CS447: Natural Language Processing
http://courses.engr.illinois.edu/cs447

Lecture 7: HMMs: The Viterbi algorithm

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

Recap: Statistical POS tagging

	she_1	promised_2	to_3	back_4	the_5	bill_6
$\mathbf{w}$ =	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_1	VBD_2	TO_3	VB_4	DT_5	NN_6

What is the most likely sequence of tags **t** for the given sequence of words **w** ?

Statistical POS tagging with HMMs

What is the most likely sequence of tags **t** for the given sequence of words **w** ?

$$\operatorname*{argmax}_{\mathbf{t}} P(\mathbf{t}|\mathbf{w}) \;=\; \operatorname*{argmax}_{\mathbf{t}} P(\mathbf{t})P(\mathbf{w}|\mathbf{t})$$

Hidden Markov Models define $P(\mathbf{t})$ and $P(\mathbf{w}|\mathbf{t})$ as:

Transition probabilities:

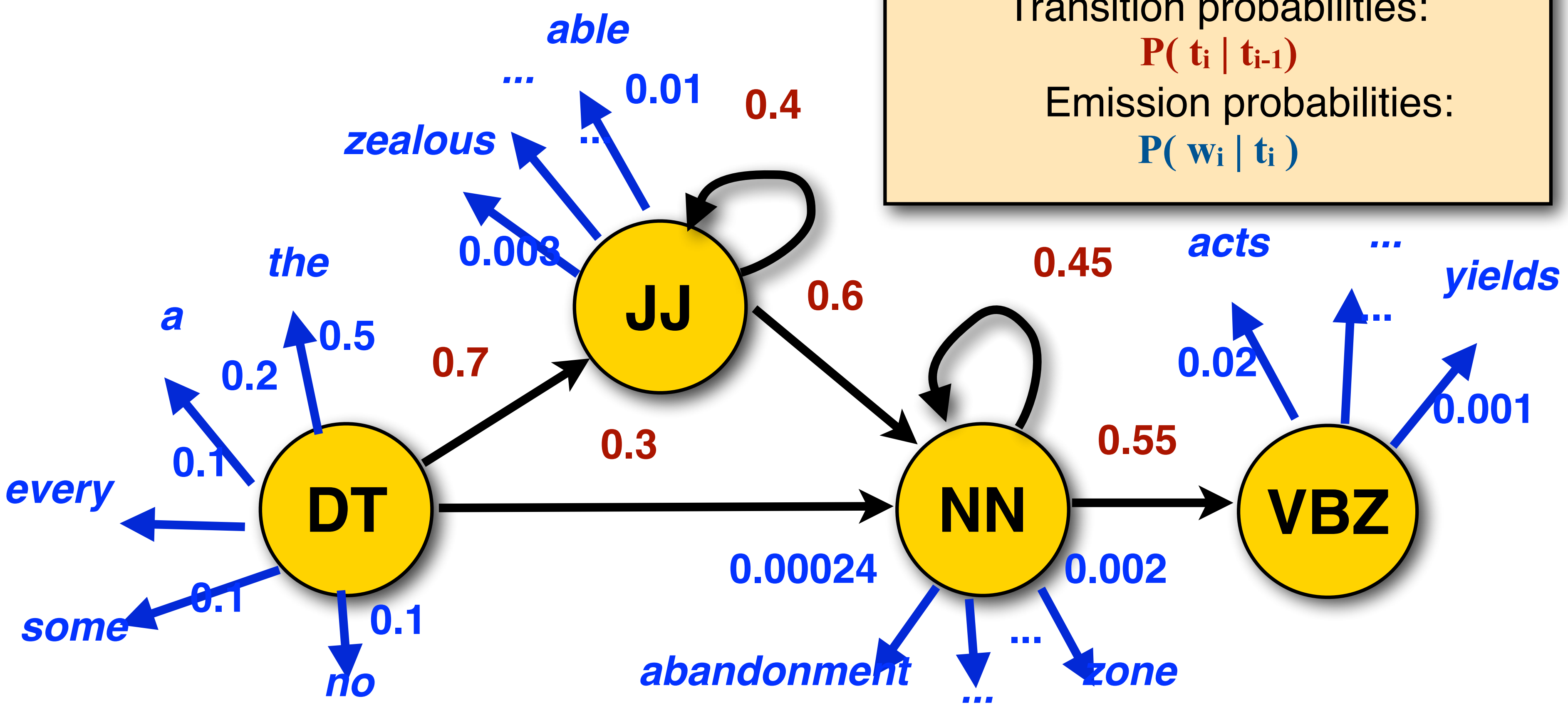
$P(\mathbf{t}) = \prod_i P(t_i \mid t_{i-1})$ [bigram HMM]

or $P(\mathbf{t}) = \prod_i P(t_i \mid t_{i-1}, t_{i-2})$ [trigram HMM]

Emission probabilities:

$P(\mathbf{w} \mid \mathbf{t}) = \prod_i P(w_i \mid t_i)$

HMMs as probabilistic automata



A (bigram) HMM defines
Transition probabilities:
$P(t_i \mid t_{i-1})$
Emission probabilities:
$P(w_i \mid t_i)$

HMMs as probabilistic automata

Transition probabilities $P(t_i \mid t_{i-1})$:

Probability of going from one state (t_{i-1}) of the automaton to the next (t_i)

"*Markov* model": We're making a Markov [independence] assumption for how to move between states of the automaton

Emission probabilities $P(w_i \mid t_i)$:

Probability of emitting a symbol (w_i) in a given state of the automaton (t_i)

"*Hidden* Markov model": We're assuming that the observed data consists only of the words **w**, and that the states of the automaton (the tags **t**) that generated the data **w** are not given to us.

HMM definitions (I)

An HMM is a **generative** probability model of word and hidden state (e.g. tag) sequences.

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

For a **bigram** HMM:

$$P(\mathbf{w} = w^{(1)}w^{(2)}\ldots w^{(N)}, \mathbf{t} = t^{(1)}t^{(2)}\ldots t^{(N)}) =$$
$$P(t^{(1)}) \cdot P(w^{(1)} \mid t^{(1)}) \cdot P(t^{(2)} \mid t^{(1)}) \cdot P(w^{(2)} \mid t^{(2)}) \cdot \ldots \cdot P(w^{(N)} \mid t^{(N)})$$

Tagging with an HMM = finding the most likely sequence of hidden states that generated **w**:

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

This search for the best **t** is also called **decoding**

HMM definitions (II)

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

- a **transition** matrix A: defines transition probabilities $P(q_i \mid q_j)$
- an **emission** matrix B: defines emission probabilities $P(w_i \mid q_j)$
- an **initial state** vector π: defines an initial state distribution $P(q_i)$

In a **bigram HMM tagger**, each state q_i corresponds to one POS tag (the tag of the current word): $q_i = t_i$

HMM definition (formally)

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**
  and $Q_F \subseteq Q$ a set of **final (accepting) 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

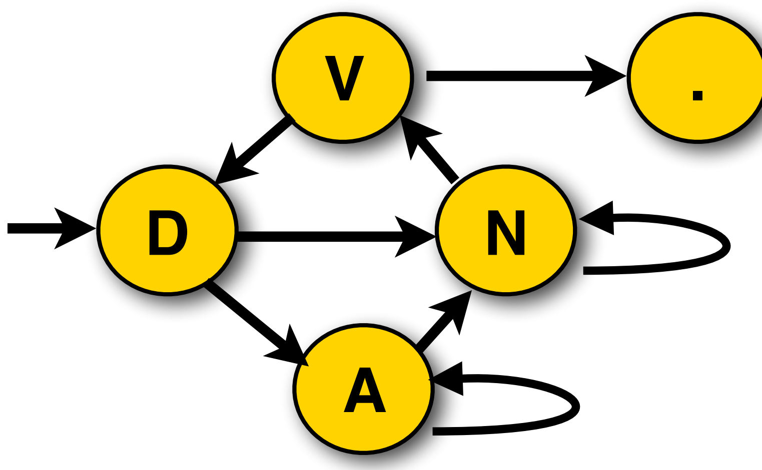
	D	N	V	A	.
D		0.8		0.2	
N		0.7	0.3		
V	0.6				0.4
A		0.8		0.2	
.					

Emission Matrix B

	the	*man*	*ball*	*throws*	*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				



Using HMMs for tagging

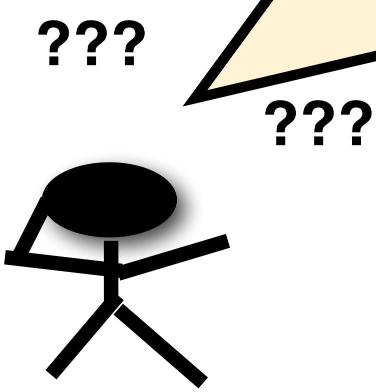
- The input to an HMM tagger is a sequence of words, **w**. The output is the most likely sequence of tags, **t**, for **w**.

- For the underlying HMM model, **w** is a sequence of output symbols, and **t** is the most likely sequence of states (in the Markov chain) that generated w.

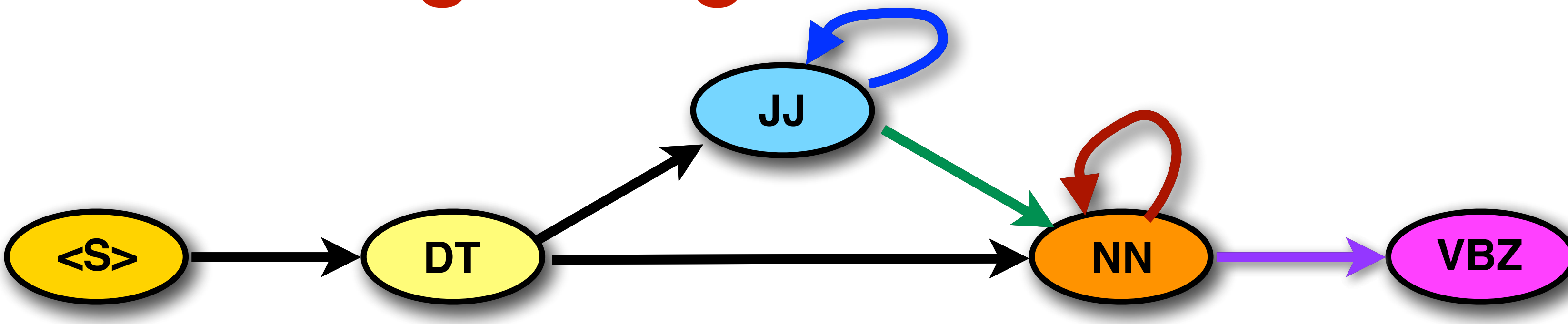
$$
\begin{aligned}
\operatorname*{argmax}_{\mathbf{t}} P(\underbrace{\mathbf{t}}_{Output_{tagger}} \mid \boxed{\underbrace{\mathbf{w}}_{Input_{tagger}}}) &= \operatorname*{argmax}_{\mathbf{t}} \frac{P(\mathbf{w},\mathbf{t})}{P(\mathbf{w})} \\
&= \operatorname*{argmax}_{\mathbf{t}} P(\mathbf{w},\mathbf{t}) \\
&= \operatorname*{argmax}_{\mathbf{t}} P(\boxed{\underbrace{\mathbf{w}}_{Output_{HMM}}} \mid \underbrace{\mathbf{t}}_{States_{HMM}}) P(\underbrace{\mathbf{t}}_{States_{HMM}})
\end{aligned}
$$

How would the automaton for a trigram HMM with transition probabilities $P(t_i \mid t_{i-2}t_{i-1})$ look like?

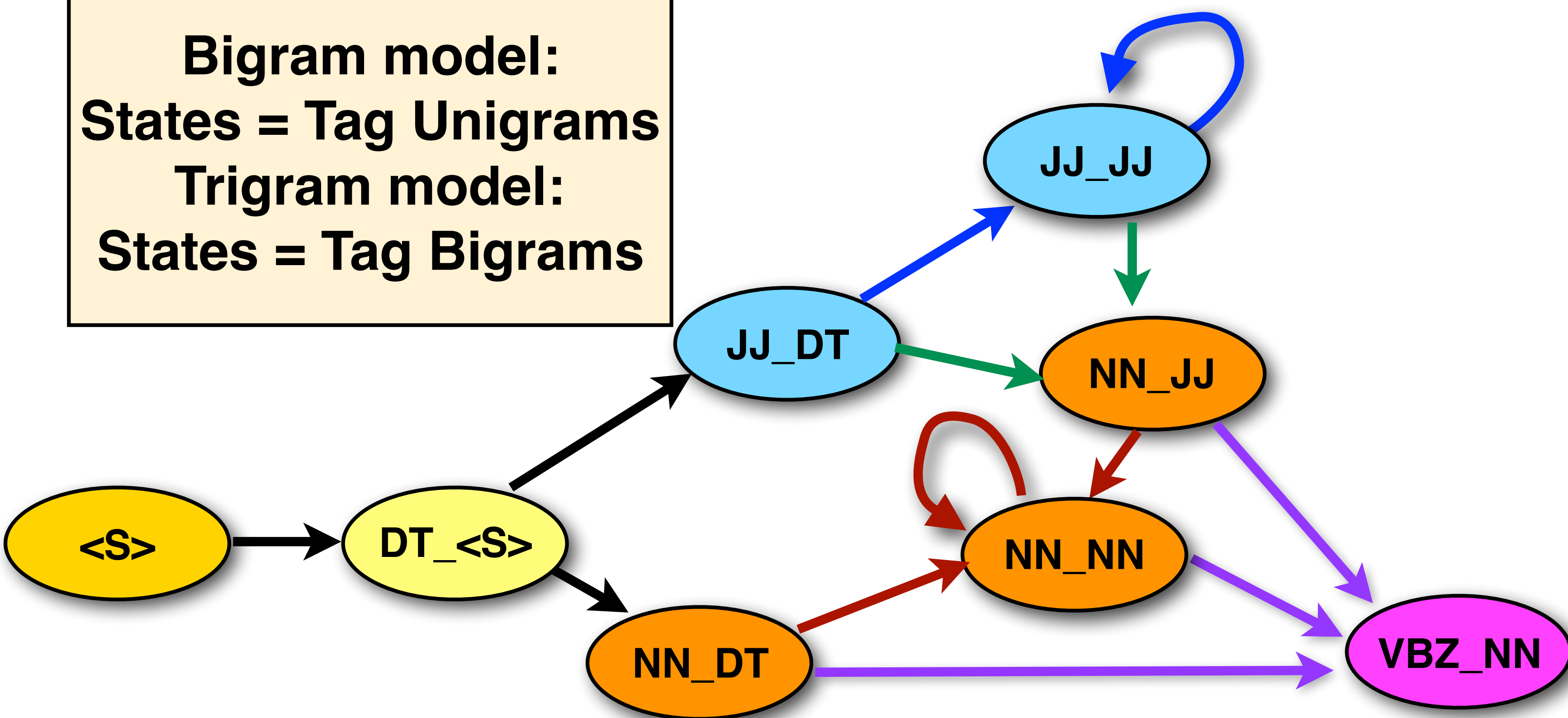
What about unigrams or n-grams?



Encoding a trigram model as FSA



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



Trigram HMMs

In a **trigram HMM tagger**, each state q_i corresponds to a POS tag bigram (the tags of the current and preceding word): $q_i = t_j t_k$

Emission probabilities depend only on the current POS tag: States $t_j t_k$ and $t_i t_k$ use the same emission probabilities $P(w_i \mid t_k)$

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)

- Define and implement a tagging algorithm that finds the best tag sequence $\mathbf{t}^*$ for each input sentence $\mathbf{w}$:
  $\mathbf{t}^* = \mathrm{argmax}_{\mathbf{t}}\, P(\mathbf{t})P(\mathbf{w} \mid \mathbf{t})$

Learning an HMM

Where do we get the transition probabilities $P(t_j | t_i)$ (matrix A) and the emission probabilities $P(w_j | t_i)$ (matrix B) from?

Case 1: We have a POS-tagged corpus.

- This is learning from labeled data, aka "supervised learning"

```
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 ._.
```

Case 2: We have a raw (untagged) corpus and a tagset.

- This is learning from unlabeled data, aka "unsupervised learning"

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

Tagset:
NNP: proper noun
CD: numeral,
JJ: adjective,...

Learning an HMM from *labeled* data

```
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 ._.
```

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

Learning the transition probabilities:

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

Learning the emission probabilities:

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

We might use some smoothing, but this is pretty trivial…

Dynamic Programming for HMMs

The three basic problems for HMMs

We observe an **output sequence** $w = w_1 \ldots w_N$:

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

Problem I (Likelihood): find $P(w \mid \lambda)$

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

Problem II (Decoding): find $Q = q_1..q_T$

Given an HMM $\lambda = (A, B, \pi)$, what is the most likely sequence of states $Q = q_1..q_N \approx t_1 \ldots t_N$ to generate $\boldsymbol{w}$?

Problem III (Estimation): find $argmax_{\lambda}\, P(w \mid \lambda)$

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

How can we solve these problems?

I. Likelihood of the input:

Compute $P(\boldsymbol{w} \mid \lambda)$ for the input $\boldsymbol{w}$ and HMM λ

II. Decoding (= tagging) the input:

Find the best tags $\boldsymbol{t}^* = argmax_{\boldsymbol{t}} P(\boldsymbol{t} \mid \boldsymbol{w}, \lambda)$ for the input $\boldsymbol{w}$ and HMM λ

III. Estimation (= learning the model):

Find the best model parameters $\lambda^* = argmax_{\lambda} P(\boldsymbol{t}, \boldsymbol{w} \mid \lambda)$
for the training data $\boldsymbol{w}$

These look like hard problems: With T tags, every input string $\boldsymbol{w}_{1...n}$ has T^n possible tag sequences

Can we find efficient (polynomial-time) algorithms?

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 solution of each subproblem in a common data structure

- Processing this data structure takes polynomial time

Dynamic programming algorithms for HMMs

I. Likelihood of the input:

Compute $P(\boldsymbol{w}|\ \lambda\ )$ for the input $\boldsymbol{w}$ and HMM λ

⇒ **Forward algorithm**

II. Decoding (=tagging) the input:

Find best tags $\boldsymbol{t}^*=argmax_{\boldsymbol{t}}\ P(\boldsymbol{t}\ |\ \boldsymbol{w},\lambda)$ for the input $\boldsymbol{w}$ and HMM λ

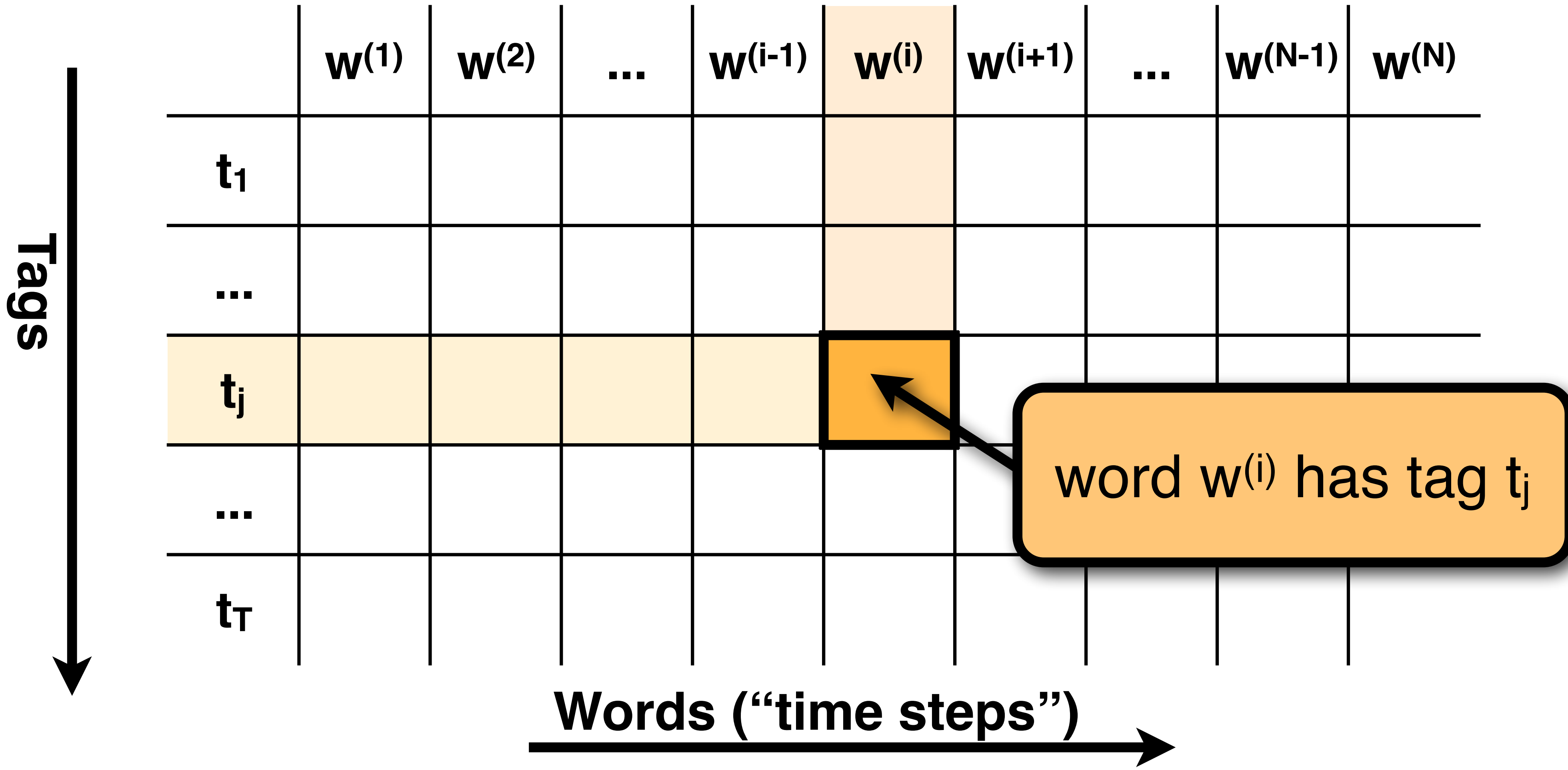
⇒ **Viterbi algorithm**

III. Estimation (=learning the model):

Find best model parameters $\lambda^*=argmax_{\lambda}\ P(\boldsymbol{t},\ \boldsymbol{w}\ |\ \lambda)$ for training data $\boldsymbol{w}$

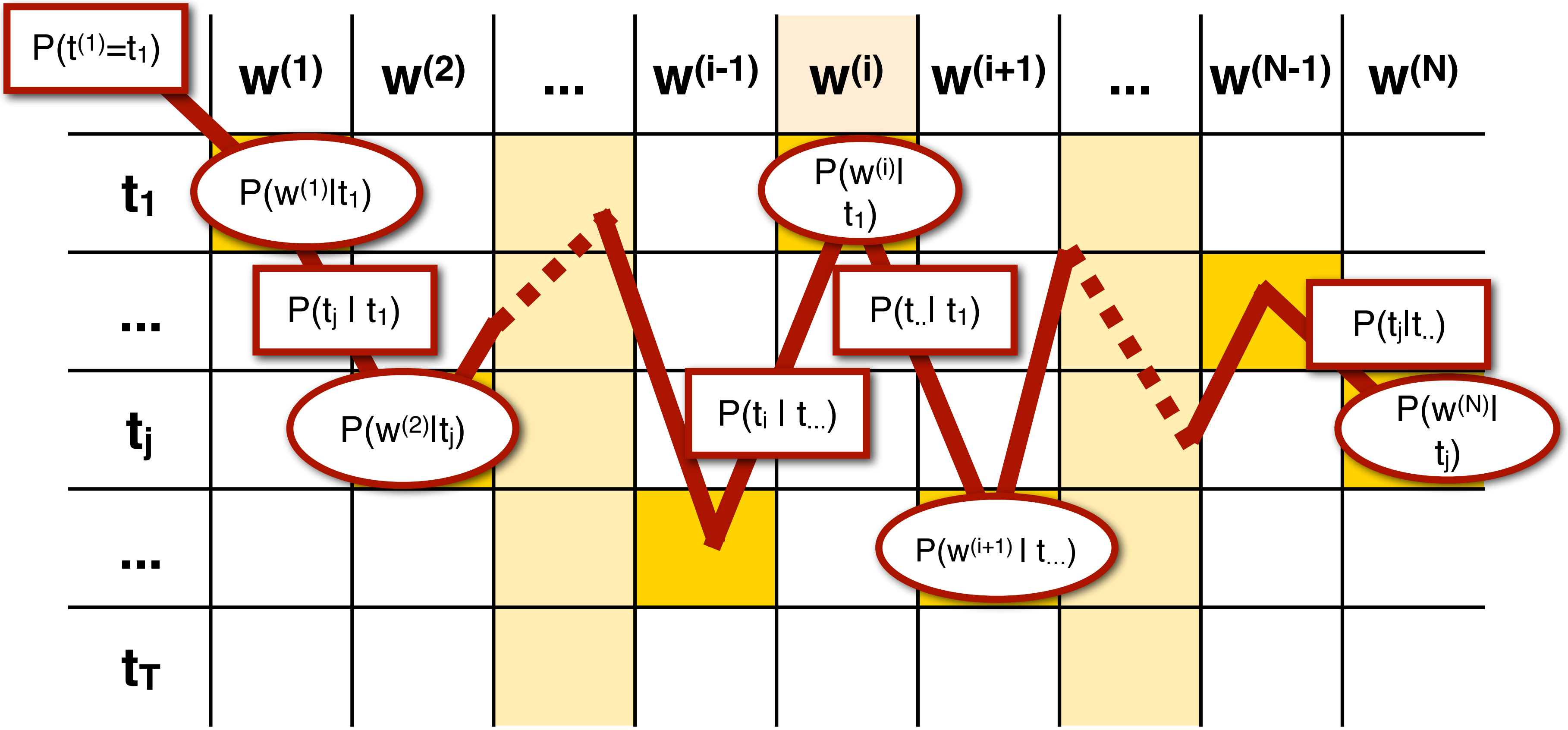
⇒ **Forward-Backward algorithm**

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.

Computing P(t,w) for one tag sequence



- One path through the trellis = one tag sequence
- We just multiply the transition and emission probabilities

The Viterbi algorithm

Finding the best tag sequence

The number of *possible* tag sequences is exponential in the length of the input sentence:

Each word can have up to T tags.
There are N words.
There are up to T^N possible tag sequences.

We cannot enumerate all T^N possible tag sequences.

But we can exploit the independence assumptions in the HMM to define an efficient algorithm that returns the tag sequence with the highest probability in linear (O(N)) time.

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 sequence

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

HMM decoding

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

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

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

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

To do this efficiently, we will use a technique called **dynamic programming** to exploit the **independence assumptions** in the HMM.

The Viterbi algorithm

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

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

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

HMM independences

1. **Emissions** depend only on the current tag:

   … $P(w^{(i)} = \text{man} \mid t^{(i)} = \text{NN})$…

We only have to multiply the emission probability $P(w^{(i)} \mid t_j)$ with **the probability of the best tag sequence** that gets us to $t^{(i)} = t_j$

HMM independences

2. **Transition probabilities to the current tag** $t^{(i)}$ depend only on the previous tag $t^{(i-1)}$:

… $P(\ t^{(i)} = NN \mid t^{(i-1)} = DT\ )$

- Assume the **probability of the best tag sequence for the prefix** $w^{(1)} \dots w^{(i-1)}$ **that ends in the tag** $t^{(i-1)} = \mathbf{t_j}$ is known, and stored in a variable $\mathbf{max[i-1][j]}$.
- To compute **the probability of the best tag sequence for** $w^{(1)} \dots \mathbf{w^{(i-1)}} \mathbf{w^{(i)}}$ **that ends in the tags** $t^{(i-1)}t^{(i)} = \mathbf{t_j t_k}$**,** multiply $\mathbf{max[i-1][j]}$ with $P(t_k \mid t_j)$ and $P(w^{(i)} \mid t_k)$
- To compute **the probability of the best tag sequence for** $w^{(1)} \dots w^{(i-1)} \mathbf{w^{(i)}}$ **that ends in** $t^{(i)} = \mathbf{t_k}$ **,** consider all possible tags $t^{(i-1)} = \mathbf{t_j}$ for the preceding word:
$\mathbf{max[i][k] = max_j}\ (\ \mathbf{max[i-1][j]}\ P(t_k \mid t_j)\ ) P(w^{(i)} \mid t_k)$

HMM independences

3. The **current tag** also determines the transition probability of the **next tag**:

$$\ldots P(\ t^{(i+1)} = \mathrm{VBZ}\ \mid t^{(i)} = \mathrm{NN}\ )\ldots$$

We cannot fix the current tag $t^{(i)}$ based on the probability of getting to $t^{(i)}$ (and producing $w^{(i)}$)

We have to wait until we have reached the last word in the sequence.
Then, we can trace back to get the best tag sequence for the entire sentence.

Using the trellis to find **t***

Let trellis[i][j] (word $w^{(j)}$ and tag t_j) store the probability of the best tag sequence for $w^{(1)} \ldots w^{(i)}$ that ends in t_j

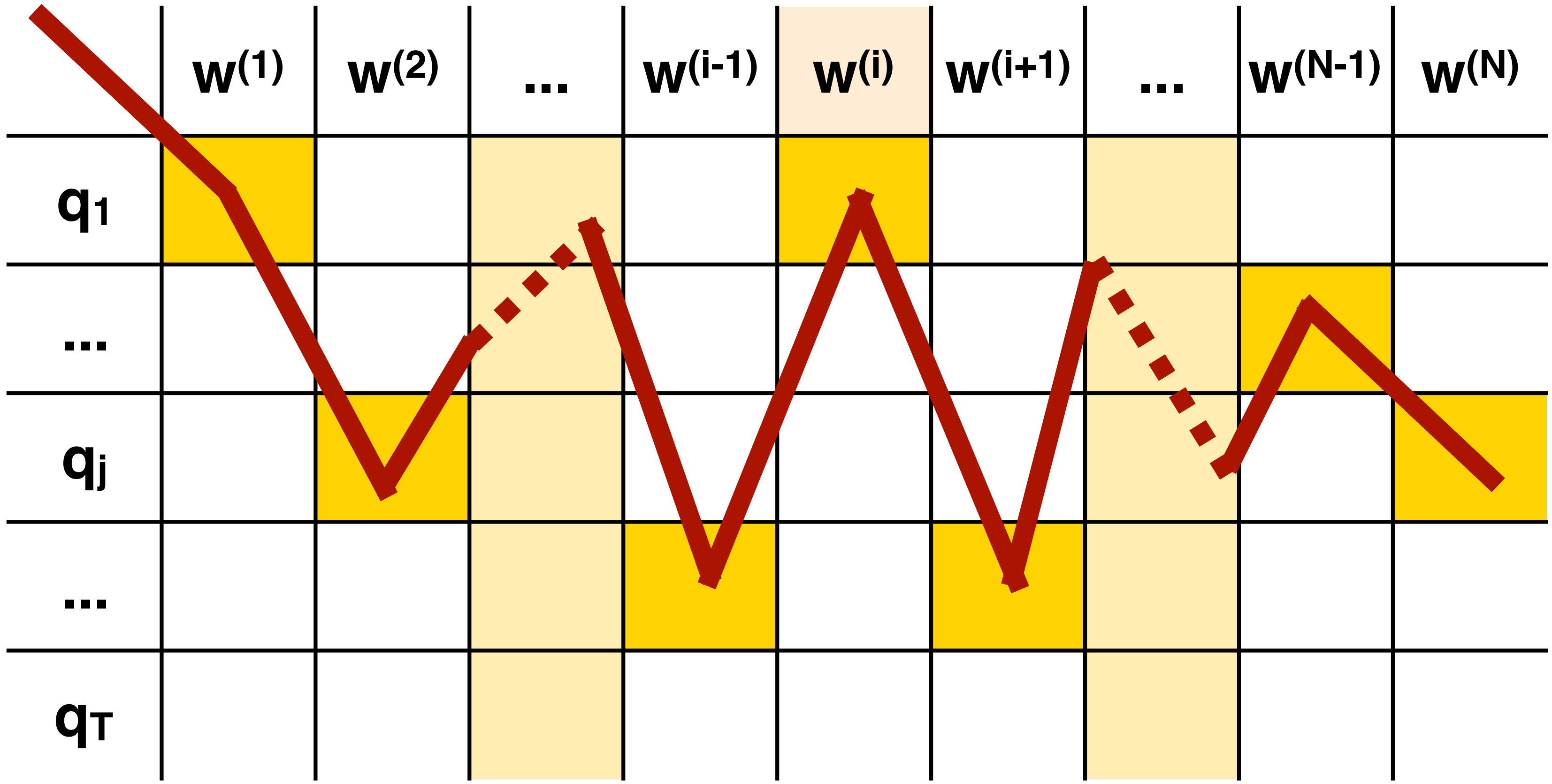
$$\text{trellis}[i][j] = \max P(w^{(1)} \ldots w^{(i)}, t^{(1)} \ldots, t^{(i)} = t_j)$$

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

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

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

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.

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);
  }
```

Unpacking the trellis

```
unpack(n, t){
  i = n;
  tags = new array[n+1];
  while (i > 0){
    tags[i] = t;
    t = trellis[i][t].backpointer;
    i--;
  }
  return tags;
}
```

Today’s key concepts

HMM taggers
Learning HMMs from labeled text
Viterbi for HMMs

- Dynamic programming
- Independence assumptions in HMMs
- The trellis

Supplementary material: Viterbi for Trigram HMMs

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:

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

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)

$\mathrm{trellis}[i]\langle j,k \rangle$: 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 $\mathrm{trellis}[i]\langle \mathbf{j},k \rangle$ to $\mathrm{trellis}[i+1]\langle l,\mathbf{j} \rangle$