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

Lecture 11: More POS tagging, Sequence labeling

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

Midterm Exam

When: Friday, October 11, 2019 in class
Where: DCL 1310 (this room)
What: Closed book exam:

- You are not allowed to use any cheat sheets, computers, calculators, phones etc.
(you shouldn't have to anyway)
- Only the material covered in lectures
- Bring a pen (black/blue) or pencil
- **Short questions — we expect short answers!**
- **Tip:** If you can't answer a question, move on to the next one. You may not be able to complete the whole exam in the time given — there will be a lot of questions, so first do the ones you know how to answer!

Question types

***Define* X:**
Provide a mathematical/formal definition of X

***Explain* X; *Explain* what X is/does:**
Use plain English to define X and say what X is/does

***Compute* X:**
Return X; Show the steps required to calculate it

***Draw* X:**
Draw a figure of X

***Show/Prove* that X is true/is the case/…:**
This may require a (typically very simple) proof.

***Discuss/Argue* whether …**
Use your knowledge (of X,Y,Z) to argue your point

Basics about language

Explain Zipf's law and why it makes NLP difficult.

Explain why we often use statistical models in NLP.

Give two examples of ambiguity and explain how they make natural language understanding difficult.

Basics about language

Explain Zipf's law and why it makes NLP difficult.

Zipf's law says that a few words are very frequent, and most words are very rare. This makes NLP difficult because we will always come across rare/unseen words.

Explain why we often use statistical models in NLP.

To handle ambiguity (and make NLP systems more robust/to deal with the coverage problem).

Give two examples of ambiguity and explain why we have to resolve them.

POS ambiguity: back = noun or verb? Need to resolve this to understand the structure of sentences.

Word sense ambiguity: bank = river bank or institution. Need to resolve this to understand the meaning of sentences.

Back to HMMs: 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"*

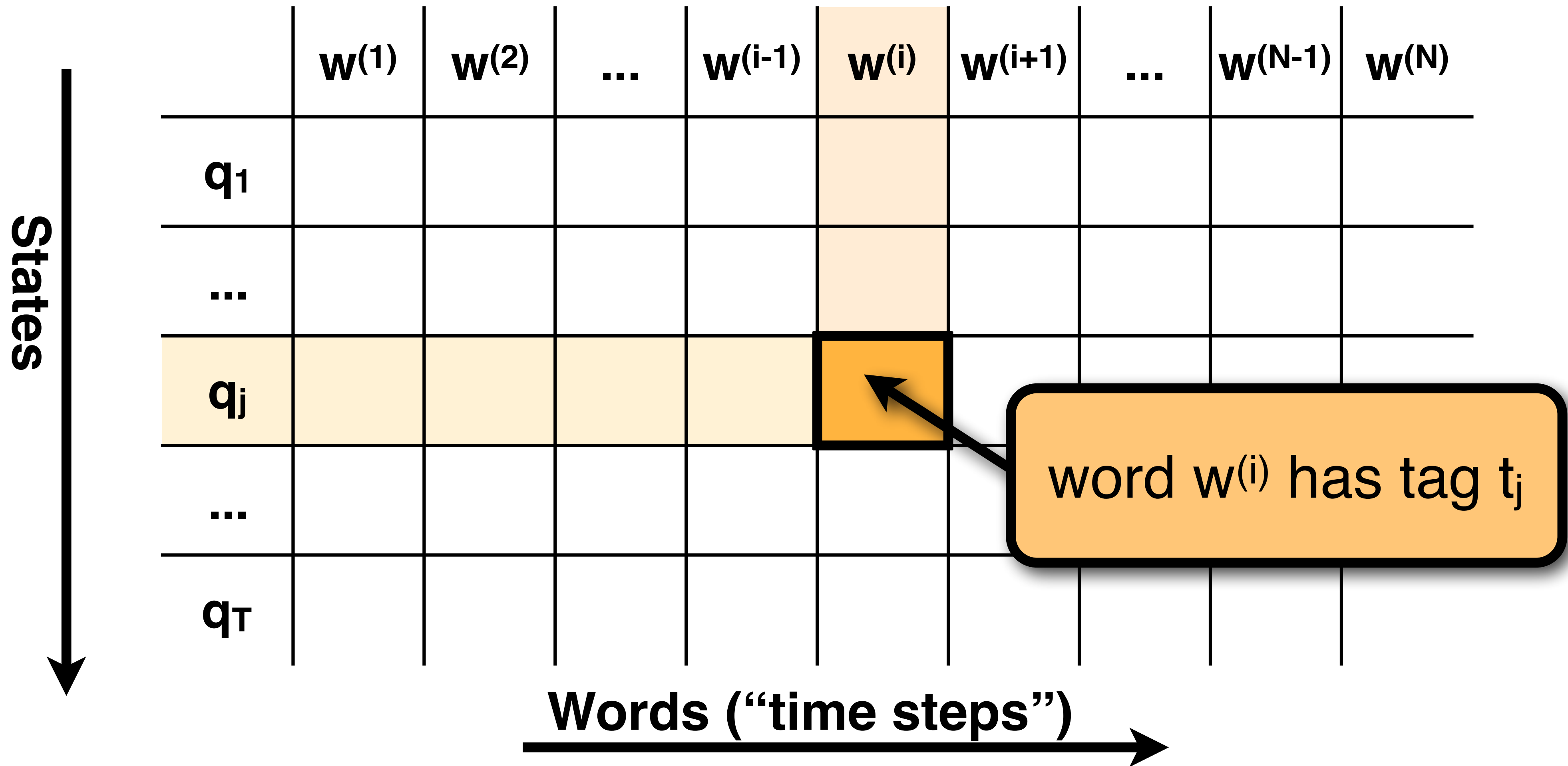
We want to use an HMM tagger to find its POS tags **t**

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

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

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

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.

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

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

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

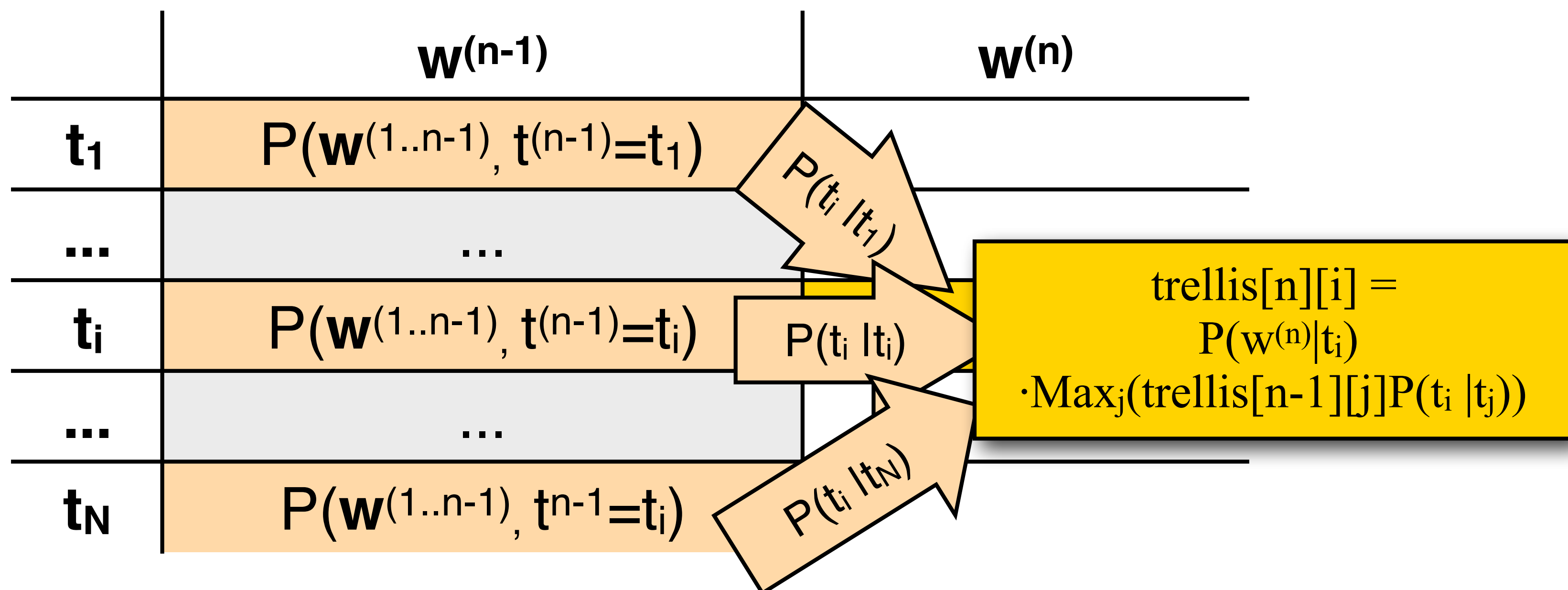
In the first column, initialize each cell $\text{trellis}[1][k]$ as

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

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

At any internal cell

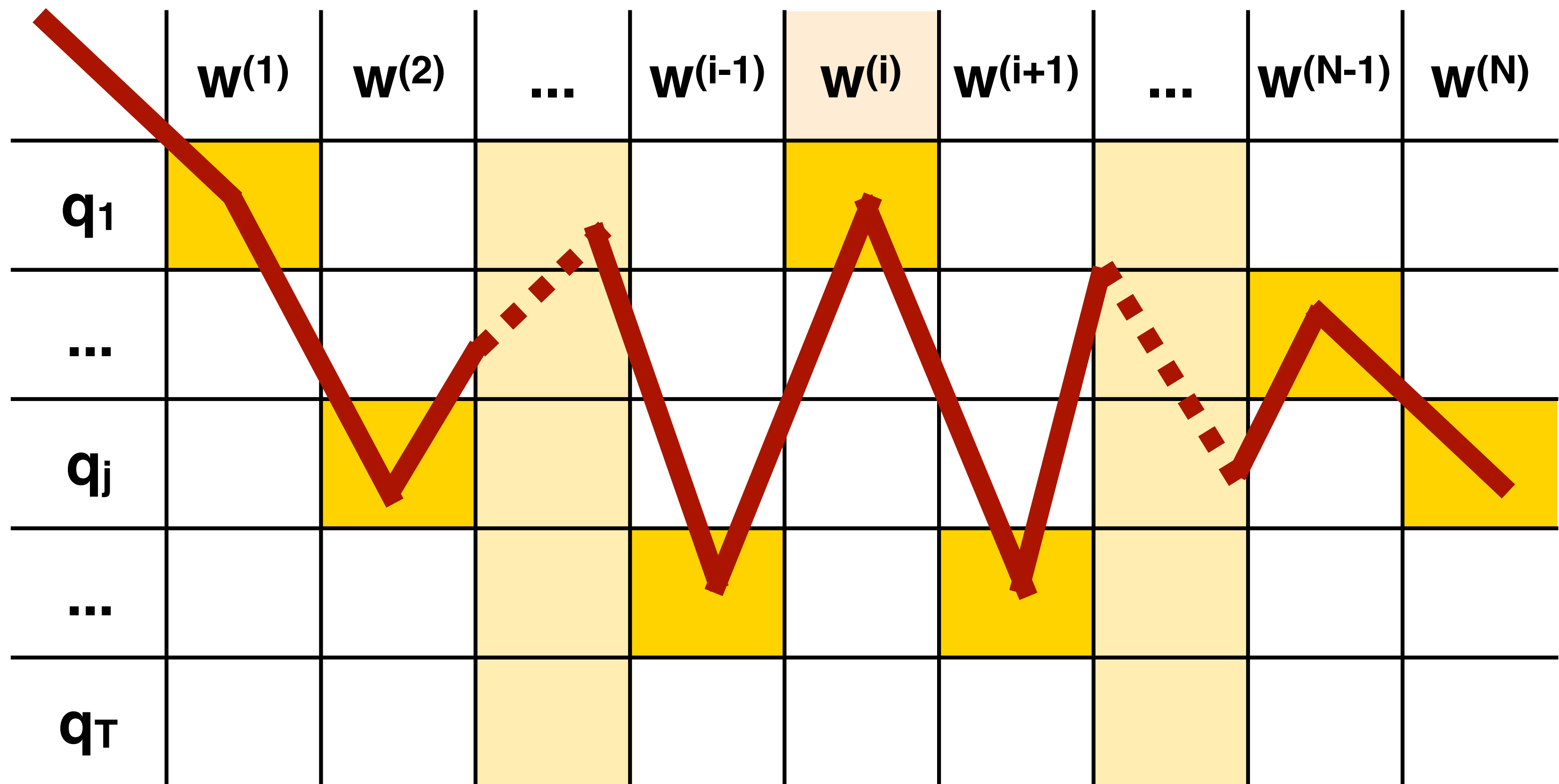
- For each cell in the preceding column: multiply its entry with the transition probability to the current cell.
- Keep a single backpointer to the best (highest scoring) cell in the preceding column
- Multiply this score with the emission probability of the current word



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



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.

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 the trellis.

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;
}
```

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:

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

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

Other HMM algorithms

The Forward algorithm:

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

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

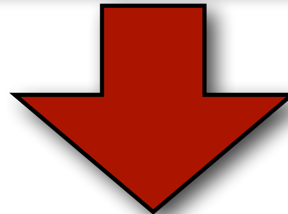
The Forward-Backward algorithm:

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 .
```

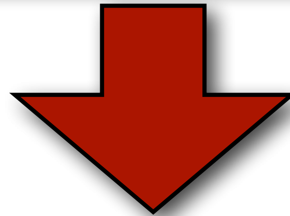


```
Pierre_NNP Vinken_NNP ,_, 61_CD years_NNS old_JJ ,_,
will_MD join_VB IBM_NNP ‘s_POS board_NN as_IN a_DT
nonexecutive_JJ director_NN Nov._NNP 29_CD ._.
```

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

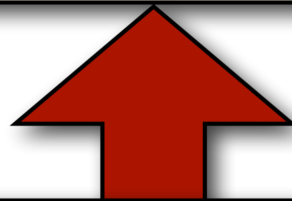
We define three new tags:

– **B-NP**: beginning of a noun phrase chunk

– **I-NP**: inside of a noun phrase chunk

– **O**: outside of a noun phrase chunk

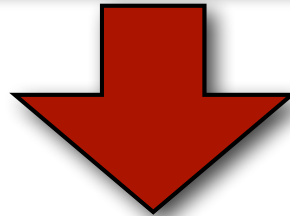
```
[NP Pierre Vinken] , [NP 61 years] old , will join
[NP IBM] ‘s [NP board] as [NP a nonexecutive director]
[NP Nov. 2] .
```



```
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

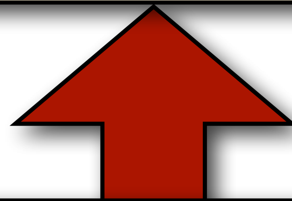
We define several new tags:

– **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

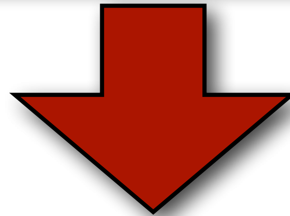
```
[NP Pierre Vinken] , [NP 61 years] old , [VP will join]
[NP IBM] ‘s [NP board] [PP as] [NP a nonexecutive
director] [NP Nov. 2] .
```



```
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

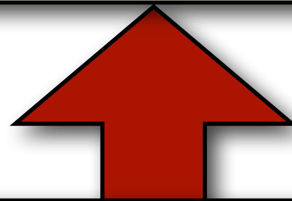
We define many new tags:

– **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 one random variable:

$P(X) =$ (X)

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

$P(Y)\,P(X \mid Y) =$ (Y) → (X)

$P(Y)\,P(Z)\,P(X \mid Y, Z) =$ (Y) → (X) ← (Z)

Shaded nodes represent observed variables.

White nodes represent hidden variables

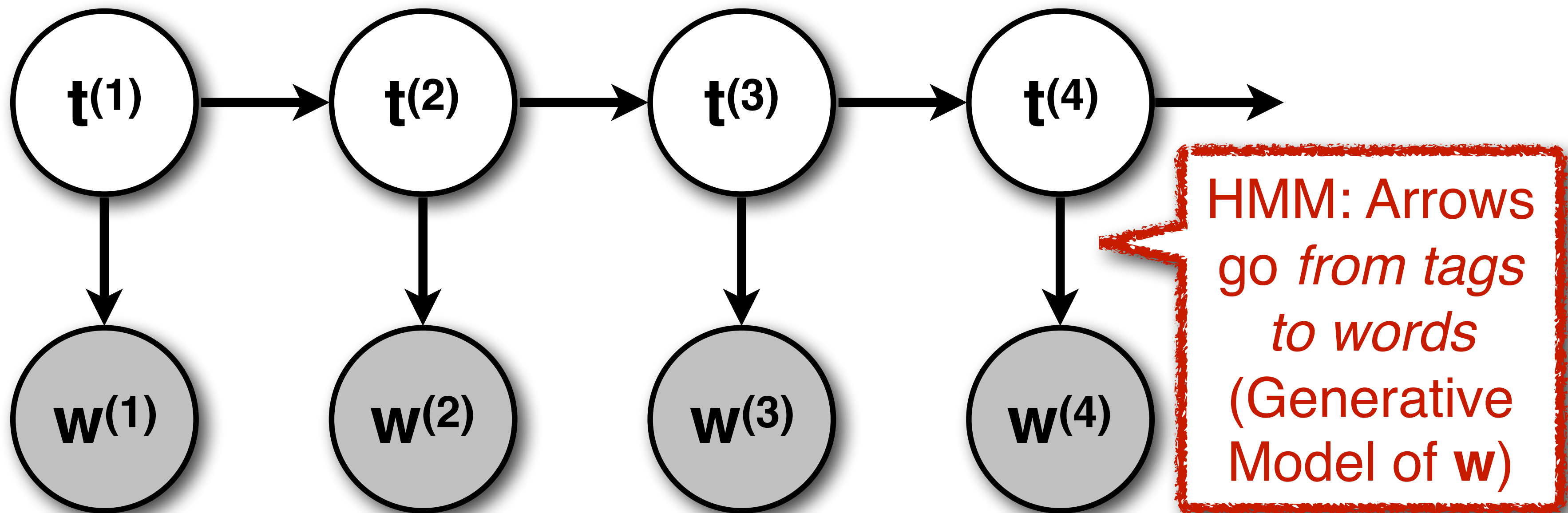
$P(Y)\,P(X \mid Y)$ with Y hidden and X observed = (Y) → (X, shaded)

HMMs as graphical models

HMMs are generative models of the observed 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 for tagging,
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

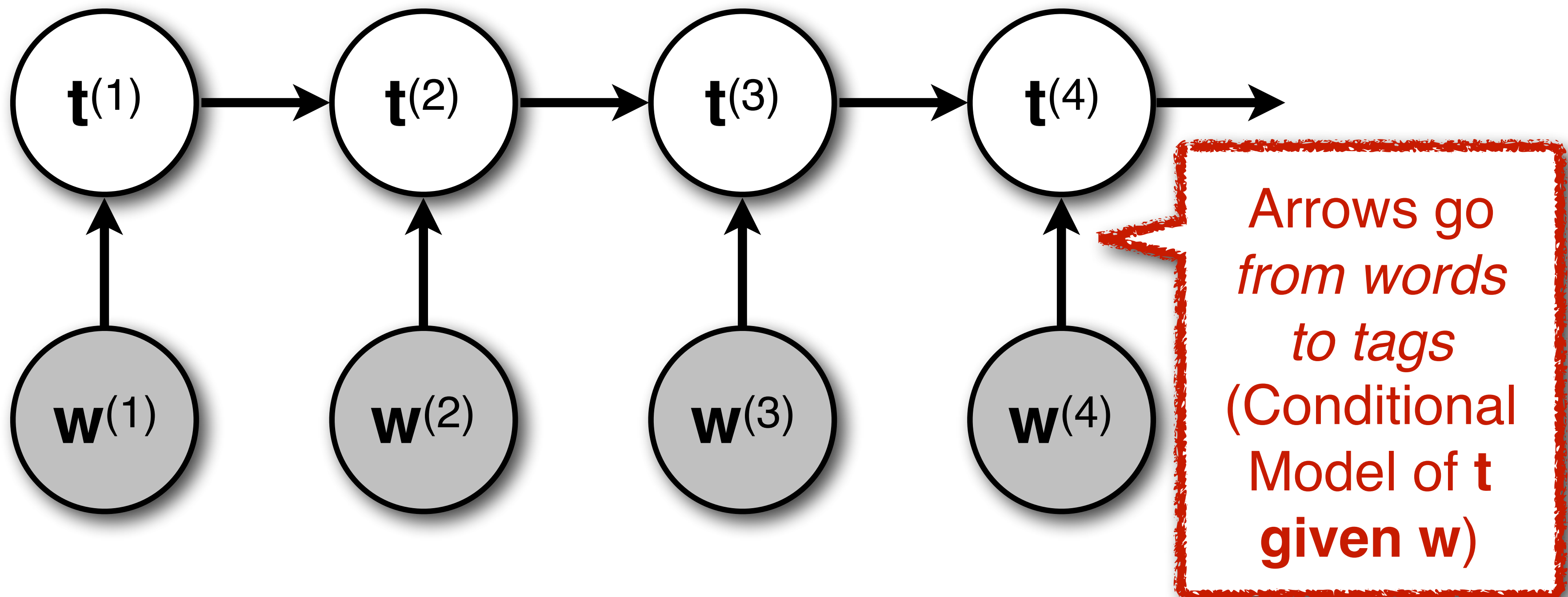
– But features may not be independent

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 $c^* = \text{argmax}_i \, P(c_i \mid \mathbf{x})$ 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).

Recall: 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 $\mathbf{x}$: $\text{score}(\mathbf{x},c) = \sum_i w_{ic} f_i(\mathbf{x})$

This score could be negative, so we exponentiate it:

$\text{score}(\mathbf{x},c) = \exp(\sum_i w_{ic} f_i(\mathbf{x}))$

To get a probability distribution over all classes c, we renormalize these scores:

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

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

Learning: finding *w*

Learning = finding weights ***w***
We use conditional maximum likelihood estimation (and standard convex optimization algorithms or gradient descent) 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 $\mathbf{x}_i$

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

NB: Terminology

Models that are of the form

$$P(c \mid \mathbf{x}) = \text{score}(\mathbf{x},c) / \sum_j \text{score}(\mathbf{x},c_j)$$
$$= \exp(\sum_i w_{ic} f_i(\mathbf{x})) / \sum_j \exp(\sum_i w_{ij} f_i(\mathbf{x}))$$

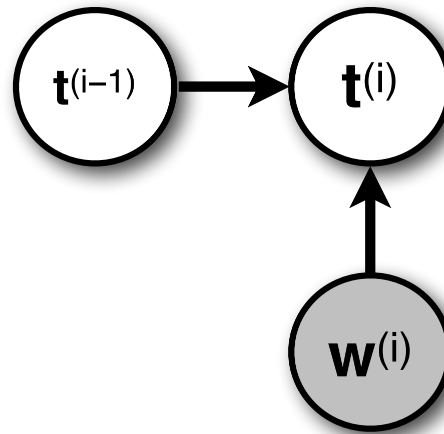
are also called loglinear models, Maximum Entropy (MaxEnt) models, or multinomial logistic regression models.

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:

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

Supplementary: 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:

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

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

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