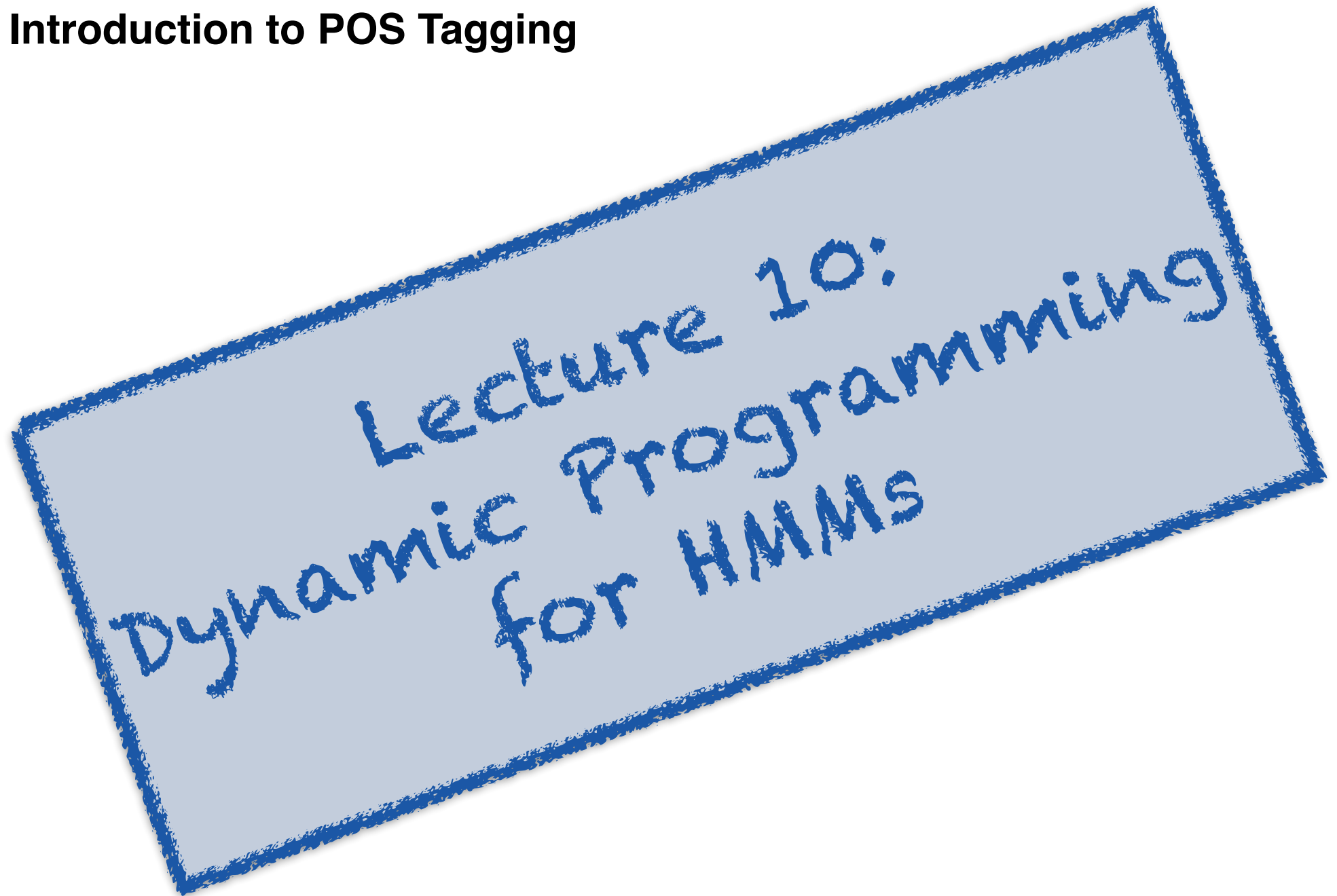


## Lecture 10: Introduction to POS Tagging



# HMM decoding (Viterbi)

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

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

We want to use an HMM tagger to find its POS tags  $\mathbf{t}$

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

$$= \operatorname{argmax}_{\mathbf{t}} P(t^{(1)}) \cdot P(w^{(1)} | t^{(1)}) \cdot P(t^{(2)} | t^{(1)}) \cdot \dots \cdot P(w^{(N)} | t^{(N)})$$

*But: with  $T$  tags,  $\mathbf{w}$  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  $\mathbf{w}$ :  $\mathbf{t}^* = \operatorname{argmax}_{\mathbf{t}} P(\mathbf{w} | \mathbf{t})P(\mathbf{t})$

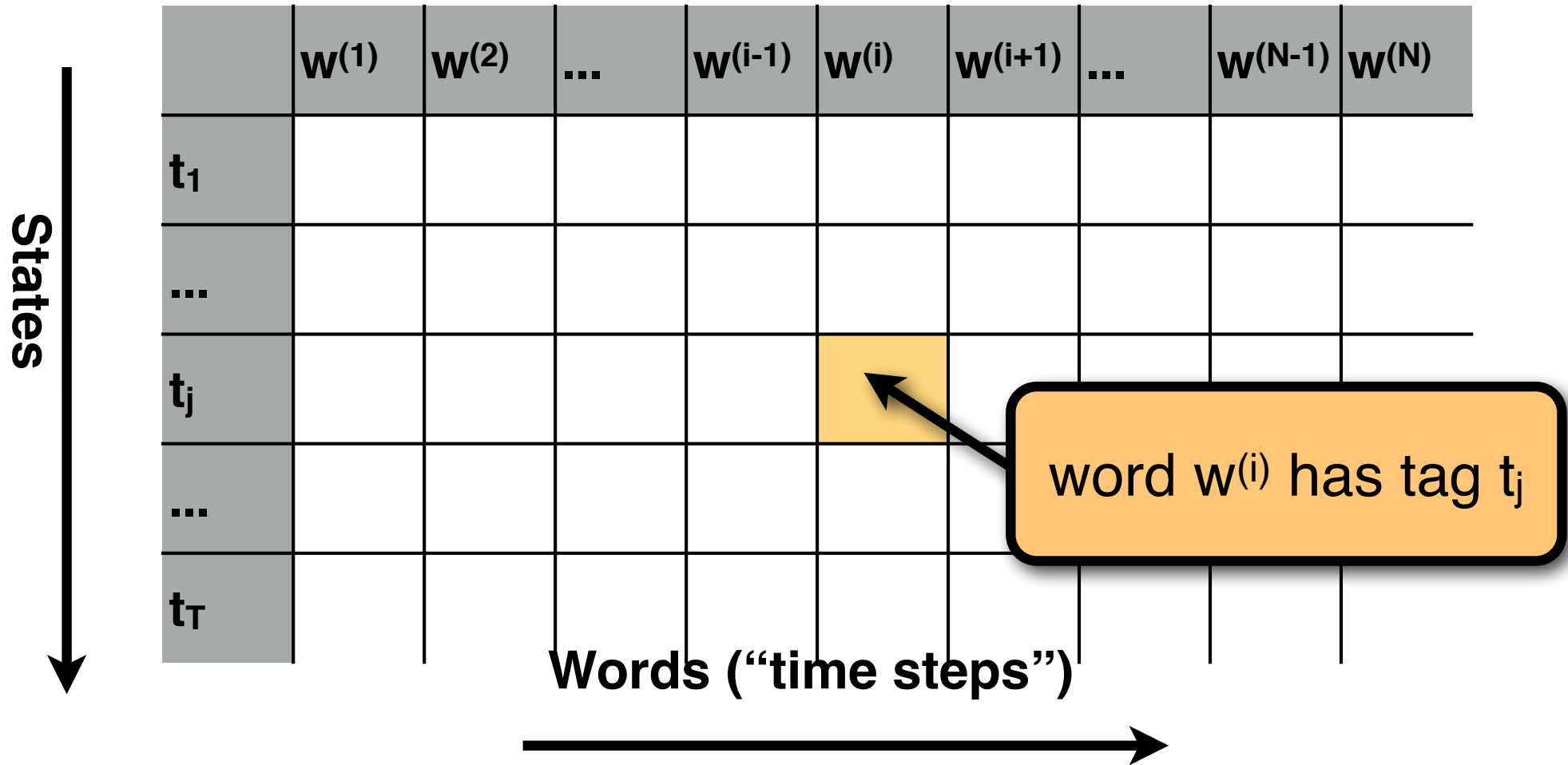
Complexity: linear in the sentence length.

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  $\mathbf{t}^* = \operatorname{argmax}_{\mathbf{t}} P(\mathbf{w} | \mathbf{t})P(\mathbf{t})$ ) into smaller subproblems.

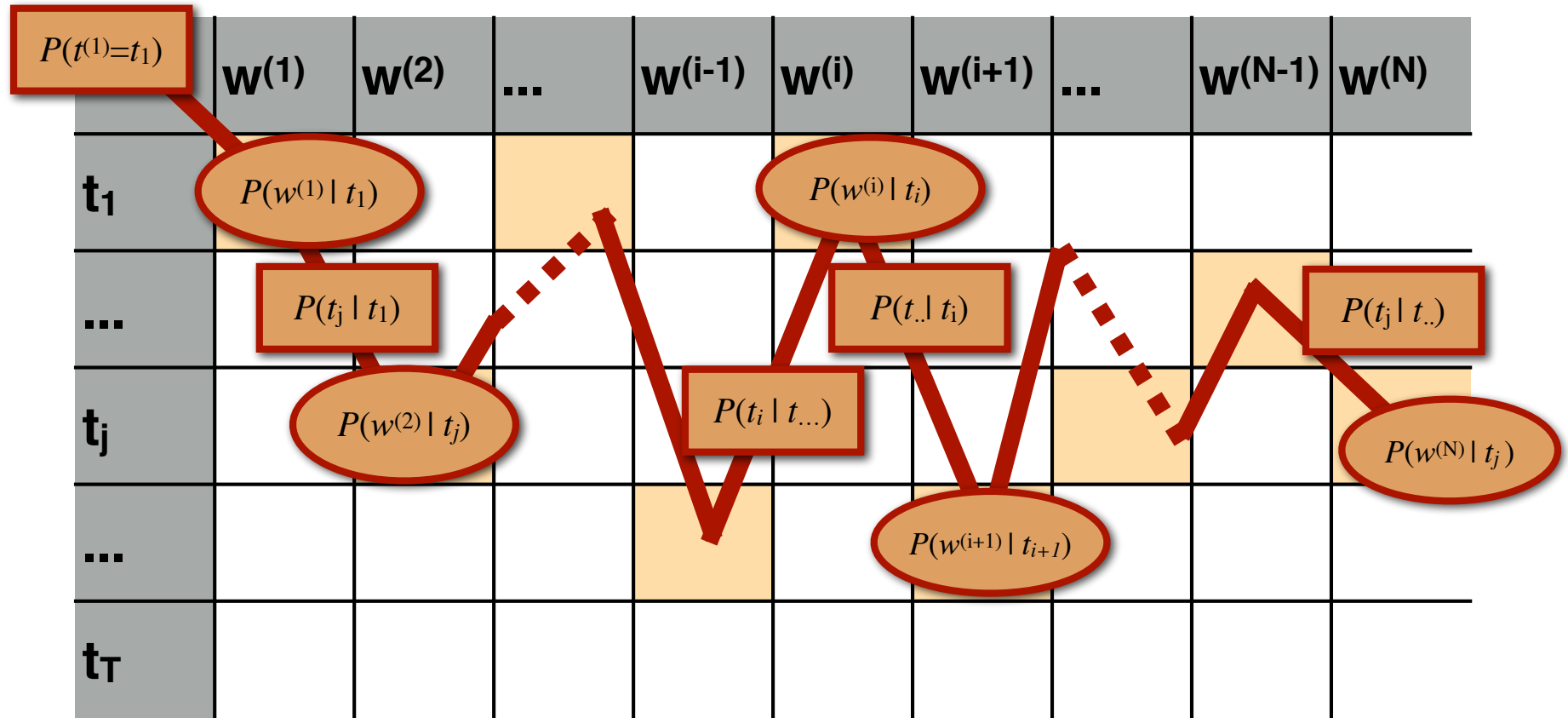
The data structure used to store the solution of these subproblems is the **trellis**.

# Bookkeeping: the trellis



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

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



One path through the trellis = one tag sequence

# Viterbi: Basic Idea

**Task:** Find the tag sequence  $t^{(1)} \dots t^{(N-1)} t^{(N)}$  that maximizes the joint probability

$$\pi(t^{(1)}) P(w^{(1)} | t^{(1)}) \prod_{i=2}^N P(t^{(i)} | t^{(i-1)}) P(w^{(i)} | t^{(i)})$$

The choice of  $t^{(1)}$  affects the probability of  $t^{(2)}$ , which in turn affects the probability of  $t^{(3)}$ , etc:

$$\pi(t^{(1)}) P(w^{(1)} | t^{(1)}) P(t^{(2)} | t^{(1)}) P(w^{(2)} | t^{(2)}) P(t^{(3)} | t^{(2)}) \dots$$

→ We **cannot fix**  $t^{(1)}$  (or any tag) until the **end of the sentence!**

# Exploiting the independence assumptions

You want to find the best tag sequence  $t^{(1)}t^{(2)}t^{(3)} \dots = t_i t_j t_k \dots$

$$\operatorname{argmax}_{t_i, t_j, t_k, \dots} \pi(t^{(1)} = t_i) P(w^{(1)} | t^{(1)} = t_i) P(t^{(2)} = t_j | t^{(1)} = t_i) P(w^{(2)} | t^{(2)} = t_j) P(t^{(3)} = t_k | t^{(2)} = t_j) P(w^{(3)} | t^{(3)} = t_k) \dots$$

This depends  
only on the choice of  $t^{(1)} = t_i$

This depends *only*  
on the choices of  
 $t^{(1)} = t_i$  and  $t^{(2)} = t_j$

This depends  
only on the choice  
of  $t^{(2)} = t_j$

This depends *only* on  
the choices of  
 $t^{(2)} = t_j$  and  $t^{(3)} = t_k$

This depends  
only on the choice  
of  $t^{(3)} = t_k$

Step 1: For any particular choice of  $t^{(1)} = t_i$  for  $w^{(1)}$ , compute  $\pi(t^{(1)} = t_i) P(w^{(1)} | t^{(1)} = t_i)$

Step 2a): for any particular choice of  $t^{(2)} = t_j$  for  $w^{(2)}$ , pick the tag  $t_i$  for  $w^{(1)}$  that gives the highest probability to  $\operatorname{argmax}_{t_i} \pi(t^{(1)} = t_i) P(w^{(1)} | t^{(1)} = t_i) P(t^{(2)} = t_j | t^{(1)} = t_i)$

Step 2b):  
Compute  $P(w^{(2)} | t^{(2)} = t_j)$

Step 3: You have already found the best  $t_i$  for any  $t^{(2)} = t_j$ .

Now, for any particular choice of  $t^{(3)} = t_k$  for  $w^{(3)}$ ,

pick the tag  $t_j$  for  $w^{(2)}$  that gives the highest probability to

$$\operatorname{argmax}_{t_j} \pi(t^{(1)} = t_i) P(w^{(1)} | t^{(1)} = t_i) P(t^{(2)} = t_j | t^{(1)} = t_i) P(w^{(2)} | t^{(2)} = t_j) P(t^{(3)} = t_k | t^{(2)} = t_j)$$

For all words  $i = 1..N$   
in the sentence:

For all tags  $j = 1..T$   
in the tag set:

Find the best  
tag sequence  $t^{(1) \dots (i)}$   
that ends in  $t^{(i)} = t_j$



# Viterbi: Basic Idea

Assume we knew (for any tag  $t_j$ ) **the maximum probability** of any *complete* sequence  $t^{(1)} \dots t^{(N)}$  that **ends in that tag**  $t^{(N)} = t_j$  [ $N$ : last word in  $\mathbf{w}$ ]

Call that probability the **Viterbi probability** of tag  $t_j$  at position  $N$ , and store it as **trellis[N][j].viterbi**

Then, the **probability of the best tag sequence** (i.e. the maximum probability of *any* complete sequence  $t^{(1)} \dots t^{(N)}$ ) **for our sentence** is  
 **$\max_{k \in \{1, \dots, T\}} (\text{trellis}[N][k].\text{viterbi})$**

# Viterbi: Basic Idea

**Viterbi probability** of tag  $t_j$  for word  $w^{(i)}$ : **trellis[i][j].viterbi**

The highest probability  $P(\mathbf{w}^{(1) \dots (i)}, \mathbf{t}^{(1) \dots (i)})$  of the prefix  $\mathbf{w}^{(1) \dots (i)}$  and *any* tag sequence  $\mathbf{t}^{(1) \dots (i)}$  ending in  $t^{(i)} = t_j$

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

The probability of the best tag sequence overall is given by:

$$\max_k \text{trellis}[N][k].\text{viterbi}$$

(the largest entry in the last column of the trellis)

The Viterbi probability  $\text{trellis}[i][j].\text{viterbi}$  (for any cell in the trellis) can easily be computed based on the cells in the preceding column,  $\text{trellis}[i-1][k].\text{viterbi}$

# Viterbi: Basic Idea

**Viterbi probability** of tag  $t_j$  for word  $w^{(i)}$ : **trellis[i][j].viterbi**

The highest probability  $P(\mathbf{w}^{(1)...(i)}, \mathbf{t}^{(1)...(i)})$  of the prefix  $\mathbf{w}^{(1)...(i)}$  and *any* tag sequence  $\mathbf{t}^{(1)...(i)}$  ending in  $t^{(i)} = t_j$

**Base case: First word  $w^{(1)}$  in the sentence**

$$\text{trellis}[1][j].\text{viterbi} = \pi(t_j)P(w^{(1)} | t_j)$$

Initial probability for tag  $t_j$   
Emission probability for  $w^{(1)}$

**Recurrence: Any other word  $w^{(i)}$  in the sentence**

$$\text{trellis}[i][j].\text{viterbi} = \max_k \left( \text{trellis}[i-1][k].\text{viterbi} \times P(t_j | t_k)P(w^{(i)} | t_j) \right)$$

Viterbi probability of tag  $t_k$  for the preceding word  $w^{(i-1)}$       Transition prob. for  $t_j$  given  $t_k$       Emission prob. for  $w^{(i)}$  given  $t_j$

# Initialization

For a bigram HMM:

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

In the first column, initialize each cell  $\text{trellis}[1][k]$  as  
$$\text{trellis}[1][k] := \pi(t_k)P(w^{(1)} \mid t_k)$$

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

# Viterbi: filling in the first column

	$w^{(1)}$
<b>DT</b>	$\pi(\text{DT}) \times P(w^{(1)} \mid \text{DT})$
...	
<b>NNS</b>	$\pi(\text{NNS}) \times P(w^{(1)} \mid \text{NNS})$
...	
<b>VBZ</b>	$\pi(\text{VBZ}) \times P(w^{(1)} \mid \text{VBZ})$

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

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

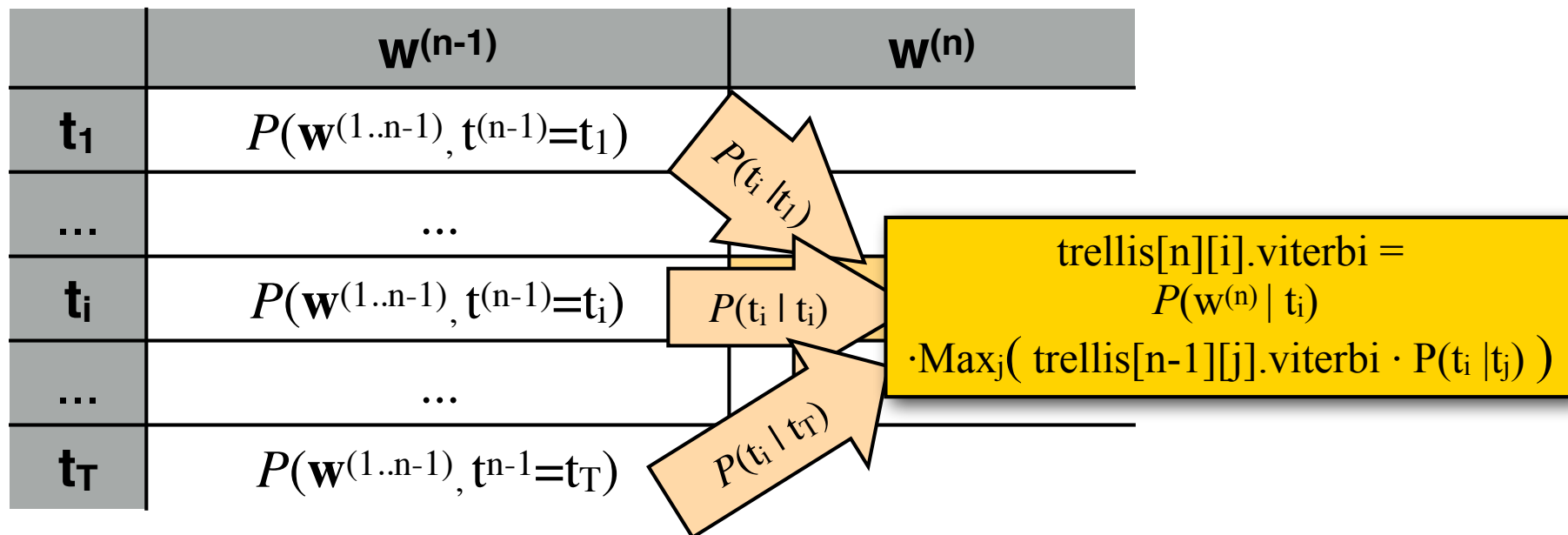
We want to find the best (most likely) tag sequence for the entire sentence.

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  $t_j$

# 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

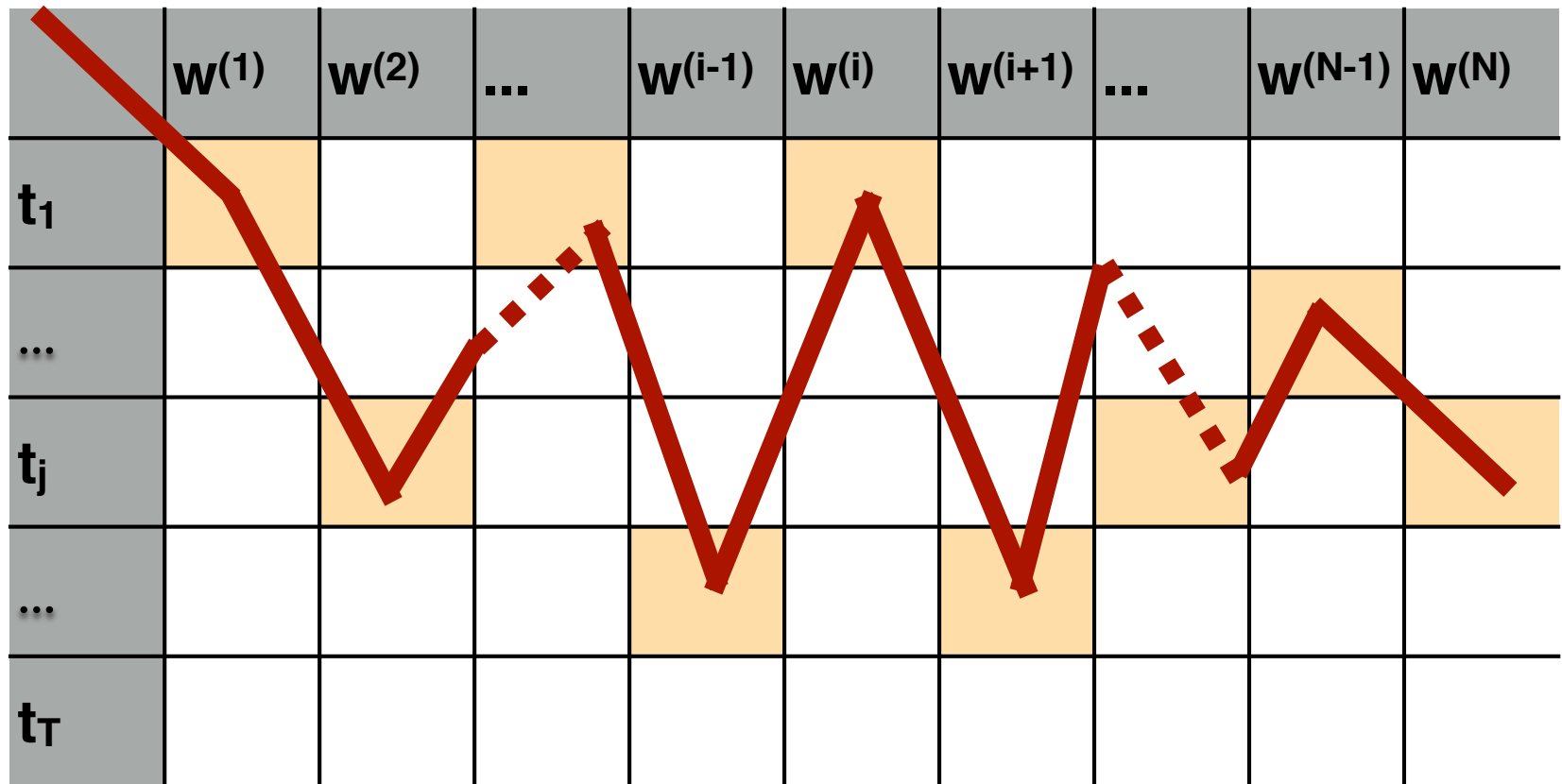


# At the end of the sentence

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 $t^* = \operatorname{argmax}_t P(t, w)$



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.



# Viterbi

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

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

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

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

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

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

# Viterbi

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

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

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

`trellis[i][j].viterbi` =

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

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					

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

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

The diagram illustrates dependencies between the words "Janet", "will", and "back" and their corresponding parts of speech (NNP, VB, NN) in the sentence "Janet will back the bill".

- Black Arrows (Grammatical Dependencies):**
  - From **NNP** (Janet) to **VB** (will).
  - From **VB** (will) to **NN** (back).
  - From **NN** (back) to **DT** (the).
- Red Arrows (Semantic Dependencies):**
  - From **NNP** (Janet) to **NN** (back).
  - From **VB** (will) to **NN** (back).
  - From **NN** (back) to **DT** (the).

	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					

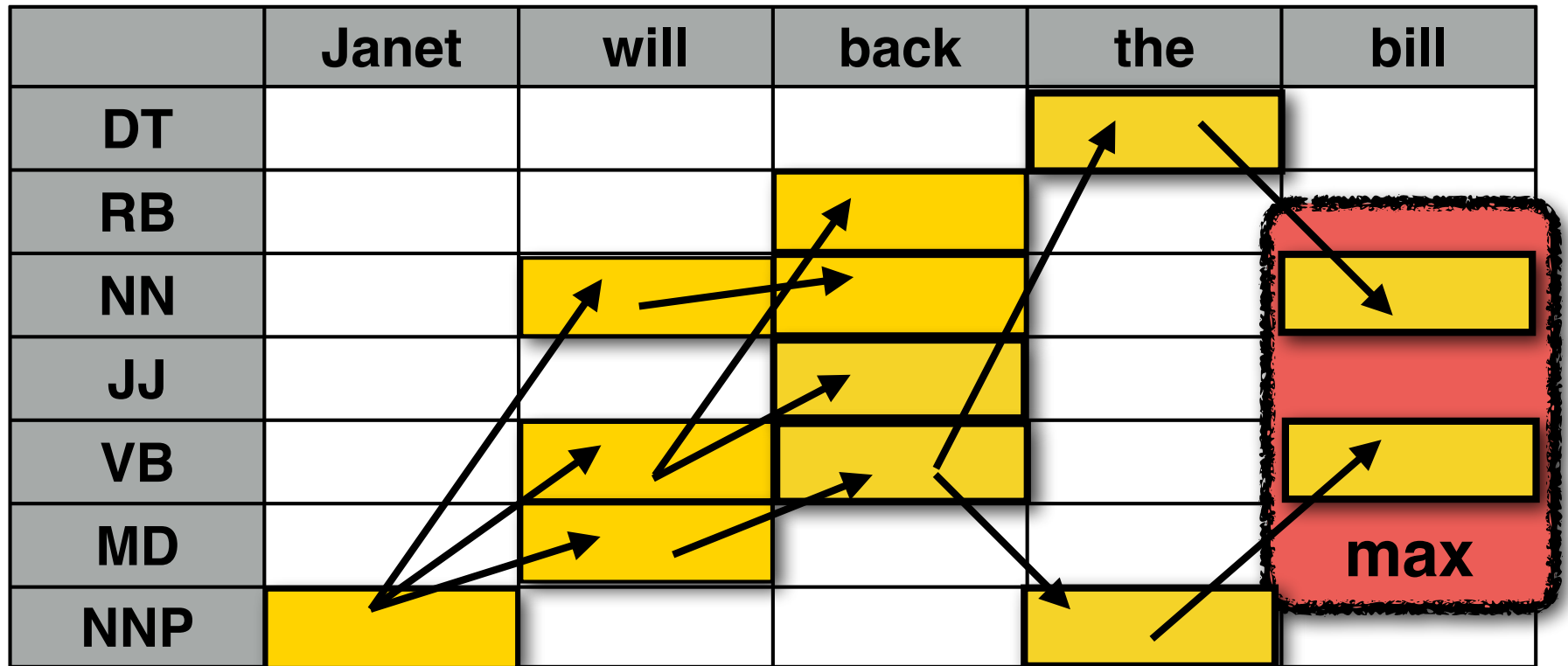


	Janet	will	back	the	bill
DT					
RB					
NN					
JJ					
VB					
MD					
NNP					

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

The diagram illustrates a parsing process using arrows from the NNP row to various cells in the table:

- An arrow from the NNP cell under "Janet" points to the NN cell under "will".
- An arrow from the NNP cell under "Janet" points to the VB cell under "will".
- An arrow from the NNP cell under "Janet" points to the MD cell under "will".
- An arrow from the NNP cell under "Janet" points to the DT cell under "the".
- An arrow from the NNP cell under "Janet" points to the NN cell under "bill".
- An arrow from the NN cell under "will" points to the RB cell under "back".
- An arrow from the VB cell under "will" points to the RB cell under "back".
- An arrow from the MD cell under "will" points to the RB cell under "back".
- An arrow from the DT cell under "the" points to the RB cell under "back".
- An arrow from the RB cell under "back" points to the NN cell under "bill".
- An arrow from the NN cell under "bill" points to the VB cell under "bill".



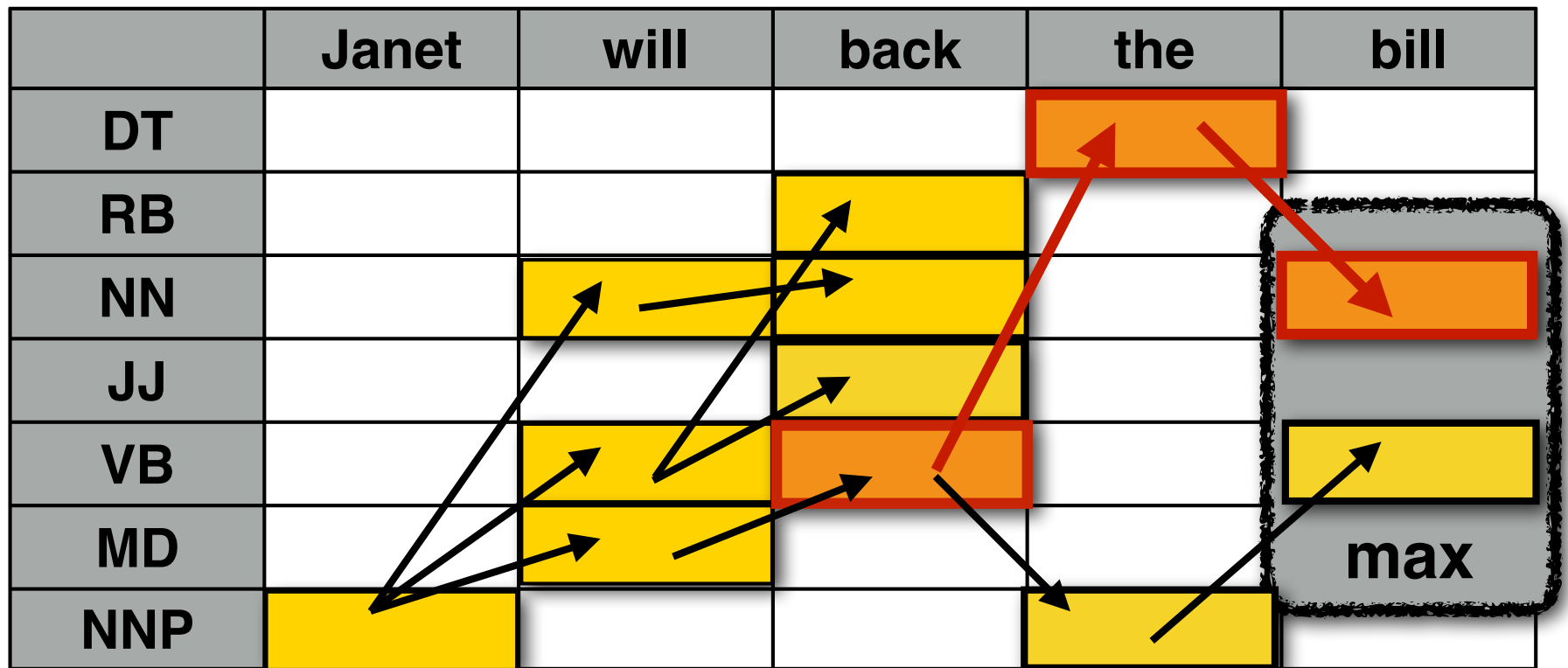
	Janet	will	back	the	bill
DT					
RB					
NN					
JJ					
VB					
MD					
NNP					

The diagram illustrates a parsing process. Arrows point from the NNP row to the NN, JJ, VB, and MD rows, and from the DT row to the NN row. A separate box on the right contains an orange box, a yellow box, and the text 'max'.

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

The diagram illustrates the parsing of the sentence "Janet will back the bill" using a shift-reduce parser. The grid shows the parts of speech (DT, RB, NN, JJ, VB, MD, NNP) for each word. The sequence of actions is as follows:

- Shift:** Janet (NNP) is shifted to the stack.
- Shift:** will (VB) is shifted to the stack.
- Shift:** back (RB) is shifted to the stack.
- Shift:** the (DT) is shifted to the stack.
- Shift:** bill (NN) is shifted to the stack.
- Reduce:** The stack contains [Janet, will, back, the, bill]. The action "max" is performed, resulting in the final parse tree structure.



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

**Janet\_NNP will\_MD back\_VB the\_DT bill\_NN**

# The Viterbi algorithm

```
Viterbi(  $w_{1...n}$  ){  
  for t (1...T) // INITIALIZATION: first column  
    trellis[1][t].viterbi =  $p\_init[t] \times p\_emit[t][w_1]$   
  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  $\times p\_trans[t'][t]$   
        if (tmp > trellis[i][t].viterbi){  
          trellis[i][t].viterbi = tmp  
          trellis[i][t].backpointer = t' } }  
      trellis[i][t].viterbi  $\times= p\_emit[t][w_i]$  } }  
  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 for Trigram HMMs

In a Trigram HMM, transition probabilities are of the form:

$$P(t^{(i)} = t_i \mid t^{(i-1)} = t_j, t^{(i-2)} = t_k)$$

The  $i$ -th tag in the sequence influences the probabilities of the  $(i+1)$ -th tag and the  $(i+2)$ -th tag:

$$\dots P(t^{(i+1)} \mid \mathbf{t^{(i)}}, t^{(i-1)}) \dots P(t^{(i+2)} \mid t^{(i+1)}, \mathbf{t^{(i)}})$$

Hence, each row in the trellis for a trigram HMM has to correspond to a pair of tags — the current and the preceding tag:

(abusing notation)

$\text{trellis}[i]\langle j, k \rangle$ : word  $w^{(i)}$  has tag  $t_j$ , word  $w^{(i-1)}$  has tag  $t_k$

The trellis now has  $T^2$  rows.

But we still need to consider only  $T$  transitions into each cell, since the current word's tag is the next word's preceding tag:

Transitions are only possible from  $\text{trellis}[i]\langle \mathbf{j}, k \rangle$  to  $\text{trellis}[i+1]\langle 1, \mathbf{j} \rangle$

# The three basic problems for HMMs

Given an **output sequence**  $\mathbf{w} = w^{(1)} \dots w^{(N)}$ :

$\mathbf{w}$  = “she promised to back the bill”

**Problem I (Likelihood):** find  $P(\mathbf{w} \mid \lambda)$

Given an HMM  $\lambda = (A, B, \pi)$ , compute the likelihood of the observed output,  $P(\mathbf{w} \mid \lambda)$

**Problem II (Decoding i.e. Tagging):** find  $Q = q^{(1)} \dots q^{(N)}$

Given an HMM  $\lambda = (A, B, \pi)$ , what is the most likely sequence of states  $Q = q^{(1)} \dots q^{(N)} \approx t^{(1)} \dots t^{(N)}$  to generate  $\mathbf{w}$ ?

**Problem III (MLE Estimation):** find  $\operatorname{argmax}_{\lambda} P(\mathbf{w} \mid \lambda)$

Find the parameters  $A, B, \pi$  which maximize  $P(\mathbf{w} \mid \lambda)$

# Dynamic programming algorithms for HMMs

## I. Likelihood of the input:

Compute  $P(\mathbf{w} | \lambda)$  for an  $\mathbf{w}$  and HMM  $\lambda$

⇒ **Forward algorithm**

## II. Decoding (=tagging) the input:

Find best tags  $\mathbf{t}^* = \operatorname{argmax}_{\mathbf{t}} P(\mathbf{t} | \mathbf{w}, \lambda)$  for input  $\mathbf{w}$  and HMM  $\lambda$

⇒ **Viterbi algorithm**

## III. Estimation (=learning the model):

Find best model parameters  $\lambda^* = \operatorname{argmax}_{\lambda} P(\mathbf{t}, \mathbf{w} | \lambda)$   
for unlabeled training data  $\mathbf{w}$

⇒ **Forward-Backward algorithm**

# Computing $P(\mathbf{w})$ : the Forward algorithm

To compute the probability of a sentence according to an HMM, we have to sum over all possible tag sequences:

$$P(\mathbf{w}) = \sum_{\mathbf{t}} P(\mathbf{w}, \mathbf{t})$$

The **Forward algorithm** computes this sum efficiently:

**Base case:** For the **first word in the sentence**, and for each tag  $j$ :

$$\text{forward}[1][j] = \pi(t_j)P(w^{(1)} | t_j)$$

**Recurrence:** For **any other word  $i$** , and for each tag  $j$ :

$$\text{forward}[i][j] = P(w^{(i)} | t_j) \sum_k \text{forward}[i-1][k]P(t_j | t_k)$$

**End:** For the **last word in the sentence**, and for all tags  $k$ :

$$P(\mathbf{w}) = \sum_k \text{forward}[N][k]$$

Same as Viterbi, except sum instead of max