CS447: Natural Language Processing

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

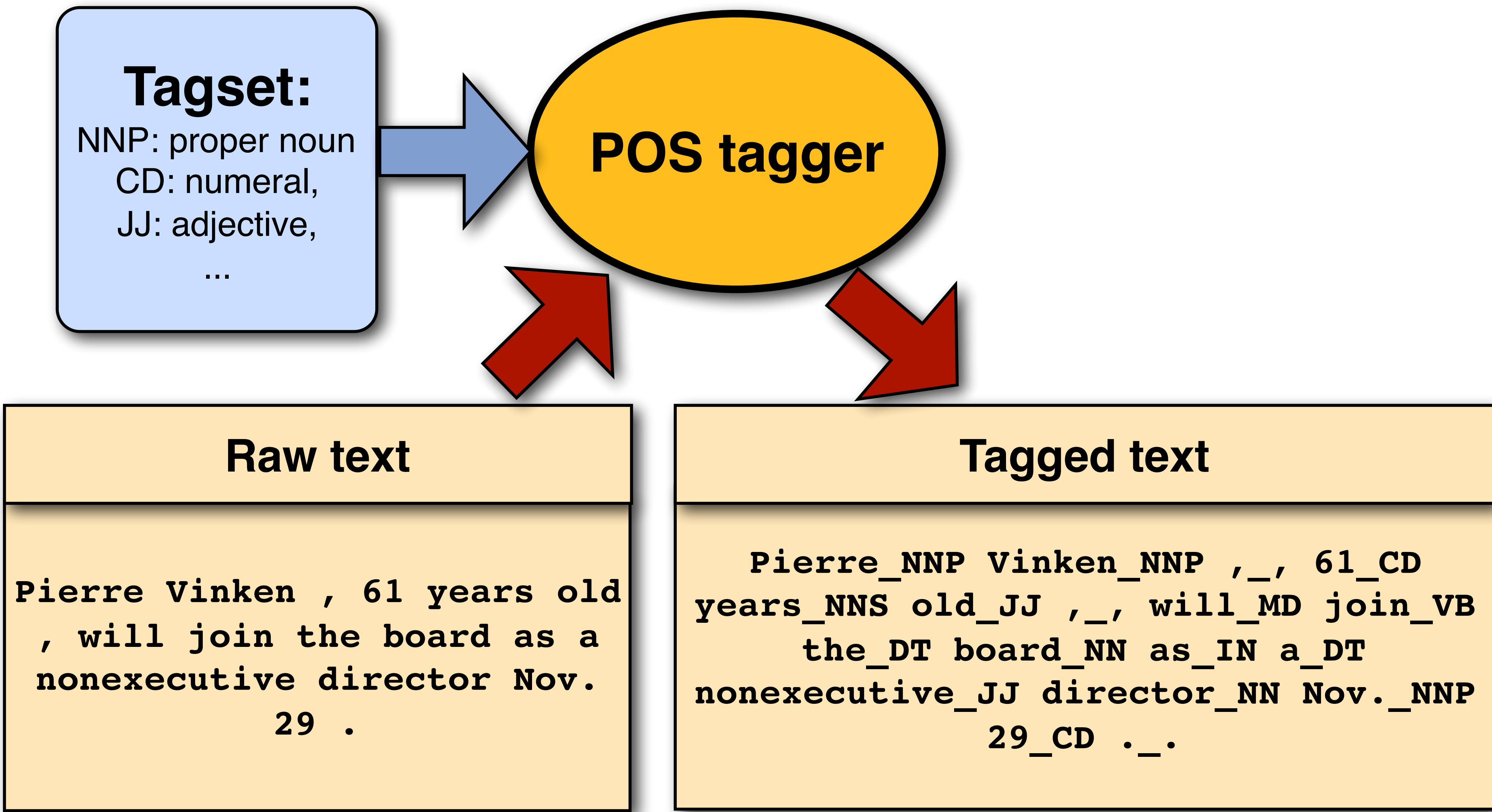
Lecture 10: Part-of-Speech Tagging

Julia Hockenmaier

juliahmr@illinois.edu

3324 Siebel Center

POS tagging


Tagset:
NNP: proper noun
CD: numeral,
JJ: adjective,
...
POS tagger
Raw text
Pierre Vinken , 61 years old , will join the board as a nonexecutive director Nov. 29 .
Tagged text
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 ._.

Why POS tagging?

POS tagging is traditionally viewed as a prerequisite for further analysis:

- Speech synthesis:

  How to pronounce "lead"?

  INsult or inSULT, OBject or obJECT, OVERflow or overFLOW, DIScount or disCOUNT, CONtent or conTENT
- Parsing:

  What words are in the sentence?
- Information extraction:

  Finding names, relations, etc.
- Machine Translation:

  The noun "content" may have a different translation from the adjective.

POS Tagging

Words often have more than one POS:

- *The **back** door* (adjective)
- *On my **back*** (noun)
- *Win the voters **back*** (particle)
- *Promised to **back** the bill* (verb)

The POS tagging task is to determine the POS tag for a particular instance of a word.

Since there is ambiguity, we cannot simply look up the correct POS in a dictionary.

These examples from Dekang Lin

Defining a tagset

Defining a tag set

We have to define an **inventory of labels for the word classes** (i.e. the tag set)

- Most taggers rely on models that have to be trained on **annotated (tagged) corpora**. Evaluation also requires annotated corpora.
- Since human annotation is expensive/time-consuming, the tag sets used in a few existing labeled corpora become the **de facto standard**.
- Tag sets need to capture **semantically or syntactically important distinctions** that can easily be made by trained human annotators.

Word classes

Open classes:

Nouns, Verbs, Adjectives, Adverbs

Closed classes:

Auxiliaries and modal verbs

Prepositions, Conjunctions

Pronouns, Determiners

Particles, Numerals

(see Appendix for details)

Defining an annotation scheme

A lot of NLP tasks require systems to map natural language text to another representation:

POS tagging: Text ⟶ POS tagged text
Syntactic Parsing: Text ⟶ parse trees
Semantic Parsing: Text ⟶ meaning representations
…: Text ⟶ …

Defining a tag set

Tag sets have different granularities:

Brown corpus (Francis and Kucera 1982): 87 tags

Penn Treebank (Marcus et al. 1993): 45 tags

Simplified version of Brown tag set
(de facto standard for English now)

NN: common noun (singular or mass): *water*, *book*
NNS: common noun (plural): *books*

Prague Dependency Treebank (Czech): 4452 tags

Complete morphological analysis:
AAFP3----3N----: *nejnezajímavějším*
Adjective Regular Feminine Plural Dative….Superlative
[Hajic 2006, VMC tutorial]

How much ambiguity is there?

Most word *types* are unambiguous:

Number of tags per word type:

		87-tag Original Brown		45-tag Treebank Brown	
Unambiguous (1 tag)		**44,019**		**38,857**	
Ambiguous (2–7 tags)		**5,490**		**8844**	
Details:	2 tags	4,967		6,731	
	3 tags	411		1621	
	4 tags	91		357	
	5 tags	17		90	
	6 tags	2	(*well*, *beat*)	32	
	7 tags	2	(*still*, *down*)	6	(*well*, *set*, *round*, *open*, *fit*, *down*)
	8 tags			4	(*'s*, *half*, *back*, *a*)
	9 tags			3	(*that*, *more*, *in*)

NB: These numbers are based on word/tag combinations in the corpus.
Many combinations that don't occur in the corpus are equally correct.

But a large fraction of word *tokens* are ambiguous

Original Brown corpus: 40% of *tokens* are ambiguous

Defining an annotation scheme

Training and evaluating models for these NLP tasks requires large **corpora annotated with the desired representations.**

Annotation at scale is expensive, so **a few existing corpora** and their **annotations** and **annotation schemes** (tag sets, etc.) often become the de facto standard for the field.

It is difficult to know what the 'right' annotation scheme should be for any particular task

How difficult is it to achieve high accuracy for that annotation?

How useful is this annotation scheme for downstream tasks in the pipeline?

⇨ We often can't know the answer until we've annotated a lot of data…

Evaluating POS taggers

Evaluation metric: test accuracy

How many words in the unseen test data can you tag correctly?

State of the art on Penn Treebank: around 97%.

➩ **How many *sentences* can you tag correctly?**

Compare your model against a **baseline**

Standard: assign to each word its most likely tag

(use training corpus to estimate P(t|w))

Baseline performance on Penn Treebank: around 93.7%

… and a **(human) ceiling**

How often do human annotators agree on the same tag?

Penn Treebank: around 97%

Is POS-tagging a solved task?

Penn Treebank POS-tagging accuracy
≈ human ceiling

Yes, but:

Other languages with more complex morphology need much larger tag sets for tagging to be useful, and will contain many more distinct word forms in corpora of the same size.

They often have much lower accuracies.

Also: POS tagging accuracy on English text from other domains can be significantly lower.

Qualitative evaluation

Generate a confusion matrix (for development data):
How often was a word with tag i mistagged as tag j:

Correct Tags

Predicted Tags

	IN	JJ	NN	NNP	RB	VBD	VBN
IN	—	.2			.7		
JJ	.2	—	**3.3**	2.1	1.7	.2	**2.7**
NN		**8.7**	—				.2
NNP	.2	**3.3**	**4.1**	—	.2		
RB	**2.2**	2.0	.5		—		
VBD		.3	.5			—	**4.4**
VBN		**2.8**				**2.6**	—

% of errors caused by mistagging VBN as JJ

See what errors are causing problems:

- Noun (NN) vs ProperNoun (NNP) vs Adj (JJ)
- Preterite (VBD) vs Participle (VBN) vs Adjective (JJ)

Building a POS tagger

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 $\mathbf{w} = w^{(1)} \ldots w^{(N)}$?

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

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 $P(\mathbf{w} \mid \mathbf{t})$:

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

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

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

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

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

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

These probabilities don't depend on the string position $^{(i)}$,
but are defined over word and tag types.
With subscripts $_{i,j,k}$, to index 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 $t^{(i)}/w^{(i)}$

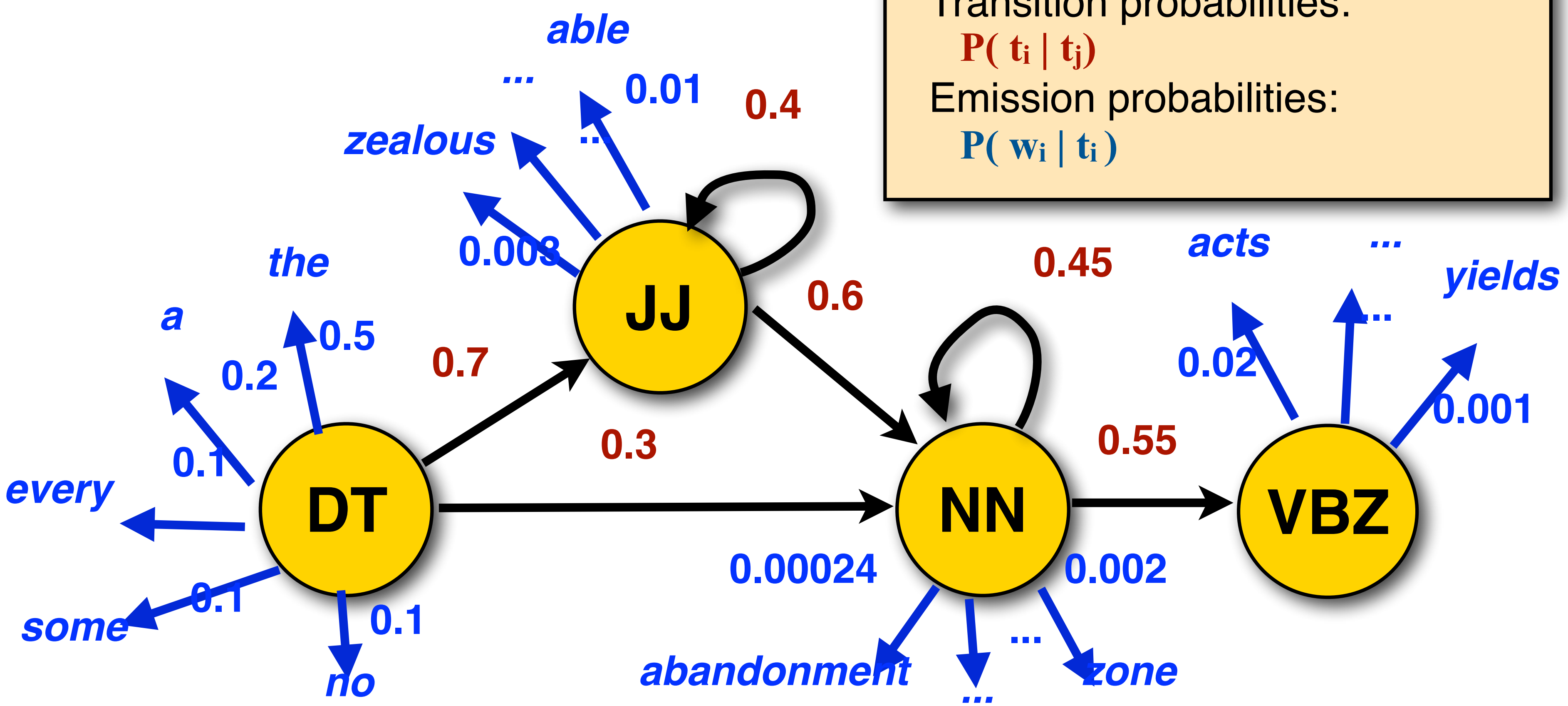
To make the distinction between the i-th word/tag in the vocabulary/tag set and the i-th word/tag in the sentence clear:

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

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

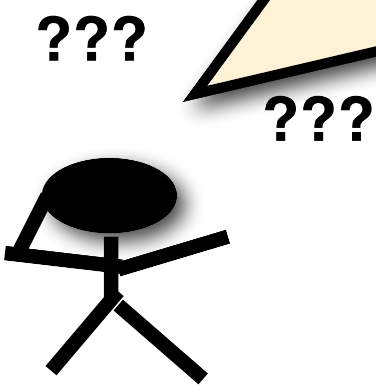
HMMs as probabilistic automata

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

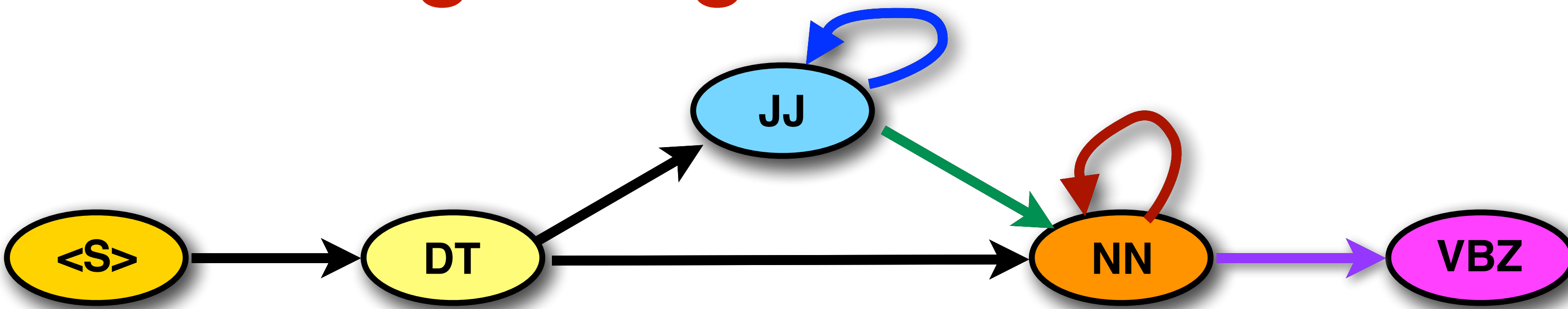


How would the
automaton for a trigram
HMM with transition probabilities
$P(t_i \mid t_j t_k)$ look like?

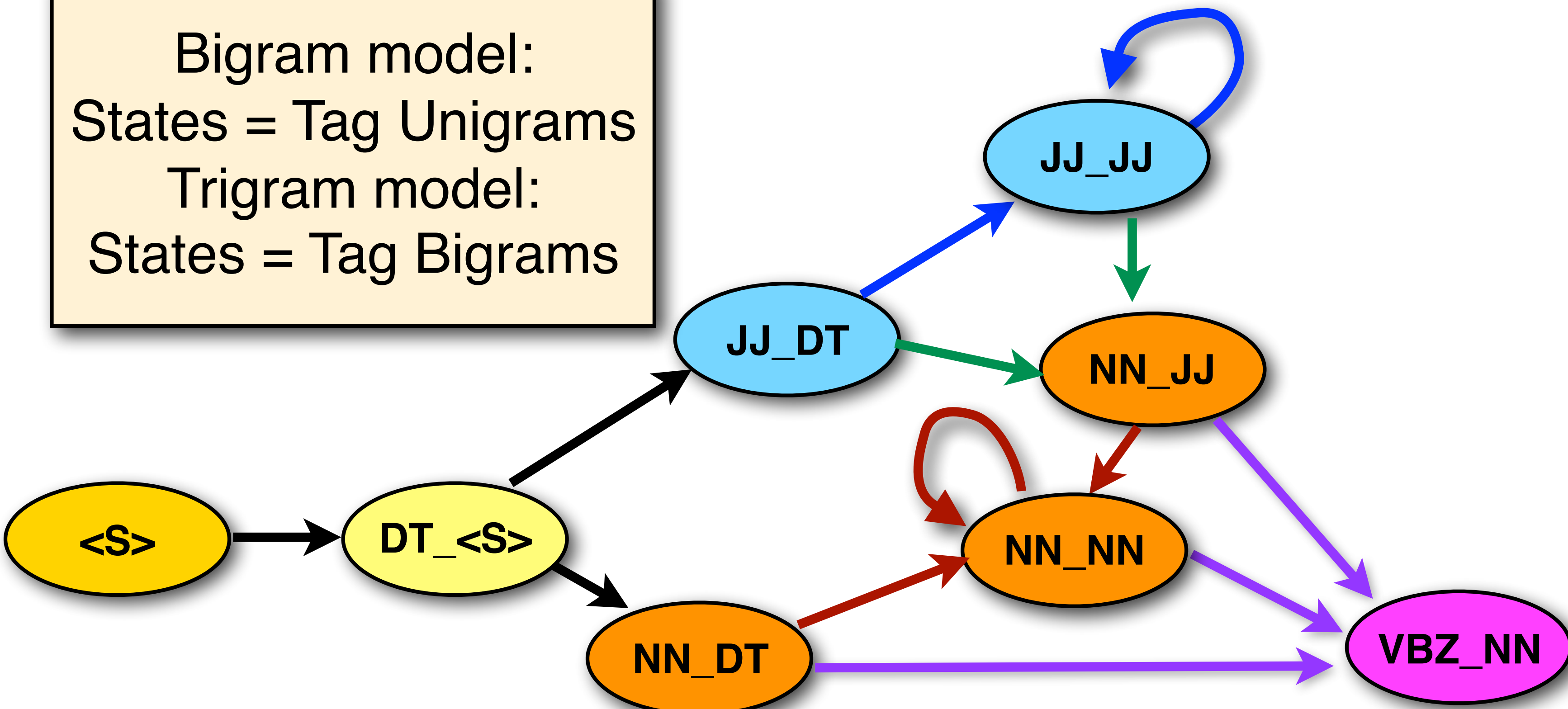
What about unigrams
or n-grams?



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**
  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

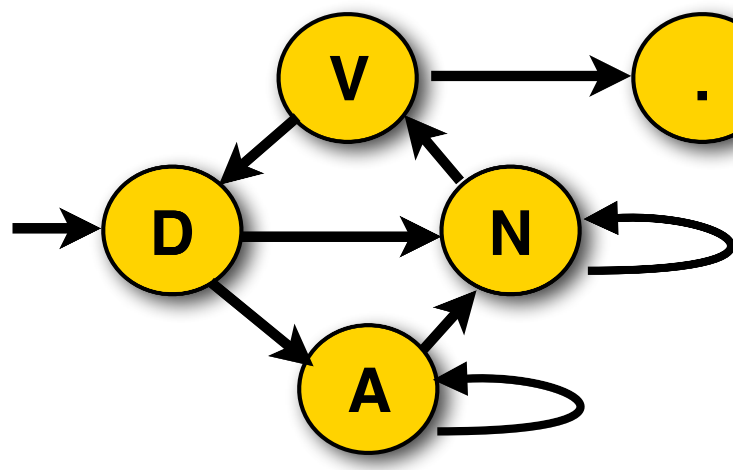
	D	N	V	A	.
D		0.8		0.2	
N		0.7	0.3		
V	0.6				0.4
A		0.8		0.2	
.					

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				



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 $\mathbf{w}$:

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

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:

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

Learning an HMM from *unlabeled* data

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

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

We can't count anymore.

We have to *guess* how often we'd *expect* to see $t_i t_j$ *etc.* in our data set. Call this expected count $\langle C(\ldots)\rangle$

- Our estimate for the transition probabilities:

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

- Our estimate for the emission probabilities:

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

These expected counts can be obtained via dynamic programming (the Inside-Outside 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 N^T possible tag sequences.

We **cannot enumerate** all N^T 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

Dynamic Programming for HMMs

The three basic problems for HMMs

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

$\mathbf{w}$=*"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, $P(\mathbf{w} \mid \lambda)$

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 $\mathbf{w}$?

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

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

How can we solve these problems?

I. **Likelihood** of the input $\boldsymbol{w}$:

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

II. **Decoding (= tagging)** the input $\boldsymbol{w}$:

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 (unlabeled) 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

I. Likelihood of the input:

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

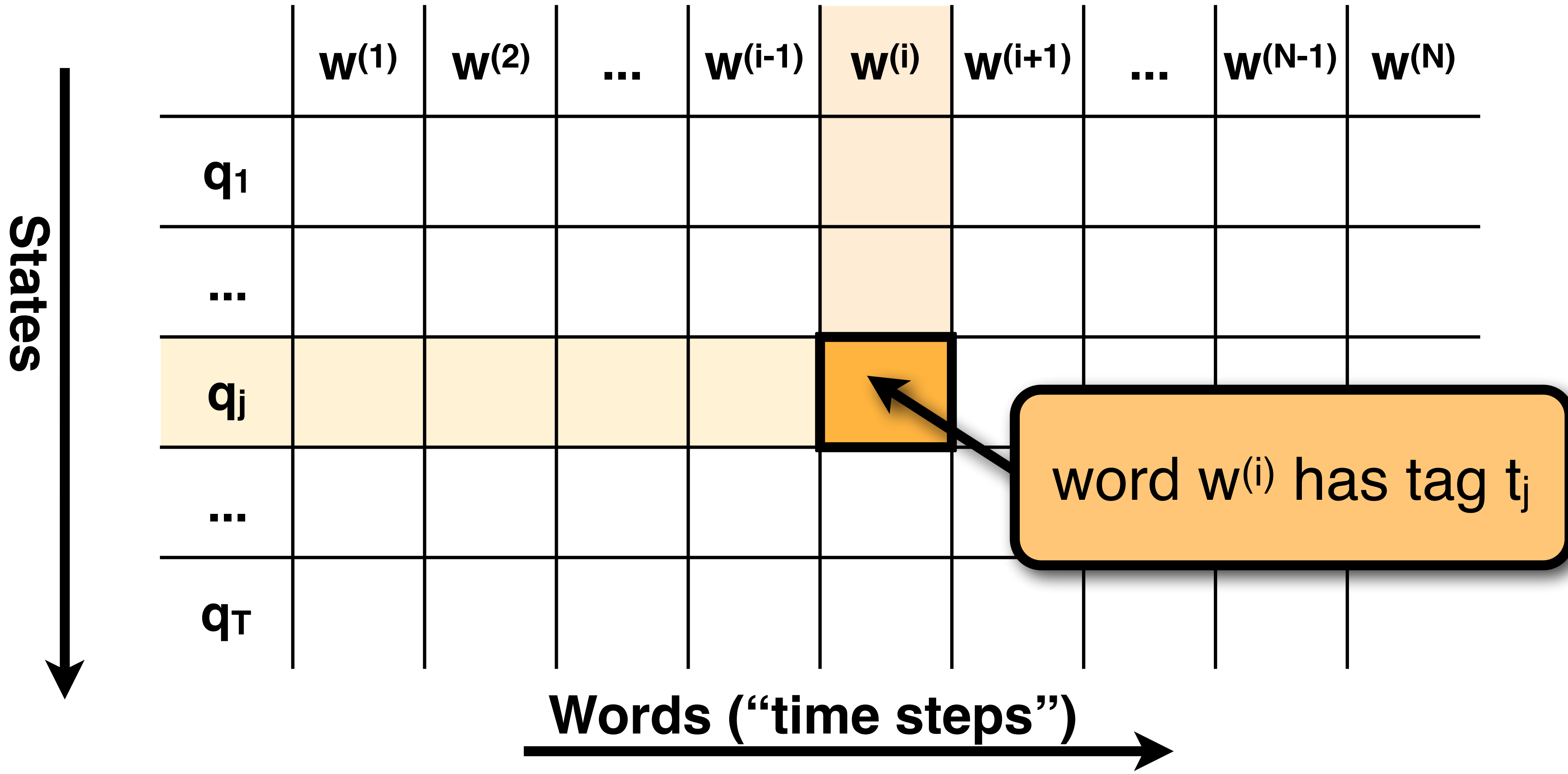
⇒ **Viterbi algorithm**

III. Estimation (=learning the model):

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

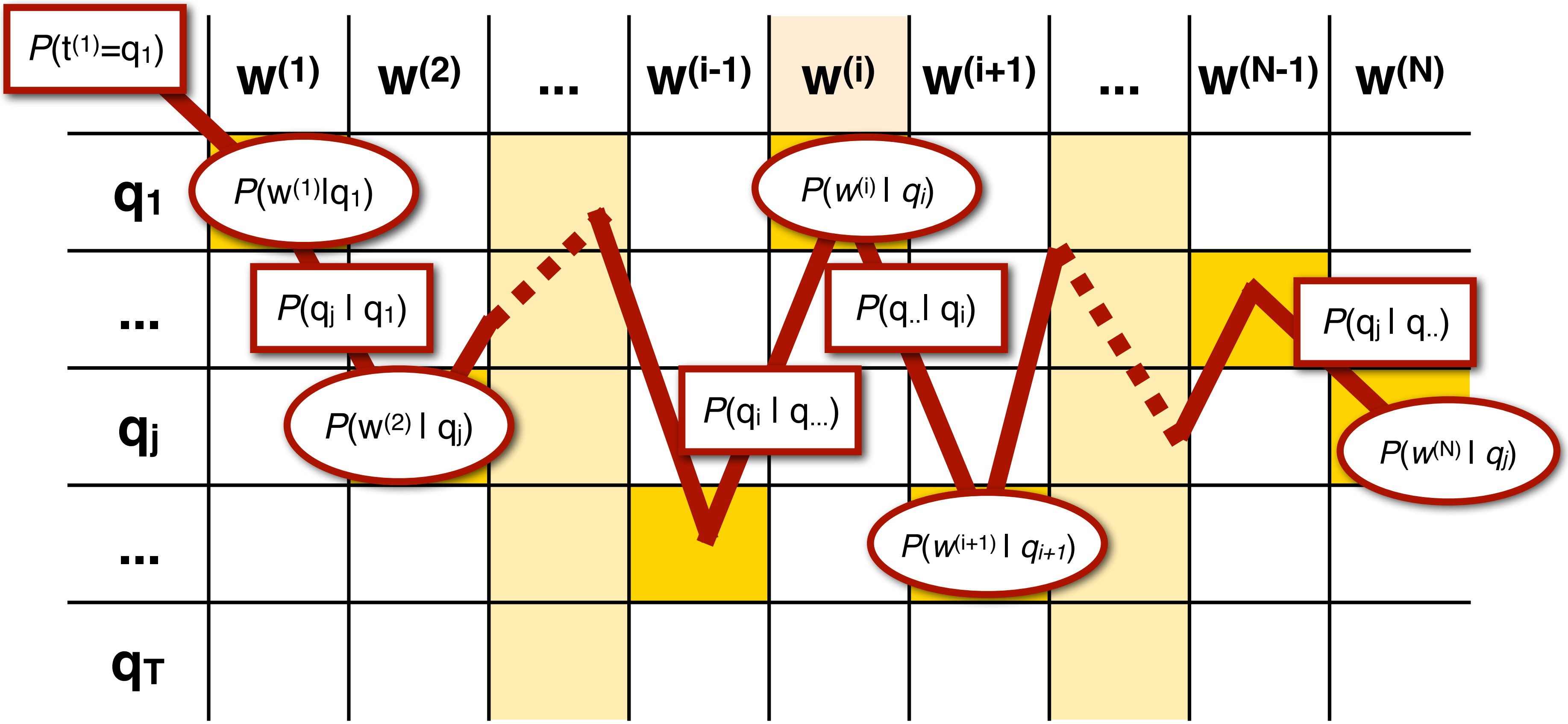
⇒ **Forward-Backward algorithm**

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



- One path through the trellis = one tag sequence
- We just multiply the probabilities as before

The Viterbi algorithm

HMM decoding (Viterbi)

We observe 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**

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

To do this efficiently, we will use a **dynamic programming** technique called the Viterbi algorithm which exploits the **independence assumptions** in the HMM.

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

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

For each cell trellis[i][j], we find the best cell in the previous column (trellis[i–1][k*]) based on the entries in the previous column and the transition probabilities $P(t_j | t_k)$

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

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, we pick the highest scoring entry in the last column of the trellis (= for the last word) and follow the backpointers

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

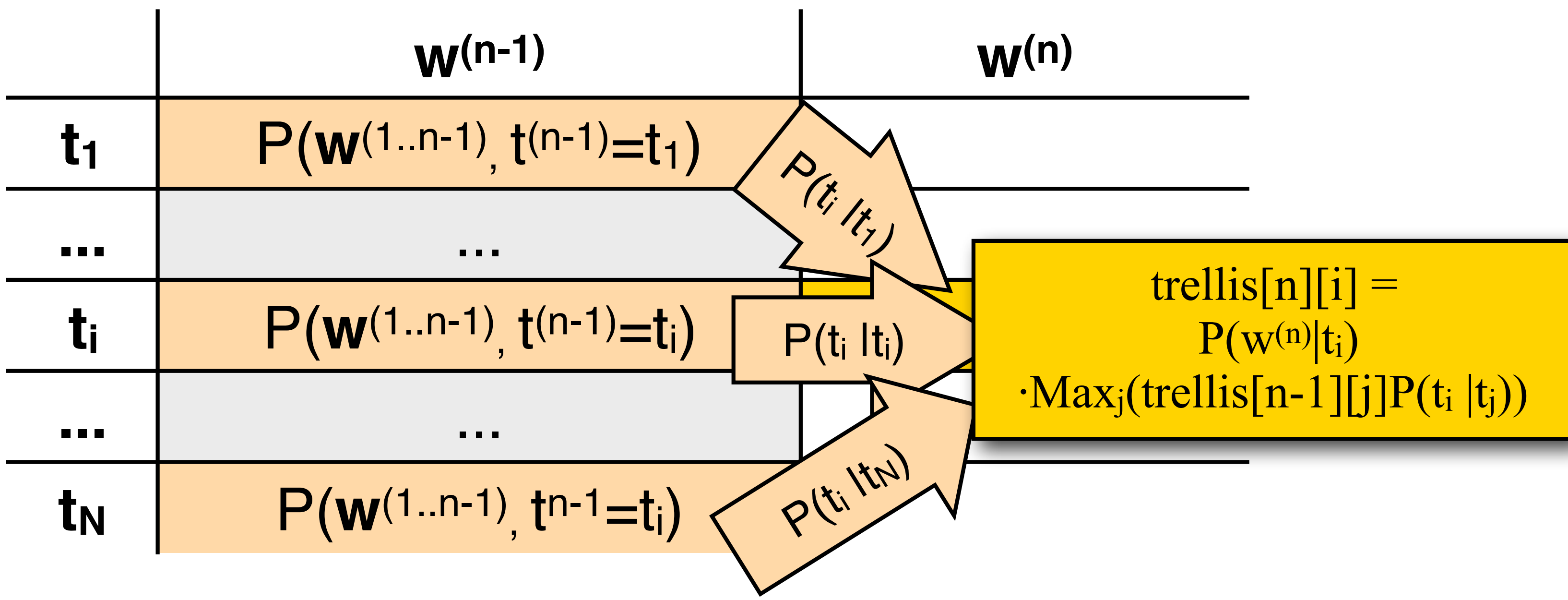
In the first column, initialize each cell $\mathrm{trellis}[1][k]$ as

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

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

At any internal cell

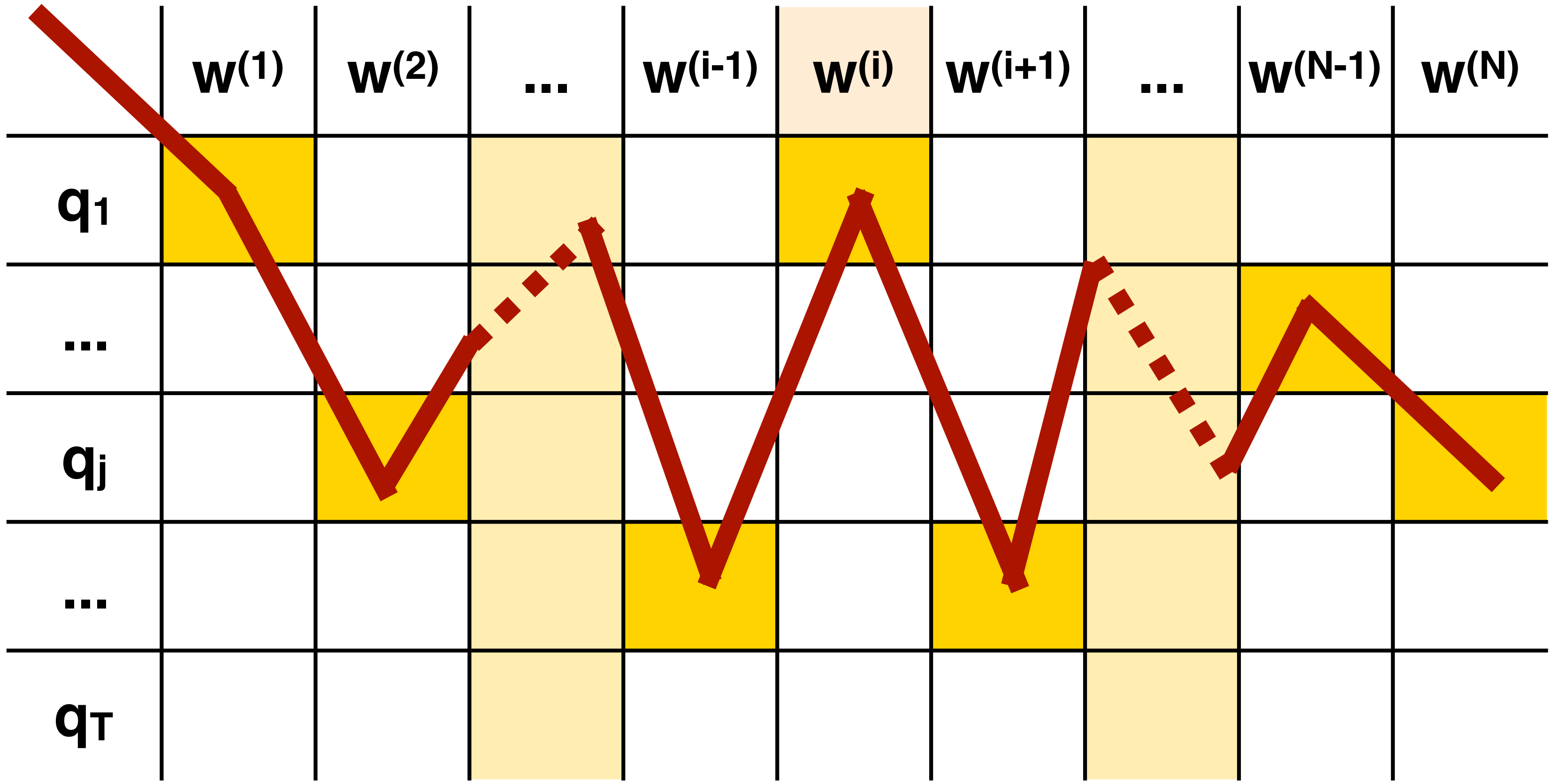
- For each cell in the preceding column: multiply its entry with the transition probability to the current cell.
- Keep a single backpointer to the best (highest scoring) cell in the preceding column
- Multiply this score with the emission probability of the current word



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



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

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.

The Viterbi algorithm

```
Viterbi( w_1...n){
  for t (1...T)  // INITIALIZATION: first column
    trellis[1][t].viterbi = p_init[t] × 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 × 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][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);
}
```

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;
}
```

Supplementary: 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:

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

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)

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

Appendix: English parts of speech

Nouns

Nouns describe entities and concepts:

Common nouns: dog, bandwidth, dog, fire, snow, information

- Count nouns have a plural (dogs) and need an article in the singular (the dog barks)
- Mass nouns don't have a plural (*snows) and don't need an article in the singular (snow is cold, metal is expensive). But some mass nouns can also be used as count nouns: Gold and silver are metals.

Proper nouns (Names): Mary, Smith, Illinois, USA, France, IBM

Penn Treebank tags:

NN: singular or mass

NNS: plural

NNP: singular proper noun

NNPS: plural proper noun

(Full) verbs

Verbs describe activities, processes, events:

eat, write, sleep, ….

Verbs have different morphological forms:
infinitive (to eat), present tense (I eat), 3rd pers sg. present tense (he eats), past tense (ate), present participle (eating), past participle (eaten)

Penn Treebank tags:

VB: infinitive (base) form
VBD: past tense
VBG: present participle
VBD: past tense
VBN: past participle
VBP: non-3rd person present tense
VBZ: 3rd person singular present tense

Adjectives

Adjectives describe properties of entities:

blue, hot, old, smelly,…

Adjectives have an...

… attributive use (modifying a noun):

the blue book

… and a predicative use (e.g. as argument of be):

The book is blue.

Many gradable adjectives also have a…

...comparative form: greater, hotter, better, worse

...superlative form: greatest, hottest, best, worst

Penn Treebank tags:

JJ: adjective   JJR: comparative   JJS: superlative

Adverbs

Adverbs describe properties of events/states.

- Manner adverbs: slowly (slower, slowest) fast, hesitantly,…
- Degree adverbs: extremely, very, highly….
- Directional and locative adverbs: here, downstairs, left
- Temporal adverbs: yesterday, Monday,…

Adverbs modify verbs, sentences, adjectives or other adverbs:

Apparently, the very ill man walks extremely slowly

NB: certain temporal and locative adverbs (yesterday, here) can also be classified as nouns

Penn Treebank tags:

RB: adverb   RBR: comparative adverb   RBS: superlative adverb

Auxiliary and modal verbs

Copula: *be* with a predicate

She is a student. I am hungry. She was five years old.

Modal verbs: *can*, *may*, *must*, *might*, *shall*,…

She can swim. You must come

Auxiliary verbs:

- *Be*, *have*, *will* when used to form complex tenses:

He was being followed. She has seen him. We will have been gone.

- *Do* in questions, negation:

Don’t go. Did you see him?

Penn Treebank tags:

MD: modal verbs

Prepositions

Prepositions occur before noun phrases
to form a prepositional phrase (PP):

on/in/under/near/towards the wall,
with(out) milk,
by the author,
despite your protest

PPs can modify nouns, verbs or sentences:

I drink [coffee [with milk]]
I [drink coffee [with my friends]]

Penn Treebank tags:

IN: preposition
TO: ‘to’ (infinitival ‘to eat’ and preposition ‘to you’)

Conjunctions

Coordinating conjunctions conjoin two elements:

X and/or/but X

[[John]NP and [Mary]NP] NP,

[[Snow is cold]S but [fire is hot]S]S.

Subordinating conjunctions introduce a subordinate (embedded) clause:

[He thinks that [snow is cold]S]S

[She wonders whether [it is cold outside]S]S

Penn Treebank tags:

CC: coordinating

IN: subordinating (same as preposition)

Particles

Particles resemble prepositions (but are not followed by a noun phrase) and appear with verbs:

come on
he brushed himself off
turning the paper over
turning the paper down

Phrasal verb: a verb + particle combination that has a different meaning from the verb itself

Penn Treebank tags:

RP: particle

Pronouns

Many pronouns function like noun phrases, and refer to some other entity:

- Personal pronouns: I, you, he, she, it, we, they
- Possessive pronouns: mine, yours, hers, ours
- Demonstrative pronouns: this, that,
- Reflexive pronouns: myself, himself, ourselves
- Wh-pronouns (question words):
  what, who, whom, how, why, whoever, which

Relative pronouns introduce relative clauses

the book that [he wrote]

Penn Treebank tags:

PRP: personal pronoun   PRP$ possessive   WP: wh-pronoun

Determiners

Determiners precede noun phrases:

the/that/a/every book

- Articles: the, an, a
- Demonstratives: this, these, that
- Quantifiers: some, every, few,…

Penn Treebank tags:

DT: determiner