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

Lecture 7: Sequence Labeling

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

Recap: Statistical POS tagging with HMMs

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

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

POS tagging with generative models

$$\begin{aligned}
\underset{\mathbf{t}}{\text{argmax}}\, P(\mathbf{t}|\mathbf{w}) &= \underset{\mathbf{t}}{\text{argmax}}\, \frac{P(\mathbf{t},\mathbf{w})}{P(\mathbf{w})} \\
&= \underset{\mathbf{t}}{\text{argmax}}\, P(\mathbf{t},\mathbf{w}) \\
&= \underset{\mathbf{t}}{\text{argmax}}\, P(\mathbf{t})P(\mathbf{w}|\mathbf{t})
\end{aligned}$$

$P(\mathbf{t},\mathbf{w})$: 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 generative models for POS tagging
(and other tasks, e.g. in speech recognition)

Independence assumptions of HMMs

$P(\mathbf{t})$ is an n-gram model over tags:

Bigram HMM: $P(\mathbf{t}) = P(t^{(1)})P(t^{(2)} \mid t^{(1)})P(t^{(3)} \mid t^{(2)}) \dots P(t^{(N)} \mid t^{(N-1)})$

Trigram HMM: $P(\mathbf{t}) = P(t^{(1)})P(t^{(2)} \mid t^{(1)})P(t^{(3)} \mid t^{(2)},t^{(1)}) \dots P(t^{(n)} \mid t^{(N-1)},t^{(N-2)})$

$P(t_i \mid t_j)$ or $P(t_i \mid t_j,t_k)$ are called transition probabilities

In $P(\mathbf{w} \mid \mathbf{t})$ each word is generated by its own tag:

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

$P(w \mid t)$ are called emission probabilities

Viterbi algorithm

Task: Given an HMM, return most likely tag sequence $t^{(1)} \dots t^{(N)}$ for a given word sequence (sentence) $w^{(1)} \dots w^{(N)}$

Data structure (Trellis): N×T table for sentence $w^{(1)} \dots w^{(N)}$ and tag set $\{t_1, \dots t_T\}$. Cell trellis[i][j] stores score of best tag sequence for $w^{(1)} \dots w^{(j)}$ that ends in tag t_j and a backpointer to the cell corresponding to the tag of the preceding word trellis[i−1][k]

Basic procedure:

Fill trellis from left to right

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

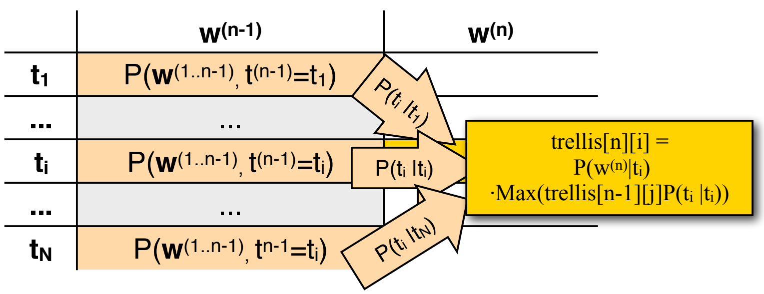
For trellis[i][j]:

- Find best preceding tag k* = argmax_k(trellis[i−1][k] × $P(t_j \mid t_k)$),
- Add backpointer from trellis[i][j] to trellis[i−1][k*];
- Set trellis[i][j] := trellis[i−1][k*] × $P(t_j \mid t_{k^*}) \times P(w^{(i)} \mid t_j)$

Return tag sequence that ends in the highest scoring cell argmax_k(trellis[N][k]) in the last column

Viterbi: At any given 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



Other HMM algorithms

The Forward algorithm:

Computes $P(\mathbf{w})$ by replacing Viterbi's max() with sum()

Learning HMMs from raw text with the EM algorithm:

- We have to replace the observed counts (from labeled data) with expected counts (according to the current model)
- Renormalizing these expected counts will give a new model
- This will be “better” than the previous model, but we will have to repeat this multiple times to get to decent model

The Forward-Backward algorithm:

A dynamic programming algorithm for computing the expected counts of tag bigrams and word-tag occurrences in a sentence under a given HMM

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...
- **O**: outside of any mention of a named entity

```
[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
```

Many NLP tasks are sequence labeling tasks

Input: a sequence of tokens/words:

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

Output: a sequence of **labeled** tokens/words:

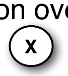
POS-tagging: `Pierre_NNP Vinken_NNP ,_, 61_CD years_NNS old_JJ ,_, will_MD join_VB IBM_NNP 's_POS board_NN as_IN a_DT nonexecutive_JJ director_NN Nov._NNP 29_CD ._.`

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`

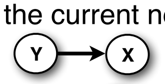
Graphical models for sequence labeling

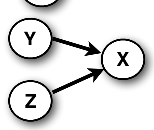
Directed graphical models

Graphical models are a **notation for probability models**.
In a **directed** graphical model, **each node** represents a distribution over a random variable:

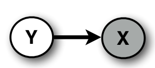
– P(X) = 

Arrows represent dependencies (they define what other random variables the current node is conditioned on)

– P(Y) P(X | Y) = 

– P(Y) P(Z) P(X | Y, Z) = 

Shaded nodes represent observed variables.
White nodes represent hidden variables

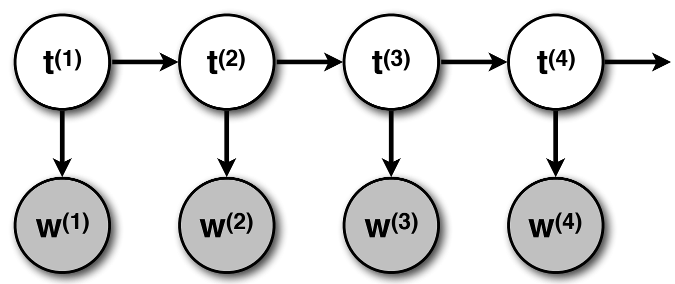
– P(Y) P(X | Y) with Y hidden and X observed = 

HMMs as graphical models

HMMs are **generative** models of the observed input string **w**

They 'generate' **w** with $P(\mathbf{w},\mathbf{t}) = \prod_i P(t^{(i)} | t^{(i-1)}) P(w^{(i)} | t^{(i)})$

When we use an HMM to tag, we observe **w**, and need to find **t**



Models for sequence labeling

Sequence labeling: Given an input sequence $\mathbf{w} = w^{(1)}\ldots w^{(n)}$, predict the best (most likely) label sequence $\mathbf{t} = t^{(1)}\ldots t^{(n)}$

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

Generative models use Bayes Rule:

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

Discriminative (conditional) models model $P(\mathbf{t}|\mathbf{w})$ directly

Advantages of discriminative models

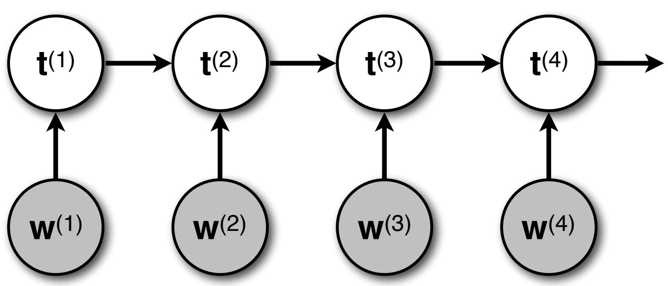
We're usually not really interested in $P(\mathbf{w} \mid \mathbf{t})$.
– $\mathbf{w}$ is given. We don't need to predict it!
Why not model what we're actually interested in: $P(\mathbf{t} \mid \mathbf{w})$

Modeling $P(\mathbf{w} \mid \mathbf{t})$ well is quite difficult:
- Prefixes (capital letters) or suffixes are good predictors for certain classes of $\mathbf{t}$ (proper nouns, adverbs,…)
- Se we don't want to model words as atomic symbols, but in terms of features
- These features may also help us deal with unknown words
- But features may not be independent

Modeling $P(\mathbf{t} \mid \mathbf{w})$ with features should be easier:
- Now we can incorporate arbitrary features of the word, because we don't need to predict $\mathbf{w}$ anymore

Discriminative probability models

A discriminative or **conditional** model of the labels **t** given the observed input string **w** models $P(\mathbf{t} \mid \mathbf{w}) = \prod_i P(t^{(i)} \mid w^{(i)}, t^{(i-1)})$ directly.



Discriminative models

There are two main types of discriminative probability models:
- Maximum Entropy Markov Models (MEMMs)
- Conditional Random Fields (CRFs)

MEMMs and CRFs:
- are both based on logistic regression
- have the same graphical model
- require the Viterbi algorithm for tagging
- differ in that MEMMs consist of independently learned distributions, while CRFs are trained to maximize the probability of the entire sequence

Probabilistic classification

Classification:
Predict a class (label) c for an input $\mathbf{x}$
There are only a (small) finite number of possible class labels

Probabilistic classification:
– Model the probability $P(c \mid \mathbf{x})$
$P(c|\mathbf{x})$ is a probability if $0 \leq P(c_i \mid \mathbf{x}) \leq 1$, and $\sum_i P(c_i \mid \mathbf{x}) = 1$
– Return the class $c^* = \mathrm{argmax}_i\ P(c_i \mid \mathbf{x})$ that has the highest probability

One standard way to model $P(c \mid \mathbf{x})$ is logistic regression (used by MEMMs and CRFs)

Using features

Think of feature functions as useful questions you can ask about the input $\mathbf{x}$:

– **Binary feature functions**:
$f_{\text{first-letter-capitalized}}(\textbf{Urbana}) = 1$
$f_{\text{first-letter-capitalized}}(\textbf{computer}) = 0$

– **Integer (or real-valued) features**:
$f_{\text{number-of-vowels}}(\textbf{Urbana}) = 3$

Which specific feature functions are useful will depend on your task (and your training data).

From features to probabilities

We associate a real-valued weight w_{ic} with each feature function $f_i(\mathbf{x})$ and output class c
Note that the feature function $f_i(\mathbf{x})$ does not have to depend on c as long as the weight does (note the double index w_{ic})
This gives us a real-valued score for predicting class c for input $\mathbf{x}$: $\mathrm{score}(\mathbf{x},c) = \sum_i w_{ic} f_i(\mathbf{x})$

This score could be negative, so we exponentiate it:
$\mathrm{score}(\mathbf{x},c) = \exp(\sum_i w_{ic} f_i(\mathbf{x}))$
To get a probability distribution over all classes c, we renormalize these scores:
$P(c \mid \mathbf{x}) = \mathrm{score}(\mathbf{x},c) / \sum_j \mathrm{score}(\mathbf{x},c_j)$
$= \exp(\sum_i w_{ic} f_i(\mathbf{x})) / \sum_j \exp(\sum_i w_{ij} f_i(\mathbf{x}))$

Learning: finding *w*

Learning = finding weights ***w***
We use conditional maximum likelihood estimation (and standard convex optimization algorithms) to find/learn ***w***
(for more details, attend CS446 and CS546)

The conditional MLE training objective:
Find the **w** that assigns highest probability to all observed outputs c_i given the inputs $\mathbf{x}_i$

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \prod_i P(c_i|\mathbf{x}_i, \mathbf{w})$$

Terminology

Models that are of the form

$P(c \mid \mathbf{x}) = \text{score}(\mathbf{x},c) / \sum_j \text{score}(\mathbf{x},c_j)$

$\quad = \exp(\sum_i w_{ic} f_i(\mathbf{x})) / \sum_j \exp(\sum_i w_{ij} f_i(\mathbf{x}))$

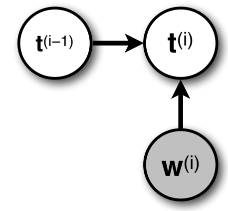
are also called loglinear models, Maximum Entropy (MaxEnt) models, or multinomial logistic regression models.

CS446 and CS546 should give you more details about these.

The normalizing term $\sum_j \exp(\sum_i w_{ij} f_i(\mathbf{x}))$ is also called the partition function and is often abbreviated as Z

Maximum Entropy Markov Models

MEMMs use a MaxEnt classifier for each $P(t^{(i)} \mid w^{(i)}, t^{(i-1)})$:



Since we use w to refer to words, let's use λ_{jk} as the weight for the feature function $f_j(t^{(i-1)}, w^{(i)})$ when predicting tag t_k:

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

Viterbi for MEMMs

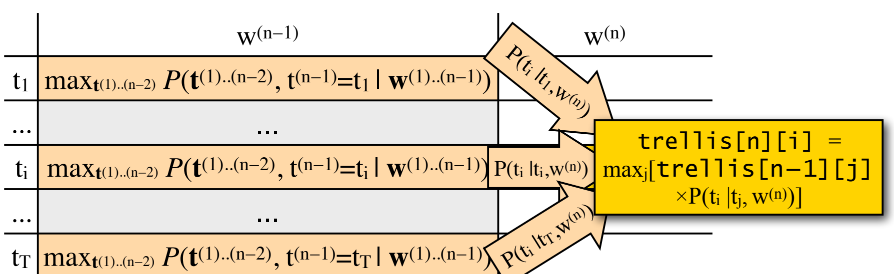
`trellis[n][i]` stores the probability of the most likely (Viterbi) tag sequence $\mathbf{t}^{(1)\ldots(n)}$ that ends in tag t_i for the prefix $w^{(1)} \ldots w^{(n)}$

Remember that we do not generate $\mathbf{w}$ in MEMMs. So:

`trellis[n][i]` $= \max_{\mathbf{t}^{(1)..(n-1)}}[P(\mathbf{t}^{(1)\ldots(n-1)}, t^{(n)}=t_i \mid \mathbf{w}^{(1)\ldots(n)})]$

$= \max_j$ [`trellis[n-1][j]` $\times P(t_i \mid t_j, w^{(n)})$]

$= \max_j [\max_{\mathbf{t}^{(1)..(n-2)}}[P(\mathbf{t}^{(1)..(n-2)}, t^{(n-1)}=t_j \mid \mathbf{w}^{(1)..(n-1)})] \times P(t_i \mid t_j, w^{(n)})]$



Today's key concepts

Sequence labeling tasks:

- POS tagging
- NP chunking
- Shallow Parsing
- Named Entity Recognition

Discriminative models:

- Maximum Entropy classifiers
- MEMMs

Supplementary material: Other HMM algorithms (very briefly…)

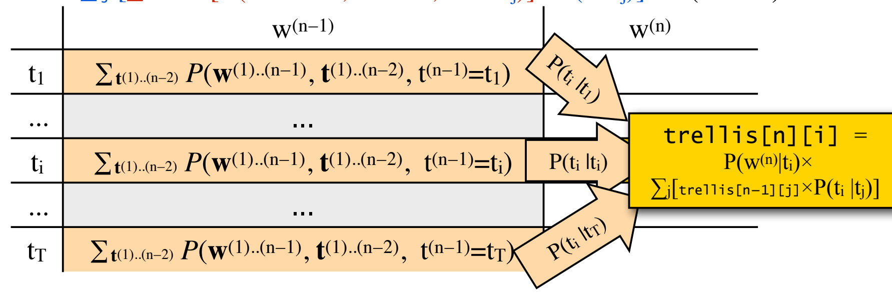
The Forward algorithm

`trellis[n][i]` stores the probability mass of all tag sequences $\mathbf{t}^{(1)\dots(n)}$ that end in tag t_i for the prefix $w^{(1)}\dots w^{(n)}$

$$\texttt{trellis[n][i]} = \sum_{\mathbf{t}^{(1)..(n-1)}} [\, P(\mathbf{w}^{(1)\dots(n)}, \mathbf{t}^{(1)\dots(n-1)}, t^{(n)}=t_i)\,]$$

$$= \sum_j [\, \texttt{trellis[n-1][j]} \times P(t_i \mid t_j)\,] \times P(w^{(n)} \mid t_i)$$

$$= \sum_j [\sum_{\mathbf{t}^{(1)..(n-2)}} [P(\mathbf{w}^{(1)..(n-1)}, \mathbf{t}^{(1)..(n-2)}, t^{(n-1)}=t_j)] \times P(t_i \mid t_j)] \times P(w^{(n)} \mid t_i)$$



Last step: computing $P(\mathbf{w})$: $P(\mathbf{w}^{(1)\dots(N)}) = \sum_j \texttt{trellis[N][j]}$

Learning an HMM from *unlabeled* text

```
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* $\langle C(\dots)\rangle$

- Our estimate for the transition probabilities:

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

- Our estimate for the emission probabilities:

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

- Our estimate for the initial state probabilities:

$$\pi(t_i) = \frac{\langle C(\text{Tag of first word } = t_i)\rangle}{\text{Number of sentences}}$$

Expected counts

Emission probabilities with *observed* counts $C(w, t)$

$$P(w \mid t) = C(w,t)/C(t) = C(w,t)/\sum_{w'} C(w', t)$$

Emission probabilities with *expected* counts $\langle C(w, t)\rangle$

$$P(w \mid t) = \langle C(w,t)\rangle / \langle C(t)\rangle = \langle C(w,t)\rangle / \sum_{w'} \langle C(w', t)\rangle$$

$\langle C(w, t)\rangle$: How often do we expect to see word w with tag t in our training data (under a given HMM)?

We know how often the word w appears in the data, but we don't know how often it appears with tag t

We need to sum up $\langle C(w^{(i)}=w, t)\rangle$ for any occurrence of w

We can show that $\langle C(w^{(i)}=w, t)\rangle = P(t^{(i)}=t \mid \mathbf{w})$

(NB: Transition counts $\langle C(t^{(i)}=t, t^{(i+1)}=t')\rangle$ work in a similar fashion)

Forward-Backward: $P(t^{(i)}=t \mid \mathbf{w}^{(1)..(N)})$

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

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

Due to HMM's independence assumptions:

$P(t^{(i)}=t, \mathbf{w}^{(1)\ldots(N)}) = P(t^{(i)}=t, \mathbf{w}^{(1)\ldots(i)}) \times P(\mathbf{w}^{(i+1)\ldots(N)} \mid t^{(i)}=t)$

The forward algorithm gives $P(\mathbf{w}^{(1)\ldots(N)}) = \sum_t \text{forward}[N][t]$

Forward trellis: $\text{forward}[i][t] = P(t^{(i)}=t, \mathbf{w}^{(1)\ldots(i)})$

Gives the total probability mass of the **prefix** $\mathbf{w}^{(1)\ldots(i)}$, summed over all tag sequences $\mathbf{t}^{(1)\ldots(i)}$ that end in tag $t^{(i)}=t$

Backward trellis: $\text{backward}[i][t] = P(\mathbf{w}^{(i+1)\ldots(N)} \mid t^{(i)}=t)$

Gives the total probability mass of the **suffix** $\mathbf{w}^{(i+1)\ldots(N)}$, summed over all tag sequences $\mathbf{t}^{(i+1)\ldots(N)}$, if we assign tag $t^{(i)}=t$ to $w^{(i)}$

The Backward algorithm

The backward trellis is filled from right to left.

backward[i][t] provides $P(\mathbf{w}^{(i+1)\ldots(N)} \mid t^{(i)}=t)$

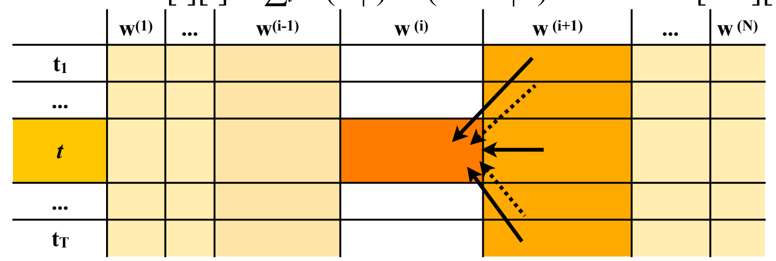
NB: $\sum_t \text{backward}[1][t] = P(\mathbf{w}^{(i+1)\ldots(N)}) = \sum_t \text{forward}[N][t]$

Initialization (last column):

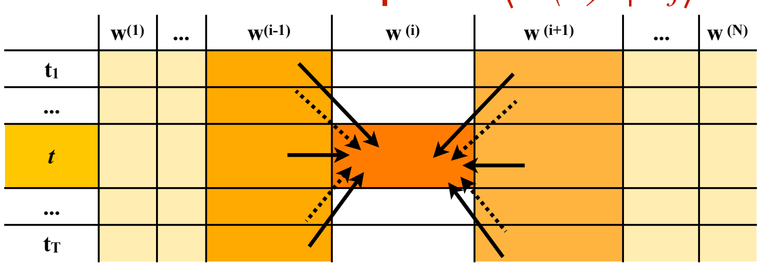
$\text{backward}[N][t] = 1$

Recursion (any other column):

$\text{backward}[i][t] = \sum_{t'} P(t' \mid t) \times P(w^{(i+1)} \mid t') \times \text{backward}[i+1][t']$



How do we compute $\langle C(t_i) \mid w_j \rangle$



$\langle C(t, w^{(i)}) \mid \mathbf{w} \rangle = P(t^{(i)}=t, \mathbf{w}) / P(\mathbf{w})$

with

$P(t^{(i)}=t, \mathbf{w}) = \text{forward}[i][t]\ \text{backward}[i][t]$

$P(\mathbf{w}) = \sum_t \text{forward}[N][t]$

The importance of tag dictionaries

Forward-Backward assumes that each tag can be assigned to any word.

No guarantee that the learned HMM bears any resemblance to the tags we want to get out of a POS tagger.

A tag dictionary lists the possible POS tags for words.

Even a partial dictionary that lists only the tags for the most common words and contains at least a few words for each tag provides enough constraints to get significantly closer to a model that produces linguistically correct (and hence useful) POS tags.

a	DT	back	JJ, NN, VB, VBP, RP
an	DT	bank	NN, VB, VBP
and	CC	…	…
America	NNP	zebra	NN