Unit 3
Part of Speech Tagging
Types of PoS Tagging, Hidden Markup Model

Natural Language Processing (NLP)
MDS 555

Objective

- Types of PoS Tagger
  - Rule Based PoSTagging
  - Stochastic PoS Tagging
  - Transformation based Tagging
- Hidden Markup Model

Reference / Reading

- **Chapter 8**

  Speech and Language Processing. Daniel Jurafsky & James H. Martin

  – https://web.stanford.edu/~jurafsky/slp3/old_oct19/8.pdf

What is PoS?

- A category to which a word is assigned in accordance with its syntactic functions
- The role a word plays in a sentence denotes what part of speech it belongs to
- In English the main parts of speech are
  - noun, pronoun, adjective, determiner, verb, adverb, preposition, conjunction, and interjection

Part of Speech

- Noun
- Adjective
- Adverb
- Verb
- Preposition
- Pronoun
- Conjunctions
- Interjections

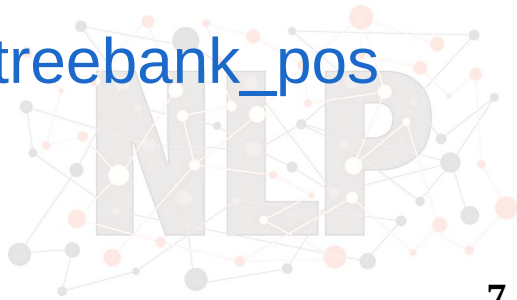
Part of Speech	Function	Examples
Nouns	Naming people, places, things & ideas/concepts	• Mike owns the fastest car • They decided to buy the house • I believe in free speech
Adjectives	Describing nouns and pronouns	• That was an exciting film • The boy in the red jumper is happy • This is the longest essay
Adverbs	Modifying verbs, adjectives & adverbs	• She walked slowly • They are extremely poor • The police arrived very quickly
Verbs	Expressing a physical action or state	• Federer plays tennis • The floods are serious • Take your time
Prepositions	Showing relationships of words and phrases	• The book is on the table • He went into the room • I'm against abortion
Pronouns	Replacing nouns	• John gave her a gift • Why did you push me? • Please look after yourself
Conjunctions	Joining words, phrases & clauses	• He's tall and slim • I'll wait until she arrives • Tom's sick so he's not here
Interjections	Showing strong emotions or feelings	• Wow, that's brilliant news! • Ouch, that really hurt • Hey! How are you doing?

PoS Tagsets

- There are many parts of speech tagsets
- Tag types
  - Coarse-grained
    - Noun, verb, adjective, …
  - Fine-grained
    - noun-proper-singular, noun-proper-plural, noun-common-mass, ..
    - verb-past, verb-present-3rd, verb-base, …
    - adjective-simple, adjective-comparative, ...

PoS Tagsets

- Brown tagset (87 tags)
  - Brown corpus
  - https://en.wikipedia.org/wiki/Brown_Corpus
- C5 tagset (61 tags)
- C7 tagset (146 tags!)
- Penn TreeBank (45 tags) – most used
  - A large annotated corpus of English tagset
  - https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
- UPenn TreeBank II - 36 tags



PoS Tag: Challenge

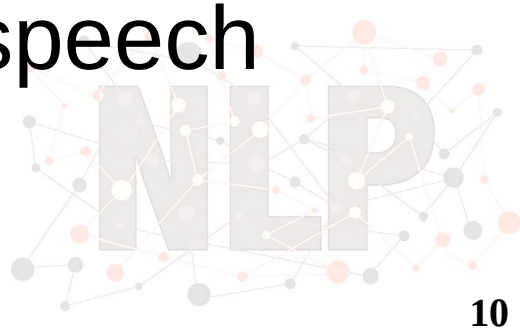
- Words often have more than one POS
- Ambiguity in POS tags
- Out of Vocabulary (OOV) in POS
- Complex grammatical structure of the language
- Lack of annotated dataset
- Inconsistencies in annotated dataset

Type of PoS Taggers

- There are different algorithms for tagging.
  - Rule Based Tagging
  - Transformation Based Tagging
  - Statistical Tagging (HMM Part-of-Speech Tagging)

Rule-Based POS 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



Rule-Based POS tagging

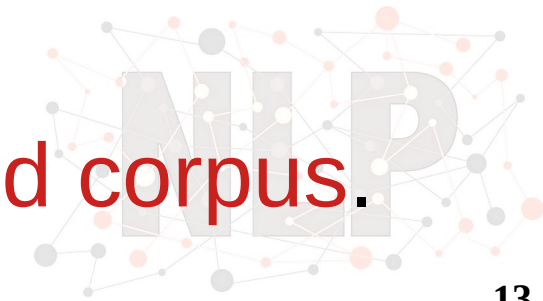
- The ENGTWOL is a rule-based tagger
  - In the first stage, uses a two-level lexicon transducer
  - In the second stage, uses hand-crafted rules (about 1100 rules).
    - Rule-1: if (the previous tag is an article)
      then eliminate all verb tags
    - Rule-2: if (the next tag is verb)
      then eliminate all verb tags

Rule-Based POS tagging

- Example: **He had a fly.**
- The fırst 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:
  - 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

Brill Tagger – How 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.

Brill Tagger – Transformation Rules?

- 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.

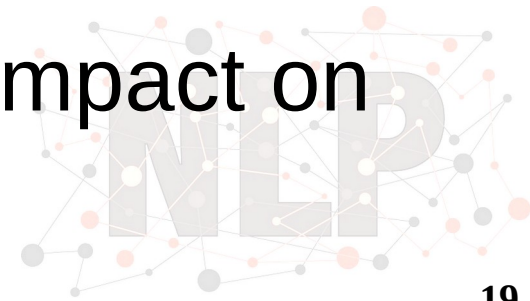
Methods of PoS Tagging

- Stochastic (Probabilistic) tagging
  - e.g., TNT [Brants, 2000]
    - Trigrams n Tags
    - Based on Markov model
  - Original Paper: https://dl.acm.org/doi/pdf/10.3115/974147.974178

Hidden Markov Model based PoS 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



Markov Chains

- Markov Assumption:
  - Consider a sequence of state variables q_1, q_2, …, q_i. 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 :
  -

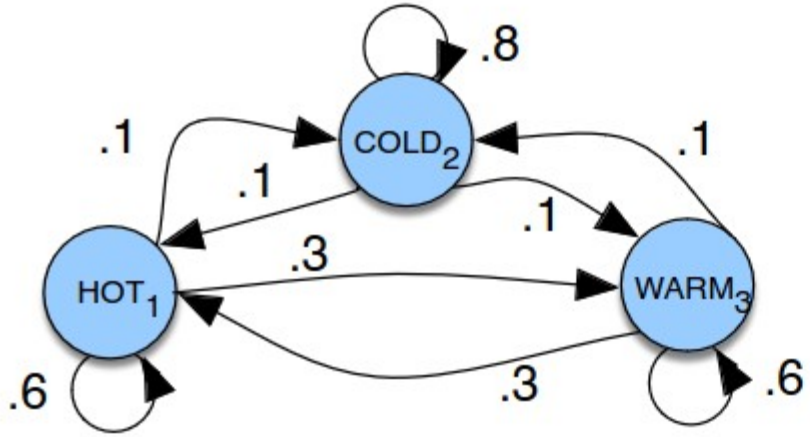
$$P(q_i \mid q_1 \ldots q_{i-1}) = P(q_i \mid q_{i-1})$$

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, $\sum_{i=1}^{n} \pi_i = 1$

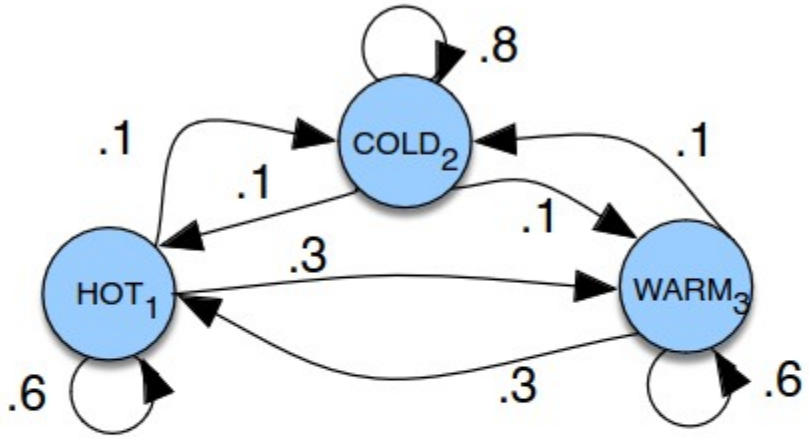
Markov Chains



- Markov chain for weather events
  - Vocabulary : HOT, COLD, and WARM
- States are represented as nodes
- Transitions, with their probabilities, as edge
- 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

Markov Chains

- Compute the probability of each of the following sequences:
  - hot hot hot hot
  - cold hot cold hot



- What does the difference in these probabilities tell you about a real-world weather fact encoded in Figure

Markov Chains

- 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.

Hidden Markov Model

- An 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 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, $\sum_{i=1}^{n} \pi_i = 1$

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

  2) Probability of an output observation o_i depends only on the state that produced the observation qi and not on any other states or any other observations:

  Output Independence: $P(o_i \mid q_1 \ldots q_i \ldots q_n, o_1 \ldots o_i \ldots o_n) = P(o_i \mid q_i)$

The components of an HMM tagger

- 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 (MD) like ***will*** are very likely to be followed by a verb in the base form (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

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

The components of an HMM tagger

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

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})} \qquad P(VB|MD) = \frac{C(MD,VB)}{C(MD)} = \frac{10471}{13124} = .80$$

- In HMM tagging, the probabilities are estimated by counting on a tagged training corpus.

The components of an HMM tagger

- 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

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

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

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

HMM tagger

- 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

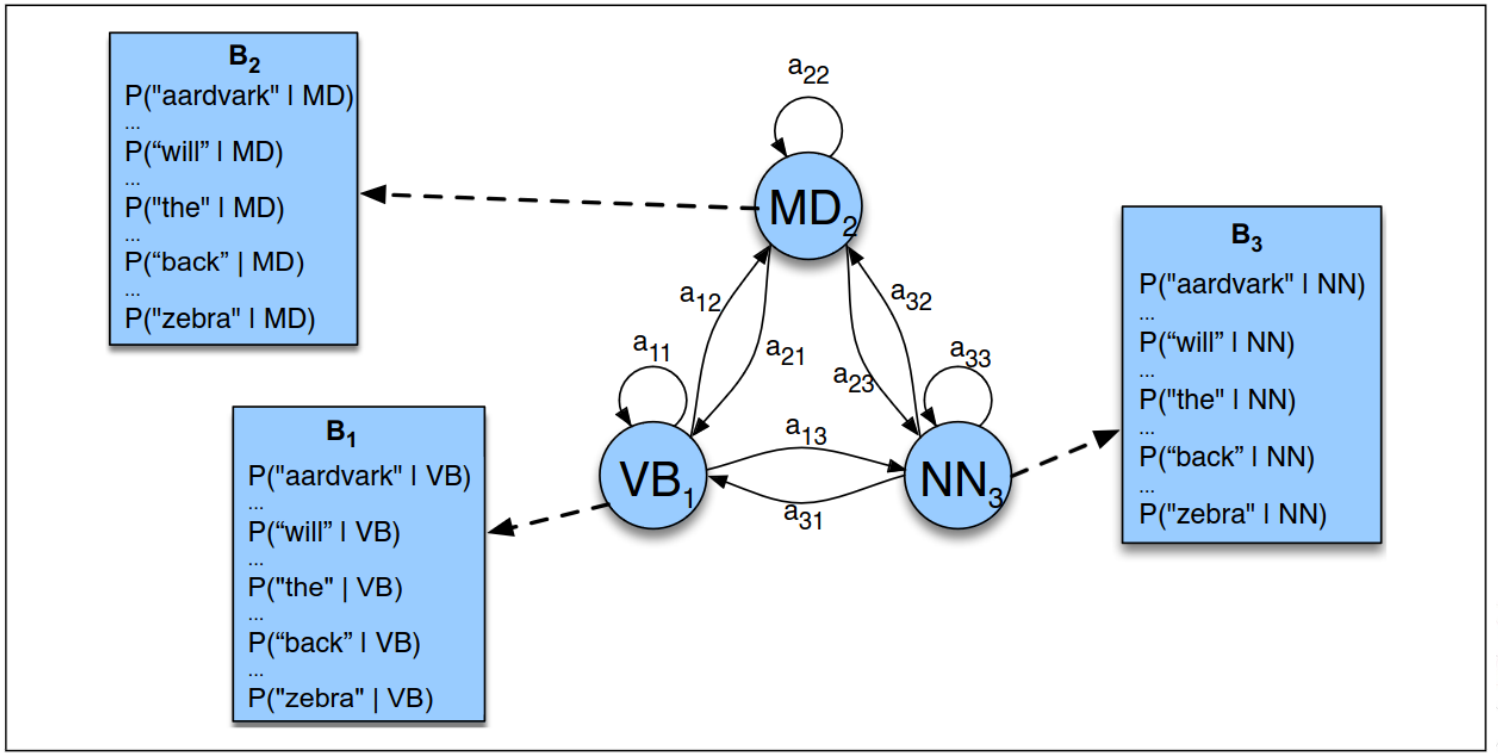


Figure 8.4 An illustration of the two parts of an HMM representation: the *A* transition probabilities used to compute the prior probability, and the *B* observation likelihoods that are associated with each state, one likelihood for each possible observation word.

HMM tagger

- 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.

$$\mathbf{P(tag_b \mid tag_a) = \frac{count(taga\ tagb)}{count(taga)}} \qquad \text{Example: } P(VB \mid MD) = \frac{count(MD\ VB)}{count(MD)}$$

- Observations: Words (Vocabulary)
  - Observation Likelihoods: Emission Probabilities P(word|tag)
  - A emission probability P(word | tag) represents probability of tag producing word.

$$\mathbf{P(word \mid tag) = \frac{count(tag, word)}{count(tag)}} \qquad \text{Example: } P(will \mid MD) = \frac{count(MD, will)}{count(MD)}$$

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

$$\mathbf{P(tag \mid <s>) = \frac{count(<s>\ tag)}{count(<s>)}}$$

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 $\lambda = (\text{TransProbs}, \text{ObsLikelihoods})$ and a sequence of observations $O = o_1,\ldots,o_T$, find the most probable sequence of states $Q = q_1,\ldots,q_T$.
- 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 - Decoding

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

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:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \tag{8.13}$$

The way we'll do this in the HMM is to use Bayes' rule to instead compute:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)} \tag{8.14}$$

Furthermore, we simplify Eq. 8.14 by dropping the denominator $P(w_1^n)$:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(w_1^n | t_1^n) P(t_1^n) \tag{8.15}$$

HMM - Decoding

- HMM taggers make two further simplifying assumption
  - The first is that 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 second assumption, the bigram assumption, is that the probability of a tag is dependent only on the previous tag, rather than the entire tag sequence;

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

HMM - Decoding

- Plugging the simplifying assumptions, results in the following equation for the most probable tag sequence from a bigram tagger:

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

- The two parts of above equation correspond neatly to the B emission probability and A transition probability

Viterbi Algorithm

- The decoding algorithm for HMMs is 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*

Figure 8.5 Viterbi algorithm for finding the optimal sequence of tags. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

Working of Viterbi Algorithm

Word Sequence O1, O2...

Number of tags

Most possible 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 $\leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Working of 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**

$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

$$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

$$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T] \quad \text{; termination step}$$

$$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] \quad \text{; termination step}$$

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

return *bestpath*, *bestpathprob*

most probable path probabilities of first word o_1 where,

PI is first tag probability of tag s and
$b_s(o_1)$ is emission probability P(word o_1 | tag s)

Working of 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 ← 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 tansition probability P(tag s | tag s_t) and
$b_s(o_t)$ is emission probability P(word o_t | tag s)

Working of 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 ← 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 - Example

- Let's tag the sentence Janet will back the bill

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

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

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

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

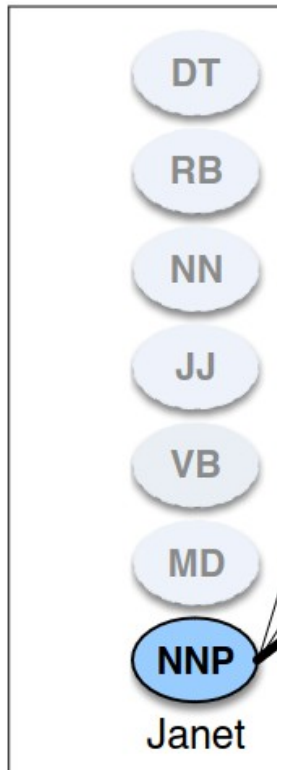
Viterbi Algorithm - Example

- Viterbi[NNP,Janet]

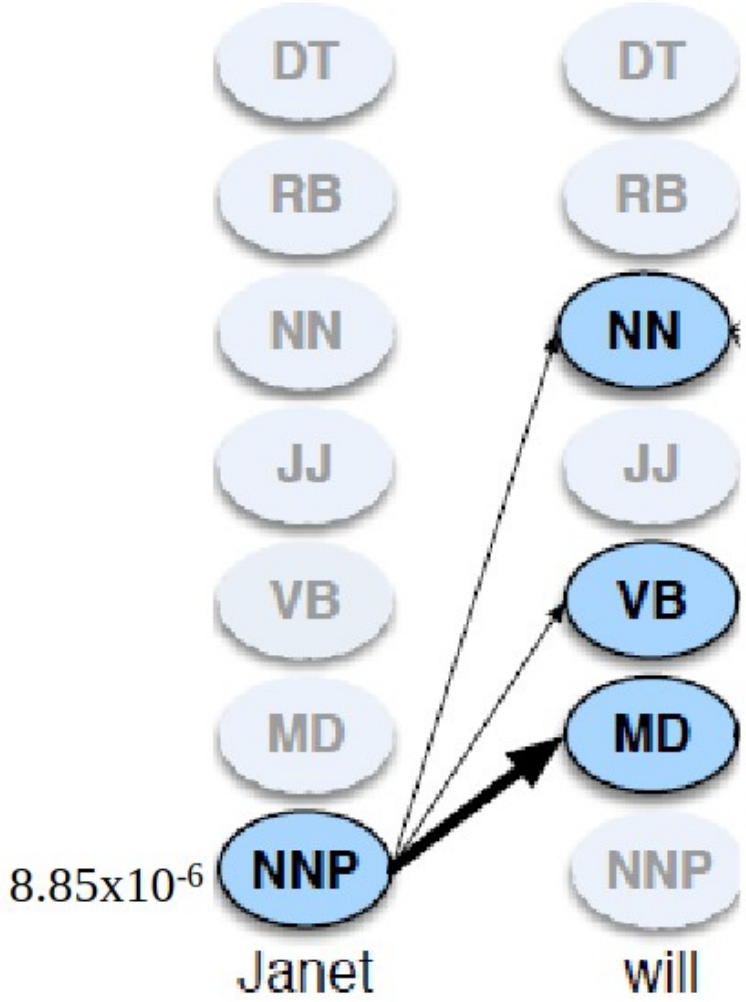
  = P(NNP|<s>)*P(Janet|NNP)

  = 0.2767*0.000032 = 0.00000885

  = 8.85x10-6



Viterbi Algorithm - Example

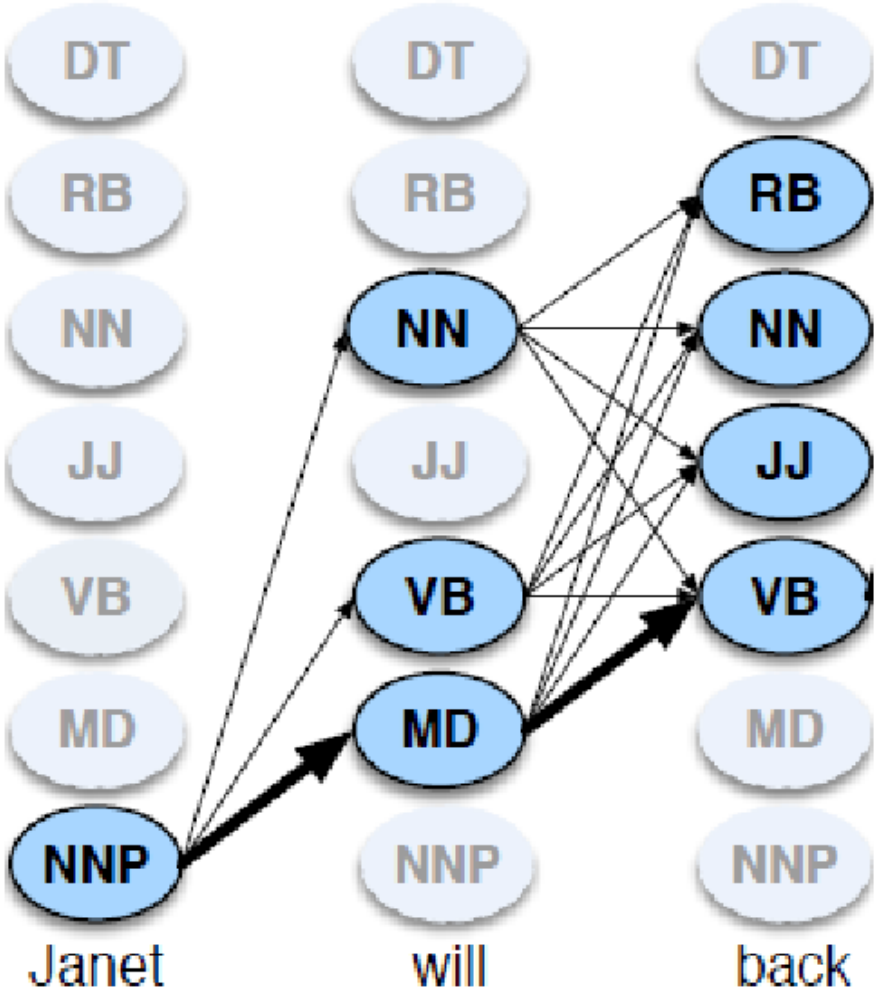


Viterbi[NN,will] = Viterbi[NNP,Janet]*P(NN|NNP)*P(will|NN)
= $8.85 \times 10^{-6} * 0.0584 * 0.0002 = 1.03368 \times 10^{-9}$

Viterbi[VB,will] = Viterbi[NNP,Janet]*P(VB|NNP)*P(will|VB)
= $8.85 \times 10^{-6} * 0.0009 * 0.000028 = 0.22302 \times 10^{-12}$

Viterbi[MD,will] = Viterbi[NNP,Janet]*P(MD|NNP)*P(will|MD)
= $8.85 \times 10^{-6} * 0.011 * 0.308431 = 0.203 \times 10^{-6}$

Viterbi Algorithm - Example



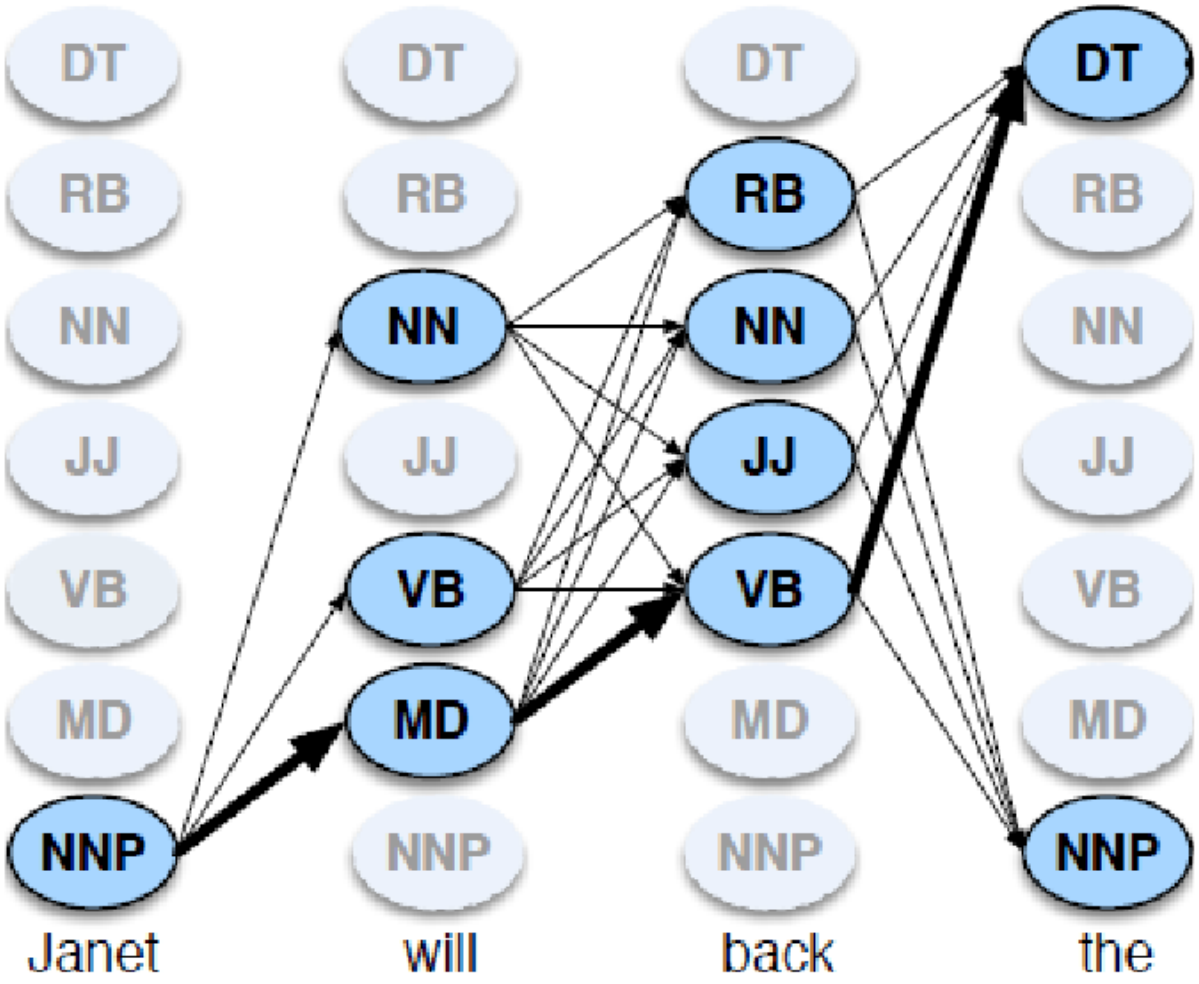
Viterbi[RB,back]=max({Viterbi[NN,will]*P(RB|NN)*P(back|RB),
Viterbi[VB,will]*P(RB|VB)*P(back|RB),
Viterbi[MD,will]*P(RB|MD)*P(back|RB)})

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)})

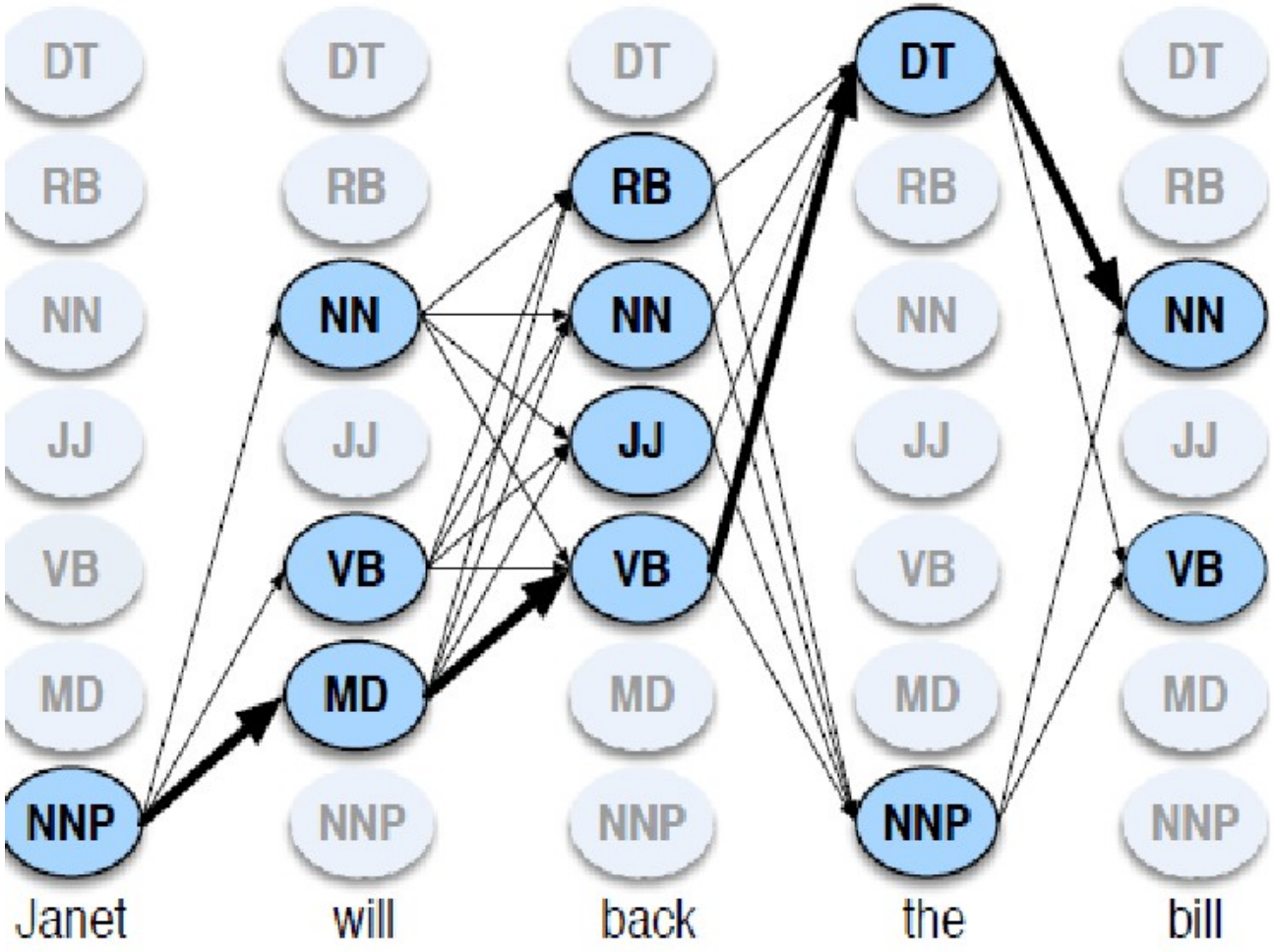
Viterbi Algorithm - Example



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 - Example



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)})

Viterbi Algorithm - Example

- Viterbi Matrix for
- Janet will back the bill
  - Janet /**NNP**
  - will /**MD**
  - back /**VB**
  - the /**DT**
  - bill /**NN**

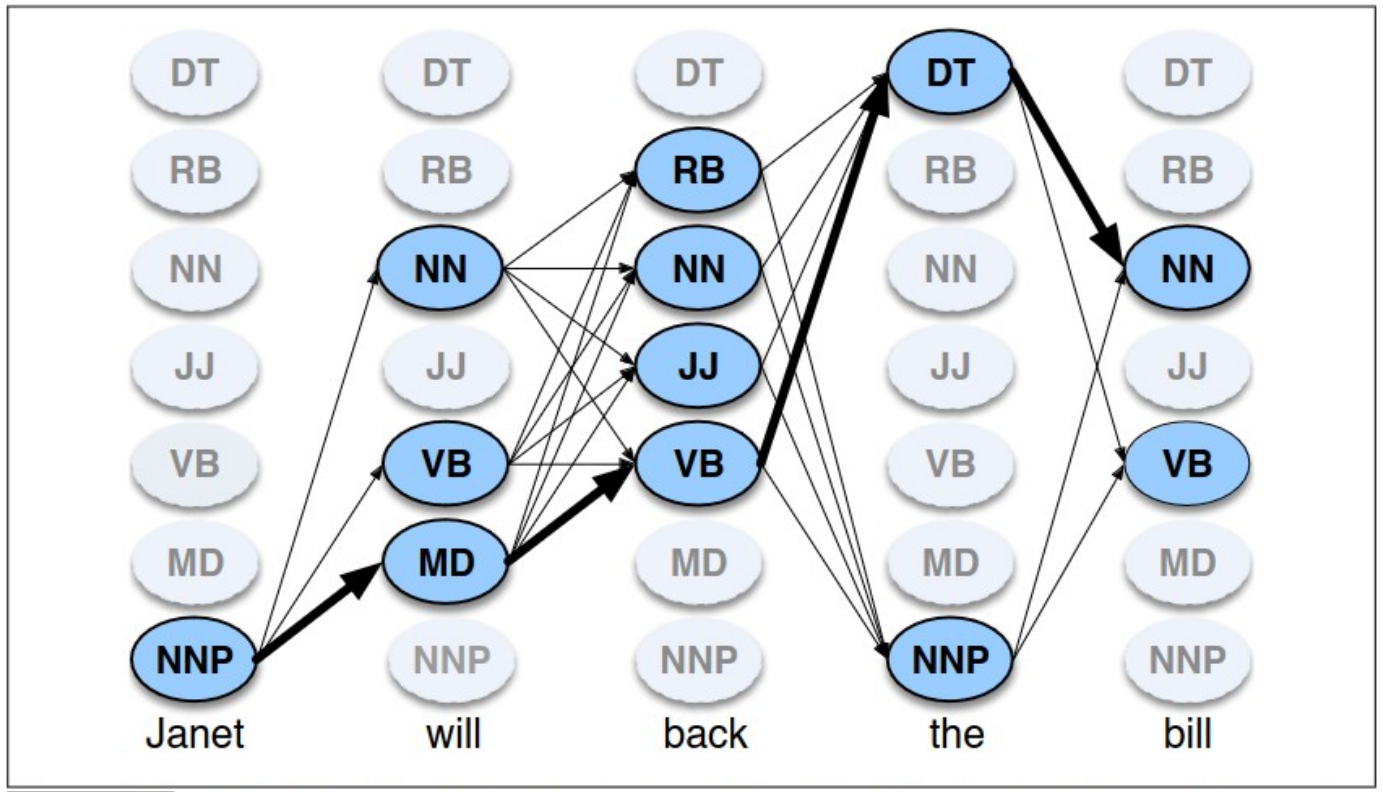


Figure 8.6 A sketch of the lattice for *Janet will back the bill*, showing the possible tags (q_i) for each word and highlighting the path corresponding to the correct tag sequence through the hidden states. States (parts of speech) which have a zero probability of generating a particular word according to the *B* matrix (such as the probability that a determiner DT will be realized as *Janet*) are greyed out.

Self Study

- 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.

Reference

- WSJ Corpus
  - https://www.spsc.tugraz.at/databases-and-tools/wall-street-journal-corpus.html
  - https://aclanthology.org/H92-1073.pdf

Thank you