

# 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 **book**.
  - Please **book** 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:
  - on the table
  - under 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:
    - I thought *that* you might like milk

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

# **Part-of-Speech 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, we look at **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.

# 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  $q_1 \dots q_{i-1} q_i$  :

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

# HMM Part-of-Speech Tagging

## *Markov Chains*

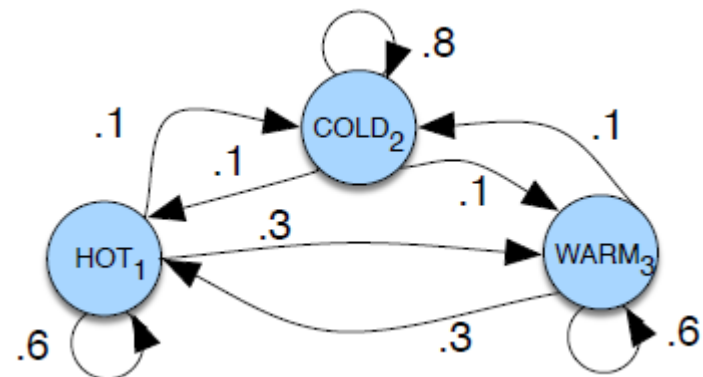
- A Markov chain is specified by the following components:

|                                               |                                                                                                                                                                                                                                                      |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $Q = q_1 q_2 \dots q_N$                       | a set of $N$ states                                                                                                                                                                                                                                  |
| $A = a_{11} a_{12} \dots a_{n1} \dots a_{nn}$ | a <b>transition probability matrix</b> $A$ , each $a_{ij}$ representing the probability of moving from state $i$ to state $j$ , s.t.<br>$\sum_{j=1}^n a_{ij} = 1 \quad \forall i$                                                                    |
| $\pi = \pi_1, \pi_2, \dots, \pi_N$            | an <b>initial probability distribution</b> over states. $\pi_i$ 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  $\pi$  is required.*
  - setting  $\pi = [0.1, 0.7, 0.2]$  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**
  - $P(\text{cold hot hot warm}) = \pi_2 * P(\text{hot}|\text{cold}) * P(\text{hot}|\text{hot}) * P(\text{warm}|\text{hot})$   
 $= 0.7 * 0.1 * 0.6 * 0.3$

# 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:

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

# First-Order HMM for Part-of-Speech Tagging

States: **Set of part-of-speech tags.**

Transition Probabilities: **Tag transition probabilities.**

- A tag transition probability  $P(\text{tag}_b | \text{tag}_a)$  represents the probability of a tag  $\text{tag}_b$  occurring given the previous tag  $\text{tag}_a$ .

- $$P(\text{tag}_b | \text{tag}_a) = \frac{\text{count}(\text{tag}_a \text{tag}_b)}{\text{count}(\text{tag}_a)}$$

Example: 
$$P(\text{VB} | \text{MD}) = \frac{\text{count}(\text{MD VB})}{\text{count}(\text{MD})}$$

Observations: **Words (Vocabulary)**

Observation Likelihoods (Emission Probabilities): **Emission Probabilities  $P(\text{word} | \text{tag})$**

- A emission probability  $P(\text{word} | \text{tag})$  represents probability of  $\text{tag}$  producing  $\text{word}$ .

- $$P(\text{word} | \text{tag}) = \frac{\text{count}(\text{tag}, \text{word})}{\text{count}(\text{tag})}$$

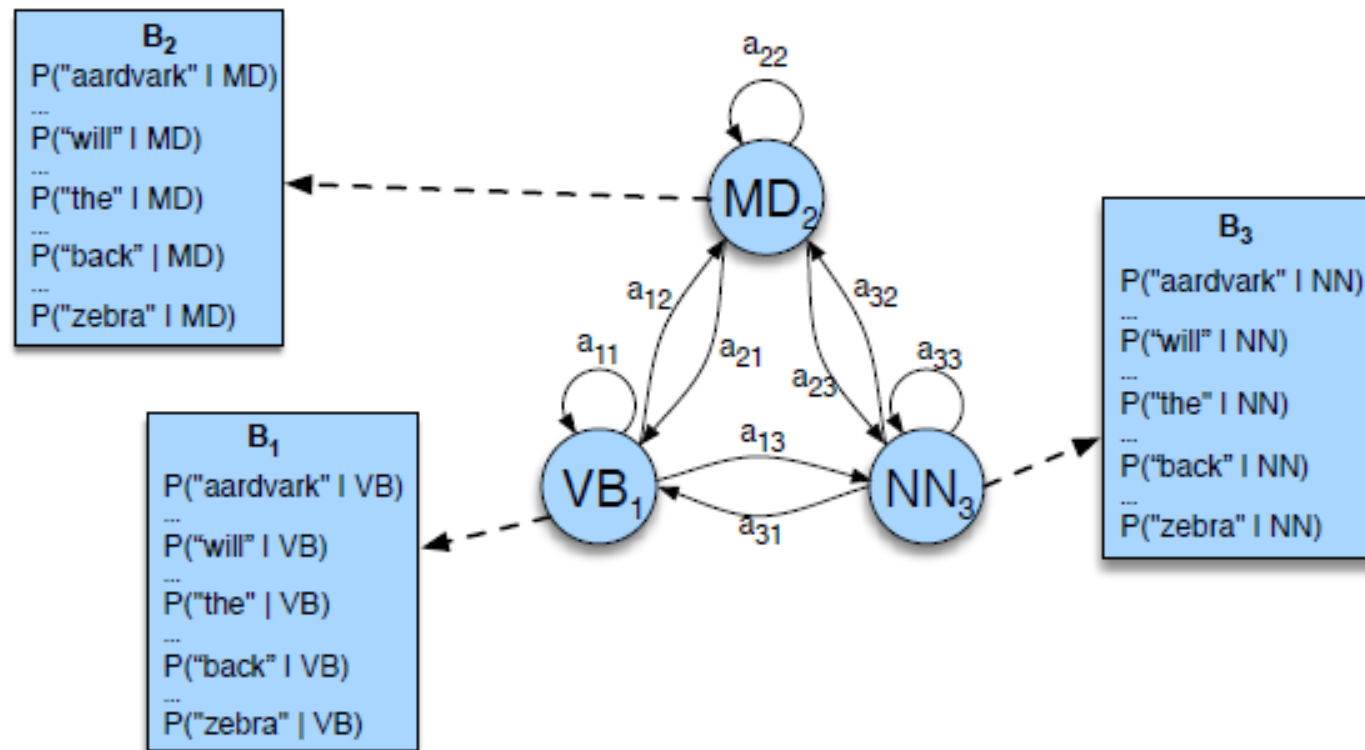
Example: 
$$P(\text{will} | \text{MD}) = \frac{\text{count}(\text{MD}, \text{will})}{\text{count}(\text{MD})}$$

Initial Probability Distribution: **First Tag Probabilities  $P(\text{tag} | \langle s \rangle)$  in sentences.**

- $$P(\text{tag} | \langle s \rangle) = \frac{\text{count}(\langle s \rangle \text{tag})}{\text{count}(\langle s \rangle)}$$

# 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  $\lambda = (\text{TransProbs}, \text{ObsLikelihoods})$  and a sequence of observations  $O = o_1, \dots, o_n$ , find the most probable sequence of states  $Q = q_1, \dots, q_n$ .

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

# HMM Tagging as Decoding

- For part-of-speech tagging, we will find **most probable tag sequence**  $\mathbf{T}=\mathbf{t}_1,\dots,\mathbf{t}_n$  for a **given sequence of n words**  $\mathbf{W}=\mathbf{w}_1,\dots,\mathbf{w}_n$ .
- **The most probable tag sequence**  $\hat{\mathbf{T}}$  (among possible tag sequences  $\boldsymbol{\tau}$ ) is:

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

- By Bayes rule:

$$\hat{\mathbf{T}} = \operatorname{argmax}_{\mathbf{T} \in \boldsymbol{\tau}} \frac{P(\mathbf{W}|\mathbf{T}) P(\mathbf{T})}{P(\mathbf{W})}$$

- Since  $P(\mathbf{W})$  is same for all tag sequences.

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

# 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:**

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

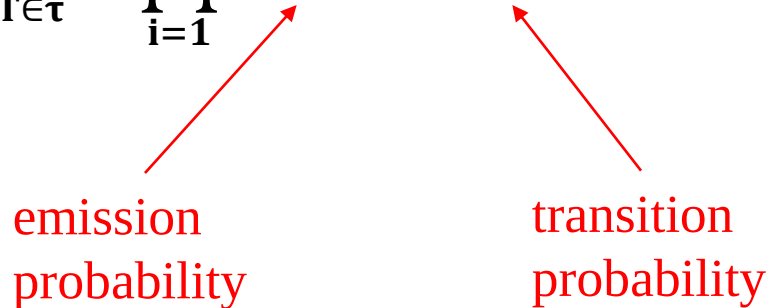
- **The second assumption, the bigram assumption (first-order HMM), is that the probability of a tag is dependent only on the previous tag, rather than the entire tag sequence:**

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

# 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)**:

$$\hat{T} = \operatorname{argmax}_{T \in \tau} P(T|W) = \operatorname{argmax}_{T \in \tau} \prod_{i=1}^n P(w_i|t_i) P(t_i|t_{i-1})$$



emission probability                      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  $\lambda = (A, B)$  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]  $\leftarrow \max_{s'=1}^N \text{viterbi}[s',t-1] * a_{s',s} * b_s(o_t)$   
        backpointer[s,t]  $\leftarrow \operatorname{argmax}_{s'=1}^N \text{viterbi}[s',t-1] * a_{s',s} * b_s(o_t)$   
  
bestpathprob  $\leftarrow \max_{s=1}^N \text{viterbi}[s,T]$                                 ; termination step  
  
bestpathpointer  $\leftarrow \operatorname{argmax}_{s=1}^N \text{viterbi}[s,T]$                         ; 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
     $bestpath \leftarrow$  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**

$viterbi[s, 1] \leftarrow \pi_s * b_s(o_1)$

$backpointer[s, 1] \leftarrow 0$

most probable path probabilities of first word  $o_1$  where  
 $\pi_s$  is first tag probability of tag  $s$  and  
 $b_s(o_1)$  is emission probability  $P(\text{word } o_1 \mid \text{tag } s)$

**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**  $bestpath$ ,  $bestpathprob$

# 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  $t$  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  $s^t$

$a_{s^t, s}$  is transition probability  $P(\text{tag } s \mid \text{tag } s^t)$  and  
 $b_s(o_t)$  is emission probability  $P(\text{word } o_t \mid \text{tag } s)$

# 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**

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

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

$bestpathprob \leftarrow \max_{s=1}^N viterbi[s, T]$

$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^N viterbi[s, T]$

← most probable path probability of  $T$  words

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

**return** *bestpath*, *bestpathprob*

# 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.

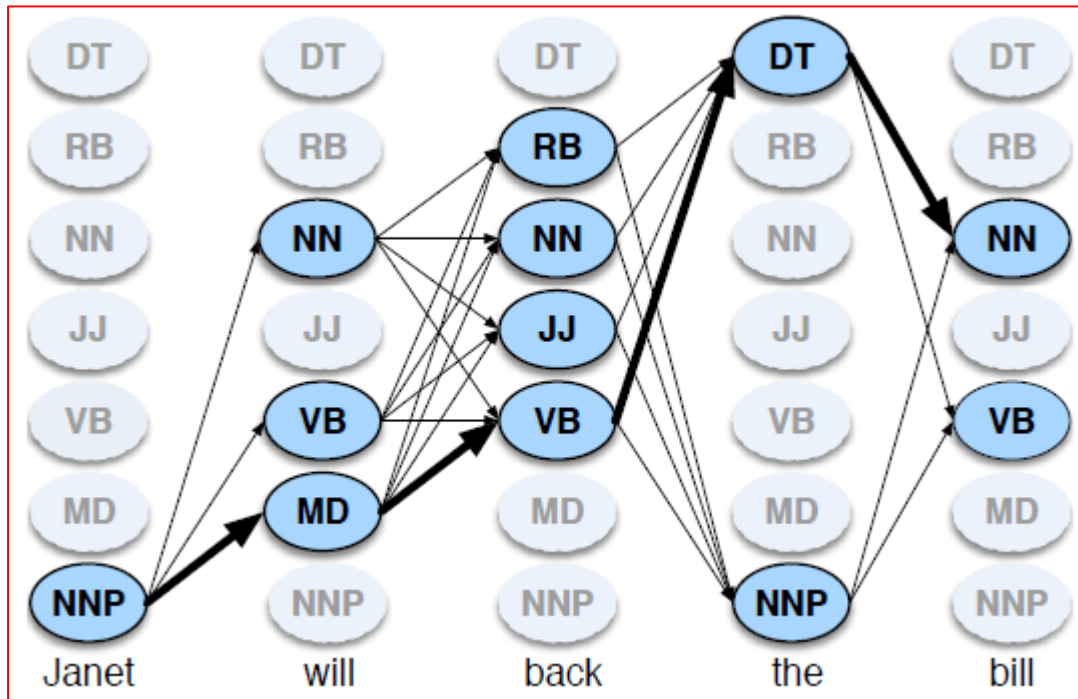
|     | NNP    | MD     | VB     | JJ     | NN     | RB     | DT     |
|-----|--------|--------|--------|--------|--------|--------|--------|
| <s> | 0.2767 | 0.0006 | 0.0031 | 0.0453 | 0.0449 | 0.0510 | 0.2026 |
| NNP | 0.3777 | 0.0110 | 0.0009 | 0.0084 | 0.0584 | 0.0090 | 0.0025 |
| MD  | 0.0008 | 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

DT

RB

NN

JJ

VB

MD

NNP

Janet

|     | 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 |

|     | 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        |

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

$\text{viterbi}[\text{NNP}, \text{Janet}] =$

$P(\text{NNP}|\text{<s>})*P(\text{Janet}|\text{NNP}) = 0.2767*0.000032 = 0.000008854 = 8.854 \times 10^{-6}$

# Viterbi Algorithm for POS Tagging

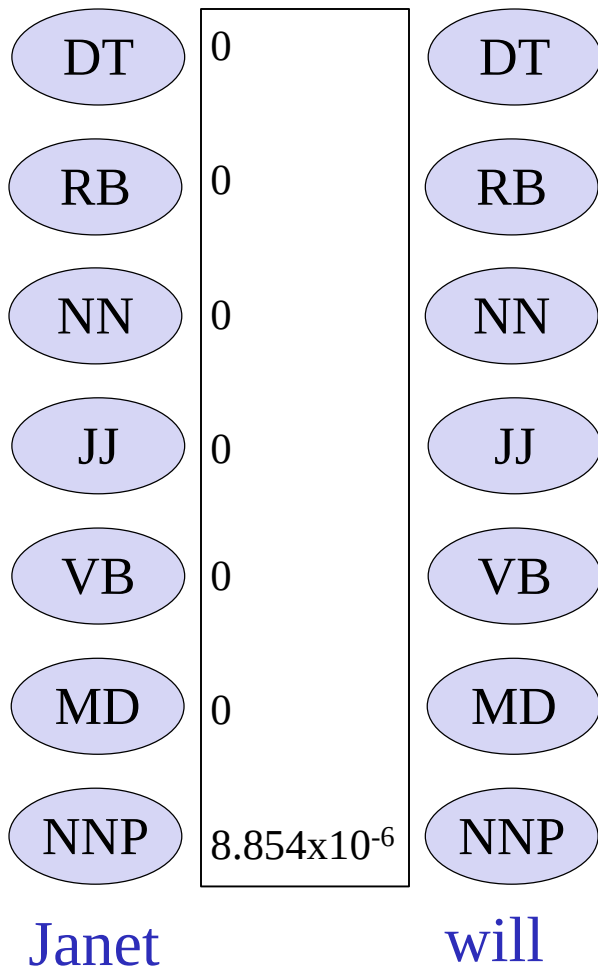
**Example: Janet will back the bill**

|     |                        |
|-----|------------------------|
| DT  | 0                      |
| RB  | 0                      |
| NN  | 0                      |
| JJ  | 0                      |
| VB  | 0                      |
| MD  | 0                      |
| NNP | $8.854 \times 10^{-6}$ |

Janet

# Viterbi Algorithm for POS Tagging

**Example: 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 |

$$\text{viterbi}[\text{NN}, \text{will}] = \text{viterbi}[\text{NNP}, \text{Janet}] * P(\text{NN}|\text{NNP}) * P(\text{will}|\text{NN})$$

$$= 8.854 \times 10^{-6} * 0.0584 * 0.0002 = 1.034 \times 10^{-10}$$

$$\text{viterbi}[\text{VB}, \text{will}] = \text{viterbi}[\text{NNP}, \text{Janet}] * P(\text{VB}|\text{NNP}) * P(\text{will}|\text{VB})$$

$$= 8.854 \times 10^{-6} * 0.0009 * 0.000028 = 2.231 \times 10^{-13}$$

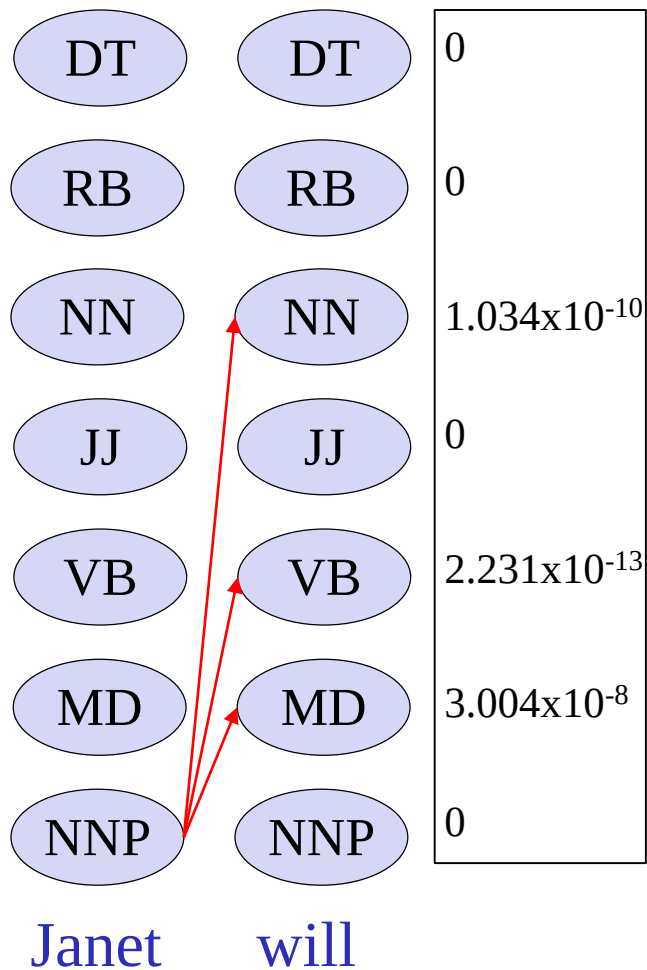
$$\text{viterbi}[\text{MD}, \text{will}] = \text{viterbi}[\text{NNP}, \text{Janet}] * P(\text{MD}|\text{NNP}) * P(\text{will}|\text{MD})$$

$$= 8.854 \times 10^{-6} * 0.011 * 0.308431 = 3.004 \times 10^{-8}$$

|     | 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        |

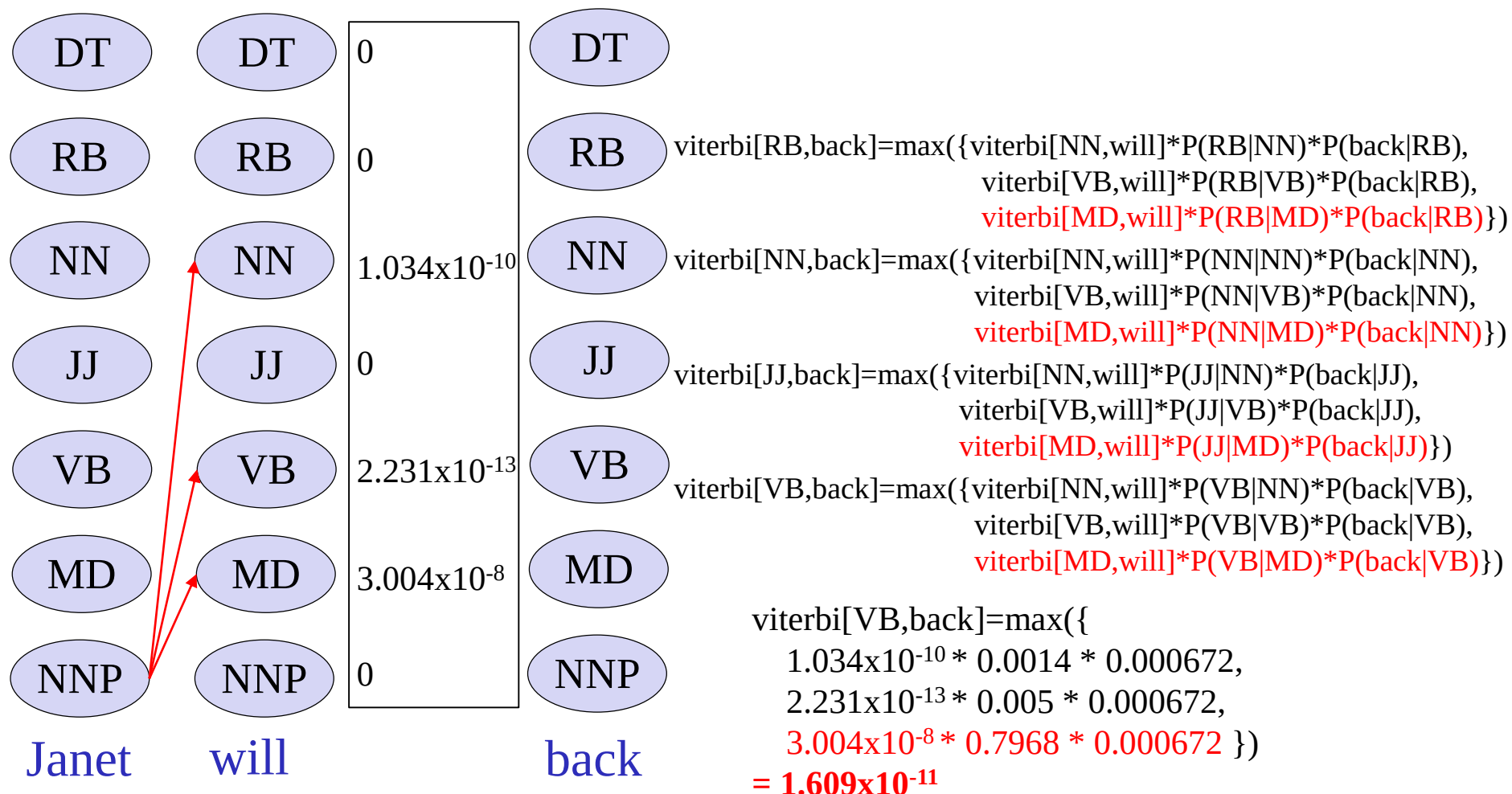
# Viterbi Algorithm for POS Tagging

**Example: Janet will back the bill**



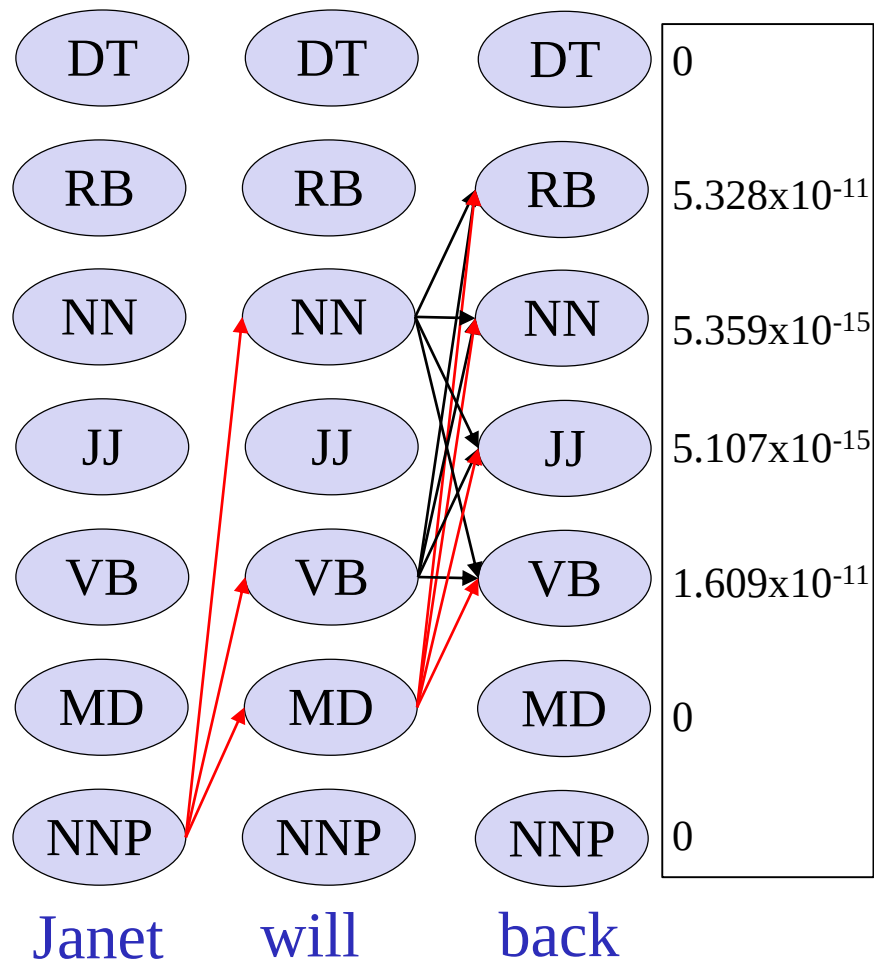
# Viterbi Algorithm for POS Tagging

Example: Janet will back the bill



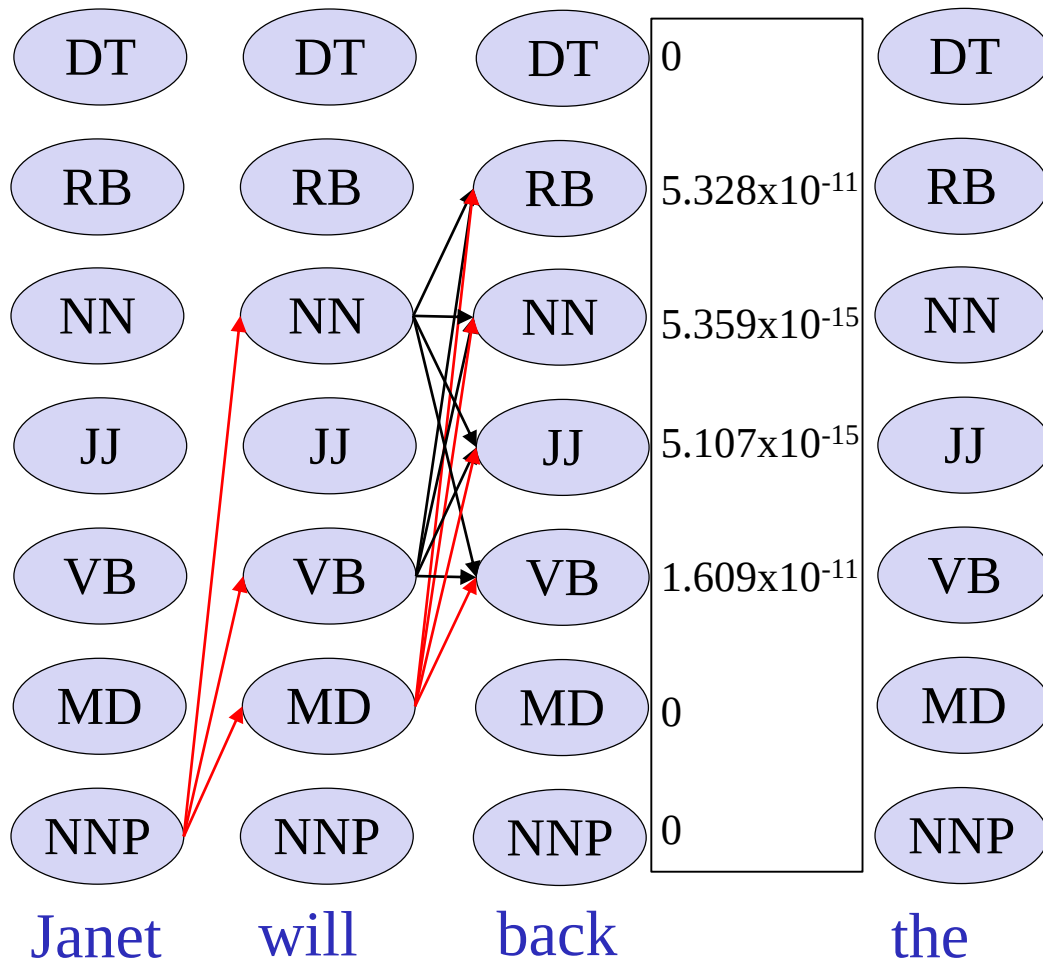
# Viterbi Algorithm for POS Tagging

**Example: Janet will back the bill**



# Viterbi Algorithm for POS Tagging

## Example: Janet will back the bill

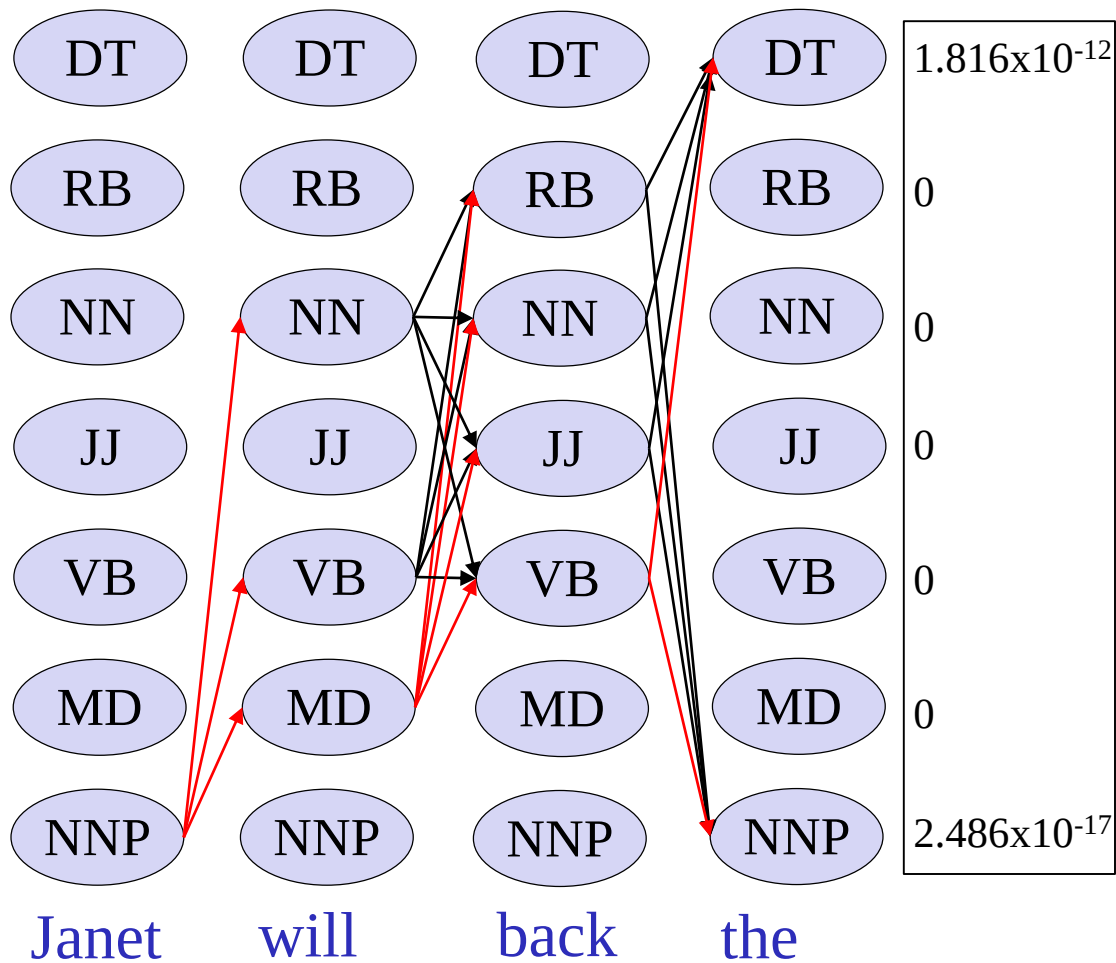


$$\text{viterbi}[\text{DT}, \text{the}] = \max(\{ \text{viterbi}[\text{RB}, \text{back}] * P(\text{DT}|\text{RB}) * P(\text{the}|\text{DT}), \\ \text{viterbi}[\text{NN}, \text{back}] * P(\text{DT}|\text{NN}) * P(\text{the}|\text{DT}), \\ \text{viterbi}[\text{JJ}, \text{back}] * P(\text{DT}|\text{JJ}) * P(\text{the}|\text{DT}), \\ \text{viterbi}[\text{VB}, \text{back}] * P(\text{DT}|\text{VB}) * P(\text{the}|\text{DT}) \})$$

$$\text{viterbi}[\text{NNP}, \text{the}] = \max(\{ \text{viterbi}[\text{RB}, \text{back}] * P(\text{NNP}|\text{RB}) * P(\text{the}|\text{NNP}), \\ \text{viterbi}[\text{NN}, \text{back}] * P(\text{NNP}|\text{NN}) * P(\text{the}|\text{NNP}), \\ \text{viterbi}[\text{JJ}, \text{back}] * P(\text{NNP}|\text{JJ}) * P(\text{the}|\text{NNP}), \\ \text{viterbi}[\text{VB}, \text{back}] * P(\text{NNP}|\text{VB}) * P(\text{the}|\text{NNP}) \})$$

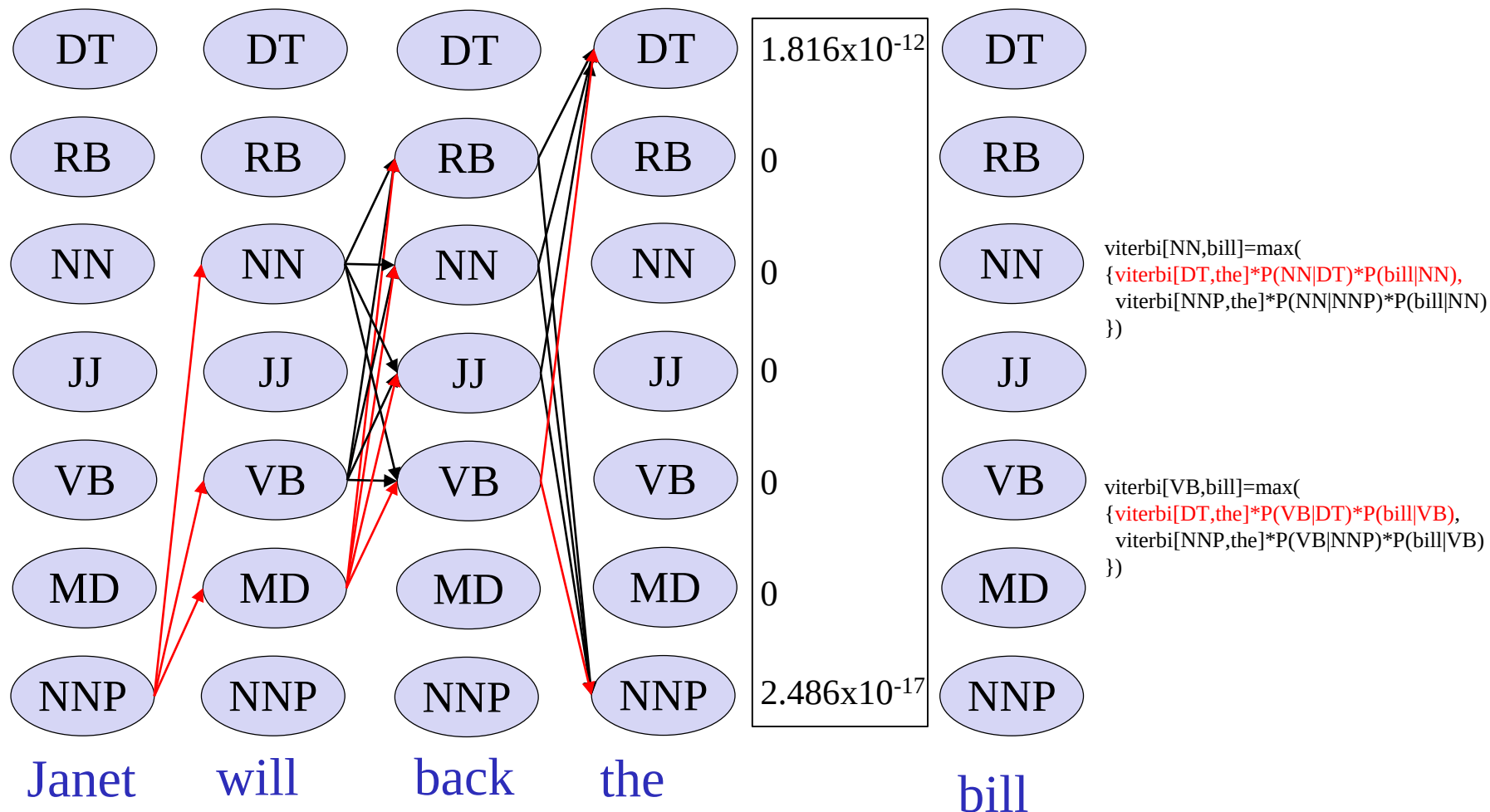
# Viterbi Algorithm for POS Tagging

**Example: Janet will back the bill**



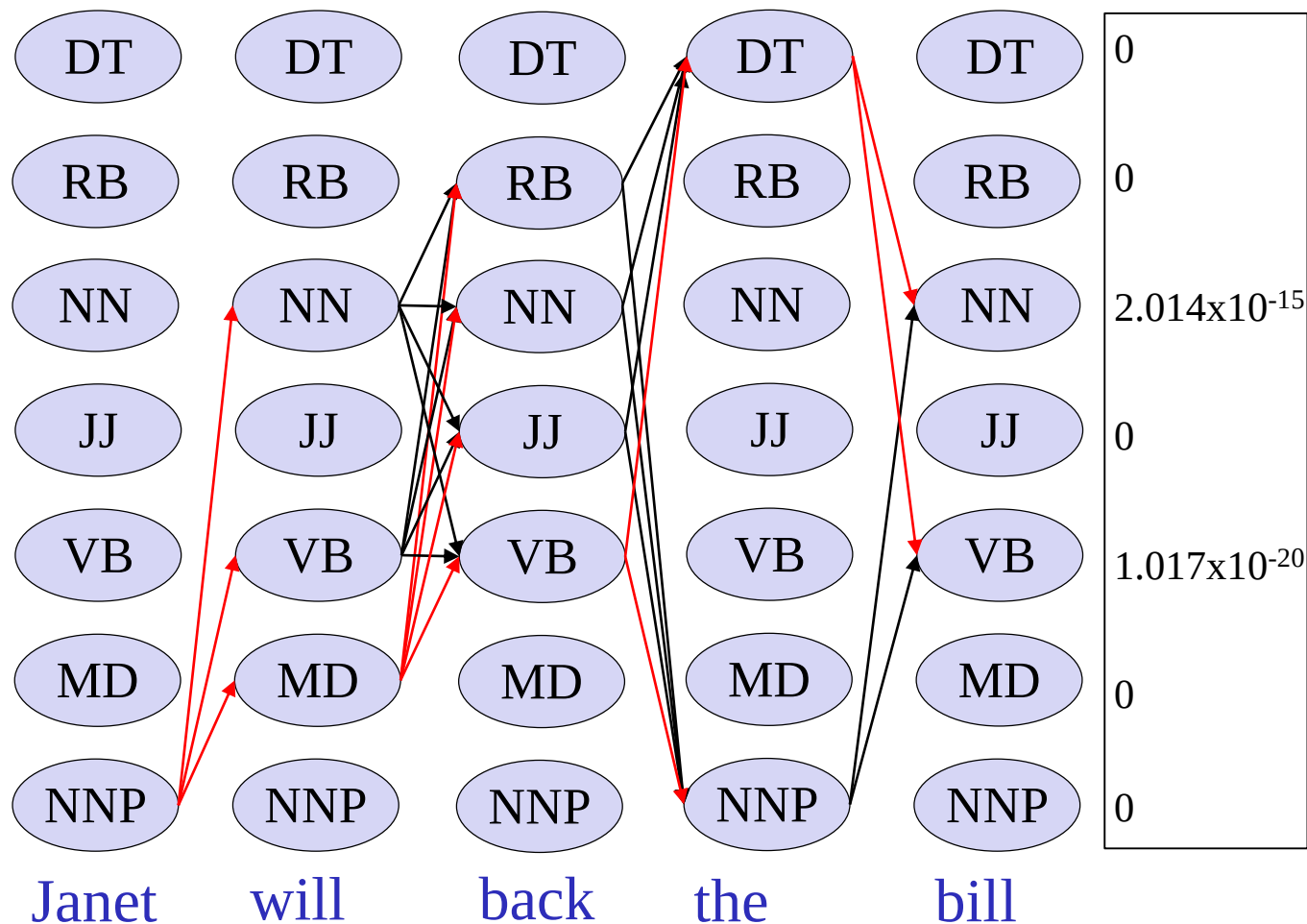
# Viterbi Algorithm for POS Tagging

**Example: Janet will back the bill**



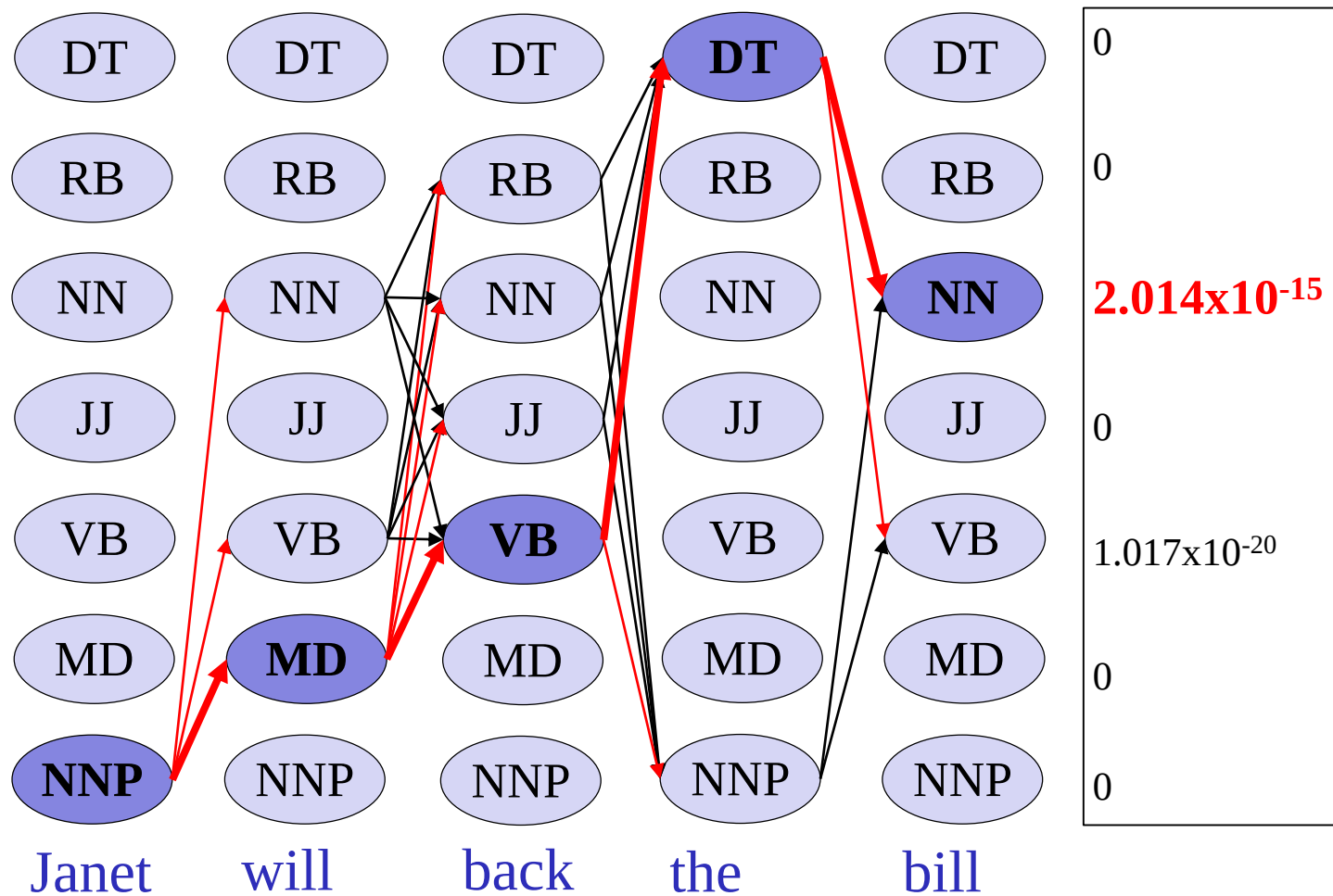
# Viterbi Algorithm for POS Tagging

**Example: Janet will back the bill**



# Viterbi Algorithm for POS Tagging

**Example: Janet will back the bill**



# 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.

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

- 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**.