CS447: Natural Language Processing

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

Lecture 8:

The Forward-Backward algorithm

Julia Hockenmaier

juliahmr@illinois.edu

3324 Siebel Center

Wednesday’s key concepts

HMM taggers

Learning HMMs from labeled text

Viterbi for HMMs

- Dynamic programming
- Independence assumptions in HMMs
- The trellis

Recap: 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):

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

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

Initial state probabilities:
$$\pi(t_i) = \frac{C(\text{Tag of first word } = t_i)}{\text{Number of sentences}}$$

Recap: The Viterbi algorithm

What: Viterbi finds the **most likely tag sequence** $\mathbf{t}^* = t^{(1)}\ldots t^{(N)}$ for an input sentence (word sequence) $\mathbf{w} = w^{(1)}\ldots w^{(N)}$

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

The most likely tag sequence is also called the Viterbi sequence

How: Viterbi is a **dynamic programming** algorithm that uses a N×T trellis (table) in which each cell `trellis[n][i]` stores:

- the **probability** of the most likely (Viterbi) tag sequence for the prefix $w^{(1)}\ldots w^{(n)}$ that ends in tag t_i
- and a **backpointer** to the cell `trellis[n−1][j]`, where $t^{(n-1)} = t_j$ is the tag of word $w^{(n-1)}$ in this Viterbi sequence

The cell `trellis[N][i]` with the largest probability in the last column tells us which tag $t^{(N)} = t_i$ the Viterbi sequence $\mathbf{t}^*$ of $\mathbf{w}$ ends in. We extract $\mathbf{t}^*$ by following the backpointers.

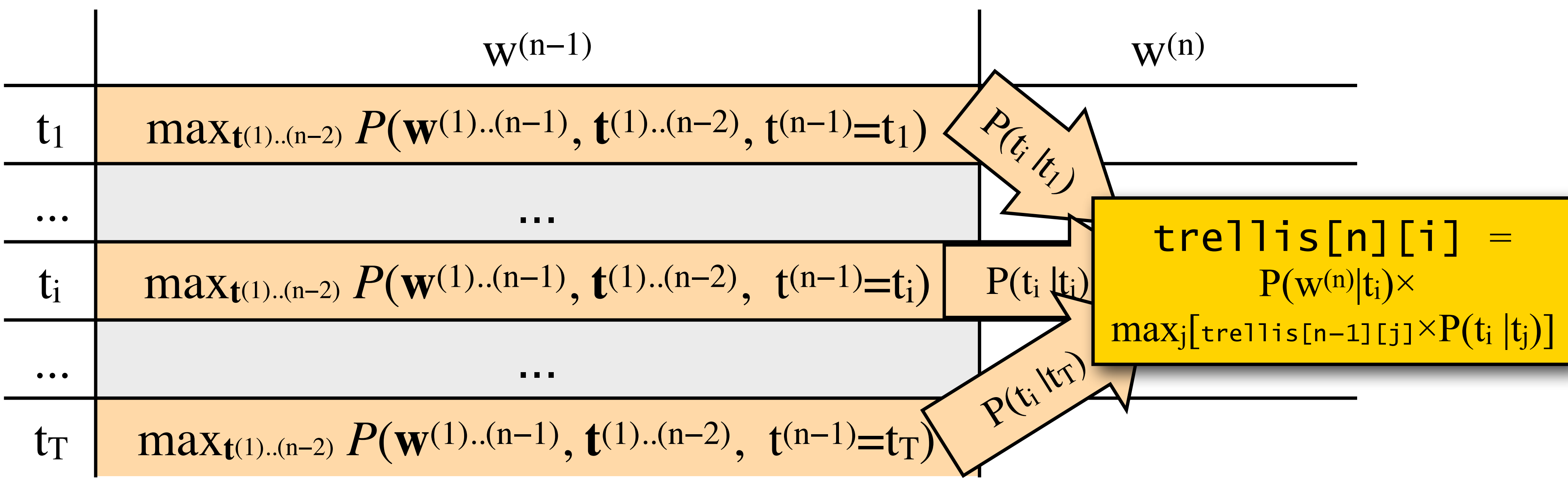
Viterbi

`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 $w^{(1)}\ldots w^{(n)}$

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

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

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



Today’s key concepts

The Forward algorithm: computing $P(\mathbf{w})$
The Forward-Backward algorithm: learning HMMs from raw text

The Forward algorithm: Computing P(**w**)

The Forward algorithm

The HMM defines a language model: $P(\mathbf{w}) = \sum_{\mathbf{t}} P(\mathbf{t},\mathbf{w})$

- To compute $P(\mathbf{w})$, sum ('*marginalize*') over all tag sequences $\mathbf{t}$

How can we compute $P(\mathbf{w})$ efficiently?

- Use dynamic programming!

In the Viterbi algorithm, we want the probability of the *best* sequence for $\mathbf{w}^{(1)\ldots(n)}$ that ends in t_i:

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

In the Forward algorithm, we want the total probability mass of *all* sequences for $\mathbf{w}^{(1)\ldots(n)}$ that end in t_i:

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

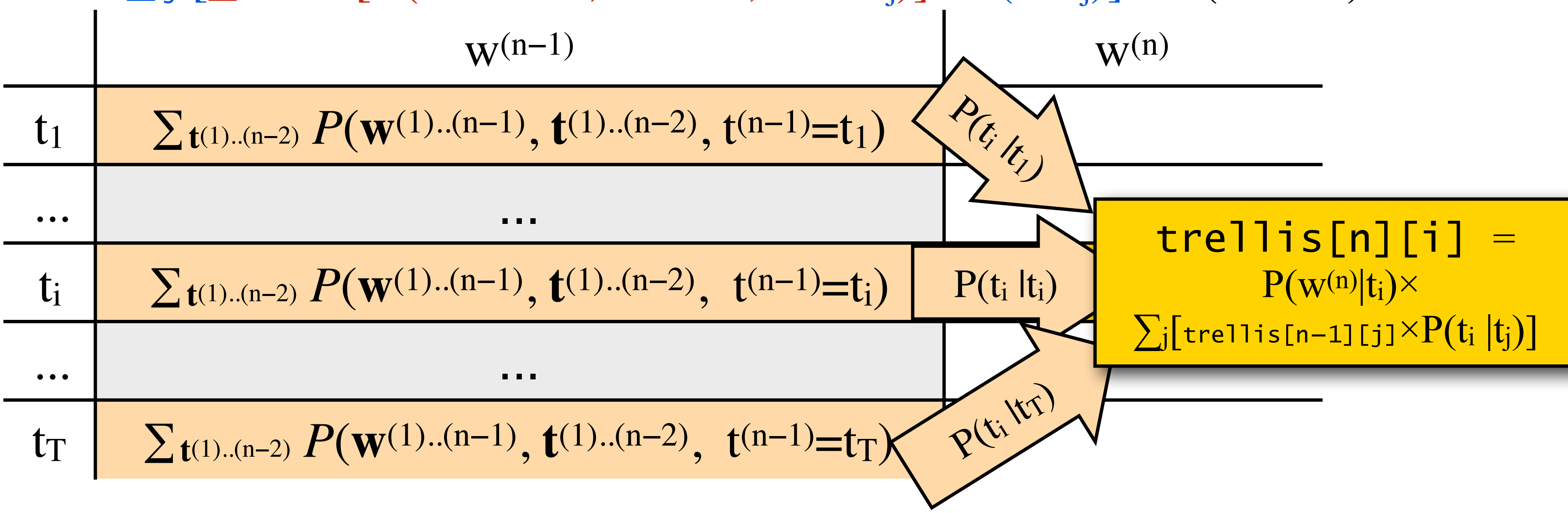
The Forward algorithm

`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 $w^{(1)}\dots w^{(n)}$

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

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

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



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

Learning an HMM from raw text

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* $\langle C(\ldots) \rangle$

- 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:

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

- Our estimate for the initial state probabilities:

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

Learning HMMs from raw text

Chicken-and-Egg problem:

We need a probability model to compute expected counts $\langle C(...)\rangle$

Solution: iterative hill-climbing

– Start with an initial model $\lambda^{(0)}$ to compute expectations.

– Use these expectations to recompute a new model.

– *Iterate*: Use this model to compute new expectations,…

(N.B.: this yields a Maximum-Likelihood estimate)

Hill-climbing:

Each iteration yields a model $\lambda^{(t+1)}$ that assigns at least as much probability (likelihood) to the training data as $\lambda^{(t)}$.

This is an instance of the **Expectation-Maximization (EM)** algorithm

Learning an HMM: the EM algorithm

Initialization:

– Take a data set S

– Guess initial parameters $A^{(0)}$, $B^{(0)}$, $\pi^{(0)}$
  These define the HMM $\lambda^{(i)} = \lambda^{(0)} = (A^{(0)}, B^{(0)}, \pi^{(0)})$

The Expectation (E) step:

– Use $\lambda^{(i)}$ to compute expected counts
  $\langle C(t) \mid \lambda^{(i)}, S \rangle$ and $\langle C(w, t) \mid \lambda^{(i)}, S \rangle$ for all words w and tags t

The Maximization (M) step

– Estimate a new HMM $\lambda^{(i+1)}$ from $\langle C(t) \mid \lambda^{(i)}, S \rangle$, $\langle C(w, t) \mid \lambda^{(i)}, S \rangle$

Repeat the E and M steps until λ *converges*

Computing $\langle C(w,t) | \lambda^{(i)}, \mathbf{S} \rangle, \langle C(t) | \lambda^{(i)}, \mathbf{S} \rangle$

$\langle C(t) \mid \lambda^{(i)}, \mathbf{S} \rangle = \sum_w \langle C(w, t) \mid \lambda^{(i)}, \mathbf{S} \rangle$

How often do we expect to see tag t in the corpus $\mathbf{S}$?

→ Sum over all words w

$\langle C(w, t) \mid \lambda^{(i)}, \mathbf{S} \rangle = \sum_j \langle C(w, t) \mid \lambda^{(i)}, S_j \rangle$

How often do we expect to see tag t with a specific word w in corpus $\mathbf{S}$?

→ Sum over all sentences S_j in $\mathbf{S}$

$\langle C(w, t) \mid \lambda^{(i)}, S_j \rangle = \sum_{k:\, w^{(k)} = w} \langle C(w, t) \mid \lambda^{(i)}, S_j \rangle$

How often do we expect to see tag t with a specific word w in sentence S_j ?

→ Sum over all positions k in S_j that are occupied by w ($w^{(k)}$ is equal to w).

Computing $\langle C(w^{(k)} = w, t^{(k)} = t) \mid \lambda^{(i)}, S_j \rangle$

$\langle C(w^{(k)} = w, t^{(k)} = t) \mid \lambda^{(i)}, S_j \rangle$:
How often do we expect to see tag t
in position k in sentence S_j ?

Supervised learning:
$w^{(k)}$ has tag $t^{(k)}$, hence $C(w^{(k)}, t^{(k)}) = 1$

Unsupervised learning:
$w^{(k)}$ can have any tag t, hence $\Sigma_i \langle C(w^{(k)}, t_i) \rangle = 1$
$\langle C(w^{(k)}, t) \rangle$ is the conditional probability of tag t in position k (in sentence S_j).

How do we compute $\langle C(t, w^{(i)}) \mid \mathbf{w} \rangle$

	$\mathbf{w}^{(1)}$	...	$\mathbf{w}^{(i-1)}$	$\mathbf{w}^{(i)}$	$\mathbf{w}^{(i+1)}$	...	$\mathbf{w}^{(N)}$
$\mathbf{t}_1$							
...							
$\boldsymbol{t}$							
...							
$\mathbf{t}_T$							

- With a slight abuse of notation, I'm using $\langle C(t, w^{(i)}) \mid \mathbf{w} \rangle$ to refer to the expected count of tag t occurring with the i-th word in $\mathbf{w} = w^{(1)} \ldots w^{(i)} \ldots w^{(N)}$
- We need to look at the k-th cell in the row corresponding to tag t

How do we compute $\langle C(t,w^{(i)}) \mid \mathbf{w} \rangle$

$\langle C(t,w^{(i)}) \mid \mathbf{w} \rangle$ is equal to the conditional probability that the i-th tag for $\mathbf{w}$ ($w^{(i)}$'s tag) is t:

$$\begin{aligned} \langle C(t,w^{(i)}) \mid \boldsymbol{w} \rangle &= P(t^{(i)} = t \mid \mathbf{w}) \\ &= P(t^{(i)} = t, \mathbf{w}) / P(\mathbf{w}) \end{aligned}$$

$P(t^{(i)} = t, \mathbf{w})$ is the total probability mass of $\mathbf{w}$ with any of the tag sequences for $\boldsymbol{w}$ where the i-th tag is t

The forward algorithm tells us how to compute $P(\mathbf{w})$

How do we compute $\langle C(t, w^{(i)}) \mid \mathbf{w} \rangle$

$P(t^{(i)} = t, \mathbf{w})$ is the total probability mass of all tag sequences for $\mathbf{w}$ where the i-th tag is t

This decomposes into two terms

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

The first term $P(t^{(i)} = t, \mathbf{w}^{(1)\dots(i)})$ is the probability mass of the prefix $\mathbf{w}^{(1)\dots(i)}$ with all tag sequences $\mathbf{t}^{(1)\dots(i)}$ that end in t

We can get this from the cell corresponding to $w^{(i)}$ and t in the forward trellis: $P(t^{(i)} = t, \mathbf{w}^{(1)\dots(i)}) = \mathrm{forward}[i][t]$

The second term $P(\mathbf{w}^{(i+1)\dots(N)} \mid t^{(i)} = t)$ is the probability mass of the suffix $\mathbf{w}^{(i+1)\dots(N)}$ with all tag sequences $\mathbf{t}^{(i+1)\dots(N)}$ given that $t^{(i)} = t$

How do we compute $\langle C(t, w^{(i)}) | \mathbf{w} \rangle$

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

$P(t^{(i)} = t, \mathbf{w}^{(1)\ldots(i)}) = \mathrm{forward}[i][\underline{t}]$
is the forward probability of t and $w^{(i)}$
computed by the forward algorithm

Correspondingly,
$P(\mathbf{w}^{(i+1)\ldots(N)} \mid t^{(i)} = t) = \mathrm{backward}[i][t]$
is the backward probability of t and $w^{(i)}$
computed by the backward algorithm

The forward algorithm

The forward trellis is filled from left to right.

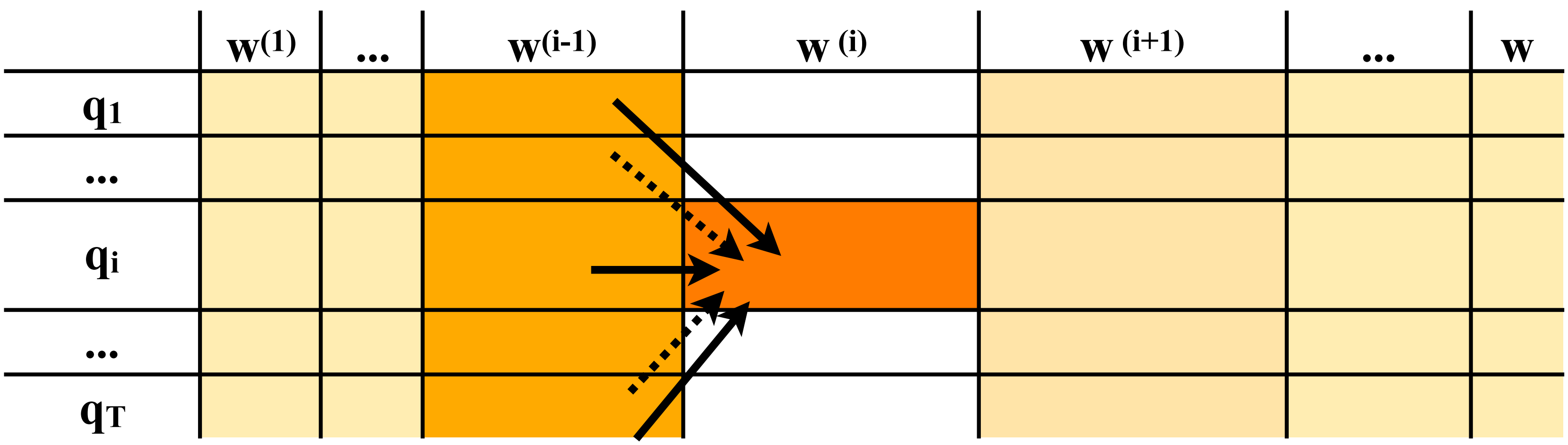
forward[i][t] provides $P(t^{(i)} = t, \mathbf{w}^{(1)\ldots(i)})$

Initialization (first column):

$$\text{forward}[1][t] = \pi(t) P(w^{(1)} \mid t)$$

Recursion (any other column):

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



The backward algorithm

The backward trellis is filled from right to left.

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

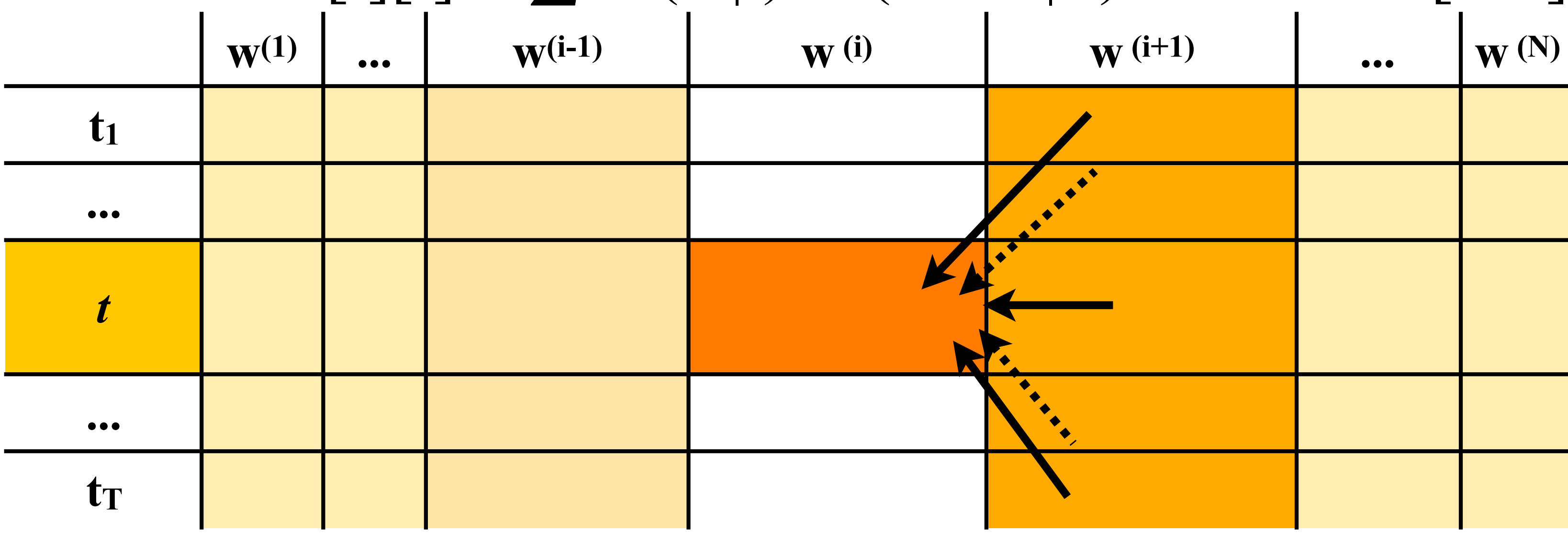
NB: $\sum_t \text{backward}[1][t] = P(\boldsymbol{w}^{(i+1)\ldots(N)}) = \sum_t \text{forward}[N][t]$

Initialization (last column):

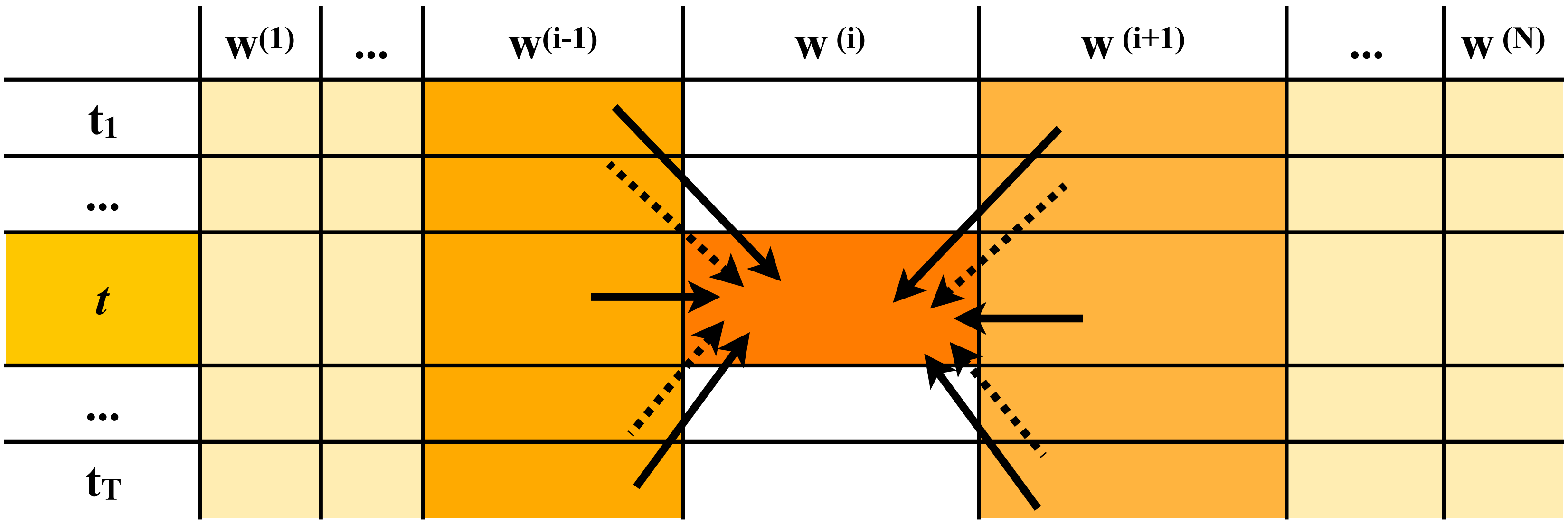
backward[N][t] = 1

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 $\langle C(t_i)\ |w_j\rangle$



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

with

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

$$P(\boldsymbol{w}) \qquad = \textstyle\sum_{t} \text{forward[N][t]}$$

How do we compute $P(t' \mid t)$?

How often do we expect tag t to transition to tag t' ?

Summing over all sentences $\boldsymbol{w}$, and all pairs of adjacent positions i, $(i+1)$, compute how often we expect the tag bigram " $t\ t'$ " starting at position i:

Compute $\langle C(t^{(i)} = t,\ t^{(i+1)} = t') \mid \boldsymbol{w} \rangle$

This is the same as the (conditional) probability mass of all tag sequences for $\boldsymbol{w}$ that have t and t' in the ith and $(i+1)$th position:

$$\langle C(t^{(i)} = t,\ t^{(i+1)} = t') \mid \boldsymbol{w} \rangle = P(\ t^{(i)} = t,\ t^{(i+1)} = t' \mid \boldsymbol{w}\ )$$
$$= P(\ t_i = t,\ t_{i+1} = t',\ \boldsymbol{w}\ ) / P(\ \boldsymbol{w}\ )$$

Computing $P(t^{(i)} = t, t^{(i+1)} = t', \mathbf{w})$

The probability of all tag sequences for $\mathbf{w}$ that have t and t' in the ith and $(i+1)$th position factors into

- the forward probability $\mathrm{forward}[i][t]$
  (i.e. the probability of the prefix $w^{(1)...(i)}$ and all tag sequences $t^{(1)...(i)}$ that end in $t^{(i)} = t$)
- the transition probability $P(t \mid t')$
- the emission probability $P(w^{(i+1)} \mid t')$
- the backward probability $\mathrm{backward}[i+1][t']$
  (i.e. the probability of the suffix $\mathbf{w}^{(i+1)...(N)}$ and all tag sequences $\boldsymbol{t}^{(i+1)...(N)}$ given that $t^{(i)} = t$)

$$P(t^{(i)} = t,\ t^{(i+1)} = t', \mathbf{w})$$
$$= P(t^{(i)} = t, \mathbf{w}^{(1)...(i)}) \times P(t' \mid t) \times P(w^{(i+1)} \mid t') \times P(w^{(i+2)...(N)} \mid t^{(i+1)} = t')$$
$$= \mathrm{forward}[i][t] \times P(t' \mid t) \times P(w^{(i+1)} \mid t') \times \mathrm{backward}[i+1][t']$$

Computing $\pi(t)$

We need to compute $\langle C(t^{(1)} = t) \mid \boldsymbol{w} \rangle = P(t^{(1)} = t \mid \boldsymbol{w})$

Again, we get the conditional probability $P(\ldots \mid \boldsymbol{w})$
by dividing the joint probability $P(t^{(1)} = t, \boldsymbol{w})$ by $P(\boldsymbol{w})$:

$$P(t^{(1)} = t \mid \boldsymbol{w}) = P(t^{(1)} = t, \boldsymbol{w})/P(\boldsymbol{w})$$

Therefore, we only need to figure out how to compute the joint probability $P(t^{(1)} = t, \boldsymbol{w})$:

$$\begin{aligned} P(t^{(1)} = t, \boldsymbol{w}) &= \pi(t) \times P(w^{(1)} \mid t) \times P(\boldsymbol{w}^{(2)\ldots(N)} \mid t^{(1)} = t) \\ &= \pi(t) \times P(w^{(1)} \mid t) \times \text{backward}[t][1] \end{aligned}$$

Numerical issues (EM and Viterbi)

Multiplying many small probabilities together leads to numerical problems, since the floating numbers are likely to underflow.

We therefore typically operate in log space:
instead of multiplying probabilities *p*(...),
sum the corresponding log probabilities *log p*(...)

We still have to compute log(X + Y)
(see next slide)

Computing log(X+Y) from log(X),log(Y)

from https://facwiki.cs.byu.edu/nlp/index.php/Log_Domain_Computations

```
public static double logAdd(double logX, double logY) {
     // 1. make X the max
     if (logY > logX) {
         double temp = logX;
         logX = logY;
         logY = temp;
     }
     // 2. now X is bigger
     if (logX == Double.NEGATIVE_INFINITY) {
         return logX;
     }
     // 3. how far "down" (think decibels) is logY from logX?
     //    if it's really small (20 orders of magnitude smaller), then ignore
     double negDiff = logY - logX;
     if (negDiff < -20) {
         return logX;
     }
     // 4. otherwise use some nice algebra to stay in the log domain
     //    (except for negDiff)
     return logX + java.lang.Math.log(1.0 + java.lang.Math.exp(negDiff));
 }
```

Today’s lecture

The Forward algorithm:

Computing P(**w**)

The Forward-Backward algorithm:

Learning HMMs from raw text

Uses the Forward algorithm and the Backward algorithm

Required reading: Ch. 6.1-5

Optional reading: Manning & Schütze, Chapter 9