CS447: Natural Language Processing

http://courses.engr.illinois.edu/cs447

Lecture 12: HMMs, Sequence Labeling

Julia Hockenmaier

juliahmr@illinois.edu

3324 Siebel Center

Assessment updates

Peer-Grading:

Starting today, you will have one week after the submission deadline to finish grading the submissions assigned to you.

4th Credit Hour Lit Review:

We put a list of suggestions on Canvas (currently buried under “Pages”), but these are not exhaustive.

We will ask you for a preliminary topic/list of papers by March 12.

Hidden Markov Models (HMMs) for POS Tagging

Statistical POS tagging

	She	promised	to	back	the	bill
$\mathbf{w} =$	$w^{(1)}$	$w^{(2)}$	$w^{(3)}$	$w^{(4)}$	$w^{(5)}$	$w^{(6)}$
$\mathbf{t} =$	$t^{(1)}$	$t^{(2)}$	$t^{(3)}$	$t^{(4)}$	$t^{(5)}$	$t^{(6)}$
	PRP	**VBD**	**TO**	**VB**	**DT**	**NN**

What is the most likely sequence of tags $\mathbf{t} = t^{(1)} \ldots t^{(N)}$ for the given sequence of words $\mathbf{w} = w^{(1)} \ldots w^{(N)}$?

$$\mathbf{t}^* = \text{argmax}_{\mathbf{t}} P(\mathbf{t} \mid \mathbf{w})$$

POS tagging with generative models

$$
\begin{aligned}
\operatorname*{argmax}_{\mathbf{t}} P(\mathbf{t}|\mathbf{w}) &= \operatorname*{argmax}_{\mathbf{t}} \frac{P(\mathbf{t},\mathbf{w})}{P(\mathbf{w})} \\
&= \operatorname*{argmax}_{\mathbf{t}} P(\mathbf{t},\mathbf{w}) \\
&= \operatorname*{argmax}_{\mathbf{t}} P(\mathbf{t})P(\mathbf{w}|\mathbf{t})
\end{aligned}
$$

$P(\mathbf{t},\mathbf{w})$: the joint distribution of the labels we want to predict ($\mathbf{t}$) and the observed data ($\mathbf{w}$).

We decompose $P(\mathbf{t},\mathbf{w})$ into $P(\mathbf{t})$ and $P(\mathbf{w} \mid \mathbf{t})$ since these distributions are easier to estimate.

Models based on joint distributions of labels and observed data are called generative models: think of $P(\mathbf{t})P(\mathbf{w} \mid \mathbf{t})$ as a stochastic process that first generates the labels, and then generates the data we see, based on these labels.

Hidden Markov Models (HMMs)

HMMs are the most commonly used generative models for POS tagging (and other tasks, e.g. in speech recognition)

HMMs make specific **independence assumptions** in $P(\mathbf{t})$ and $P(\mathbf{w}|\ \mathbf{t})$:

1) $P(\mathbf{t})$ is an *n*-gram (typically **bigram** or **trigram**) model over tags:

$$P_{\text{bigram}}(\mathbf{t}) = \prod_i P(t^{(i)} \mid t^{(i-1)})$$

$$P_{\text{trigram}}(\mathbf{t}) = \prod_i P(t^{(i)} \mid t^{(i-1)}, t^{(i-2)})$$

$P(t^{(\mathrm{i})} \mid t^{(\mathrm{i}-1)})$ and $P(t^{(\mathrm{i})} \mid t^{(\mathrm{i}-1)}, t^{(\mathrm{i}-2)})$ are called **transition probabilities**

2) In $P(\mathbf{w} \mid \mathbf{t})$, each $\mathrm{w}^{(\mathrm{i})}$ depends only on [is generated by/conditioned on] $\mathrm{t}^{(\mathrm{i})}$:

$$P(\mathbf{w} \mid \mathbf{t}) = \prod_i P(w^{(i)} \mid t^{(i)})$$

$P(w^{(\mathrm{i})} \mid t^{(\mathrm{i})})$ are called **emission probabilities**

These probabilities don't depend on the position in the sentence $^{(\mathrm{i})}$,
but are defined over word and tag types.
With subscripts $_{\mathrm{i,j,k}}$, to index word/tag types, they become $P(t_i \mid t_j)$, $P(t_i \mid t_j, t_k)$, $P(w_i \mid t_j)$

Notation: t_i/w_i vs $t^{(i)}/w^{(i)}$

To make the distinction between the i-th word/tag in the vocabulary/tag set and the i-th word/tag in the sentence clear:

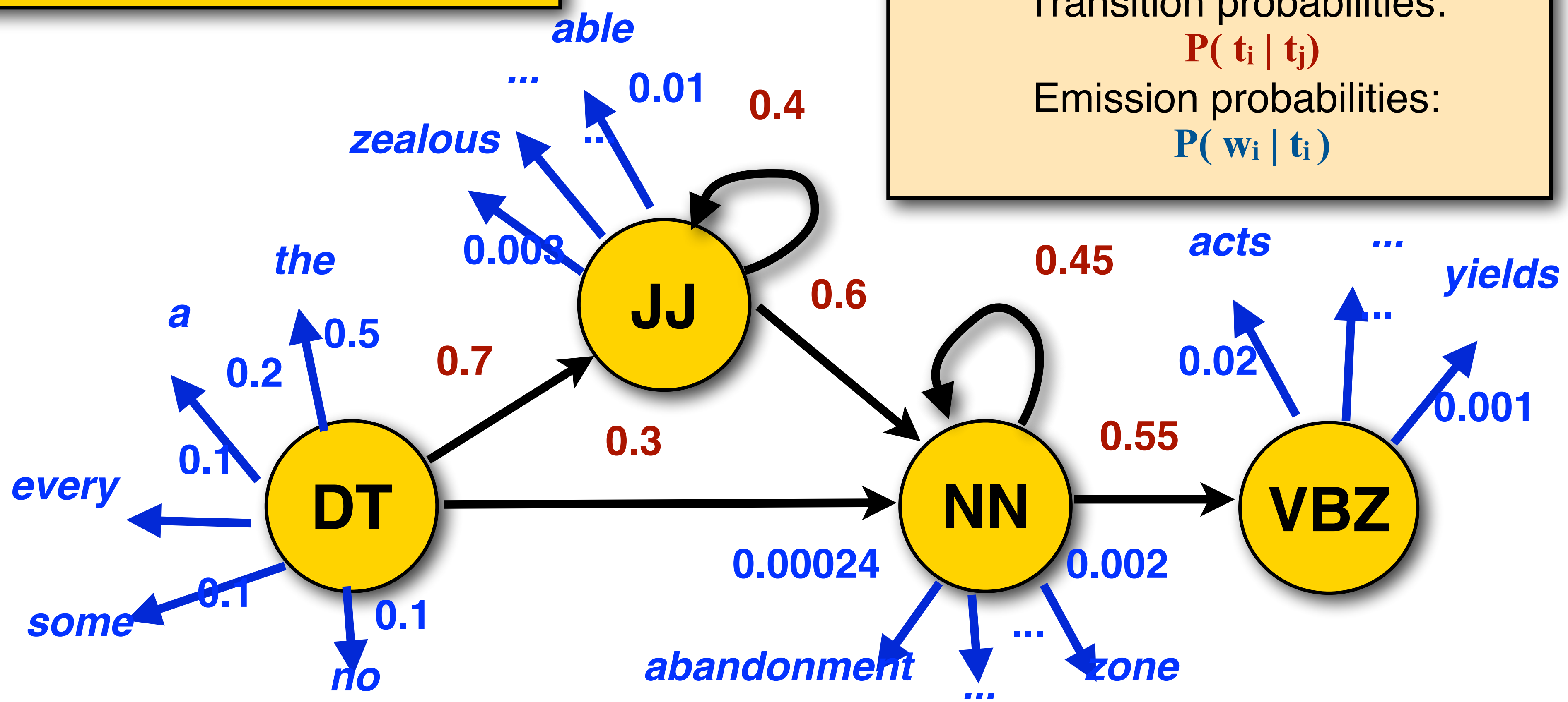
Use **superscript notation** $w^{(i)}$ for the **i-th token** in the sentence/sequence

and **subscript notation** w_i for the **i-th type** in the inventory (tagset/vocabulary)

HMMs as probabilistic automata

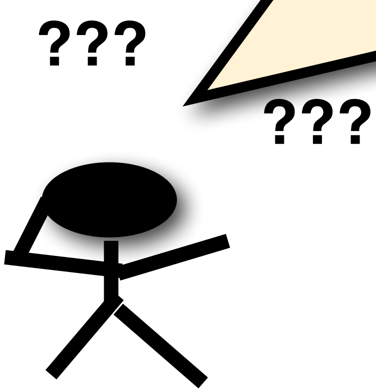
DT: Determiner
JJ: Adjective
NN: Common noun (singular)
VBZ: Verb (3rd pers sing. present tense)

An HMM defines
Transition probabilities:
$P(t_i \mid t_j)$
Emission probabilities:
$P(w_i \mid t_i)$

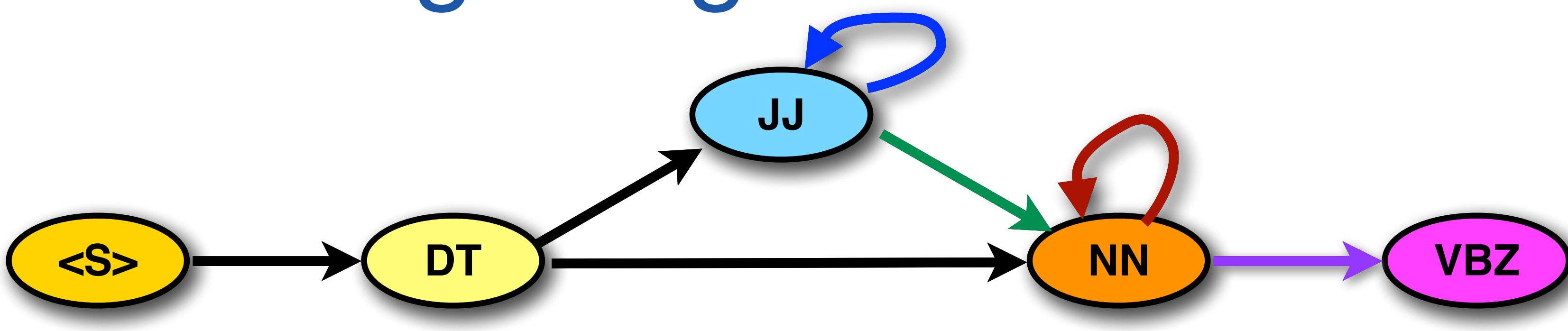


How would the automaton for a trigram HMM with transition probabilities $P(t_i \mid t_j t_k)$ look like?

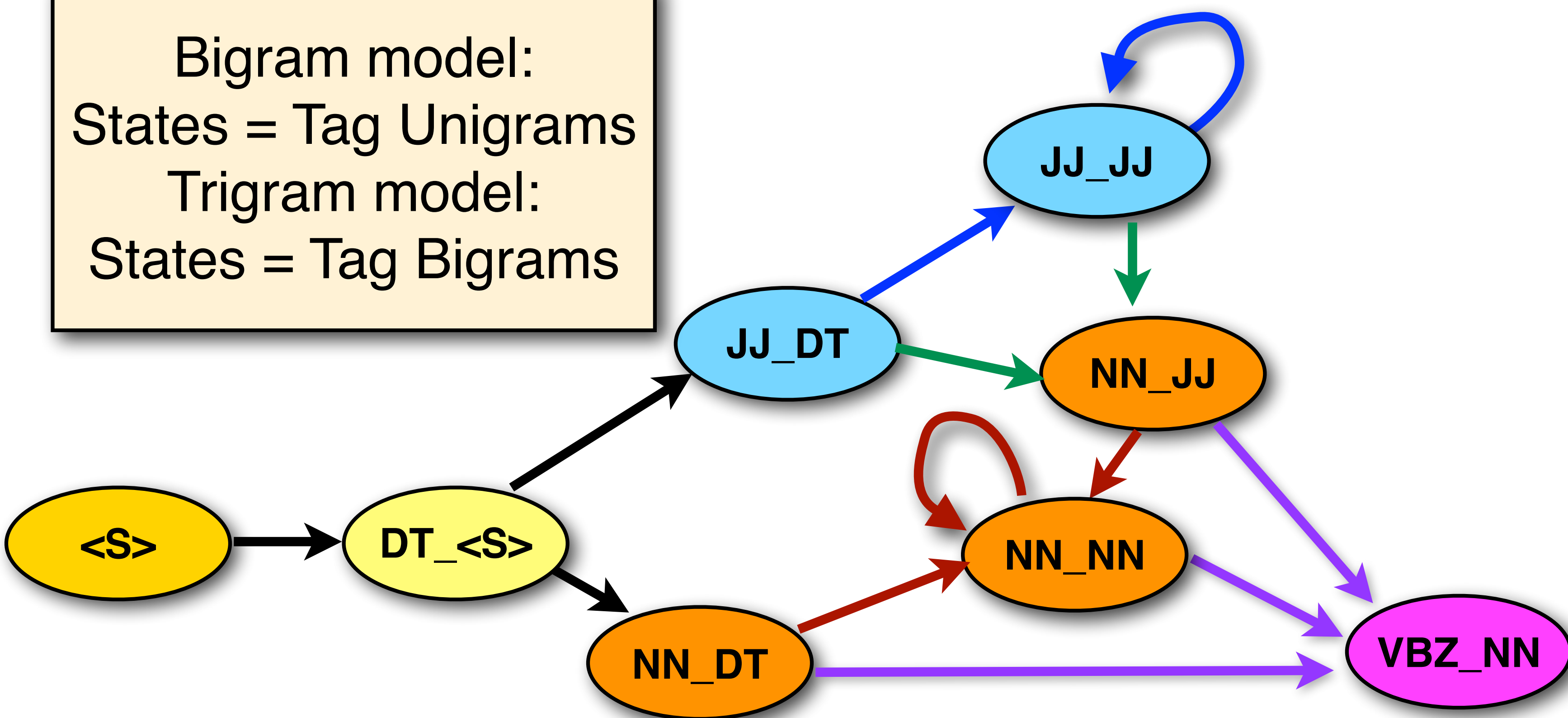
What about unigrams or n-grams?



Encoding a trigram model as FSA



Bigram model:
States = Tag Unigrams
Trigram model:
States = Tag Bigrams



HMM definition

A HMM $\lambda = (A, B, \pi)$ consists of

- a set of N **states** $Q = \{q_1, ...q_N\}$
  with $Q_0 \subseteq Q$ a set of **initial states**

- an **output vocabulary** of M items $V = \{v_1, ...v_m\}$

- an $N \times N$ **state transition probability matrix** A
  with a_{ij} the probability of moving from q_i to q_j.
  $(\sum_{j=1}^{N} a_{ij} = 1 \ \forall i; \quad 0 \leq a_{ij} \leq 1 \ \forall i, j)$

- an $N \times M$ **symbol emission probability matrix** B
  with b_{ij} the probability of emitting symbol v_j in state q_i
  $(\sum_{j=1}^{N} b_{ij} = 1 \ \forall i; \quad 0 \leq b_{ij} \leq 1 \ \forall i, j)$

- an **initial state distribution vector** $\pi = \langle \pi_1, ..., \pi_N \rangle$
  with π_i the probability of being in state q_i at time $t = 1$.
  $(\sum_{i=1}^{N} \pi_i = 1 \quad 0 \leq \pi_i \leq 1 \ \forall i)$

An example HMM

Transition Matrix A

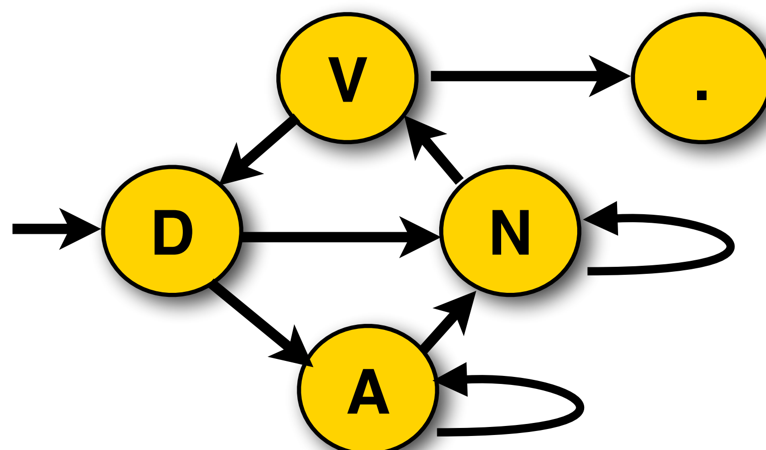
	D	**N**	**V**	**A**	**.**
D		0.8		0.2	
N		0.7	0.3		
V	0.6				0.4
A		0.8		0.2	
.					

Emission Matrix B

	the	**man**	**ball**	**throw**	**sees**	**red**	**blue**	**.**
D	1							
N		0.7	0.3					
V				0.6	0.4			
A						0.8	0.2	
.								1

Initial state vector π

	D	**N**	**V**	**A**	**.**
π	1				



Building an HMM tagger

To build an HMM tagger, we have to:

Train the model, i.e. estimate its parameters (the transition and emission probabilities)

Easy case: We have a corpus labeled with POS tags (supervised learning)
Harder case: We have a corpus, but it's just raw text without tags (unsupervised learning). In that case it really helps to have a dictionary of which POS tags each word can have

Define and implement a **tagging algorithm** that finds the best tag sequence $\mathbf{t}^*$ for each input sentence $\mathbf{w}$:

$$\mathbf{t}^* = \text{argmax}_{\mathbf{t}} P(\mathbf{t})P(\mathbf{w} \mid \mathbf{t})$$

[Viterbi]

Learning an HMM from *labeled* data

We count how often we see $t_i t_j$ and $w_j\_t_i$ etc. in the data (use relative frequency estimates):

```
Pierre_NNP Vinken_NNP ,_, 61_CD years_NNS
old_JJ ,_, will_MD join_VB the_DT board_NN
as_IN a_DT nonexecutive_JJ director_NN Nov._NNP
29_CD ._.
```

Learning the transition probabilities:

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

Learning the emission probabilities:

$$P(w_j|t_i) = \frac{C(w_j\_t_i)}{C(t_i)}$$

The Viterbi Algorithm

HMM decoding (Viterbi)

We are given a sentence $\mathbf{w} = w^{(1)} \ldots w^{(N)}$

$\mathbf{w}$= *"she promised to back the bill"*

We want to use an HMM tagger to find its POS tags **t**

$$\mathbf{t}^* = \text{argmax}_{\mathbf{t}}\ \mathrm{P}(\mathbf{w}, \mathbf{t})$$

$$= \text{argmax}_{\mathbf{t}}\ \mathrm{P}(t^{(1)}) \cdot \mathrm{P}(w^{(1)} | t^{(1)}) \cdot \mathrm{P}(t^{(2)} | t^{(1)}) \cdot \ldots \cdot \mathrm{P}(w^{(N)} | \mathrm{t}^{(N)})$$

*But: with T tags, **w** has $O(T^N)$ possible tag sequences!*

To do this efficiently (in $O(T^2N)$ time), we will use a **dynamic programming** technique called the **Viterbi algorithm** which exploits the **independence assumptions** in the HMM.

Dynamic programming

Dynamic programming is a general technique to solve certain complex search problems by memoization

1.) Recursively decompose the large search problem into smaller **subproblems** that can be solved efficiently

- There is only a polynomial number of subproblems.

2.) Store (memoize) the solutions of each subproblem in a **common data structure**

- Processing this data structure takes polynomial time

The Viterbi algorithm

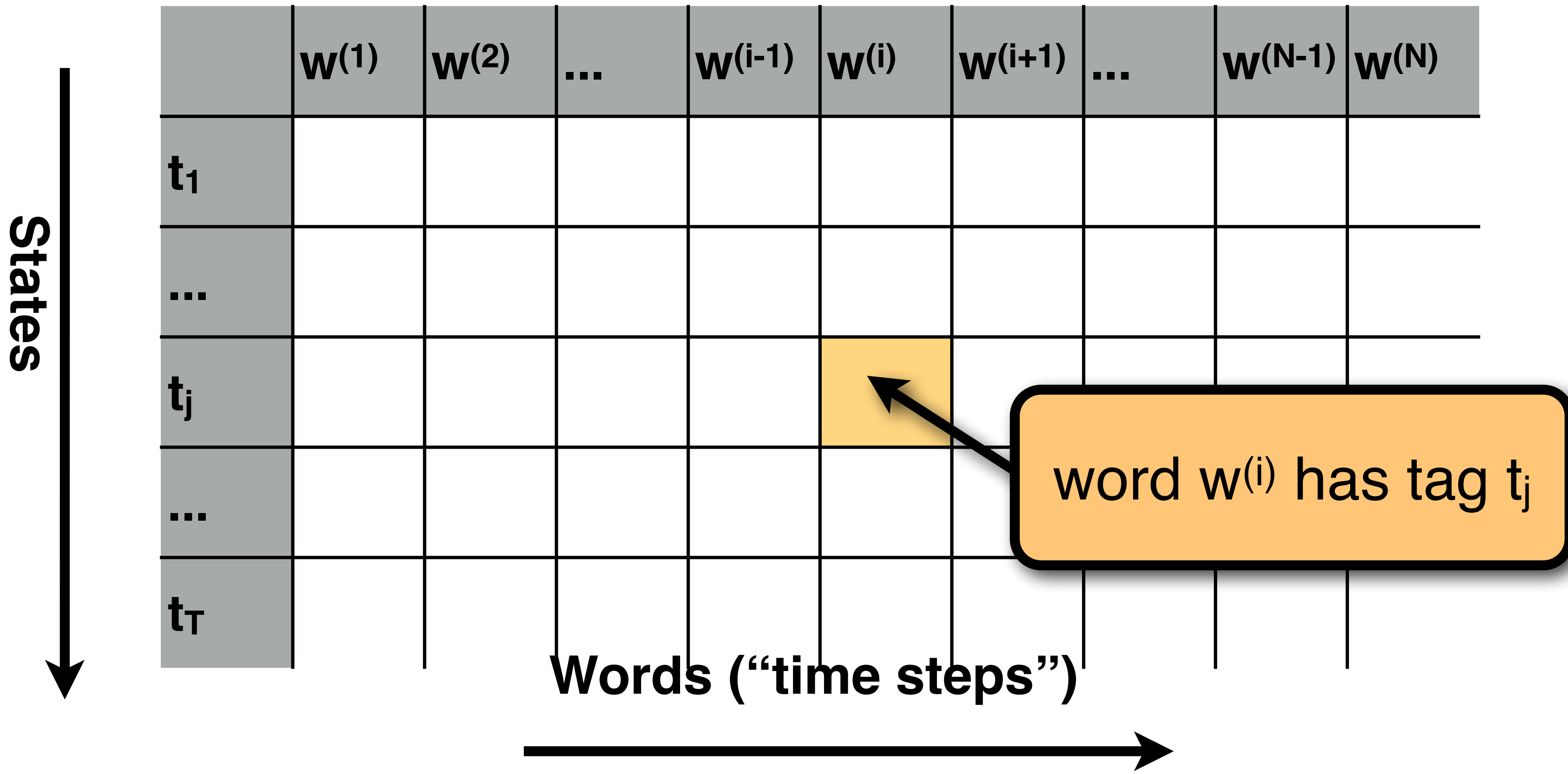
A dynamic programming algorithm which finds the best (=most probable) tag sequence $\mathbf{t}^*$ for an input sentence $\mathbf{w}$: $\mathbf{t}^* = \text{argmax}_{\mathbf{t}} P(\mathbf{w} \mid \mathbf{t})P(\mathbf{t})$

Complexity: linear in the sentence length.
With a bigram HMM, Viterbi runs in $O(T^2N)$ steps for an input sentence with N words and a tag set of T tags.

The independence assumptions of the HMM tell us how to break up the big search problem (find $\mathbf{t}^* = \text{argmax}_{\mathbf{t}} P(\mathbf{w} \mid \mathbf{t})P(\mathbf{t})$) into smaller subproblems.

The data structure used to store the solution of these subproblems is the trellis.

Bookkeeping: the trellis



We use a N×T table ("**trellis**") to keep track of the HMM.
The HMM can assign one of the T tags to each of the N words.

Viterbi: filling in the first column

	$w^{(1)}$
DT	$\pi(\mathrm{DT}) \times P(w^{(1)} \mid \mathrm{DT})$
...	
NNS	$\pi(\mathrm{NNS}) \times P(w^{(1)} \mid \mathrm{NNS})$
...	
VBZ	$\pi(\mathrm{VBZ}) \times P(w^{(1)} \mid \mathrm{VBZ})$

$\pi(\mathrm{DT})$: probability that a sentence starts with DT

$P(w^{(1)} \mid \mathrm{DT})$: probability that tag DT emits word $w^{(1)}$

We want to find the best (most likely) tag sequence for the entire sentence.

Each cell **trellis[i][j]** (corresponding to word $w^{(i)}$ with tag t_j) contains:

— **trellis[i][j].viterbi:** the probability of the best sequence ending in t_j

— **trellis[i][j].backpointer:** to the cell k in the previous column that corresponds to the best tag sequence ending in t_j

Initialization

For a bigram HMM:

Given an N-word sentence $w^{(1)} \ldots w^{(N)}$ and a tag set consisting of T tags, create a trellis of size N×T

In the first column, initialize each cell $\mathrm{trellis}[1][k]$ as

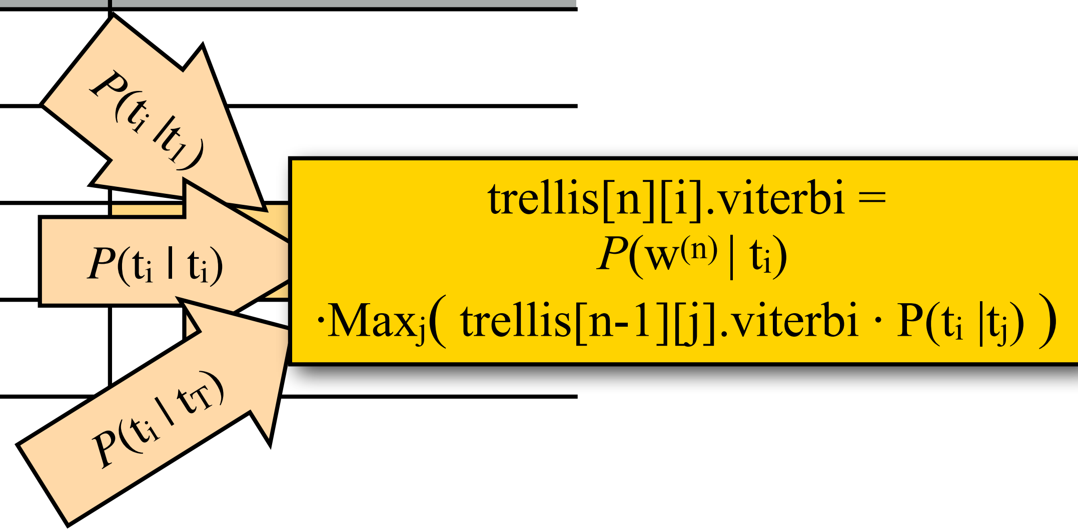
$$\mathrm{trellis}[1][k] := \pi(t_k) P(w^{(1)} \mid t_k)$$

(there is only a single tag sequence for the first word that assigns a particular tag to that word)

At any internal cell

– For each cell in the preceding column: multiply its Viterbi probability with the transition probability to the current cell.
– Keep a single backpointer to the best (highest scoring) cell in the preceding column
– Multiply this score with the emission probability of the current word

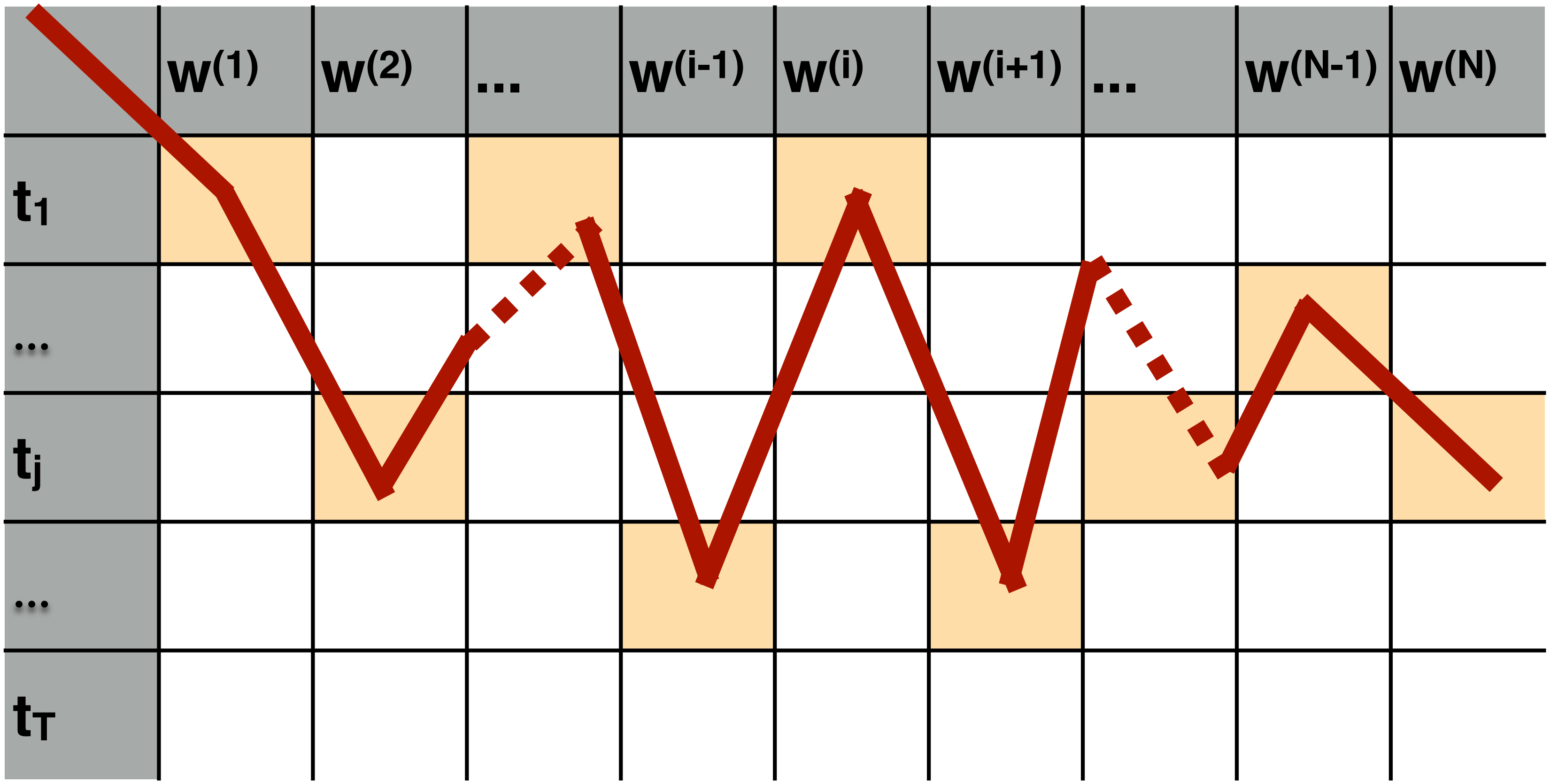
	$\mathbf{w}^{(n-1)}$	$\mathbf{w}^{(n)}$
$\mathbf{t_1}$	$P(\mathbf{w}^{(1..n-1)}, t^{(n-1)}=t_1)$	
...	...	
$\mathbf{t_i}$	$P(\mathbf{w}^{(1..n-1)}, t^{(n-1)}=t_i)$	
...	...	
$\mathbf{t_T}$	$P(\mathbf{w}^{(1..n-1)}, t^{n-1}=t_T)$	



At the end of the sentence

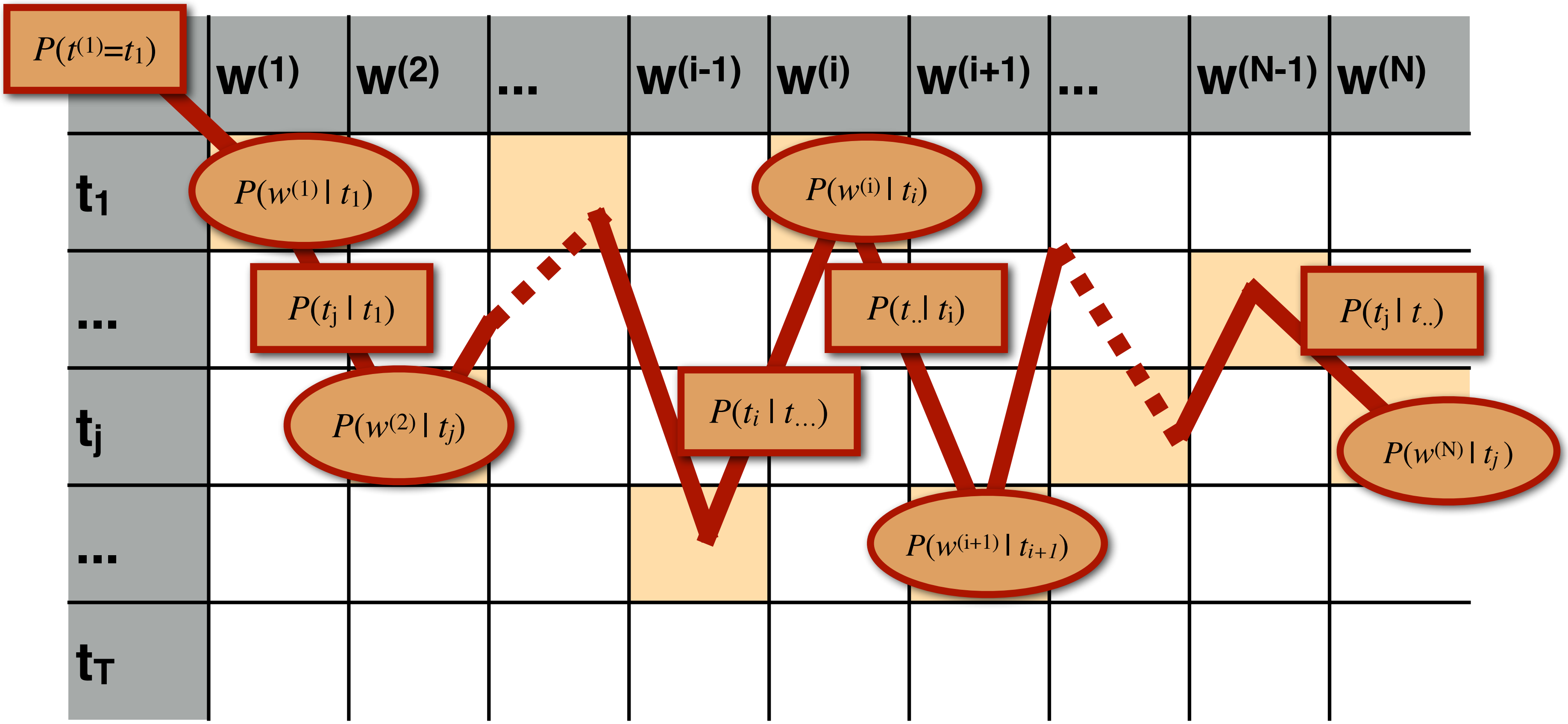
In the last column (i.e. at the end of the sentence) pick the cell with the highest entry, and trace back the backpointers to the first word in the sentence.

Retrieving $\mathbf{t}^* = \text{argmax}_{\mathbf{t}}\, P(\mathbf{t},\mathbf{w})$



By keeping **one backpointer** from each cell to the cell in the previous column that yields the highest probability, we can retrieve the most likely tag sequence when we're done.

Computing $P(\mathbf{t},\mathbf{w})$ for one tag sequence



One path through the trellis = one tag sequence

Viterbi

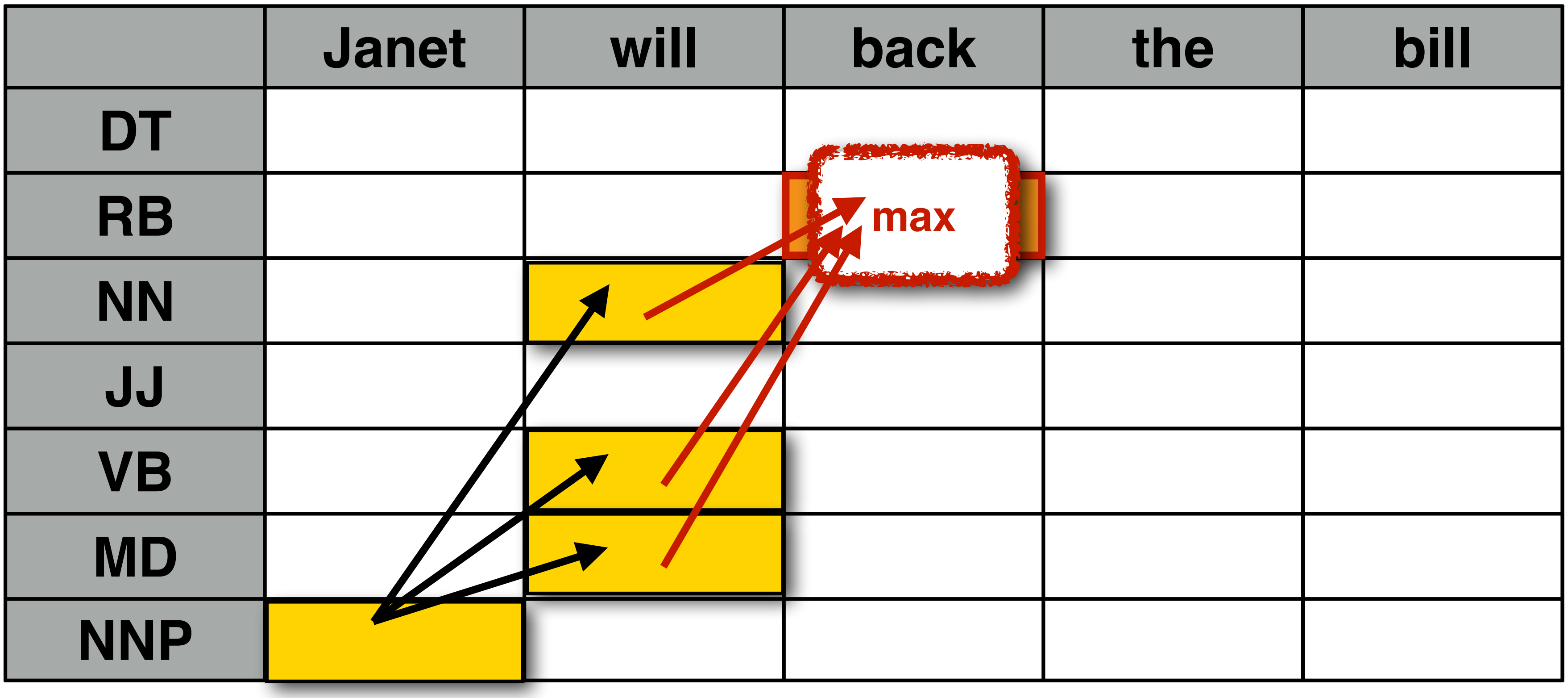
trellis[i][j].viterbi (word $w^{(j)}$, tag t_j) stores the probability of the best tag sequence for $w^{(1)}\ldots w^{(i)}$ that ends in t_j

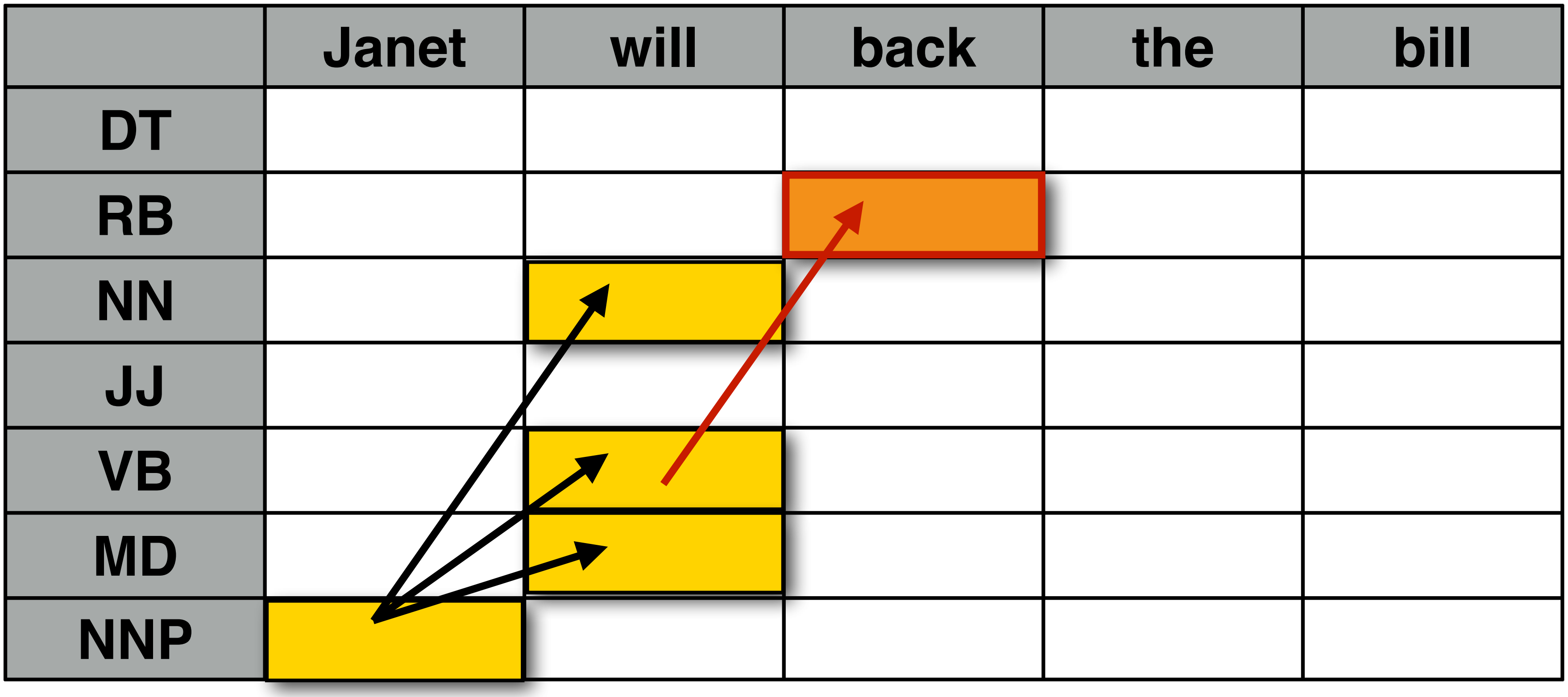
$$\text{trellis[i][j].viterbi} = \max P(w^{(1)}\ldots w^{(i)}, t^{(1)}\ldots, t^{(i)} = t_j)$$

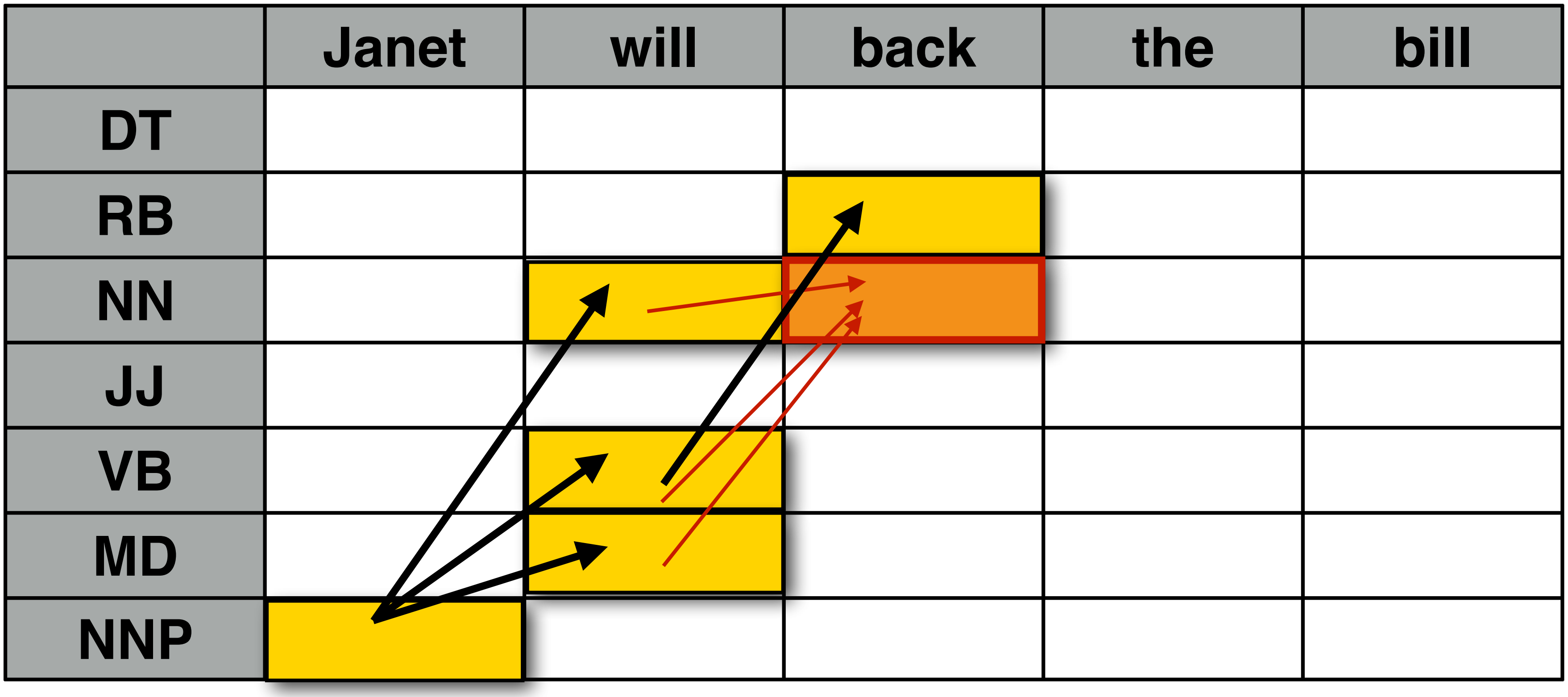
We can recursively compute trellis[i][j].viterbi from the entries in the previous column trellis[i-1][j].viterbi

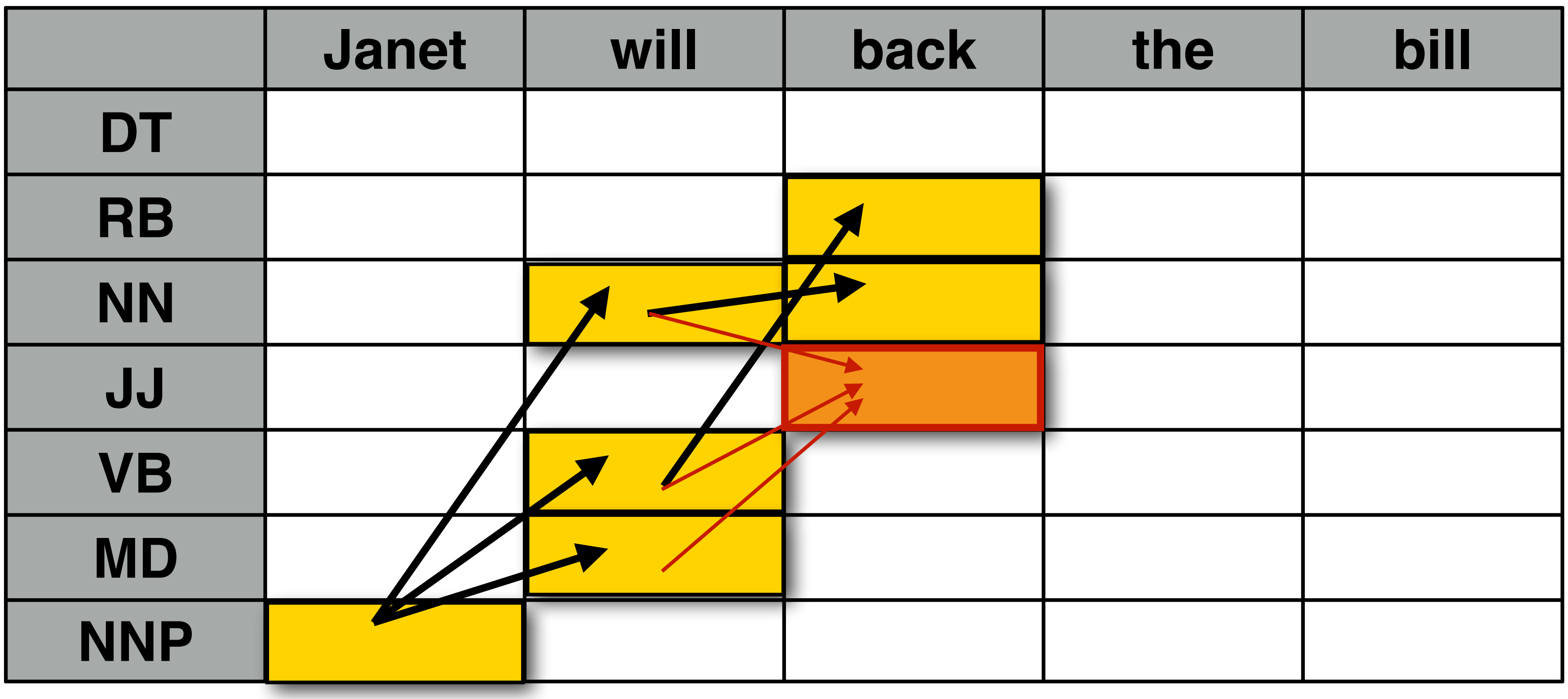
$$\text{trellis[i][j].viterbi} = P(w^{(i)} \mid t_j) \cdot \mathrm{Max}_k(\text{ trellis[i-1][k].viterbi} P(t_j \mid t_k))$$

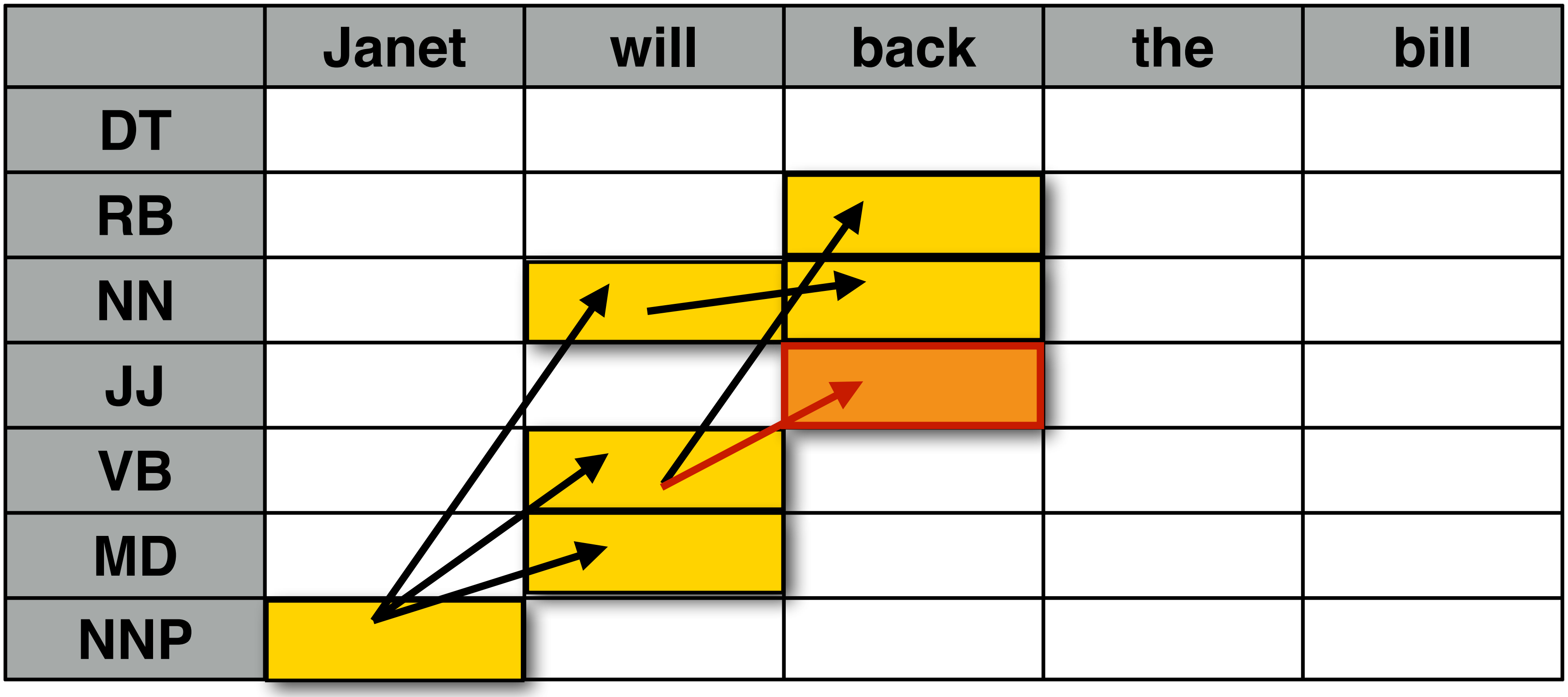
At the end of the sentence, we pick the highest scoring entry in the last column of the trellis

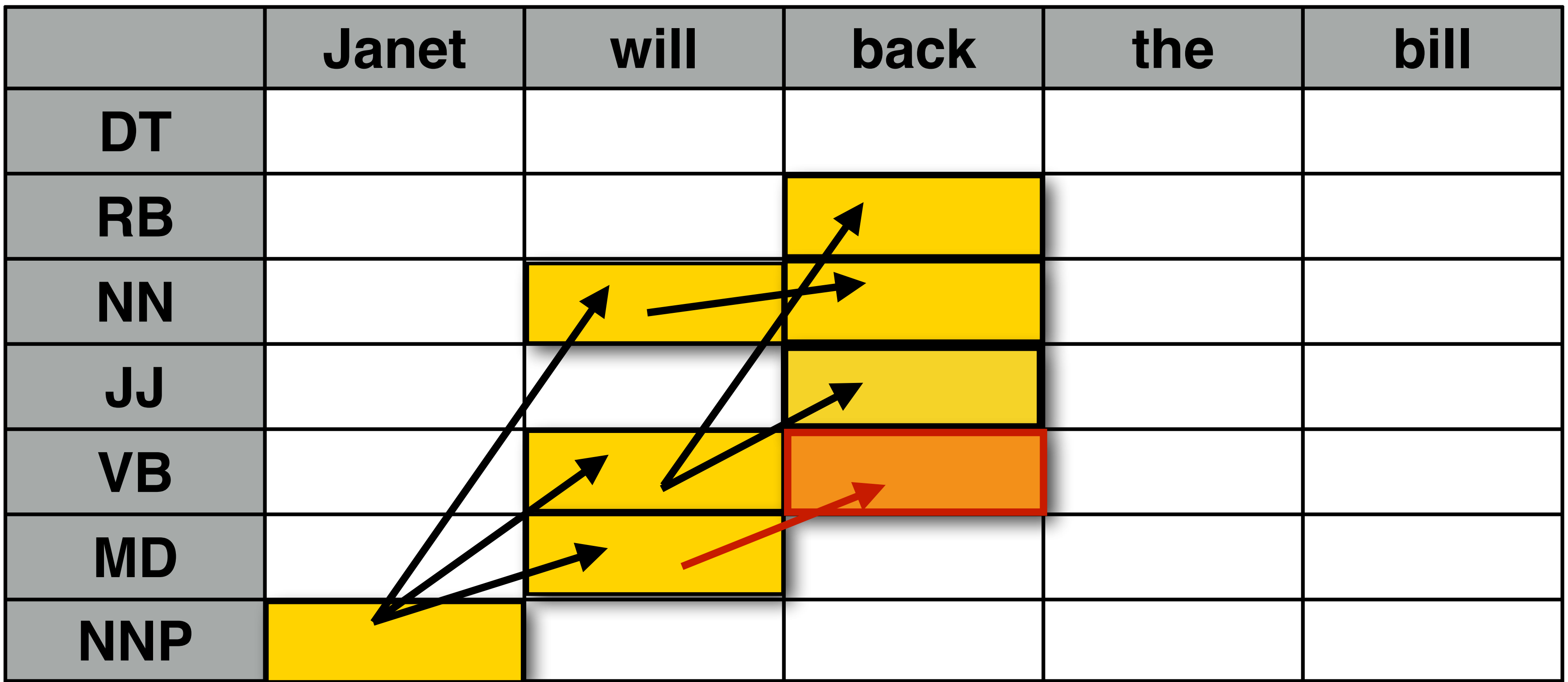

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP
max

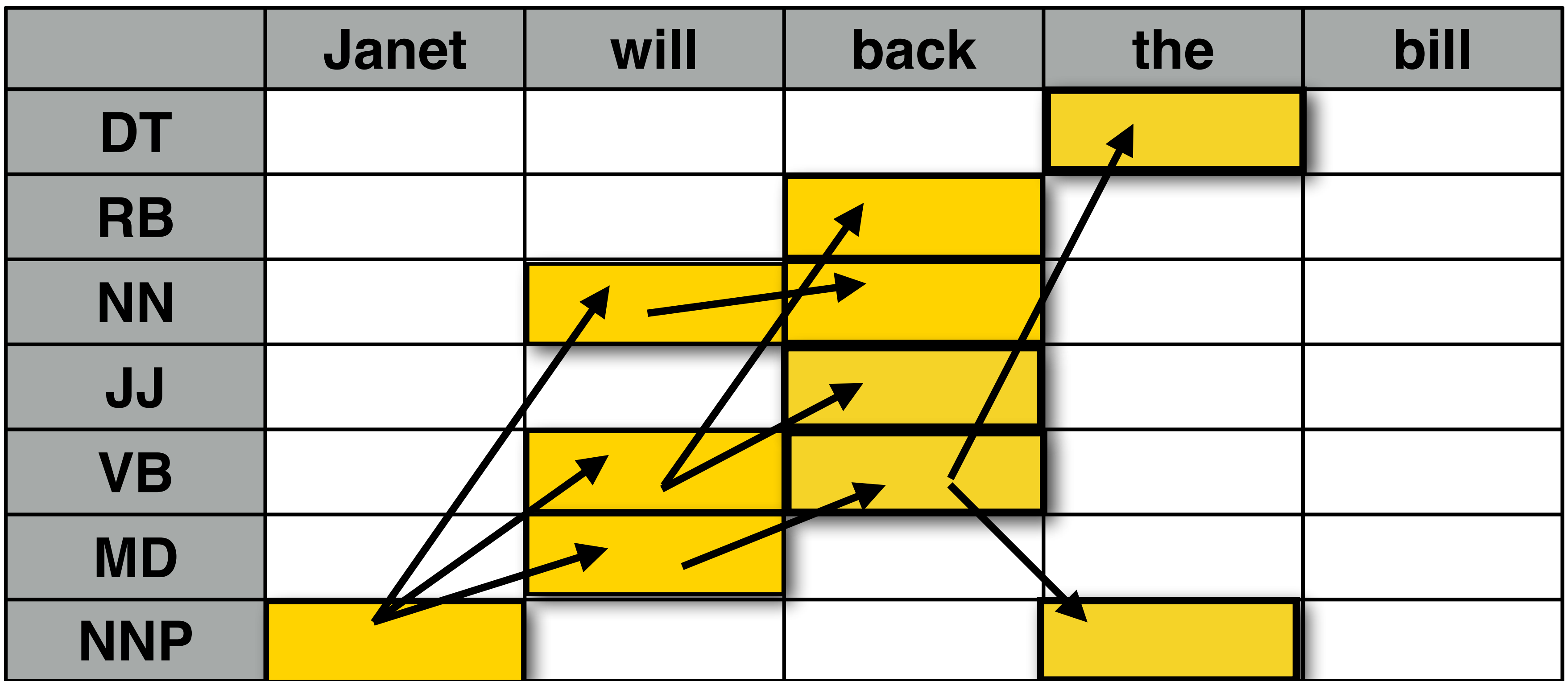

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP

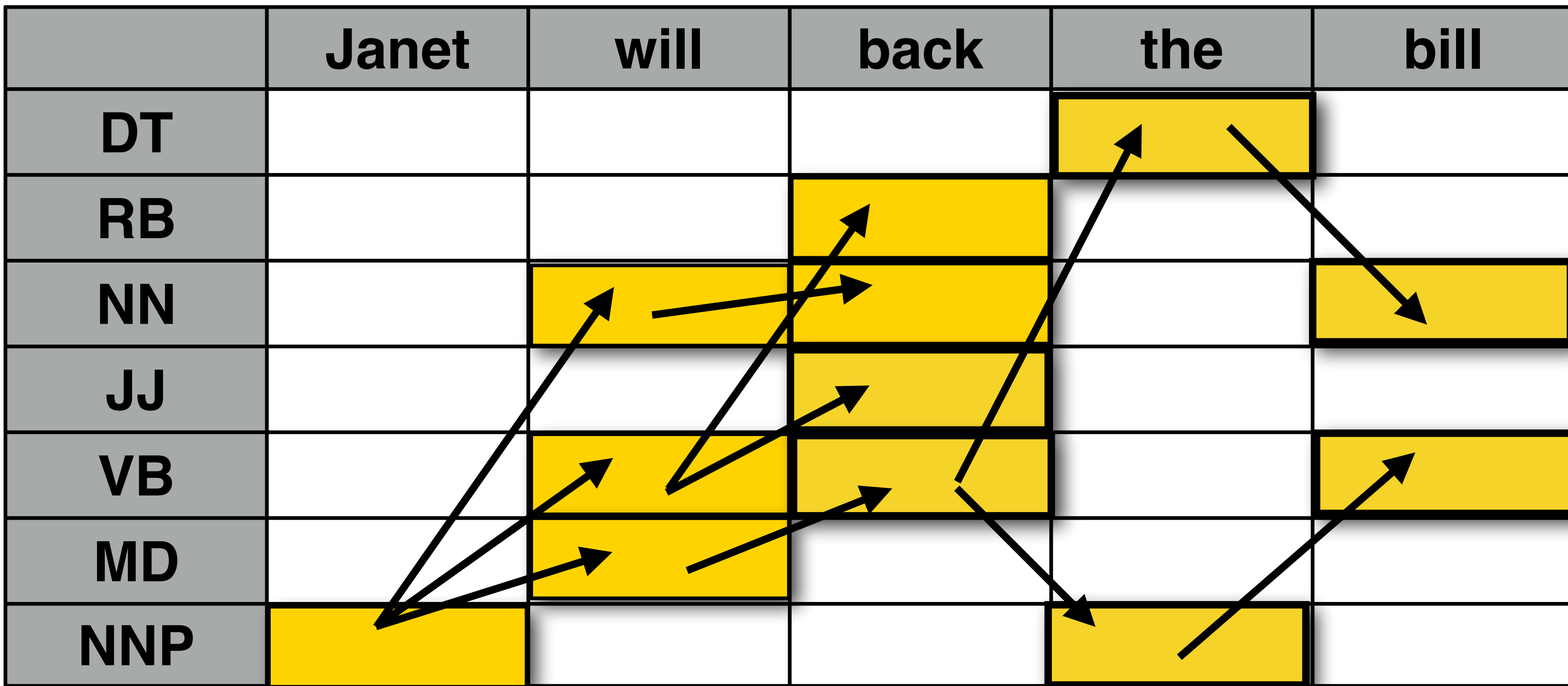

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP

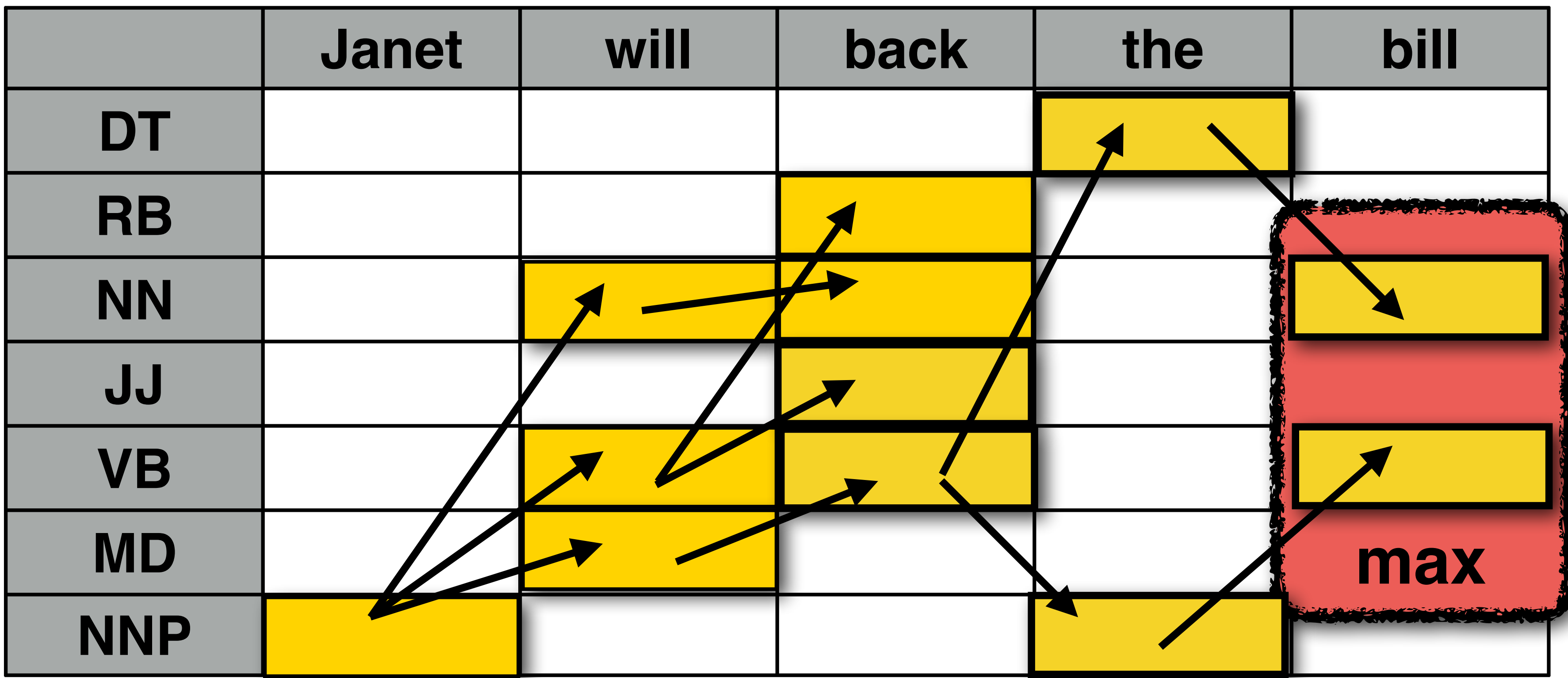

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP

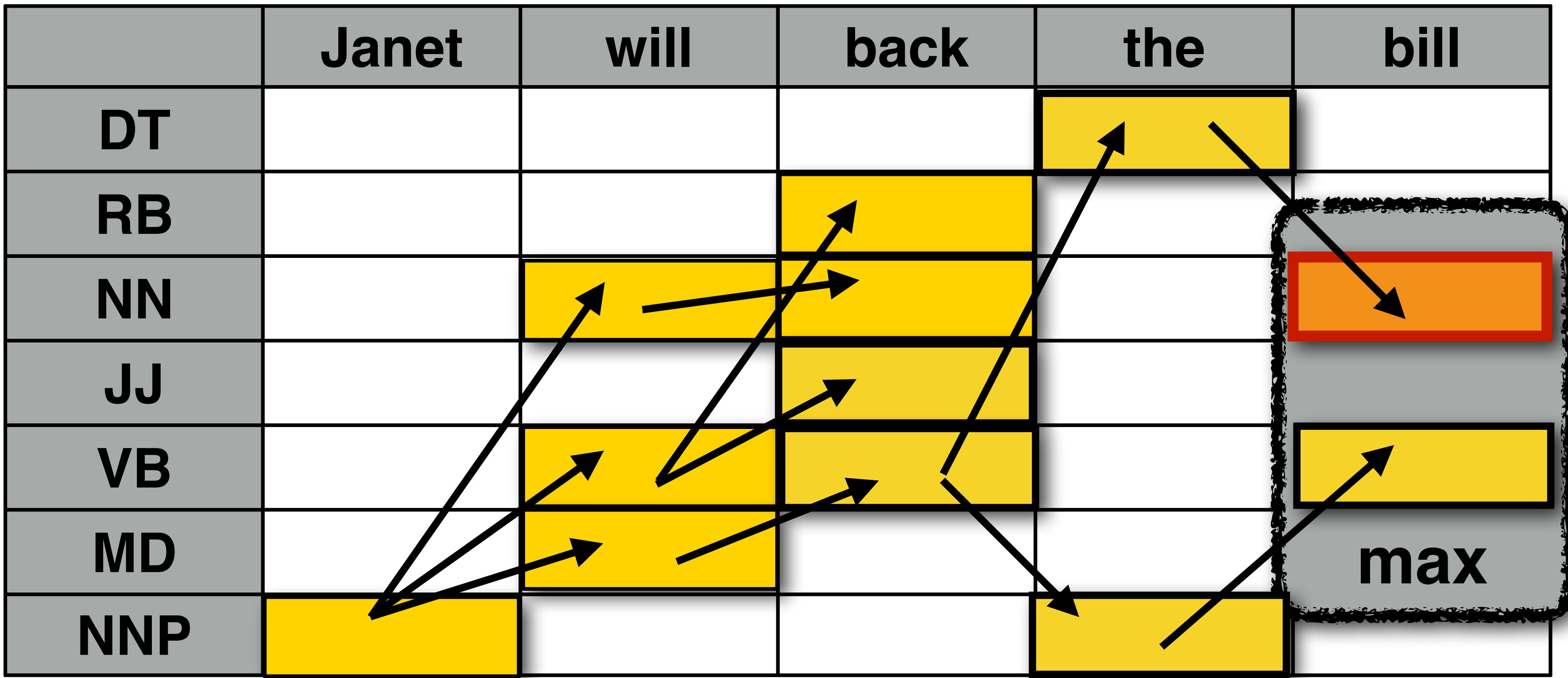

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP


Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP

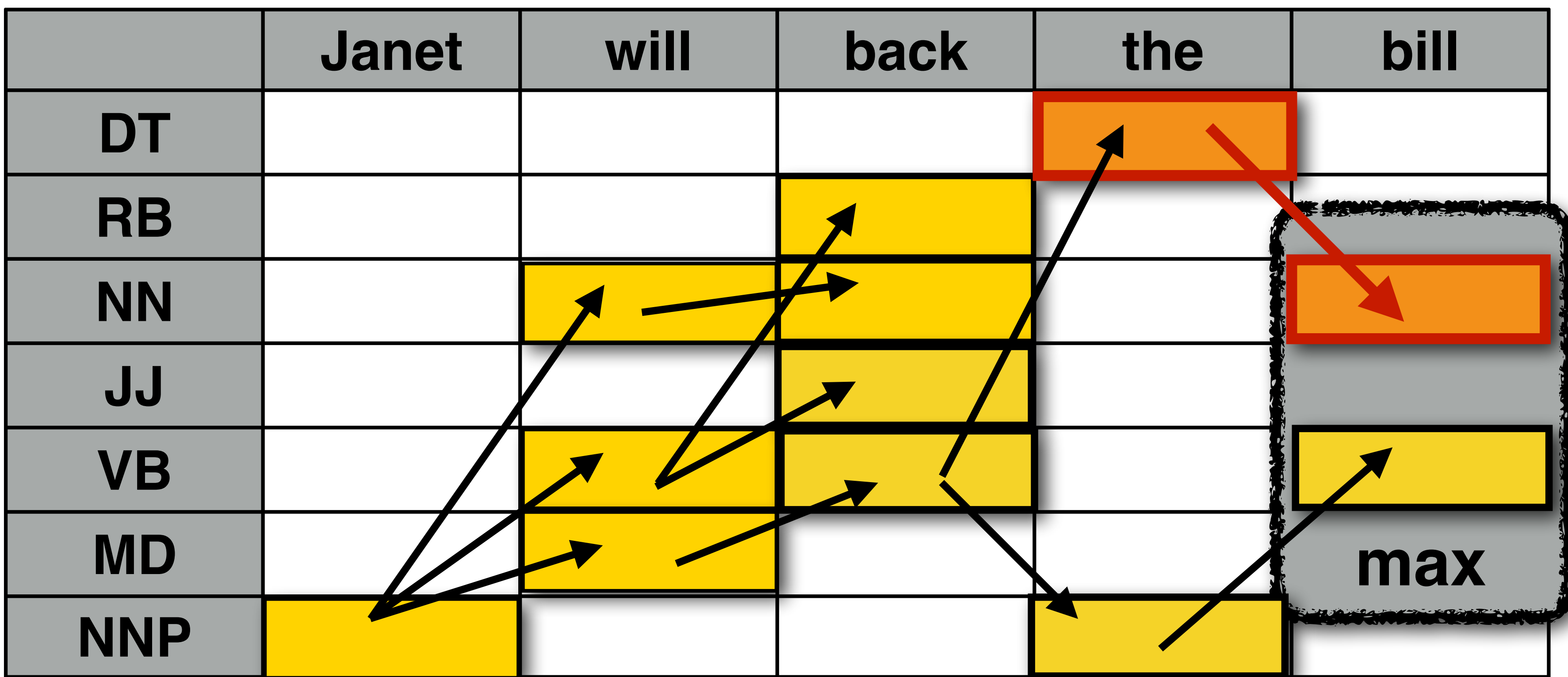

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP

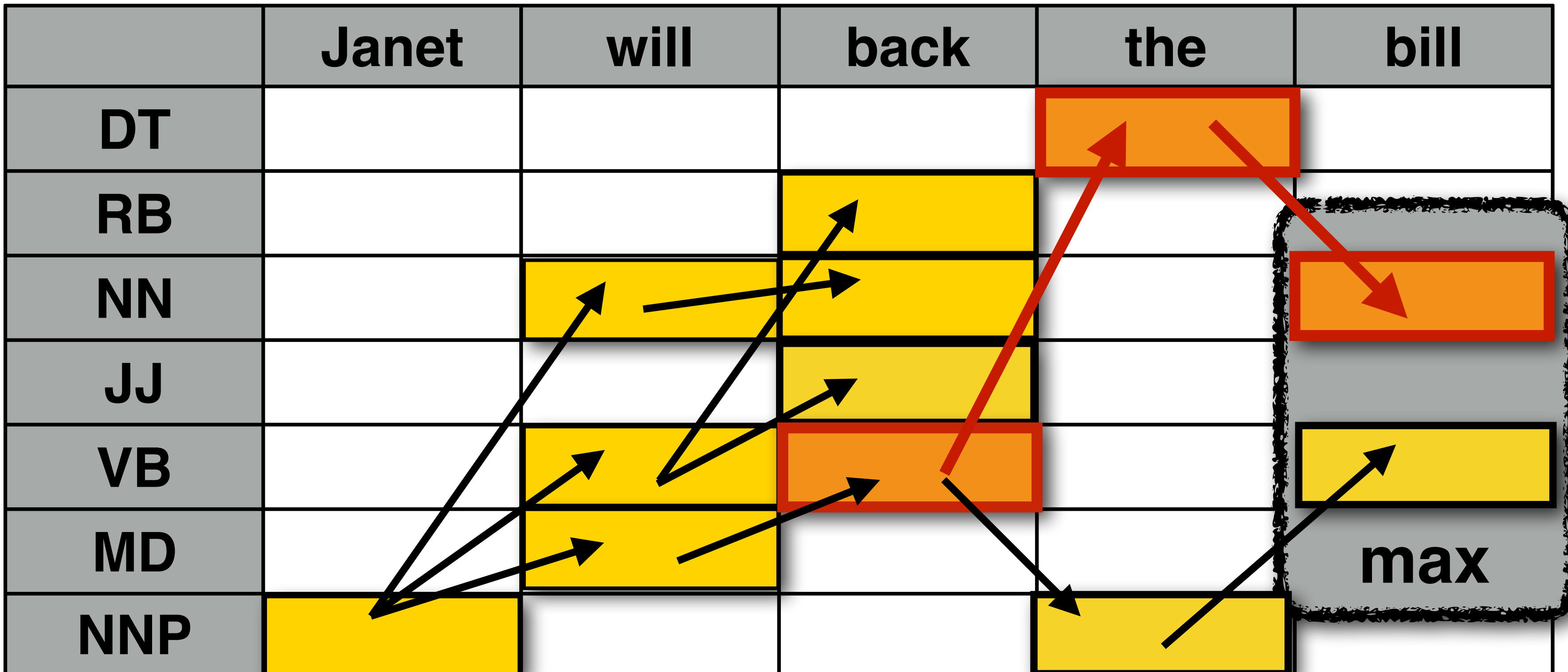

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP

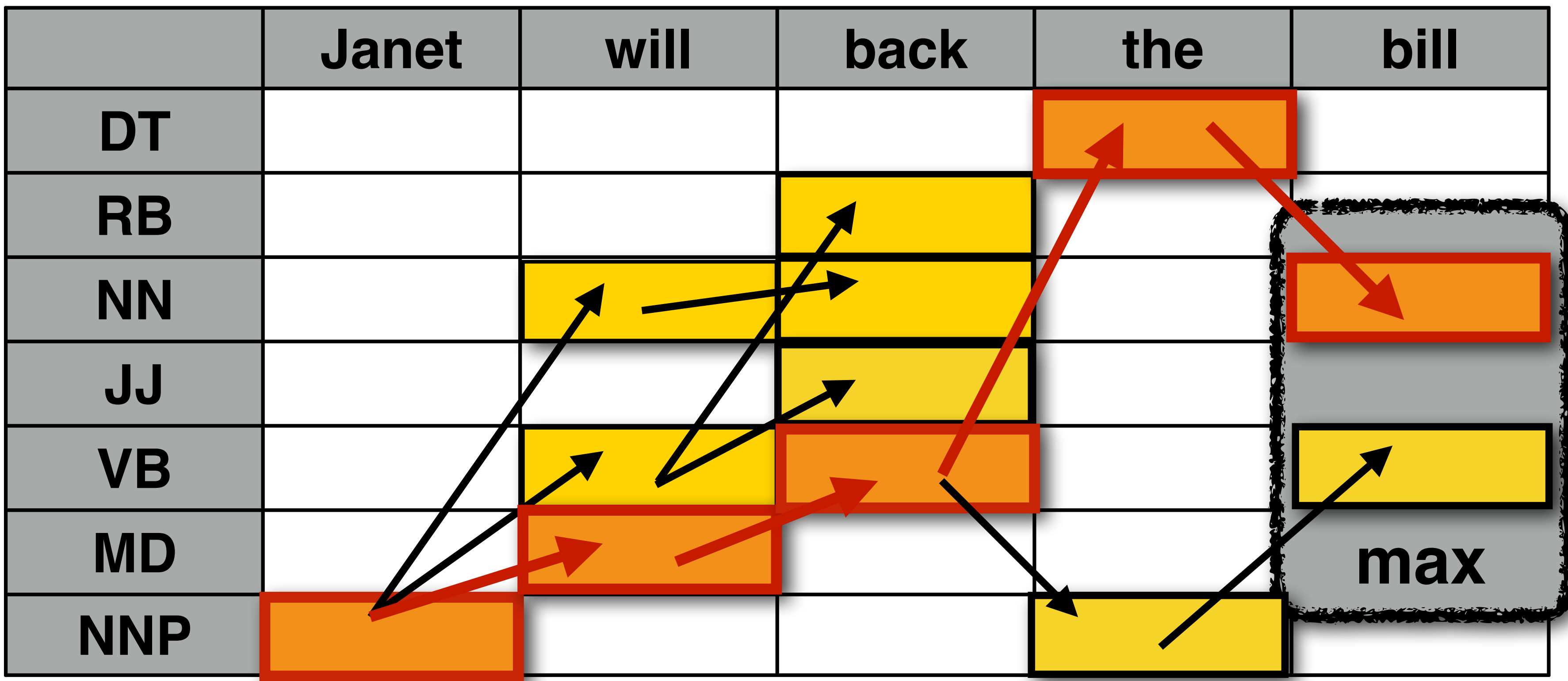

Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP
max



Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP
max


Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP
max


Janet
will
back
the
bill
DT
RB
NN
JJ
VB
MD
NNP
max



Janet_NNP will_MD back_VB the_DT bill_NN

The Viterbi algorithm

```
Viterbi( w1...n){
  for t (1...T)  // INITIALIZATION: first column
    trellis[1][t].viterbi = p_init[t] × p_emit[t][w1]
  for i (2...n){  // RECURSION: every other column
     for t (1....T){
        trellis[i][t] = 0
        for t’ (1...T){
           tmp = trellis[i-1][t’].viterbi × p_trans[t’][t]
           if (tmp > trellis[i][t].viterbi){
              trellis[i][t].viterbi = tmp
              trellis[i][t].backpointer = t’}}
        trellis[i][t].viterbi ×= p_emit[t][wi]}}
  t_max = NULL, vit_max = 0;  // FINISH: find the best cell in the last column
  for t (1...T)
     if (trellis[n][t].vit > vit_max){t_max = t; vit_max = trellis[n][t].value }
  return unpack(n, t_max);
  }
```

Viterbi

Each cell trellis[i][j] (word $w^{(i)}$ with tag t_j) **contains**:

— **The Viterbi probability** trellis[i][j].viterbi:
The maximum probability $P(w^{(1)} \ldots w^{(i)}, t^{(1)}, \ldots, t^{(i)} = t_j)$
of any tag sequence that ends in t_j for the prefix $w^{(1)\ldots(i)}$

— **A backpointer** trellis[i][j].backpointer = k*
to the cell trellis[i–1][k*] in the preceding column
that corresponds to the tag

To fill trellis[i][j], find the best cell in the previous column (trellis[i–1][k*]) based on the previous column and the transition probabilities $P(t_j \mid t_k)$

$$k^* \text{ for trellis[i][j]} := \text{Max}_k \left(\text{trellis[i–1][k]} \cdot P(t_j \mid t_k) \right)$$

The entry in trellis[i][j] includes the emission probability $P(w^{(i)} \mid t_j)$

$$\text{trellis[i][j]} := P(w^{(i)} \mid t_j) \cdot (\text{trellis[i–1][k*]} \cdot P(t_j \mid t_{k^*}))$$

We also associate a backpointer from trellis[i][j] to trellis[i–1][k*]
Finally, return the highest scoring entry in the last column of the trellis (= for the last word) and follow its backpointers

Sequence Labeling

POS tagging

```
Pierre Vinken , 61 years old , will join IBM ‘s board
as a nonexecutive director Nov. 29 .
```

```
Pierre_NNP Vinken_NNP ,_, 61_CD years_NNS old_JJ ,_,
will_MD join_VB IBM_NNP ‘s_POS board_NN as_IN a_DT
nonexecutive_JJ director_NN Nov._NNP 29_CD ._.
```

Task: assign POS tags to words

Noun phrase (NP) chunking

```
Pierre Vinken , 61 years old , will join IBM ‘s board
as a nonexecutive director Nov. 29 .
```

```
[NP Pierre Vinken] , [NP 61 years] old , will join
[NP IBM] ‘s [NP board] as [NP a nonexecutive director]
[NP Nov. 2] .
```

Task: identify all non-recursive NP chunks

The BIO encoding

We define three new tags:

– **B-NP**: beginning of a noun phrase chunk
– **I-NP**: inside of a noun phrase chunk
– **O**: outside of a noun phrase chunk

```
[NP Pierre Vinken] , [NP 61 years] old , will join
[NP IBM] ‘s [NP board] as [NP a nonexecutive director]
[NP Nov. 2] .
```

```
Pierre_B-NP Vinken_I-NP ,_O 61_B-NP years_I-NP
old_O ,_O will_O join_O IBM_B-NP ‘s_O board_B-NP as_O
a_B-NP nonexecutive_I-NP director_I-NP Nov._B-NP
29_I-NP ._O
```

Shallow parsing

```
Pierre Vinken , 61 years old , will join IBM ‘s board
as a nonexecutive director Nov. 29 .
```

```
[NP Pierre Vinken] , [NP 61 years] old , [VP will join]
[NP IBM] ‘s [NP board] [PP as] [NP a nonexecutive
director] [NP Nov. 2] .
```

Task: identify all non-recursive NP,
verb (“VP”) and preposition (“PP”) chunks

The BIO encoding for shallow parsing

We define several new tags:

- **B-NP B-VP B-PP**: beginning of an NP, “VP”, “PP” chunk
- **I-NP I-VP I-PP**: inside of an NP, “VP”, “PP” chunk
- **O**: outside of any chunk

```
[NP Pierre Vinken] , [NP 61 years] old , [VP will join]
[NP IBM] ‘s [NP board] [PP as] [NP a nonexecutive
director] [NP Nov. 2] .
```

```
Pierre_B-NP Vinken_I-NP ,_O 61_B-NP years_I-NP
old_O ,_O will_B-VP join_I-VP IBM_B-NP ‘s_O board_B-NP
as_B-PP a_B-NP nonexecutive_I-NP director_I-NP Nov._B-
NP 29_I-NP ._O
```

Named Entity Recognition

```
Pierre Vinken , 61 years old , will join IBM ‘s board
as a nonexecutive director Nov. 29 .
```

```
[PERS Pierre Vinken] , 61 years old , will join
[ORG IBM] ‘s board as a nonexecutive director
[DATE Nov. 2] .
```

Task: identify all mentions of named entities (people, organizations, locations, dates)

The BIO encoding for NER

We define many new tags:

– **B-PERS**, **B-DATE, ...:** beginning of a mention of a person/ date...

– **I-PERS**, **I-DATE, ...:** inside of a mention of a person/date...

```
[PERS Pierre Vinken] , 61 years old , will join
[ORG IBM] ‘s board as a nonexecutive director
[DATE Nov. 2] .
```

```
Pierre_B-PERS Vinken_I-PERS ,_O 61_O years_O old_O ,_O
will_O join_O IBM_B-ORG ‘s_O board_O as_O a_O
nonexecutive_O director_O Nov._B-DATE 29_I-DATE ._O
```

Sequence Labeling

Input: a sequence of *n* tokens/words:

`Pierre Vinken , 61 years old , will join IBM ‘s board as a nonexecutive director Nov. 29`

Output: a sequence of *n* labels, such that each token/word is associated with a label:

POS-tagging: Pierre_**NNP** Vinken_**NNP** ,_**,** 61_**CD** years_**NNS** old_**JJ** ,_**,** will_**MD** join_**VB** IBM_**NNP** ‘s_**POS** board_**NN** as_**IN** a_**DT** nonexecutive_**JJ** director_**NN** Nov._**NNP** 29_**CD** ._**.**

Named Entity Recognition: Pierre_**B-PERS** Vinken_**I-PERS** ,_**O** 61_**O** years_**O** old_**O** ,_**O** will_**O** join_**O** IBM_**B-ORG** ‘s_**O** board_**O** as_**O** a_**O** nonexecutive_**O** director_**O** Nov._**B-DATE** 29_**I-DATE** ._**O**

BIO encodings in general

BIO encoding can be used to frame any task that requires the identification of non-overlapping and non-nested text spans as a sequence labeling problem, e.g.:

- NP chunking
- Shallow Parsing
- Named entity recognition

Sequence labeling algorithms

Statistical models:

— Maximum Entropy Markov Models (MEMMs)

— Conditional Random Fields (CRFs)

Neural models:

— Recurrent networks (or transformers) that predict a label at each time step, possibly with a CRF output layer.

Maximum Entropy Markov Models

MEMMs use a **logistic regression** ("Maximum Entropy") classifier for each $P(t^{(i)} | w^{(i)}, t^{(i-1)})$

$$P(t^{(i)} = t_k \mid t^{(i-1)}, w^{(i)}) = \frac{\exp(\sum_j \lambda_{jk} f_j(t^{(i-1)}, w^{(i)}))}{\sum_l \exp(\sum_j \lambda_{jl} f_j(t^{(i-1)}, w^{(i)}))}$$

Here, $t^{(i)}$: label of the i-th word vs. t_i = i-th label in the inventory

This requires the definition of a **feature function** $f(t^{(i-1)}, w^{(i)})$ that returns an n-dimensional feature vector for predicting label $t^{(i)}=t_j$ given inputs $t^{(i-1)}$ and $w^{(i)}$

Training returns weights λ_{jk} for each feature j used to predict label t_k

Conditional Random Fields (CRFs)

Conditional Random Fields have the same mathematical definition as MEMMs, but:

— CRFS are trained globally to maximize the probability of the overall sequence,
— MEMMs are trained locally to maximize the probability of each individual label

This requires dynamic programming
- Training: akin to the Forward-Backward algorithm used to train HMMs from unlabeled sequences)
- Decoding: Viterbi