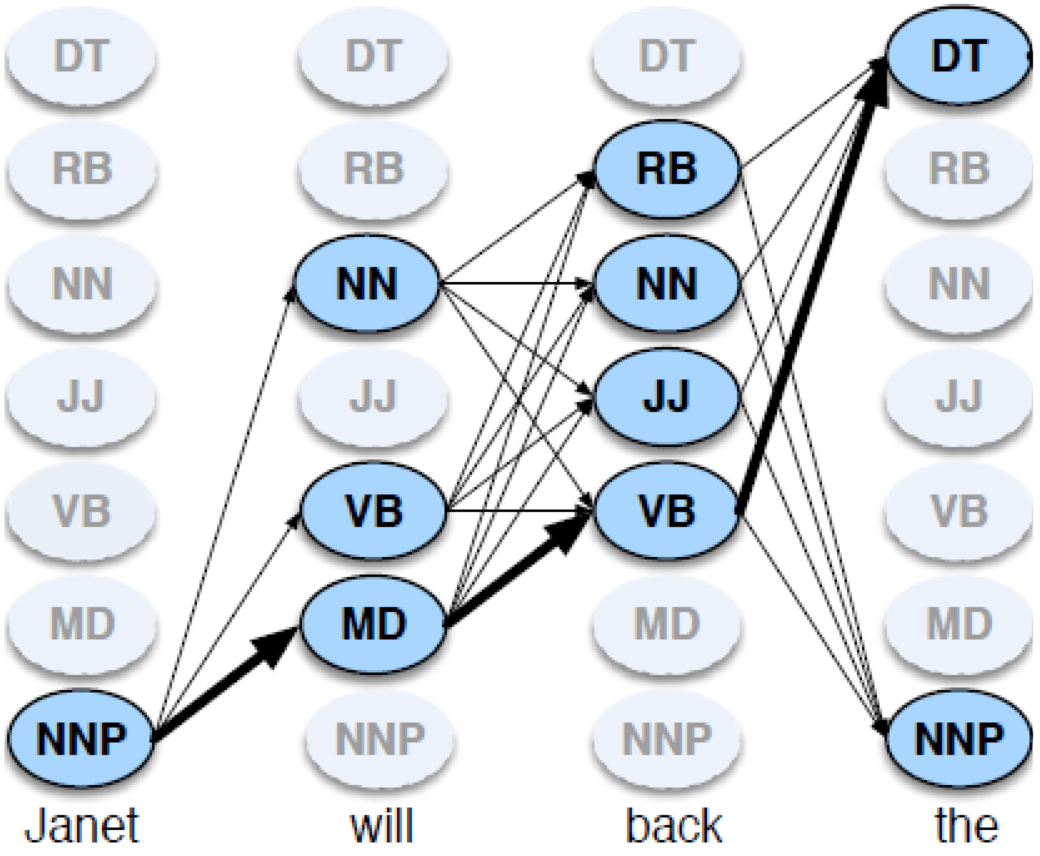
Viterbi[RB,back]=max({Viterbi[NN,will]*P(RB|NN)*P(back|RB),
Viterbi[VB,will]*P(RB|VB)*P(back|RB),
Viterbi[MD,will]*P(RB|MD)*P(back|RB)})

Viterbi[NN,back]=max({Viterbi[NN,will]*P(NN|NN)*P(back|NN),
Viterbi[VB,will]*P(NN|VB)*P(back|NN),
Viterbi[MD,will]*P(NN|MD)*P(back|NN)})

Viterbi[JJ,back]=max({Viterbi[NN,will]*P(JJ|NN)*P(back|JJ),
Viterbi[VB,will]*P(JJ|VB)*P(back|JJ),
Viterbi[MD,will]*P(JJ|MD)*P(back|JJ)})

Viterbi[VB,back]=max({Viterbi[NN,will]*P(VB|NN)*P(back|VB),
Viterbi[VB,will]*P(VB|VB)*P(back|VB),
Viterbi[MD,will]*P(VB|MD)*P(back|VB)})

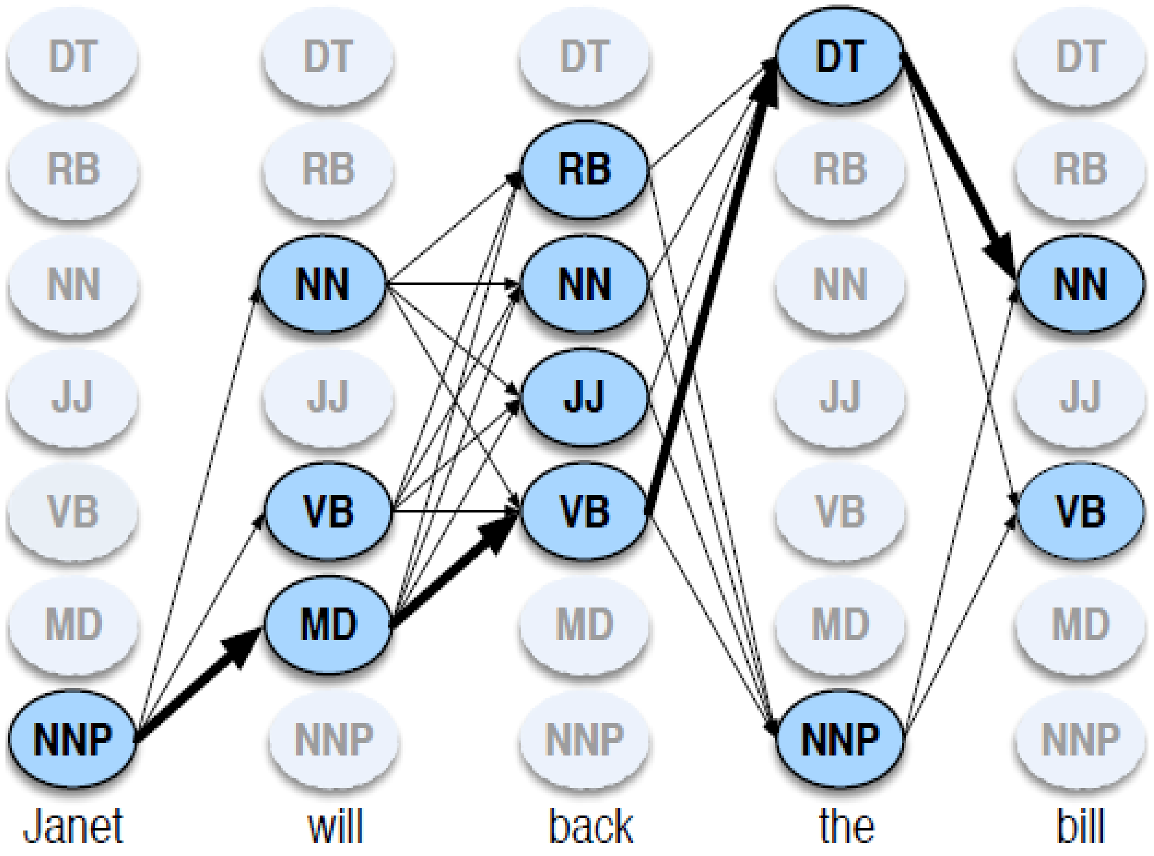
Viterbi Algorithm for POS Tagging
Example: Janet will back the bill



Viterbi[DT,the]=
max({Viterbi[RB,back]*P(DT|RB)*P(the|DT),
Viterbi[NN,back]*P(DT|NN)*P(the|DT),
Viterbi[JJ,back]*P(DT|JJ)*P(the|DT),
Viterbi[VB,back]*P(DT|VB)*P(the|DT)})

Viterbi[NNP,the]=
max({Viterbi[RB,back]*P(NNP|RB)*P(the|NNP),
Viterbi[NN,back]*P(NNP|NN)*P(the|NNP),
Viterbi[JJ,back]*P(NNP|JJ)*P(the|NNP),
Viterbi[VB,back]*P(NNP|VB)*P(the|NNP)})

Viterbi Algorithm for POS Tagging
Example: Janet will back the bill



Viterbi[NN,bill]=
max({Viterbi[DT,the]*P(NN|DT)*P(bill|NN),
Viterbi[NNP,the]*P(NN|NNP)*P(bill|NN)})

Viterbi[VB,bill]=
max({Viterbi[DT,the]*P(VB|DT)*P(bill|VB),
Viterbi[NNP,the]*P(VB|NNP)*P(bill|VB)})

HMM Part-of-Speech Tagging
In Practice

- Practical HMM taggers may use **higher-order (such as tri-gram) models** instead of the first-order HMM (bi-gram) model.

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

- When the number of states grows very large for trigram taggers, Viterbi algorithm can be slow.
  - The complexity of Viterbi algorithm $O(N^2T)$.
  - One solution is the usage of **Beam Search** where only few best states are propagated forward instead of all non-zero states at each time step.

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

Part-of-Speech Tagging for Other Languages

- Highly inflectional languages have much more information than English coded in word morphology, like case (nominative, accusative, …) or gender.
  - Because this information is important for part-of-speech taggers for morphologically rich languages, they need to label words with case and gender information.
- Tagsets for morphologically rich languages are therefore sequences of morphological tags rather than a single primitive tag.
- For Turkish, some tags can be:
  - Noun+A3sg+Pnon+Gen
  - Noun+A3sg+P2sg+Nom
  - Noun+A3sg+Pnon+Nom

Part-of-Speech Tagging: Summary

- Languages generally have a small set of **closed class** words that are highly frequent, ambiguous, and **open-class** words like nouns, verbs, adjectives.
  - Various part-of-speech tagsets exist for English, of between 40 and 200 tags.
  - For Turkish, the size of a tagset can be more than 1000.
- **Part-of-speech tagging** is the process of assigning a part-of-speech label to each of a sequence of words.
- The **probabilities in HMM taggers** are estimated by maximum likelihood estimation on tag-labeled training corpora.
  - **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 (MEMM) taggers*** are **another types of taggers** that train logistic regression models to pick the best tag given a word, its context and its previous tags using **feature templates**.