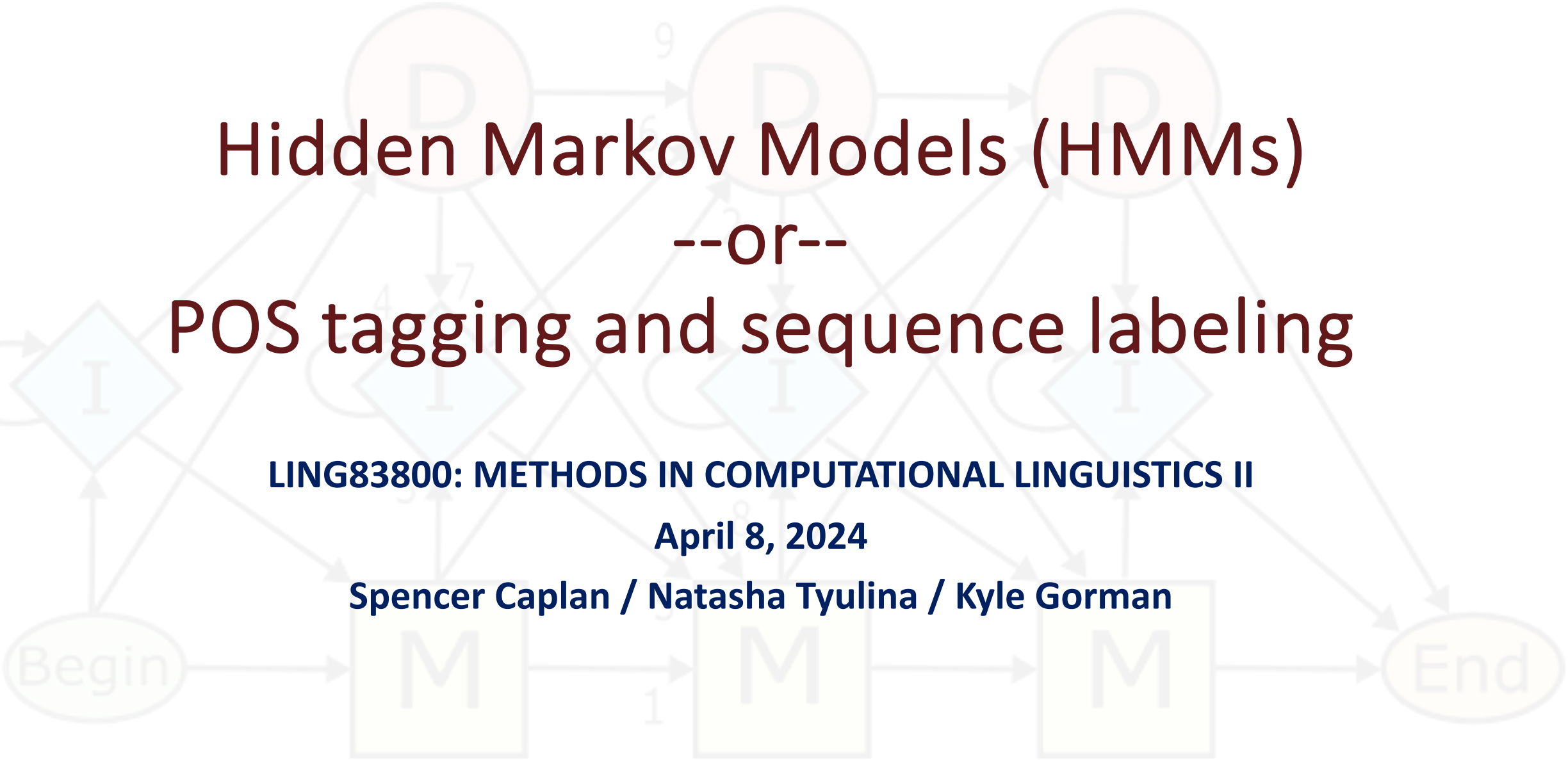
Hidden Markov Models (HMMs)
--or--
POS tagging and sequence labeling



LING83800: METHODS IN COMPUTATIONAL LINGUISTICS II

April 8, 2024

Spencer Caplan / Natasha Tyulina / Kyle Gorman

Today

1. Word classes and part-of-speech (POS) tagging
2. Tagset design and tradeoffs
3. Tagging methods: Back-of-the-envelope math for what's possible
4. Sequence labeling tasks
   - Hidden-Markov Models (HMMs)
5. Visualizing HMMs
   - Bayes Net
   - Probabilistic Automaton
   - Trellis
6. The three classic HMM problems
   - Viterbi algorithm for decoding

Word classes

- 8 (ish) traditional parts of speech
  - Noun, verb, adjective, preposition, adverb, article, interjection, pronoun, conjunction, etc
  - This idea has been around for over 2000 years (Dionysius Thrax of Alexandria, c. 100 B.C.)

- Called: parts-of-speech, lexical category, word classes, morphological classes, lexical tags, POS
  - We'll use **POS** most frequently

For engineering purposes, simply words that behave "alike"

- Appear in similar contexts
- Perform similar functions in sentences
- Undergo similar transformations

POS Examples

• N	noun	chair, bandwidth, pacing
• V	verb	study, debate, munch
• ADJ	adjective	purple, tall, ridiculous
• ADV	adverb	unfortunately, slowly,
• P	preposition	of, by, to
• PRO	pronoun	I, me, mine
• DET	determiner	the, a, that, those

Open vs. Closed classes

- Open:
    - Nouns, Verbs, Adjectives, Adverbs.
    - Why “open”?
- Closed:
    - determiners: a, an, the
    - pronouns: she, he, I
    - prepositions: on, under, over, near, by, …

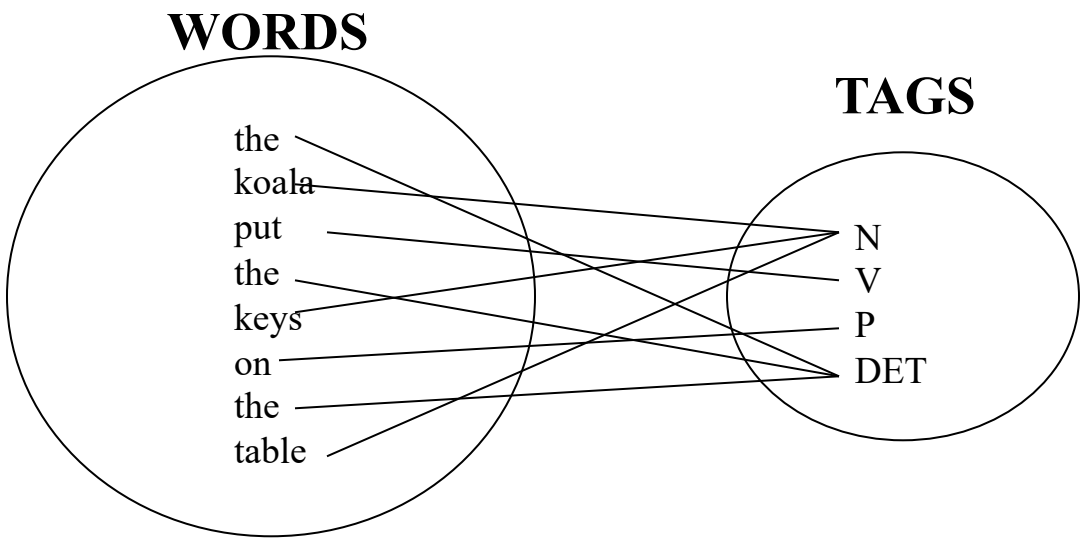
Determining part of speech tags

- *The Problem:*

Word	POS listing in Brown Corpus		
heat	noun	***verb***	
oil	***noun***		
in	***prep***	noun	adv
a	***det***	noun	noun-proper
large	***adj***	noun	adv
pot	***noun***		

POS Tagging: Definition

The process of assigning a part-of-speech or lexical class marker to each word in a sentence, corpus, etc.:



Tagging is a disambiguation task

Words often have more than one POS: *back*

- The *back* door = JJ
- On my *back* = NN
- Win the voters *back* = RB
- Promised to *back* the bill = VB

Why such a large gap between type- and token rate of ambiguity?

		WSJ	Brown
Types:			
Unambiguous	(1 tag)	44,432 (**86%**)	45,799 (**85%**)
Ambiguous	(2+ tags)	7,025 (**14%**)	8,050 (**15%**)
Tokens:			
Unambiguous	(1 tag)	577,421 (**45%**)	384,349 (**33%**)
Ambiguous	(2+ tags)	711,780 (**55%**)	786,646 (**67%**)

What POS tagging good for?

- Parsing:
  - gives us the terminal nodes
- Information retrieval:
  - knowing a word is a N tells you it gets plurals (query: “aardvarks” actually get (aardvark|aardvarks)

Speech synthesis:

How to pronounce “lead”?

INsult	inSULT
OBject	obJECT
OVERflow	overFLOW
DIScount	disCOUNT
CONtent	conTENT

POS Tagging

Dr Mitch never got around to joining

All we gotta do is go around the corner

Chateau Petrus costs around 250

POS Tagging

Dr/NNP Mitch/NNP never/RB got/VBD around/RP to/TO joining/VBG

All/DT we/PRP gotta/VBN do/VB is/VBZ go/VB around/IN the/DT corner/NN

Chateau/NNP Petrus/NNP costs/VBZ around/RB 250/CD

Roadmap

1. Word classes and part-of-speech (POS) tagging
2. **Tagset design and tradeoffs**
3. Tagging methods: Back-of-the-envelope math for what's possible
4. Sequence labeling tasks
   - Hidden-Markov Models (HMMs)
5. Visualizing HMMs
   - Bayes Net
   - Probabilistic Automaton
   - Trellis
6. The three classic HMM problems
   - Viterbi algorithm for decoding

Part-of-speech tagging

Part-of-speech tagging assigns part-of-speech labels to each token in a sentence.

Helps to resolve local ambiguities, identify the who/what/why, and is a preparation step for later linguistic analyses (including parsing).

Tagsets are language- (and corpus-) dependent:

- Brown corpus (American English): 85 tags
- Penn Treebank (PTB; American English): 36 tags
- Sinica (Chinese): 294 tags
- Szeged (Hungarian): 744 tags
- “Universal” tagset: 12 tags (plus mappings from other tagsets)

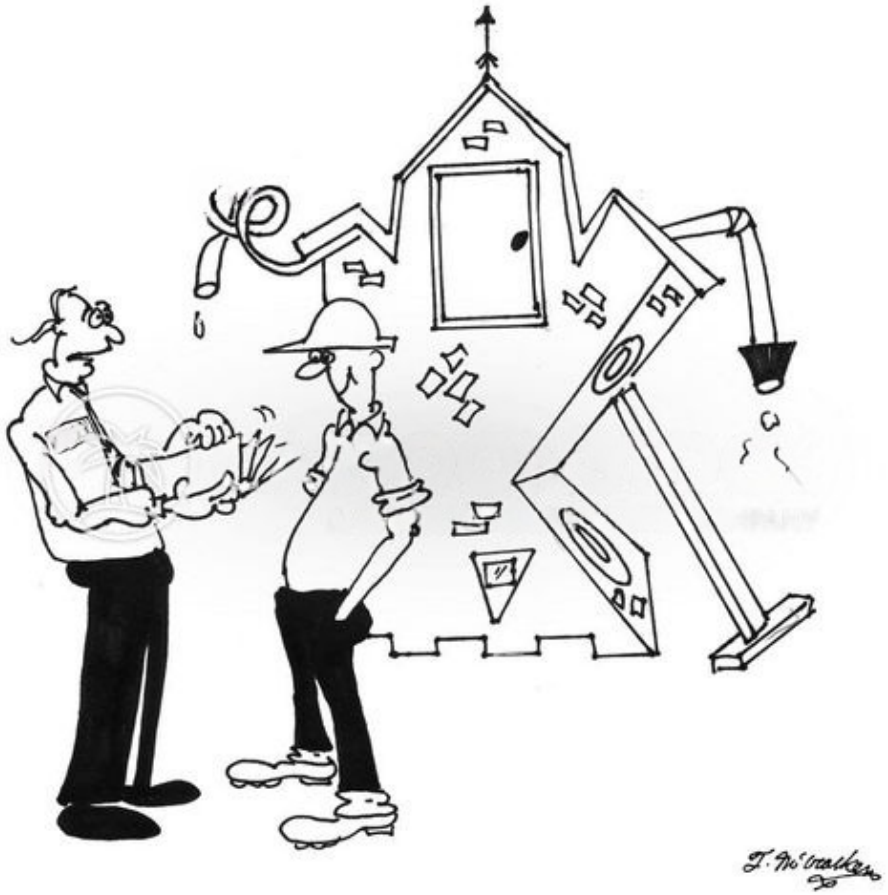
Brief History

- Brown Corpus (Kucera & Francis, 1967): 1m tagged words from various genres, including much of Robert Heinlein's 1964 science fiction novel *Stranger In A Strange Land*, initially distributed as a book
- Penn Treebank (Marcus, Santorini, & Marcinkiewicz, 1993): 1m tagged and parsed words of the 1989 Wall St. Journal, distributed on CD-ROM (later releases added dyadic phone conversations)
- OntoNotes (Weischedel et al., 2011): most of the Wall St. Journal data, corrected by a senior syntax professor
- The Universal Dependencies project* (Nivre et al. 2015): free tagged and parsed data for roughly 100 treebanks in 60 languages

*http://universaldependencies.org/

How did this happen?

How is there such a large gap in tagset size? E.g.



Szeged (Hungarian): 744 tags
Vs
“Universal Dependencies” set: 12 tags

Rationale behind “British & European” tagsets

To provide “distinct codings for all classes of words having distinct grammatical behaviour” – Garside et al. 1987

- The Lund tagset for adverb distinguishes between
  - Adjunct – Process, Space, Time
  - Wh-type – Manner, Reason, Space, Time, Wh-type + ‘S
  - Conjunct – Appositional, Contrastive, Inferential, Listing, …
  - Disjunct – Content, Style
  - Postmodifier – “else”
  - Negative – “not”
  - Discourse Item – Appositional, Expletive, Greeting, Hesitator, …

Motivations for keeping a smaller tag set

- Many tags are unique to particular lexical items, and can be recovered automatically if desired.

Brown Tags For Verbs		
be/BE	have/HV	sing/VB
is/BEZ	has/HVZ	sing/VBZ
was/BED	had/HVD	sang/VBD
being/BEG	having/HVG	singing/VBG
been/BEN	had/HVN	sung/VBN

Penn Treebank Tags For Verbs		
be/VB	have/VB	sing/VB
is/VBZ	has/VBZ	sing/VBZ
was/VBD	had/VBD	sang/VBD
being/VBG	having/VBG	singing/VBG
been/VBN	had/VBN	sung/VBN

Note: tag sets are not linguistic theories per se, although of course they encode linguistic information

This is a practical decision rather than a scientific one

The universal tagset: open-class tags

- `ADJ`: adjective
- `ADV`: adverb
- `NOUN`: noun
- `VERB`: verb

The universal tagset: closed-class tags

- `ADP`: adposition
- `CONJ`: conjunction
- `DET`: determiner
- `NUM`: numeral
- `PRT`: particle
- `PRON`: pronoun

The universal tagset: miscellaneous tags

- `PUNCT`: punctuation
- `X`: other

Universal tagset (automatic tagging)

Rolls-Royce	NOUN	to	PRT
Motor	NOUN	remain	VERB
Cars	NOUN	steady	ADJ
Inc.	NOUN	at	ADP
said	VERB	about	ADP
it	PRON	1,200	NUM
expects	VERB	cars	NOUN
its	PRON	in	ADP
U.S.	NOUN	1990	NUM
sales	NOUN	.	PUNCT

Linguistic critique (-KBG/SPC)

The proper noun/common noun distinction is really useful, and present in most tagsets: it should have been preserved.

Penn Treebank: open-class tags (1/2)

- JJ (ADJ): adjective
- JJR (ADJ): comparative adjective
- JJS (ADJ): superlative adjective
- NN (NOUN): singular common or mass noun
- NNS (NOUN): plural common or mass noun
- NNP (NOUN): singular proper noun
- NNPS (NOUN): plural proper noun
- RB (ADV): adverb
- RBR (ADV): comparative adverb
- RBS (ADV): superlative adverb

Penn Treebank: open-class tags (2/2)

- VB (VERB): verb (base form)
- VBD (VERB): simple past tense verb
- VBG (VERB): gerund or present participle
- VBN (VERB): past participle
- VBP (VERB): non 3rd-person singular present
- VBZ (VERB): 3rd-person singular present
- MD (VERB): modal

Penn Treebank: closed-class tags

- `CC` (`CONJ`): coordinating conjunction
- `CD` (`NUM`): numeral
- `DT` (`DET`): determiner
- `EX` (`DET`): existential *there*
- `IN` (`ADP`): preposition (and subordinating conjunctions)
- `PDT` (`DET`): "predeterminer" (e.g., *both* in *both the boys*)
- `PRP` (`PRON`): pronoun
- `PRP$` (`PRON`): possessive pronoun
- `WDT` (`DET`): *wh*-determiner
- …

Penn Treebank: miscellaneous tags (1/2)

- # (PUNCT): #
- $ (PUNCT): $
- `` (PUNCT): left quotation mark
- '' (PUNCT): right quotation mark
- : (PUNCT): colon and semicolon
- -LRB- (PUNCT): left bracket
- -RRB- (PUNCT): right bracket
- , (PUNCT): comma
- . (PUNCT): sentential punctuation

Penn Treebank: miscellaneous tags (2/2)

- `FW` (`X`): foreign word
- `LS` (`X`): list item marker
- `SYM` (`X`): symbol
- `UH` (`X`): interjection

Linguistic critiques (-KBG/SPC)

- There are way too many punctuation tags.
- Existential *there* doesn't need its own tag (`EX`); cf. existential *it* (`PRP`).
- The `TO` tag is massively polysemous (e.g., infinitive marker, preposition).
- Yet, there are many distinctions which could be trivially recovered from the tokens or the parse:
  - `IN`: subordinating conjunction (heading a clause) vs. preposition (heading a prepositional phrase)
  - `UH`: interjections (*yes!*) vs. filled pause (*uh*, *um*)
  - `DT`: articles (*a*, *an*, *the*) vs. demonstratives (*these*, *those*)
  - `PRP`: actual personal pronouns (*I*, *her*) vs. reflexive pronouns (*myself*)

PTB tagset (automatic tagging)

Rolls-Royce	NNP	to	TO
Motor	NNP	remain	VB
Cars	NNPS	steady	JJ
Inc.	NNP	at	IN
said	VBD	about	IN
it	PRP	1,200	CD
expects	VBZ	cars	NNS
its	PRP$	in	IN
U.S.	NNP	1990	CD
sales	NNS	.	.

Roadmap

1. Word classes and part-of-speech (POS) tagging
2. Tagset design and tradeoffs
3. **Tagging methods: Back-of-the-envelope math for what's possible**
4. Sequence labeling tasks
   - Hidden-Markov Models (HMMs)
5. Visualizing HMMs
   - Bayes Net
   - Probabilistic Automaton
   - Trellis
6. The three classic HMM problems
   - Viterbi algorithm for decoding

Classical idea

- The Problem:

Word	POS listing in Brown		
heat	noun	***verb***	
oil	***noun***		
in	***prep***	noun	adv
a	***det***	noun	noun-proper
large	***adj***	noun	adv
pot	***noun***		

Classical idea

- The Old Solution: *Depth First search.*
  - If each of *n* words has *k* tags on average, try the n^k combinations until one works.
  - (Can define "works" in a number of different ways)

- Machine Learning (Statistical NLP) Solutions: *Automatically learn* Part of Speech (POS) assignment.
  - The best techniques achieve >97% accuracy per word on new materials, given large training corpora.

Simple statistical approach: Idea 1

Simply assign each word its most likely POS.

Success rate: 91%!

Word	POS listings in Brown		
heat	noun/89	**verb/5**	
oil	**noun/87**		
in	**prep/20731**	noun/1	adv/462
a	**det/22943**	noun/50	noun-proper/30
large	**adj/354**	noun/2	adv/5
pot	**noun/27**		

Let that sink in

- How many words in the unseen test data can be tagged correctly? **(Tag Accuracy)**

- Baseline is already 91%
  - (Baseline: performance on the “stupidest possible method”)
    - Tag each word with it’s most frequent tag
    - Tag all unknown words as nouns
  - Partly easy because:
    - Many words are unambiguous
    - You get lots of points for the frequent easy cases (the, a, etc.) and for punctuation

Simple statistical approach: Idea 2

For a string of words

$$W = w_1 w_2 w_3 \ldots w_n$$

find the string of POS tags

$$T = t_1 t_2 t_3 \ldots t_n$$

which maximizes $P(T \mid W)$

- i.e., the most likely POS tag t_i for each word w_i given its surrounding context

The sparse data problem…

A **Simple, but Impossible** Approach to Compute $P(T \mid W)$:

Count up instances of the string:

e.g. *"heat oil in a large pot"*

in the training corpus, and pick the *most common tag assignment* to the string

A BOTEC Estimate of What Works

What parameters can we estimate with a million words of hand tagged training data?

- Assume a uniform distribution of 5000 words and 40 part of speech tags..

Event	Count	Estimate Quality?
tags	40	Excellent
tag bigrams	1600	Excellent
tag trigrams	64,000	OK
tag 4-grams	2.5M	Poor
words	5000	Very Good
word bigrams	25M	Poor
word x tag pairs	200,000	OK

We can get reasonable estimates of

- *Tag* bigrams
- *Word x tag pairs*

How to use the estimates we can get

We can get reasonable estimates of

- *Tag* bigrams
- *Word x tag pairs*

- Let's turn this intuition into a formal / implementable system

Roadmap

1. Word classes and part-of-speech (POS) tagging
2. Tagset design and tradeoffs
3. Tagging methods: Back-of-the-envelope math for what's possible
4. **Sequence labeling tasks**
   - **Hidden-Markov Models (HMMs)**
5. Visualizing HMMs
   - Bayes Net
   - Probabilistic Automaton
   - Trellis
6. The three classic HMM problems
   - Viterbi algorithm for decoding

Sequence labeling

- A sequence-labeling problem has a sequence of length n as input
  - $\mathrm{X} = (x_1, \ldots, x_n)$
- Output is another sequence also of length n
  - $Y = (y_1, \ldots, y_n)$
- Each $y_i \in Y$ is the ”label” of x_i

General framework for formalizing many language-processing tasks!

Sequence labeling applications

Noun-phrase chunking: Each x_i is a word in the sentence and its corresponding y_i indicates whether x_i is in the beginning, middle or end of a noun phrase (NP) chunk.

$\boldsymbol{y}$:	[NP	NP	NP]	_	[NP]	.
$\boldsymbol{x}$:	the	big	cat	bit	Sam	.

In this task, '*[NP*' labels the beginning of a noun phrase — the notation is intended to conveys the intuitive idea that the labeled word is the start of an NP. Similarly, '*[NP]*' labels a word that is both the start and end of a noun phrase.

Sequence labeling applications

Named entity detection: The elements of $\boldsymbol{x}$ are the words of a sentence, and $\boldsymbol{y}$ indicates whether they are in the beginning, middle or end of a noun phrase (NP) chunk that is the name of a person, company or location.

$\boldsymbol{y}$:	[CO	CO]	_	[LOC]	_	[PER]	_
$\boldsymbol{x}$:	XYZ	Corp.	of	Boston	announced	Spade's	resignation

Sequence labeling applications

Speech recognition: The elements of $\boldsymbol{x}$ are 100 msec. time slices of acoustic input, and those of $\boldsymbol{y}$ are the corresponding phonemes (i.e., y_i is the phoneme being uttered in time slice x_i). A *phoneme* is (roughly) the smallest unit of sound that makes up words.

Sequence labeling for POS

Part-of-speech tagging (abbreviated POS tagging): Each x_i in $\boldsymbol{x}$ is a word of the sentence, and each y_i in $\boldsymbol{y}$ is a part of speech (e.g., '*NN*' is common noun, '*JJ*' is adjective. etc.).

$\boldsymbol{y}$:	DT	JJ	NN	VBD	NNP	.
$\boldsymbol{x}$:	the	big	cat	bit	Sam	.

Hidden markov models (HMMs) are a great technique for accomplishing such sequence labeling tasks

(Ordinary) markov models

- A markov model (e.g. a bigram model) generates a string:
  - $X = (x_1, \ldots, x_n)$
  - $x_0 = \triangleright$ and $x_{n+1} = \triangleleft$

$$P(X) = \prod_{i=1}^{n+1} P(x_i | x_{i-1})$$

$$= \prod_{i=1}^{n+1} \Theta_{x_{i-1}, x_i}$$

Hidden Markov Model (HMM)

- In a *hidden* Markov model (HMM) we observe a string X, but in general its label sequence Y is "hidden" (not observed)

- Just as in an ordinary markov model we imagine that the label sequence Y is padded with START and STOP tokens
  - $y_0 = \triangleright$ and $y_{n+1} = \triangleleft$

- An HMM is a generative model that jointly generates both the label sequence Y and the observation sequence X

Specifically, the label sequence Y is generated by a Markov model.

Then the observations X are generated from the Y.

Hidden Markov Model (HMM)

$$P(Y) = \prod_{i=1}^{n+1} P(y_i|y_{i-1})$$

$$P(X|Y) = \prod_{i=1}^{n+1} P(x_i|y_i)$$

$$= \prod_{i=1}^{n+1} \sigma_{y_{i-1},y_i}$$

$$= \prod_{i=1}^{n+1} \tau_{y_i,x_i}$$

$\sigma_{y,y'}$ (Sigma) is a parameter estimating the probability that label y is followed by label y'

$\tau_{y,x}$ (Tau) is a parameter estimating the probability that label y generated output X

Check in: does it make sense

1. Our bigram language model is a markov model
2. In *hidden* Markov models (HMMs) we observe a string X, but in general its label sequence Y is “hidden” (not observed)
3. For us, the observed string is the output words and the label sequence is the part-of-speech tags

Hidden Markov Model (HMM)

$$P(Y) = \prod_{i=1}^{n+1} P(y_i|y_{i-1})$$

$$P(X|Y) = \prod_{i=1}^{n+1} P(x_i|y_i)$$

$$= \prod_{i=1}^{n+1} \sigma_{y_{i-1},y_i}$$

$$= \prod_{i=1}^{n+1} \tau_{y_i,x_i}$$

$\sigma_{y,y'}$ (Sigma) is a parameter estimating the probability that label y is followed by label y'

Think of σ (Sigma) as a "state-to-state" transition

$\tau_{y,x}$ (Tau) is a parameter estimating the probability that label y generated output X

Think of τ (Tau) as a "state-to-token" emission

Hidden Markov Model (HMM)

$\sigma_{y,y'}$ (Sigma) is a parameter estimating the probability that label y is followed by label y'

Think of σ (Sigma) as a "state-to-state" transition

Other sources use the term A for the "transition probability"

$$P(X,Y) = P(Y)P(X|Y)$$

$$= \prod_{i=1}^{n+1} \sigma_{y_{i-1},y_i} \tau_{y_i,x_i}$$

$\tau_{y,x}$ (Tau) is a parameter estimating the probability that label y generated output X

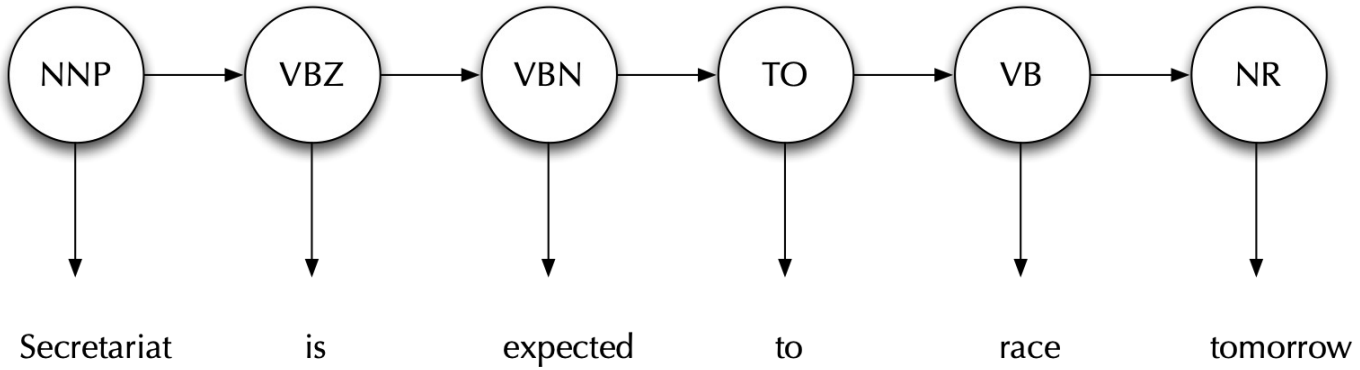
Think of τ (Tau) as a "state-to-tag" emission

Other sources use the term B for the "emission probability"

Generative story of HMMs: generate the next label y_i with probability $P(y_i|y_{i-1})$, and then generate the next member of the sequence x_i with probability $P(x_i|y_i)$

Tags can be thought of as *hidden* states...

Observed words can be thought of as emissions...



Roadmap

1. Word classes and part-of-speech (POS) tagging
2. Tagset design and tradeoffs
3. Tagging methods: Back-of-the-envelope math for what's possible
4. Sequence labeling tasks
   - Hidden-Markov Models (HMMs)
5. **Visualizing HMMs**
   - **Bayes Net**
   - **Probabilistic Automaton**
   - **Trellis**
6. The three classic HMM problems
   - Viterbi algorithm for decoding

Three ways to visualize HMMs

- Bayes-net
- Probabilistic automaton
- Trellis

Three ways to visualize HMMs: Bayes-Net

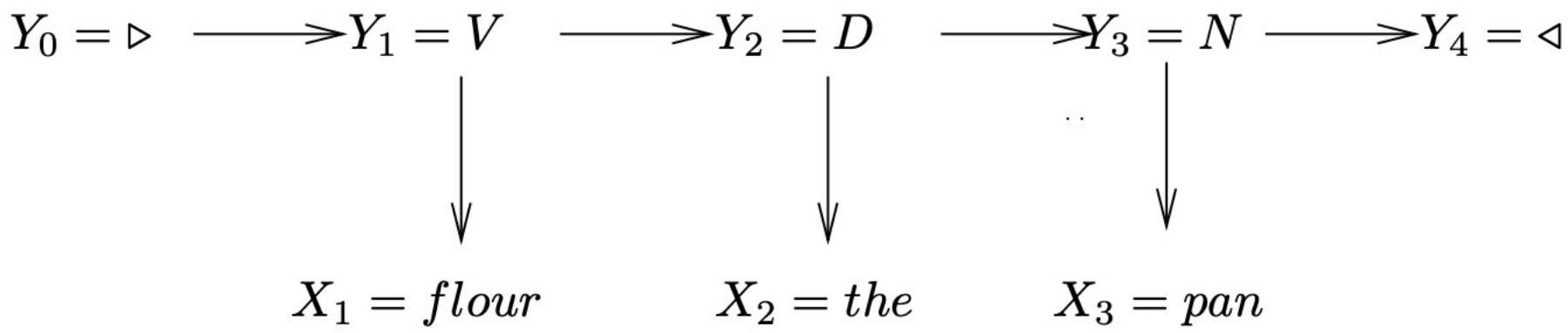


Figure 3.1: The Bayes-net representation of an HMM generating '*flour the pan*' from the labels '*V D N*'.

Three ways to visualize HMMs: Probabilistic Automata

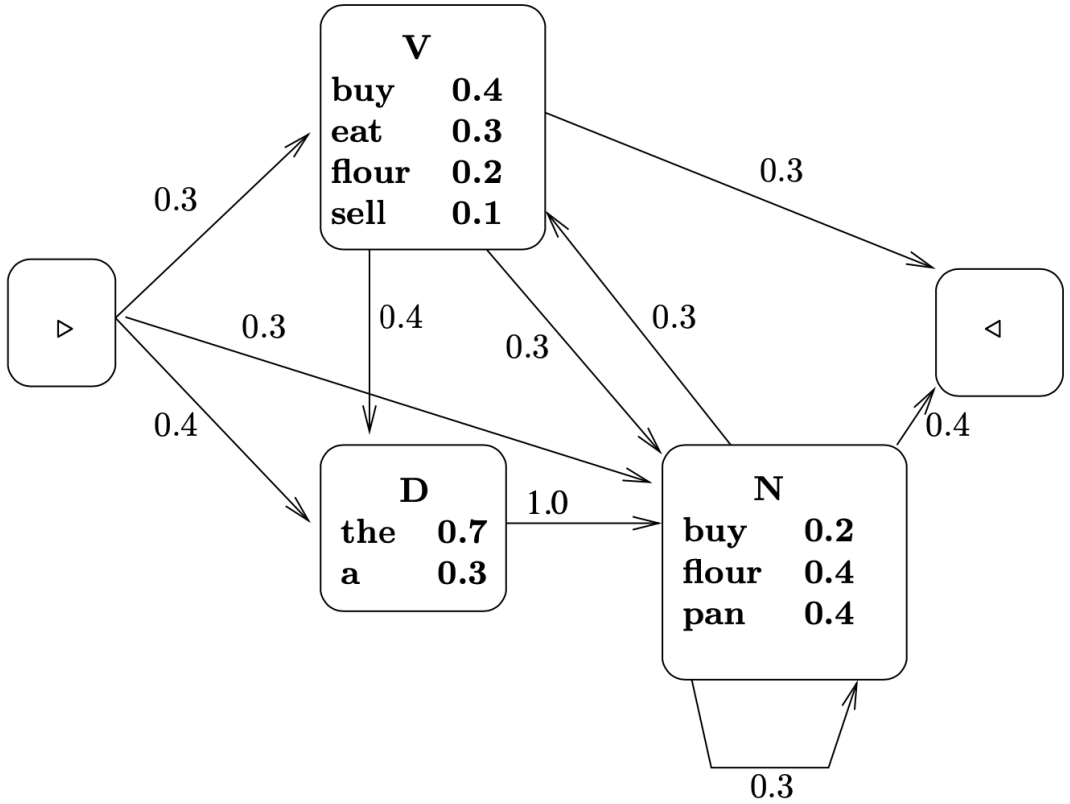


Figure 3.2: Example of HMM for POS tagging '*flour pan*', '*buy flour*'

Exercise

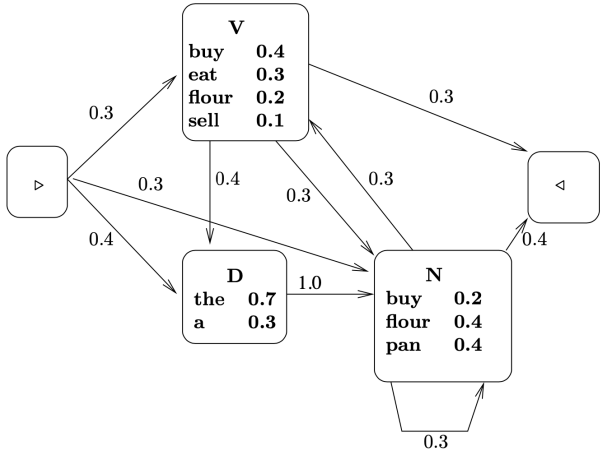


Figure 3.2: Example of HMM for POS tagging '*flour pan*', '*buy flour*'

Example 3.1: What is the probability of the sequence '*flour pan*' when the state sequence is $< \triangleright, V, N \triangleleft >$? (So '*flour pan*' is a command to coat the pan with flour.) That is, we want to compute

$$\mathrm{P}(< flour, pan >, < \triangleright, V, N, \triangleleft >)$$

Exercise

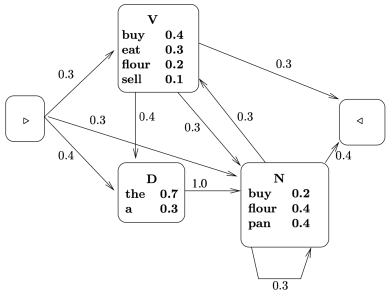


Figure 3.2: Example of HMM for POS tagging '*flour pan*', '*buy flour*'

Example 3.1: What is the probability of the sequence '*flour pan*' when the state sequence is $< \triangleright, V, N \triangleleft >$? (So '*flour pan*' is a command to coat the pan with flour.) That is, we want to compute

$$\mathrm{P}(< flour, pan >, < \triangleright, V, N, \triangleleft >)$$

Joint probability of the labeling and the string

$$\begin{aligned} \mathrm{P}(< flour, pan >, < V, N, \triangleleft >) &= \sigma_{\triangleright,V}\ \tau_{V,flour}\ \sigma_{V,N}\ \tau_{N,pan}\ \sigma_{N,\triangleleft} \\ &= 0.3 \cdot 0.2 \cdot 0.3 \cdot 0.4 \cdot 0.4. \end{aligned}$$

Three ways to visualize HMMs: Trellis

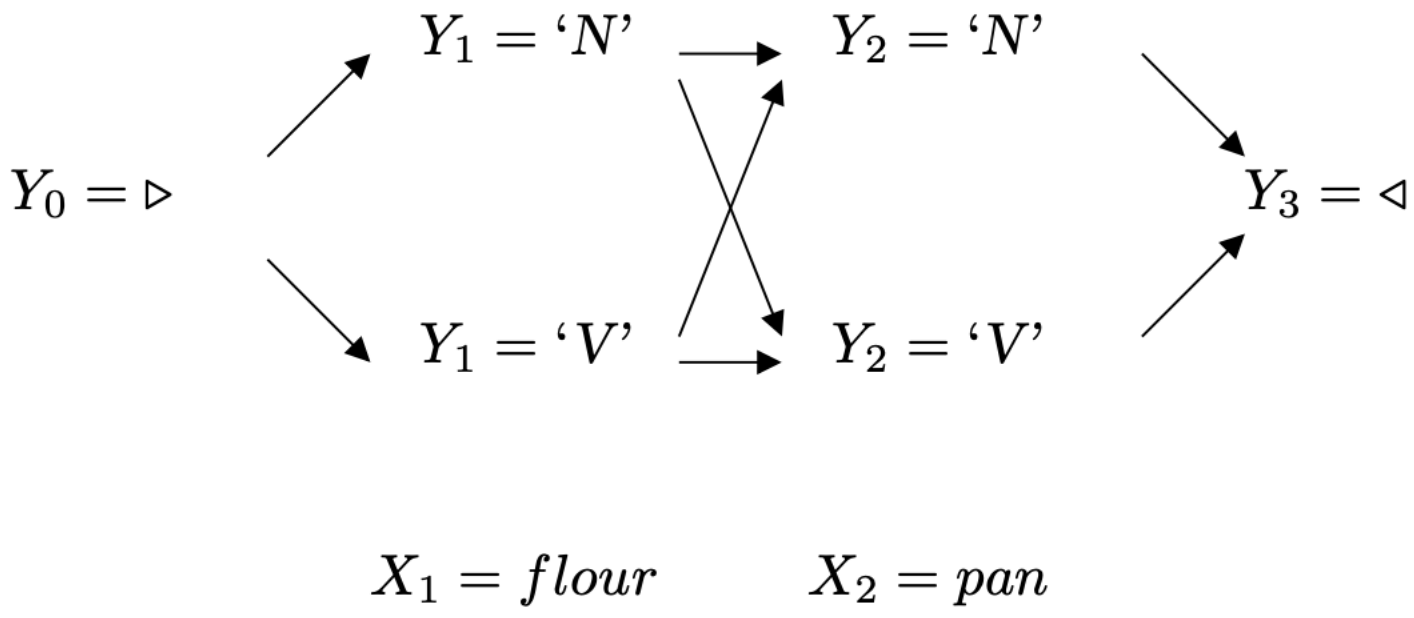
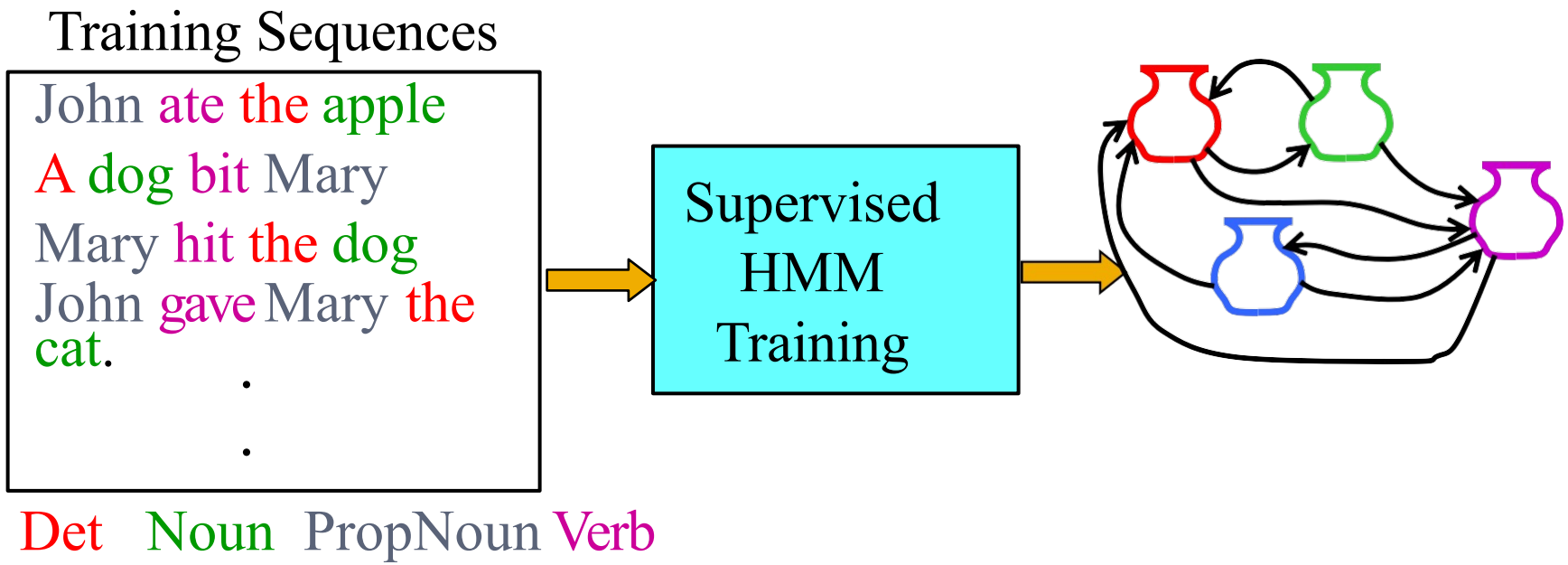


Figure 3.3: The trellis representation of an HMM generating '*flour pan*'

Supervised HMM Training

If training sequences are labeled (tagged) with the underlying state sequences that generated them, then the parameters, $\lambda=\{\sigma,\tau\}$ can all be estimated directly.



Likelihood and Prior

HMM Taggers choose tag sequence that maximizes this formula:

- P(word|tag) × P(tag|previous n tags)

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

How to get our estimates?

- To estimate the parameters of this model, given an annotated training corpus:

To estimate $\mathbf{P(t_i|t_{i-1})}$**:**

$$\frac{Count(t_{i-1}t_i)}{Count(t_{i-1})}$$

To estimate $\mathbf{P(w_i|t_i)}$**:**

$$\frac{Count(\ w_i \text{ tagged } t_i)}{Count(\text{ all words tagged } t_i)}$$

- Because many of these counts are small, *smoothing* is necessary for best results…

Two kinds of probabilities

- Tag transition probabilities $p(t_i | t_{i-1})$
  - Determiners likely to precede adjs and nouns
    - That/DT flight/NN
    - The/DT yellow/JJ hat/NN
    - So we expect P(NN|DT) and P(JJ|DT) to be high
    - But P(DT|JJ) to be pretty low
  - Compute P(NN|DT) by counting in a labeled corpus:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

$$P(NN|DT) = \frac{C(DT, NN)}{C(DT)} = \frac{56,509}{116,454} = .49$$

Two kinds of probabilities

Word likelihood "emission" probabilitiesp($w_i | t_i$)

VBZ (3sg Pres verb) likely to be "is"

Compute P(is|VBZ) by counting in a labeled corpus:

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

$$P(is|VBZ) = \frac{C(VBZ, is)}{C(VBZ)} = \frac{10,073}{21,627} = .47$$

Let's review that (HMMs)

$\sigma_{y,y'}$ (Sigma) is a parameter estimating the probability that label y is followed by label y'

$$P(X,Y) = P(Y)P(X|Y)$$

$\tau_{y,x}$ (Tau) is a parameter estimating the probability that label y generated output X

You'll also see the term A for the "transition probability"

$$= \prod_{i=1}^{n+1} \sigma_{y_{i-1},y_i} \tau_{y_i,x_i}$$

You'll also see the the term B for the "emission probability"

Generative story of HMMs: generate the next label y_i with probability $P(y_i|y_{i-1})$, and then generate the next member of the sequence x_i with probability $P(x_i|y_i)$

Practice "Quiz"

Suppose for some HMM when applied to the sequence $\boldsymbol{x}$, $\mathrm{P}(\boldsymbol{x} \mid \boldsymbol{y}) = 0$ for all $\boldsymbol{y} =< \cdots, Y_i = a, \cdots >$. That is, any sequence of states that goes through state a at position i assigns zero probability to the string $\boldsymbol{x}$. Does it follow that $\tau_{a,x_i} = 0$?

Parameters of an HMM

- *States*: A set of states $S=s_1, \ldots s_n$
- *Transition probabilities (Sigma)*: $A= a_{1,1}, a_{1,2}, \ldots, a_{n,n}$ Each $a_{i,j}$ represents the probability of transitioning from state s_i to s_j.
- *Emission probabilities (Tau):* a set B of functions of the form $b_i(o_t)$ which is the probability of observation o_t being emitted by s_i
- *Initial state distribution*: π_i is the probability that s_i is a start state

Roadmap

1. Word classes and part-of-speech (POS) tagging
2. Tagset design and tradeoffs
3. Tagging methods: Back-of-the-envelope math for what's possible
4. Sequence labeling tasks
   - Hidden-Markov Models (HMMs)
5. Visualizing HMMs
   - Bayes Net
   - Probabilistic Automaton
   - Trellis
6. **The three classic HMM problems**
   - **Viterbi algorithm for decoding**

The Three Basic HMM Problems

- *Problem 1 (Evaluation):* Given the observation sequence $O=o_1,\ldots,o_T$ and an HMM model $\lambda = (A, B, \pi)$, how do we compute the probability of O given the model?

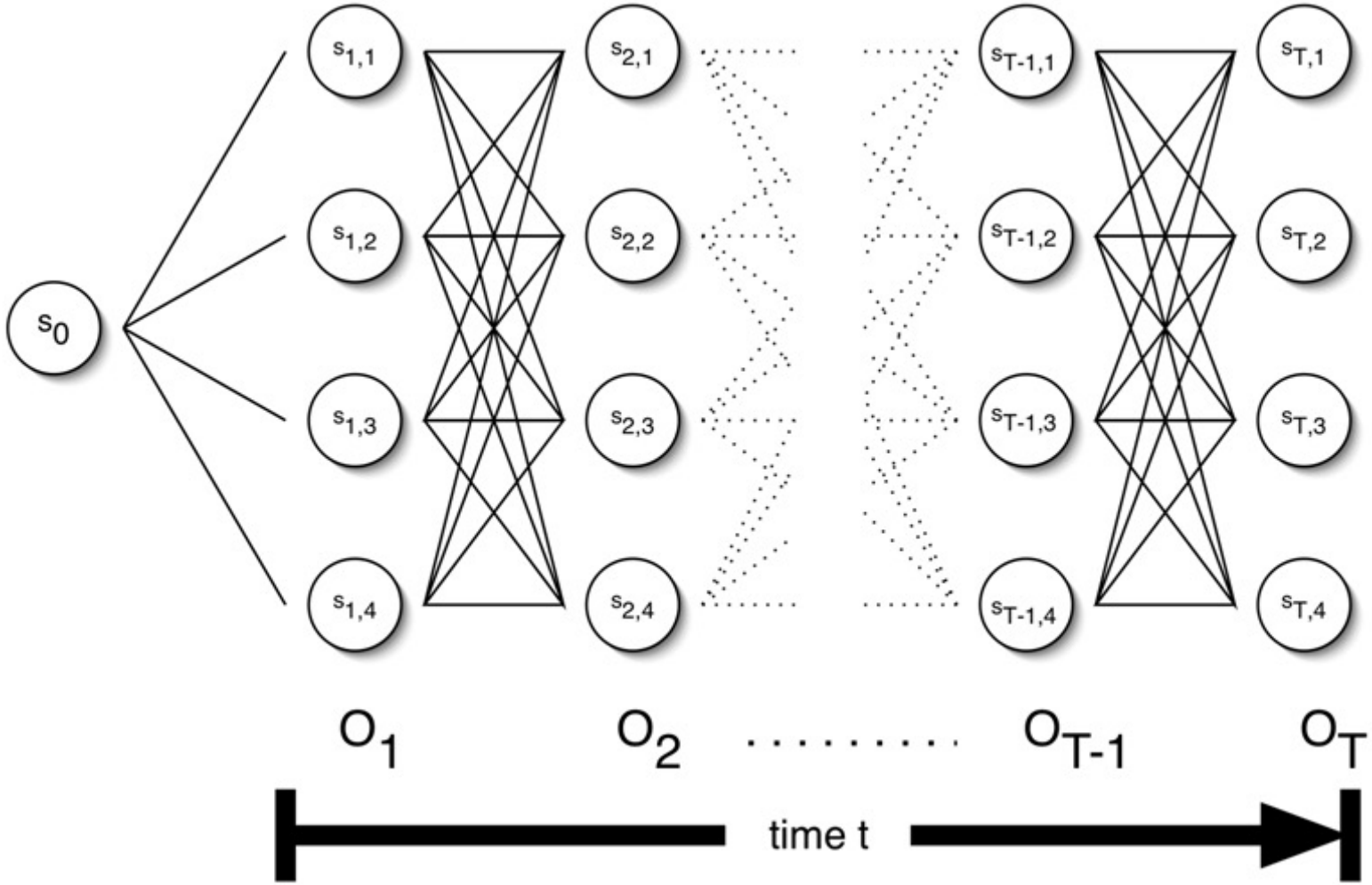
- *Problem 2 (Decoding):* Given the observation sequence O and an HMM model λ, how do we find the state sequence that best explains the observations?

- *Problem 3 (Learning):* How do we adjust the model parameters $\lambda = (A, B, \pi)$, to maximize $P(O|\lambda)$?

Problem 1: Probability of an Observation Sequence

- Q: What is $P(O|\lambda)$?
- A: the sum of the probabilities of all possible state sequences in the HMM.

Crucial Data Structure: the Trellis



Dynamic Programming Solution: Forward Algorithm

Simply sum the probabilities rather than picking the most likely one

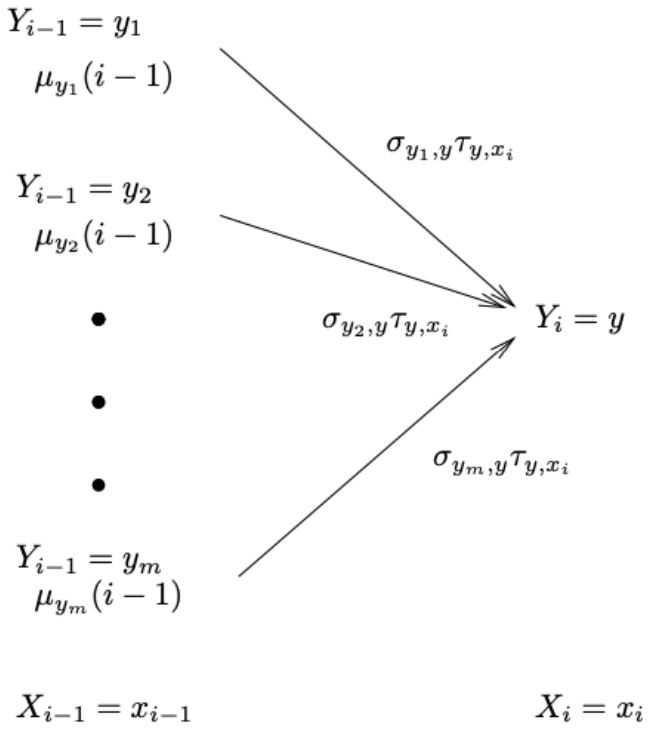


Figure 3.4: A piece of a trellis showing how to compute $\mu_y(i)$ if all of the $\mu_{y'}(i-1)$ are known.

Forward Algorithm

- The Forward algorithm gives the *sum of all paths* through an HMM efficiently.
- This is the total probability of the output sequence

$$P(O|\lambda)$$

Discussion question:

How is this different from just using an N-gram language model to get the probability of the output sequence? Is this better?

Using HMMs

- **Viterbi algorithm*** is an efficient solution to finding the most probable sequence of hidden states that could have generated the observed sequence
  - Also called the *"Viterbi labeling"*

- We'll also briefly mention two additional algorithms:
  - Finding the total probability of an observed string according to an HMM
  - Finding the most likely state at any given point

https://en.wikipedia.org/wiki/Andrew_Viterbi

Most likely labels

- Given an HMM (σ, τ) and an observed sequence of words X, what is the most likely label sequence $\hat{y}$?

$$\hat{y} = arg_y max(x, y)$$

- In principle we could solve this by enumerating all possible Y, and finding the one that maximized P(x,y)
  - But this grows exponentially with the length of the sentence (n)

How bad is the brute force solution?

- Assume that every word had exactly two possible tags.

- Then a string of length one has two possible sequences,
- A sequence of two word has 4 possible state sequences
- … a sequence of n words has 2^n possible state sequences

Pick a random sentence out of the New York Times: it has 38 words.

- $2^{38} \cong 10^{12}$ i.e. a *trillion*.

Dynamic Programming Solution: Viterbi decoding

Finding the probability of the most likely solution for the prefix **X** up to position **i** that ends in state **y**

$$\mu_y(i) = \mathrm{P}(x_{1,i}, Y_i = y)$$

We can compute all our **μ_y(i)** by starting on the left and working our way to the right

$$\mu_{\triangleright}(0) = 1.0$$

We next go from time (i-1) to time i as follows:

$$\mu_y(i) = \max_{y'=1}^{m} \mu_{y'}(i-1)\sigma_{y',y}\tau_{y,x_i}$$

Dynamic Programming Solution: Viterbi decoding

At each stage we need look backward only one step because the new maximum probability must be the continuation from the maximum probability at one of the previous states.

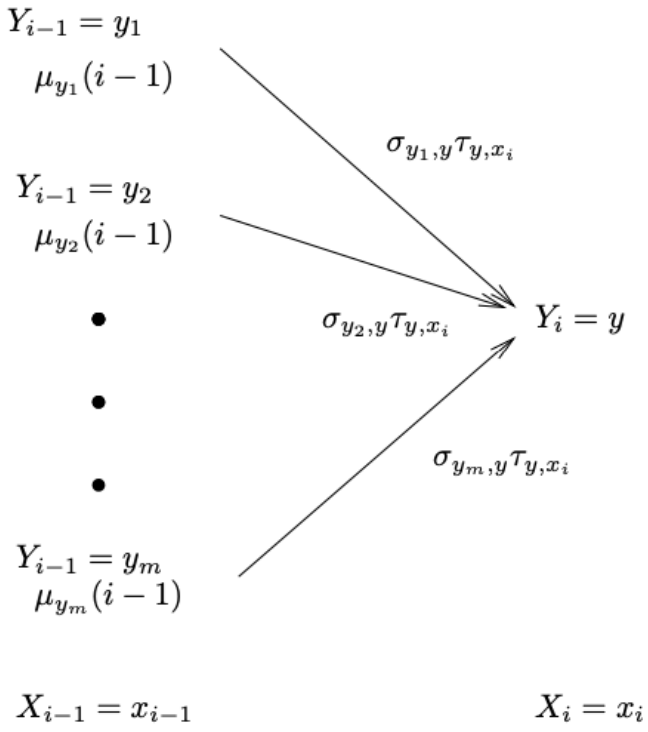


Figure 3.4: A piece of a trellis showing how to compute $\mu_y(i)$ if all of the $\mu_{y'}(i-1)$ are known.

Dynamic Programming Solution: Viterbi decoding

- Computation starts at the left by setting $\mu_{\triangleright}(0) = 1$
- Moving to word 1, we need to compute $\mu_N(1)$ and $\mu_V(1)$
  - This is straightforward since there is only one possible prior state
  - $\mu_N(1) = \mu_{\triangleright}(0) * \tau_{N,flour} * \sigma_{\triangleright,N}$
  - $\mu_N(1) = 1.0 * 0.3 * 0.4$
  - $\mu_N(1) = 0.12$
- Moving to word 2, there is only one possible generating state with non-zero value (namely 'N'), but we need to compute the two possible predecessors at i=1
  - …
- Moving to word 3…

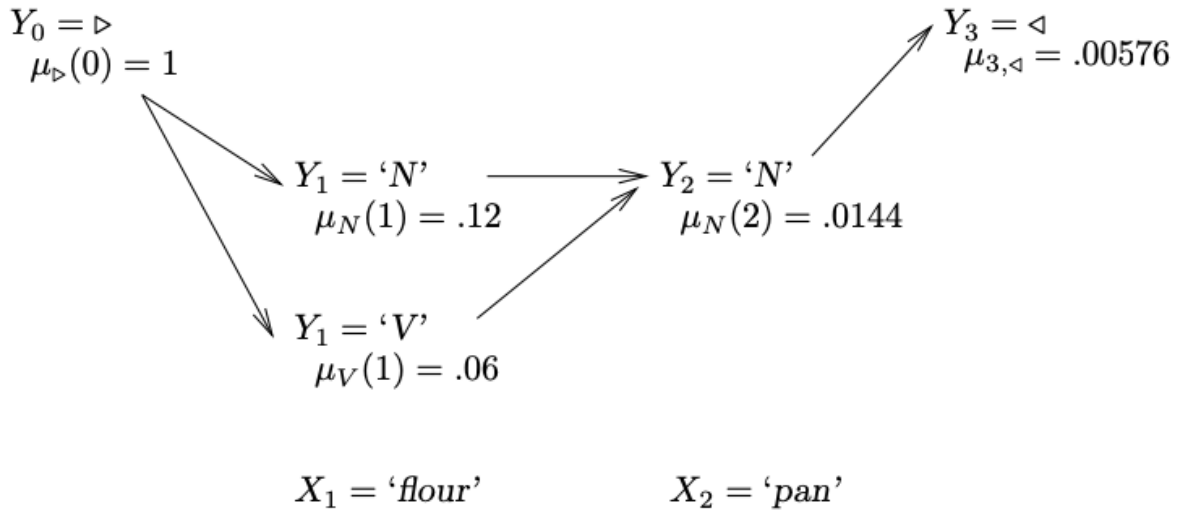


Figure 3.5: The trellis for '*flour pan*' showing the $\mu_y(i)$'s

Viterbi Algorithm Complexity

- Naïve approach requires exponential time to evaluate all N^T state sequences
- Forward algorithm using dynamic programming takes $O(N^2T)$ computations

(Where T is the length of the sentence, and N is the size of the tag set)

How did the Forward algorithm (for sequence likelihood) related to Viterbi (for most probably tag path)?

Problem 2: Decoding

- The Forward algorithm gives the *sum of all paths* through an HMM efficiently.
- Here, we want to find the *highest probability path*.
- We want to find the state sequence $Q=q_1 \ldots q_T$, such that

$$Q = \underset{Q'}{\operatorname{argmax}} P(Q' \mid O, \lambda)$$

Viterbi Algorithm

- Just like the forward algorithm, but instead of *summing* over transitions from incoming states, compute the *maximum*

- Forward:

$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i)\, a_{ij} \right] b_j(o_t)$$

- Viterbi Recursion:

$$\delta_t(j) = \left[\max_{1 \le i \le N} \delta_{t-1}(i)\, a_{ij} \right] b_j(o_t)$$

Viterbi Algorithm

- Just like the forward algorithm, but instead of *summing* over transitions from incoming states, compute the *maximum*

Tau: probability of generating output word "o" from tag state "j" (at time "t")

- Forward:

$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i)\, a_{ij} \right] b_j(o_t)$$

Sigma: probability of transitioning from state "i" to state "j"

- Viterbi Recursion:

$$\delta_t(j) = \left[\max_{1 \le i \le N} \delta_{t-1}(i)\, a_{ij} \right] b_j(o_t)$$

Probability of getting to state "i" at time "t-1"

Not quite what we want…

- Viterbi recursion computes the *maximum probability* path to state j at time t given that the partial observation $o_1 \ldots o_t$ *has been* generated

$$\delta_t(j) = \left[\max_{1 \le i \le N} \delta_{t-1}(i)\, a_{ij} \right] b_j(o_t)$$

- But we want the *path itself* that gives the maximum probability
- *Solution:*
  1. ***Keep backpointers***
  2. ***Find*** $\arg\max_j \delta_T(j)$
  3. ***Chase backpointers from state j at time T to find state sequence (backwards)***

Viterbi Algorithm

Let T = # of part-of-speech tags
W = # of words in the sentence

/* Initialization Step */

for t = 1 to T
Score(t, 1) = $Pr(Word_1| Tag_t) * Pr(Tag_t| \varphi)$
BackPtr(t, 1) = 0;

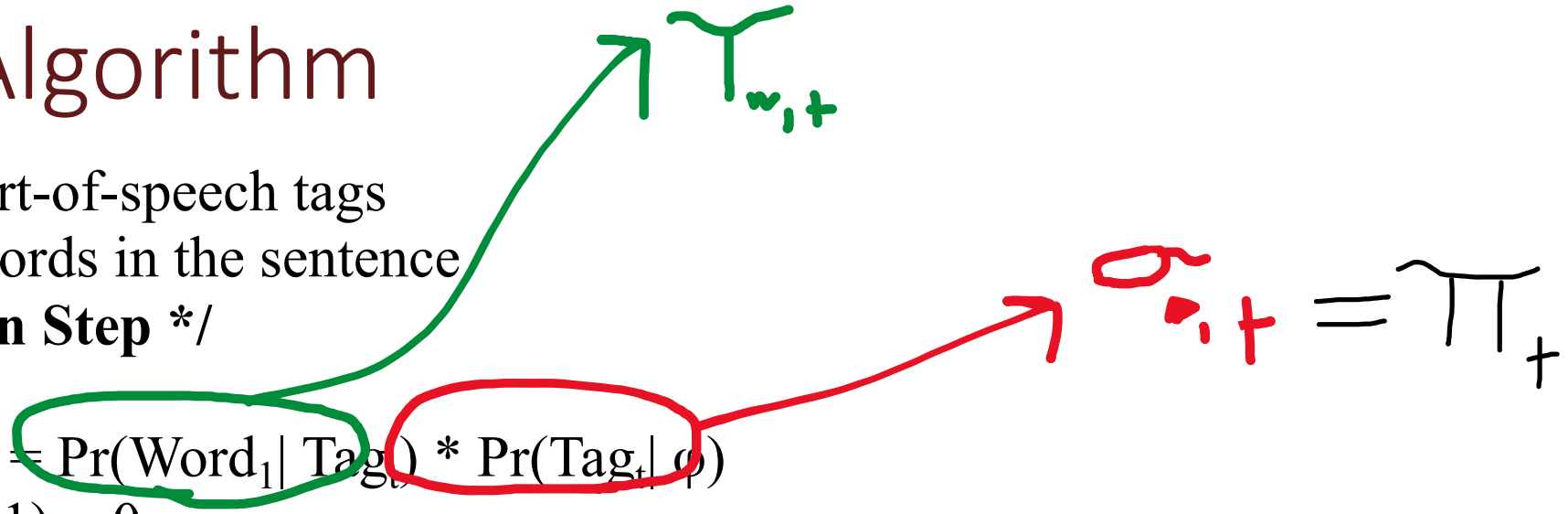
/* Iteration Step */

for w = 2 to W
for t = 1 to T
Score(t, w) = $Pr(Word_w| Tag_t) * MAX_{j=1,T}(Score(j, w-1) * Pr(Tag_t| Tag_j))$
BackPtr(t, w) = index of j that gave the max above

/* Sequence Identification */

Seq(W) = t that maximizes Score(t,W)
for w = W -1 to 1
Seq(w) = BackPtr(Seq(w+1),w+1)

Viterbi Algorithm



Let T = # of part-of-speech tags
W = # of words in the sentence
/* Initialization Step */
for t = 1 to T
Score(t, 1) = $Pr(Word_1 | Tag_t) * Pr(Tag_t | \phi)$
BackPtr(t, 1) = 0;
/* Iteration Step */
for w = 2 to W
for t = 1 to T
Score(t, w) = $Pr(Word_w | Tag_t) * MAX_{j=1,T}(Score(j, w-1) * Pr(Tag_t | Tag_j))$
BackPtr(t, w) = index of j that gave the max above
/* Sequence Identification */
Seq(W) = t that maximizes Score(t,W)
for w = W -1 to 1
Seq(w) = BackPtr(Seq(w+1),w+1)

Viterbi Algorithm

Let T = # of part-of-speech tags
W = # of words in the sentence
/* Initialization Step */
for t = 1 to T
Score(t, 1) = $Pr(Word_1 | Tag_t) * Pr(Tag_t | \phi)$
BackPtr(t, 1) = 0;
/* Iteration Step */
for w = 2 to W
for t = 1 to T
Score(t, w) = $Pr(Word_w | Tag_t) * MAX_{j=1,T}(Score(j, w-1) * Pr(Tag_t | Tag_j))$
BackPtr(t, w) = index of j that gave the max above
/* Sequence Identification */
Seq(W) = t that maximizes Score(t,W)
for w = W -1 to 1
Seq(w) = BackPtr(Seq(w+1),w+1)

Another inner loop, But we're only keeping the best path to each possible prior tag --> still an exponential reduction in computation compared to the brute-force solution

Probability up to this point for the prior tag j

(The backpointers keep track of the total path to that tag, but here we "don't care")

Viterbi Algorithm

Let T = # of part-of-speech tags
W = # of words in the sentence

/* Initialization Step */

for t = 1 to T
Score(t, 1) = $Pr(Word_1 | Tag_t) * Pr(Tag_t | \varphi)$
BackPtr(t, 1) = 0;

/* Iteration Step */

for w = 2 to W
for t = 1 to T
Score(t, w) = $Pr(Word_w | Tag_t) * MAX_{j=1,T}(Score(j, w-1) * Pr(Tag_t | Tag_j))$
BackPtr(t, w) = index of j that gave the max above

/* Sequence Identification */

Seq(W) = t that maximizes Score(t,W)
for w = W -1 to 1
Seq(w) = BackPtr(Seq(w+1),w+1)

Just need to recurse through the back-pointers

Done once we find the best tag that could generate last word

Check in: Given the pseudo-code from the previous slide could you implement the Viterbi algorithm right now?

If not, what is the most confusing part?

The Three Basic HMM Problems

- *Problem 1 (Evaluation):* Given the observation sequence $O=o_1,\ldots,o_T$ and an HMM model $\lambda = (A, B, \pi)$, how do we compute the probability of O given the model?

- *Problem 2 (Decoding):* Given the observation sequence O and an HMM model λ, how do we find the state sequence that best explains the observations?

- *Problem 3 (Learning):* How do we adjust the model parameters $\lambda = (A, B, \pi)$, to maximize $P(O|\lambda)$?

So, we have *O*, and know what our state vocabulary is...

... but we don't know transition or emission probabilities.

We can use *Baum-Welch algorithm* (a.k.a. *Forward-Backward algorithm*) to iteratively estimate *A* and *B*.

Leonard Baum

Lloyd Welch

Recall from before that the *forward probability* $\alpha_t(i)$ is the probability of ending up in state *i* given observations $O_{1:t}$.

A related property is the *backward probability* $\beta_t(i)$, which represents the probability of seeing observations $O_{t:T}$, given that we are currently in state *i* at time *t*.

This is calculated using the *backward algorithm*, which is very similar to the forward algorithm (but in reverse!).

Forward-Backward

We're not going to cover the Forward-Backward algorithm here… but just know that it:

is a dynamic programming solution (like Viterbi) for estimating the optimal parameter values for an HMM (transition and emission / Sigma and Tau probability) ***without any labeled training data***

The intuition behind how this works is as follows

Forward-Backward intuition

- The intuition for how this works is that we start with a (very bad) guess for Sigma and Tau.
- We use those to estimate

$$\hat{a}_{ij} = \frac{\text{expected \# transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}$$

- If we had an estimate of the probability of transition i→j occurring at each time t, we could sum them to get the total count for i→j.

- We use those estimates to calculate new values for Sigma and Tau
- Which in turns gives us new transition estimates
- Which in turns gives us new Sigma and Tau estimates
- …
- You repeat that process until the algorithm converges (extremely little change in the parameters from one iteration to the next) or we give up at stop after a large number of iterations

Unsupervised tagging

Obviously this just sounds **cool (😎)**. A machine learning model *without* any labeled training data?

- However, the results are often “strange” (they represent *some* way to categorize the data, but not exactly the same linguistically constrained way that we’d like)
- And perform worse compared to results from models trained on labeled data (supervised training)

- E.g., Garrette & Baldridge (2013) outperformed an unsupervised tagger with just two hours worth of annotation time, in two languages—Kinyarwanda and Malagasy—they did not know ahead of time.