6.863J Natural Language Processing
Lecture 6: The Red Pill or the Blue Pill, Episode 1: part-of-speech tagging

Instructor: Robert C. Berwick
berwick@ai.mit.edu

The Menu Bar

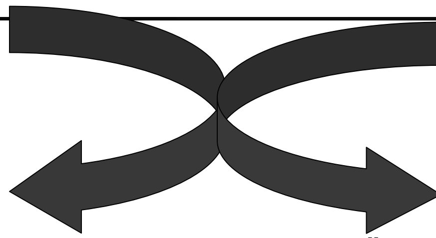
- Administrivia:
  - Schedule alert: Lab1b due today
  - Lab 2a, released today; Lab 2b, this Weds

Agenda:

Red vs. Blue:

- Ngrams as models of language
- Part of speech 'tagging' via statistical models
- Ch. 6 & 8 in Jurafsky

The Great Divide in NLP: the red pill or the blue pill?



"Knowledge Engineering" approach
Rules built by hand w/ K of Language
"Text understanding"

"Trainable Statistical" Approach
Rules inferred from lots of data ("corpora")
"Information retrieval"

Two ways

- Probabilistic model - some constraints on morpheme sequences using prob of one character appearing before/after another

  prob(ing | stop) vs. prob(ly| stop)

- Generative model – concatenate then fix up joints
  - stop + -ing = stopping,   fly + s = flies
  - Use a cascade of transducers to handle all the fixups

The big picture II

- In general: 2 approaches to NLP
- Knowledge Engineering Approach
  - Grammars constructed by hand
  - Domain patterns discovered by human expert via introspection & inspection of ‘corpus’
  - Laborious tuning
- Automatically Trainable Systems
  - Use statistical methods when possible
  - Learn rules from annotated (or o.w. processed) corpora

Preview of tagging

- What is tagging?
- Input: word sequence:
  Police police police
- Output: classification (binning) of words -
  Noun Verb Noun *or*
  [Help!]

Preview of tagging & pills: red pill and blue pill methods

- Method 1: statistical (n-gram)
- Method 2: more symbolic (but still includes some probabilistic training + fixup) – ‘example based’ learning

What is part of speech tagging & why?

Input: **`the lead paint is unsafe`**

Output: **`the/Det lead/N paint/N is/V unsafe/Adj`**

Or: BOS **`the lyric beauties of Schubert ‘s Trout Quintet : its elemental rhythms and infectious melodies : make it a source of pure pleasure for almost all music listeners ./`**

Tagging for this..

```
The/DT lyric/JJ beauties/NNS of/IN
 Schubert/NNP 's/POS Trout/NNP Quintet/NNP
--/:
 its/PRP$ elemental/JJ rhythms/NNS
and/CC   infectious/JJ melodies/NNS
--/: make/VBP   it/PRP
 a/DT source/NN  of/IN  pure/JJ pleasure/NN
for/IN almost/RB  all/DT music/NN listeners/NNS  ./.
```


Tagging words

- Well defined
- Easy, but not too easy (not AI-complete)
- Data available for machine learning methods
- Evaluation methods straightforward

Why should we care?

- The first statistical NLP task
- Been done to death by different methods
- Easy to evaluate (how many tags are correct?)
- Canonical finite-state task
  - Can be done well with methods that look at local context
  - Though should "really" do it by parsing!

Why should we care?

- "Simplest" case of recovering surface, underlying form via statistical means
- We are modeling p(word seq, tag seq)
- The tags are hidden, but we see the words
- Is tag sequence T likely given these words?

Tagging as n-grams

- Most likely word? Most likely tag *t* given a word *w?* = P(tag|word) – not quite
- Task of predicting the next word
- Woody Allen:

  "I have a gub"

But in general: predict the N^{th} tag from the preceding n-1 word (tags) aka N-gram

Summary of n-grams

- n-grams define a probability model over sequences
- we have seen examples of sequences of words, but one can also look at characters
- n-grams deal with sparse data by using the Markov assumption

Markov models: the 'pure' statistical model…

- 0th order Markov model: $P(w_i)$
- 1st order Markov model: $P(w_i|w_{i-1})$
- 2nd order Markov model: $P(w_i|w_{i-1}\, w_{i-2})$

…

- Where do these probability estimates come from?
- Counts: $P(w_i \mid w_{i-1}) = \mathit{count}(w_i, w_{i-1})/\mathit{count}(w_{i-1})$

(so-called maximum likelihood estimate - MLE)

N-grams

- But…How many possible distinct probabilities will be needed?, i.e. parameter values
- Total number of word tokens in our training data
- Total number of unique words: word types is our vocabulary size

n-gram Parameter Sizes – large!

- Let V be the vocabulary, size of V is |V|, 3000 distinct types say
- $P(W_i=x)$ how many different values for W_i ?

- $P(W_i = x \mid W_j = y)$, # distinct doubles =

  $3x10^3 \times 3x10^3 = 9 \times 10^6$

$P(W_i = x \mid W_k = z, W_j = y)$, how many distinct triples?
27×10^9

Choosing *n*

Suppose we have a vocabulary (V) = 20,000 words

n	Number of bins
2 (bigrams)	400,000,000
3 (trigrams)	8,000,000,000,000
4 (4-grams)	1.6×10^{17}

How far into the past should we go?

- "*long distance___*"
- Next word? *Call?*
- $p(w_n|w\ldots)$
- Consider special case above
- Approximation says that

  | long distance call|/|distance call| ≈ |distance call|/|distanc
- If context 1 word back = bigram

But even better approx if 2 words back: *long distance___*

Not always right: long distance runner/long distance call

Further you go: *collect long distance_____*

Parameter size vs. corpus size

- Corpus: said the joker to the thief

  |V| = 5
- What's the max # of parameters?
- What's observed? (All pairs)

- We observe only |V| many bigrams!
- V had better be <u>large</u> wrt # parameters

Reliability vs. discrimination

"large green ___________"

tree? mountain? frog? car?

"swallowed the large green ________"

pill? broccoli?

Reliability vs. discrimination

- larger *n*: more information about the context of the specific instance (greater discrimination)
- smaller *n*: more instances in training data, better statistical estimates (more reliability)

Statistical estimators

Example:

Corpus: five Jane Austen novels

N = 617,091 words

V = 14,585 unique words

Task: predict the next word of the trigram "inferior to ____"

from test data, *Persuasion*:

"[In person, she was] inferior to *both* [sisters.]"

Shakespeare in lub…
The unkindest cut of all

- Shakespeare: 884,647 words or *tokens* (Kucera, 1992)
- 29,066 *types* (incl. proper nouns)
- So, # bigrams is $29{,}066^2$ > 844 million. 1 million word training set doesn't cut it – only 300,000 difft bigrams appear
- Most entries are <u>zero</u>
- So we can't go very far…

Bigram models in practice

- P(Bush read a book) =
  P(Bush | BOS) x
  P(read | Bush) x
  P(a | read) x
  P(book | a) x
  P(EOS | book)

Estimate via <u>counts</u> $P(w_i \mid w_{i-1}) = \mathit{count}(w_i , w_{i-1})/\mathit{count}(w_{i-1})$

On unseen data, $\mathit{count}(w_i , w_{i-1})$ or worse, $\mathit{count}(w_{i-1})$ could be zero! What to do?

How to Estimate?

- p(z | xy) = ?
- Suppose our training data includes
  … xya ..
  … xyd …
  … xyd …
  but never xyz
- Should we conclude
  p(a | xy) = 1/3?
  p(d | xy) = 2/3?
  p(z | xy) = 0/3?
- NO! Absence of xyz might just be bad luck.

Smoothing

Smoothing deals with events that have been observed zero times

- Smoothing algorithms also tend to improve the accuracy of the model
- Not just unobserved events: what about events observed once?

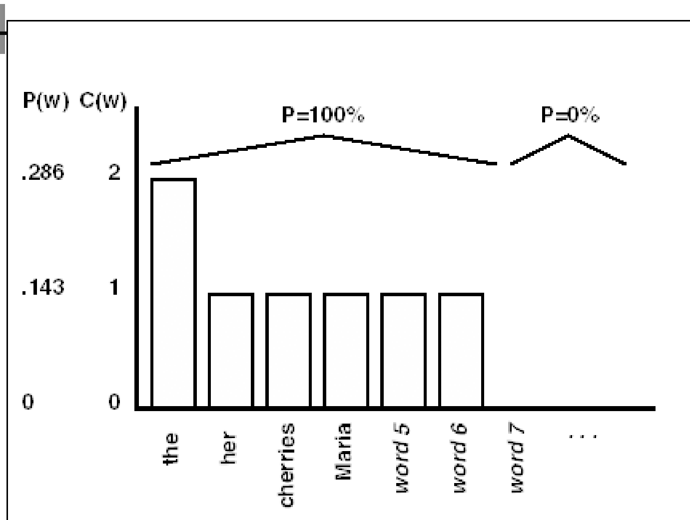
Smoothing the Estimates

- Should we conclude
  - p(a | xy) = 1/3? *reduce this*
  - p(d | xy) = 2/3? *reduce this*
  - p(z | xy) = 0/3? *increase this*
- Discount the positive counts somewhat
- Reallocate that probability to the zeroes
- Especially if the denominator is small …
  - 1/3 probably too high, 100/300 probably about right
- Especially if numerator is small …
  - 1/300 probably too high, 100/300 probably about right

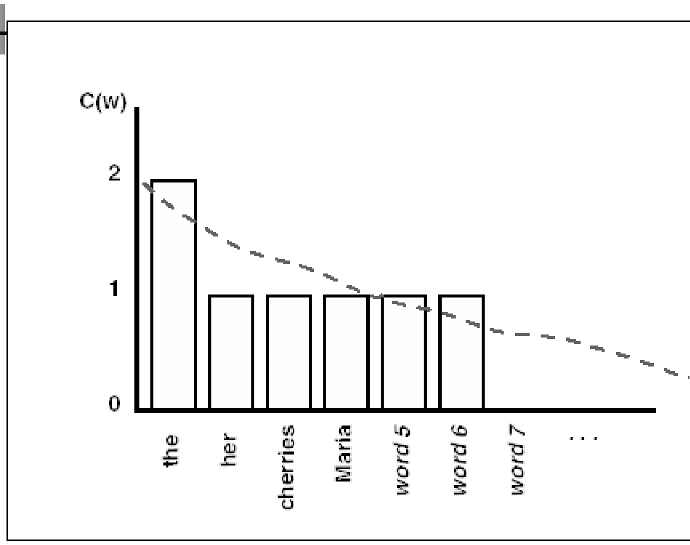
Add-one smoothing

- Let V be the number of words in our vocabulary
- Remember that we observe only V many bigrams
- Assigns count of 1 to unseen bigrams

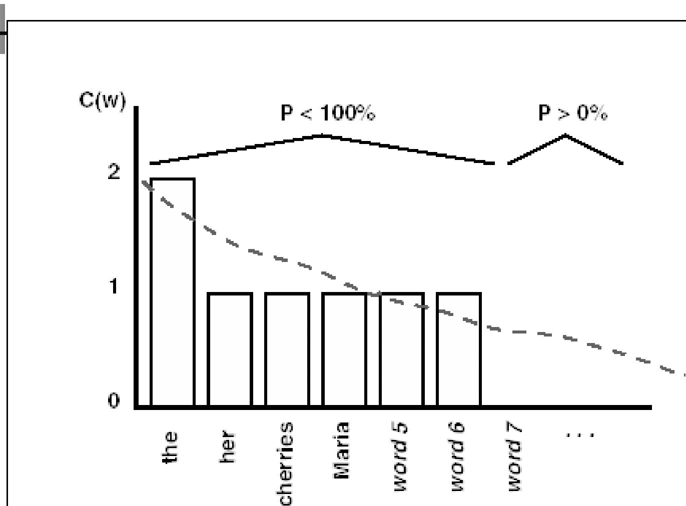
Maximum likelihood estimate



Actual probability distribution



Comparison



Add-One Smoothing

xya	1	1/3	2	2/29
xyb	0	0/3	1	1/29
xyc	0	0/3	1	1/29
xyd	2	2/3	3	3/29
xye	0	0/3	1	1/29
…				
xyz	0	0/3	1	1/29
Total xy	3	3/3	29	29/29

Add-one smoothing

Example: Bush reads a book

- P(Bush reads a book)
- Without smoothing:
- With add-one smoothing (assuming c(Bush)=1 but c(Bush, read) =0

Add-One Smoothing

300 observations instead of 3 – better data, less smoothing

xya	100	100/300	101	101/326
xyb	0	0/300	1	1/326
xyc	0	0/300	1	1/326
xyd	200	200/300	201	201/326
xye	0	0/300	1	1/326
…				
xyz	0	0/300	1	1/326
Total xy	300	300/300	326	326/326

Add-One Smoothing

Suppose we're considering 20000 word types, not 26 letters

xya	1	1/3	2	2/29
xyb	0	0/3	1	1/29
xyc	0	0/3	1	1/29
xyd	2	2/3	3	3/29
xye	0	0/3	1	1/29
…				
xyz	0	0/3	1	1/29
Total xy	3	3/3	29	29/29

Add-One Smoothing

As we see more word types, smoothed estimates keep falling

see the abacus	1	1/3	2	2/20003
see the abbot	0	0/3	1	1/20003
see the abduct	0	0/3	1	1/20003
see the above	2	2/3	3	3/20003
see the Abram	0	0/3	1	1/20003
…				
see the zygote	0	0/3	1	1/20003
Total	3	3/3	20003	20003/20003

Problems…too many mouths to feed

- Suppose we're dealing with a vocab of 20000 words
- As we get more and more training data, we see more and more words that need probability – the probabilities of existing words keep dropping, instead of converging
- This can't be right – eventually they drop too low

Good-Turing smoothing

- Add-1 works horribly in practice – adding 1 seems too large
- So…imagine you're sitting at a sushi bar with a conveyor belt

- How likely are you to see a <u>new</u> kind of seafood appear?

The sushi bar

[redacted]

10 tuna, 3 unagi, 2 salmon, 1 shrimp, 1 octopus, 1 yellowtail

How likely are you to see another salmon? < 2/18

Good-Turing smoothing

- How likely are you to see a <u>new</u> type of seafood?
- How many types of seafood (submarines, words) were seen only once? Use this to predict probabilities for unseen events
- Let n_1 be the # of events that occurred <u>once</u>, then the initial est. of this is, $p_0 = n_1/N$
- Let n_2 be the # of events that occurred <u>twice</u>

Good-Turing smoothing

- 10 tuna, 3 unagi, 2 salmon, 1 shrimp, 1 octopus, 1 yellowtail
- $p_0 = n_1/N = 3/18$
- Now how likely is *octopus*?
- Good-Turing estimate: for any n-gram that occurs *r* times, we pretend it occurs *r** times,

At the sushi bar

- 10 tuna, 3 unagi, 2 salmon, 1 shrimp, 1 octopus, 1 yellowtail
- Octopus occurs 1 time, r=1 so we adjust to 1*
- We need n_1 # of things that occur once = 3
- We need n_2 # of things that occur twice = 1
- Then

Is that the final word?

- No – what happens if something is not seen at all?
- Then you must backoff to bigrams, unigrams…
- Many other new smoothing methods available (see book) – various weighting/discounting schemes (we shall revisit: EM algorithm)
- Are we done? Can we use trigrams now?
- Not quite…

Time must have a stop

- Note that in the trigram model the length of the sentence *n* is <u>variable</u>
- But then, what is the event space for calculating probabilities?
- Suppose our alphabet is just *a, b,* and the language is all strings over this, eg, ε, a, b, aa, bb, ab, …
- Assume unigram model, P(a)=P(b) = 0.5
- Then P(aa)= 0.5^2 = 0.25 = P(bb), etc…
- But then, adding, P(a)+P(b)+P(aa)+P(bb) =
  1.5 ????

What went wrong???

Must have a stop

- No probability for P(ε)
- Add special stop symbol:
  - P(a)=P(b)= 0.25; P(stop)= 0.5
- Now it works out: P(a stop) = P(b stop) = 0.25 x 0.5 = 0.125; P(aa stop) = 0.25^2 = .03125, etc.
- Exercise: show the P sum is now 1.

OK, back to sushi

- Tagging for the Trout:

Tagging for this..

```
The/DT lyric/JJ beauties/NNS of/IN
 Schubert/NNP 's/POS Trout/NNP Quintet/NNP
--/:
 its/PRP$ elemental/JJ rhythms/NNS
and/CC   infectious/JJ melodies/NNS
--/: make/VBP   it/PRP
 a/DT source/NN  of/IN  pure/JJ pleasure/NN
for/IN almost/RB  all/DT music/NN listeners/NNS  ./.
```


(Next step: bracketing…)

```
 [The/DT lyric/JJ beauties/NNS ]
of/IN
[ Schubert/NNP 's/POS Trout/NNP Quintet/NNP ]
--/:
[ its/PRP$ elemental/JJ rhythms/NNS ]
and/CC  [ infectious/JJ melodies/NNS ]
--/: make/VBP  [ it/PRP ]
[ a/DT source/NN ] of/IN [ pure/JJ pleasure/NN ]
for/IN almost/RB  [ all/DT music/NN listeners/NNS ] ./.
```

Tagging methods

- Statistical Tagger T3
- Error-driven Transformation-based Tagger TBT
- Maximum Entropy Tagger MET
- Example-based tagger ET

Why tagging? Beyond part of speech

- Syntactic word class
- Word sense
- Attachment
- Shallow parsing
- Sentence boundary detection

Two approaches

1. Statistical model
2. Deterministic baseline tagger composed with a cascade of fixup transducers

These two approaches are the guts of Lab 2

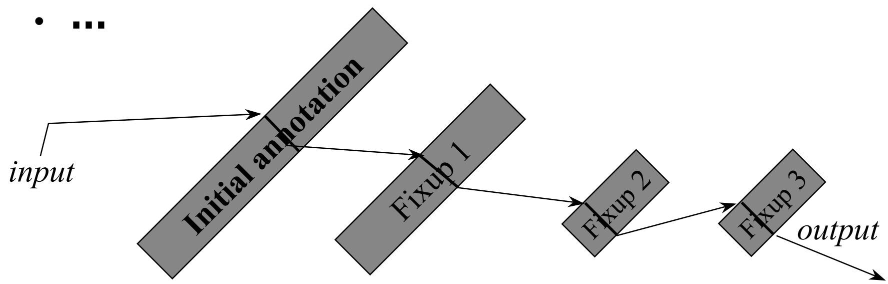
(lots of others methods: decision trees, …)

Statistical Model

- Prob(Tag sequence, word sequence) – based on n-grams: train on marked up text
- We shall see how to do this in detail in a moment

Another FST Paradigm: Successive Fixups

- Like successive markups but *alter*
- Morphology
- Phonology
- Part-of-speech tagging
- …



Evaluation Criteria

- Precision/recall
- Accuracy/ambiguity

An exemplar for the divide: “tagging” text

- Input: the lead paint is unsafe
  Output: the/Det lead/N paint/N is/V unsafe/Adj

- Can be challenging:
  I know that
  I know that block
  I know that blocks the sun

- new words (OOV= out of vocabulary); words can be whole phrases (“I can’t believe it’s not butter”)

What are tags?

- Bridge from words to parsing – but not quite the morphemic details that Kimmo provides (but see next slide)
- Idea is more divide-and-conquer – and depends on task
- “Shallow” analysis for “shallow parsing”

More sophisticated – use features

- Word form: $A^+ \rightarrow 2^{(L,C1,C2,\ldots,Cn)} \rightarrow T$
  - He always books the violin concert tickets early.
    - books $\rightarrow$ {(book-1,Noun,Pl,-,-),(book-2,Verb,Sg,Pres,3)}
    - tagging (disambiguation): ... $\rightarrow$ (Verb,Sg,Pres,3)
  - ...was pretty good. However, she did not realize...
    - However $\rightarrow$ {(however-1,Conj/coord,-,-,-), (however- 2,Adv,-,-,-)}
    - tagging: ... $\rightarrow$ (Conj/coord,-,-,-)

Two approaches

1. Noisy Channel Model (statistical) – what's that?? (we will have to learn some statistics)

2. Deterministic baseline tagger composed with a cascade of fixup transducers

These two approaches will the guts of Lab 2

(lots of others: decision trees, …)

Example tagsets

- 87 tags - Brown corpus
- Three most commonly used:

1. Small: 45 Tags - Penn treebank (Medium size: 61 tags, British national corpus
2. Large: 146 tags

Big question: have we thrown out the right info? Impoverished? How?

Brown/Upenn corpus tags

J. text,
p. 297
Fig 8.6
1M words
60K tag
counts

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	+,%, &
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	*#*
PDT	Predeterminer	*all, both*	"	Left quote	(' or ")
POS	Possessive ending	*'s*	"	Right quote	(' or ")
PP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PP$	Possessive pronoun	*your, one's*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	*up, off*			

Current (computer/human) performance

Input: `the lead paint is unsafe`

Output: `the/Det lead/N paint/N is/V unsafe/Adj`

- How many tags are correct?
  - About 97% currently
  - But baseline is already 90%:
    - Baseline is Homer Simpson algorithm:
    - Tag every word with its most frequent tag (Unigram frequency)
    - Tag unknown words as nouns
- How well do people do?

Ok, what should we look at?

correct tags

PN	Verb	Det	Noun	Prep	Noun	Prep	Det	Noun
Bill	directed	a	cortege	of	autos	through	the	dunes
PN	Adj	Det	Noun	Prep	Noun	Prep	Det	Noun
Verb	Verb	Noun	Verb					
			Adj					
			Prep					
			…?					

some possible tags for each word (maybe more)

Each unknown tag is <u>constrained</u> by its word and by the tags to its immediate left and right. But those tags are unknown too …

Ok, what should we look at?

correct tags

PN	Verb	Det	Noun	Prep	Noun	Prep	Det	Noun
Bill	directed	a	cortege	of	autos	through	the	dunes
PN	Adj	Det	Noun	Prep	Noun	Prep	Det	Noun
Verb	Verb	Noun	Verb					
			Adj					
			Prep					
			...?					

some possible tags for each word (maybe more)

Each unknown tag is constrained by its word
and by the tags to its immediate left and right.
But those tags are unknown too …

Ok, what should we look at?

correct tags

PN	Verb	Det	Noun	Prep	Noun	Prep	Det	Noun
Bill	directed	a	cortege	of	autos	through	the	dunes
PN	Adj	Det	Noun	Prep	Noun	Prep	Det	Noun
Verb	Verb	Noun	Verb					
			Adj					
			Prep					
			...?					

some possible tags for each word (maybe more)

Each unknown tag is constrained by its word
and by the tags to its immediate left and right.
But those tags are unknown too …

Ok, what *should* we look at?

correct tags

PN	Verb	Det	Noun	Prep	Noun	Prep	Det	Noun
Bill	directed	a	cortege	of	autos	through	the	dunes
PN	Adj	Det	Noun	Prep	Noun	Prep	Det	Noun
Verb	Verb	Noun	Verb					
			Adj					
			Prep					
			…?					

some possible tags for each word (maybe more)

Each unknown tag is constrained by its word and by the tags to its immediate left and right. But those tags are unknown too …

We can use n-grams for tagging

- Replace ‘words’ with ‘tags’
- Find best <u>maximum likelihood estimates</u>
- Estimates calculated this way:
  - P(noun|det) = p(det, noun)/p(det) replace:
  - $\approx \frac{\text{count(det at position i-1 \& noun at i)}}{\text{count(det at position i-1)}}$
  - Correction: include frequency of context word

    $\approx \frac{\text{count(det at position i-1 \& noun at i)}}{\text{count(det at position i-1)*count (noun at i)}}$

Find optimal path – highest p, using dynamic programming algorithm, approx. linear in length of tag sequence

Example

The guy still saw her
Det NN NN NN PPO
VB VB VBD PP$
RB

Table 2 from DeRose (1988)
Det=determiner, NN=noun, VB=verb, RB=adverb, VBD=past-tense-verb, PPO=object pronoun and PP$=possessive pronoun

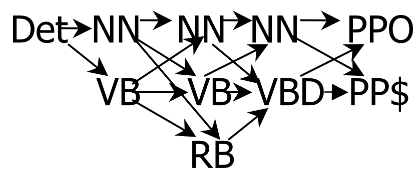
Find the Max likelihood estimate (MLE) path through this ‘trellis’

Transitional probability estimates from counts

	DT	NN	PPO	PP$	RB	VB	VBD
DT	0	186	0	8	1	8	9
NN	40	1	3	40	9	66	186
PPO	7	3	16	164	109	16	313
PP$	176	0	0	5	1	1	2
RB	5	3	16	164	109	16	313
VB	22	694	146	98	9	1	59
VBD	11	584	143	160	2	1	91

Tagging search tree (trellis)

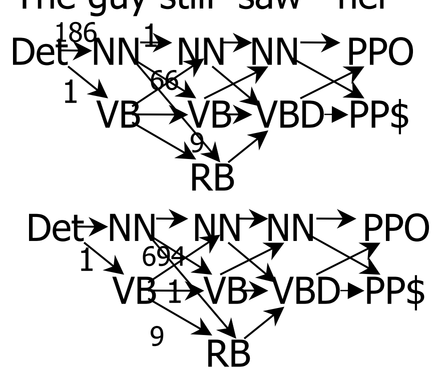
The guy still saw her



Step 1. c(DT-NN)= 186
c(DT-VB) = 1
Keep both paths. (Why?)
Step 2. Pick *max* to each of the tags NN, VB, RB
need keep only the *max.* Why?

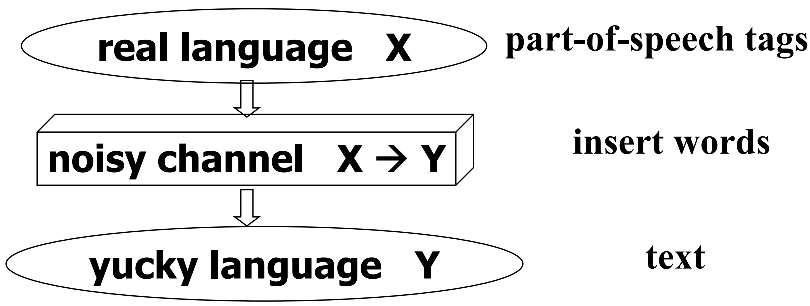
Trellis search

The guy still saw her



Finite-state approaches

- Noishy Chunnel Muddle (statistical)



want to recover X from Y

So far, then…

- n-gram models are a.k.a. Markov models/chains/processes.
- They are a model of how a sequence of observations comes into existence.
- The model is a probabilistic walk on a FSA.
- *Pr(a|b)* = probability of entering state *a*, given that we're currently in state *b*.

How well does this work for tagging?

- 90% accuracy (for unigram) pushed up to 96%
- So what?
- How *good* is this? Evaluation!

Evaluation of systems

- The principal measures for information extraction tasks are recall and precision.
- *Recall* is the number of answers the system got right divided by the number of possible right answers
  - It measures how complete or comprehensive the system is in its extraction of relevant information
- *Precision* is the number of answers the system got right divided by the number of answers the system gave
  - It measures the system's correctness or accuracy
  - Example: there are 100 possible answers and the system gives 80 answers and gets 60 of them right, its recall is 60% and its precision is 75%.

A better measure - Kappa

- Takes baseline & complexity of task into account – if 99% of tags are Nouns, getting 99% correct no great shakes
- Suppose no “Gold Standard” to compare against?
- P(A) = proportion of times hypothesis *agrees* with standard (% correct)
- P(E) = proportion of times hypothesis and standard would be *expected* to agree by chance (computed from some other knowledge, or actual data)

Kappa [p. 315 J&M text]

- Note K ranges between 0 (no agreement, except by chance; to complete agreement, 1)
- Can be used even if no ‘Gold standard’ that everyone agrees on
- K> 0.8 is good

Kappa

- *A* = actual agreement; *E* = expected agreement
- consistency is more important than "truth"
- methods for raising consistency
  - style guides (often have useful insights into data)
  - group by task, not chronologically, etc.
  - annotator acclimatization

Statistical Tagging Methods

- Simple bigram – ok, done
- Combine bigram and unigram

- OUR GOAL: maximize P(T,w) where T=a tag sequence (guessed); and w= the <u>observed</u> word sequence – note this is a <u>joint</u> probability
- So, why not use our formula for joint probabilities…

Our first try…

- $P(T,w) = P(T)\,P(w \mid T)$

Our plan to find P(T,w)

- Find best P(T) – probability of a tag sequence
- How? A: use bigrams
- Find best P(w|T) --???? How?

Punchline – recovering (words, tags)

tags X→

BOS PN Verb Det Noun Prep Noun Prep Det Noun **EOS**

Bill directed a cortege of autos through the dunes

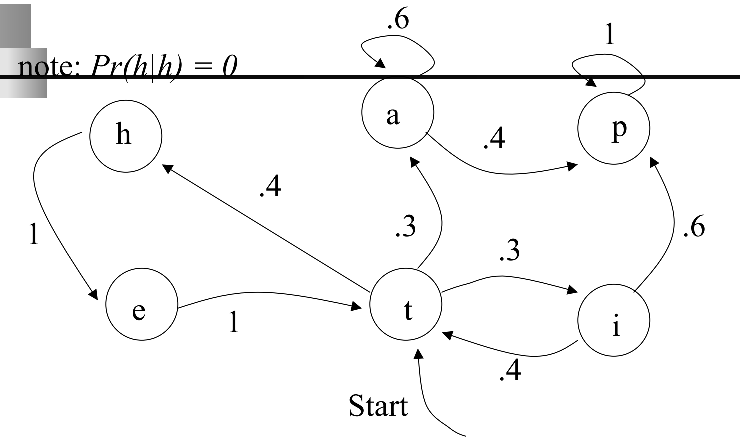
words Y→

GOAL: Find tag sequence X that maximizes probability product

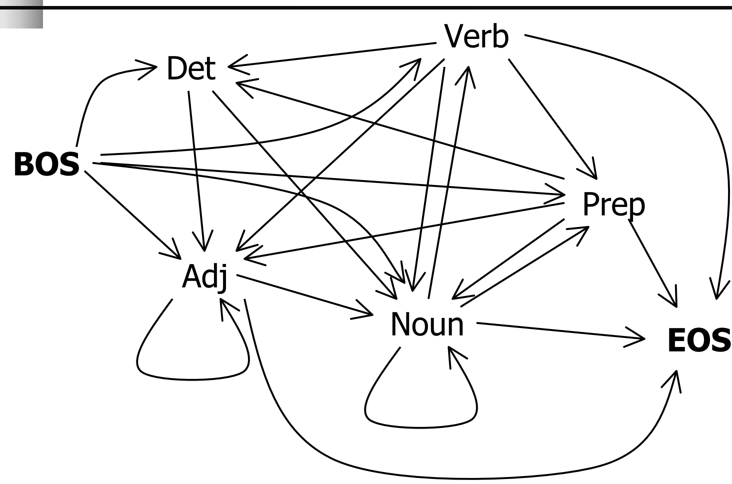
Break apart in 2 stages

- If we were just predicting tags, we could just use bigrams
- We can model this as a Markov process, in particular, an fsa with probabilities on the arcs…

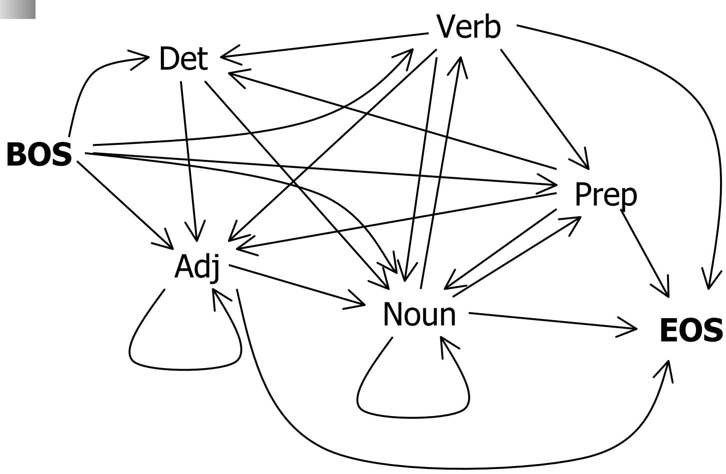
Markov chain…pr of letter sequences



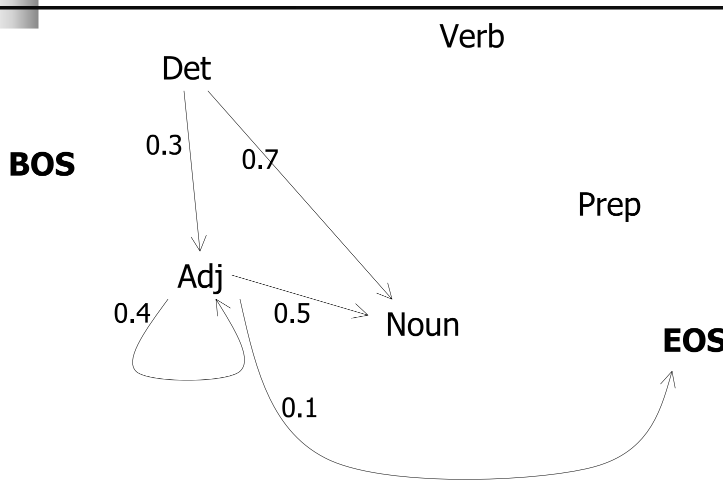
First-order Markov (bigram tag) model as fsa



First-order Markov (bigram tag) model as fsa

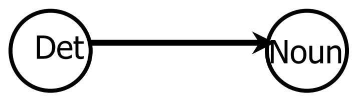


Add in transition probs from training data - sum to 1

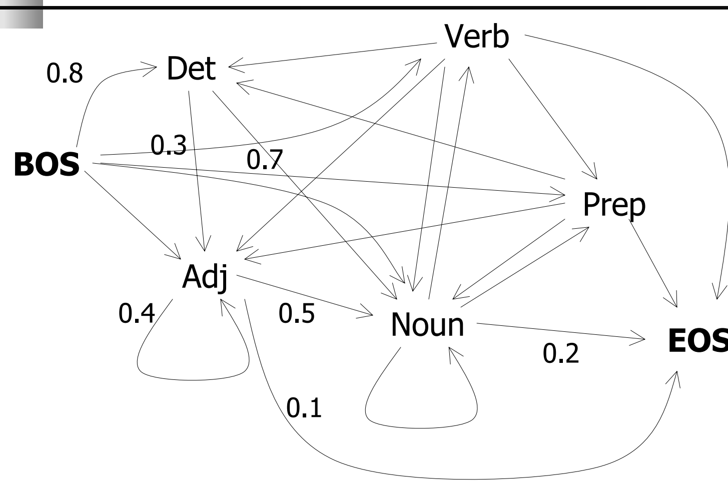


Same as bigram…estimate the same way

P(Noun|Det)=0.7 ≡

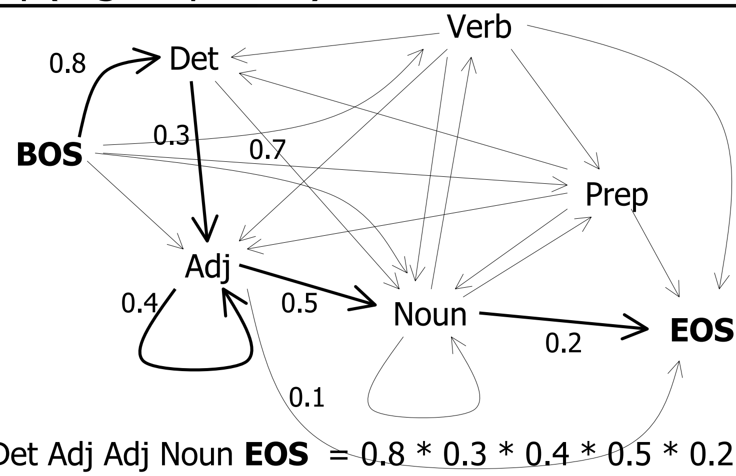


Add in start & etc.



Markov Model – bigram tag sequence

p(tag sequence)

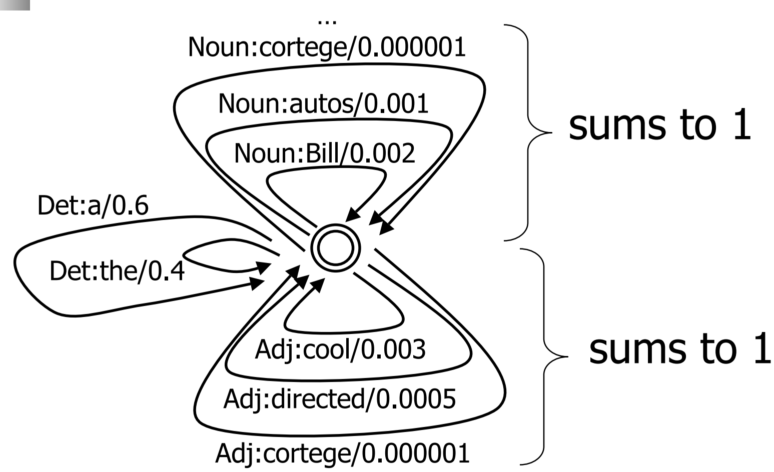


BOS Det Adj Adj Noun **EOS** = 0.8 * 0.3 * 0.4 * 0.5 * 0.2

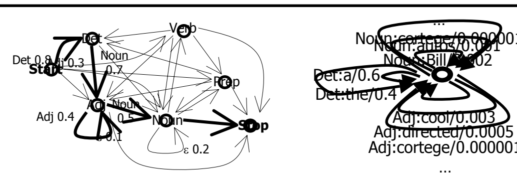
So what?

- We want more!
- We cannot observe the tag sequence –
- But we <u>can</u> estimate P(words | tags)
- Also use an fsa – just unigrams

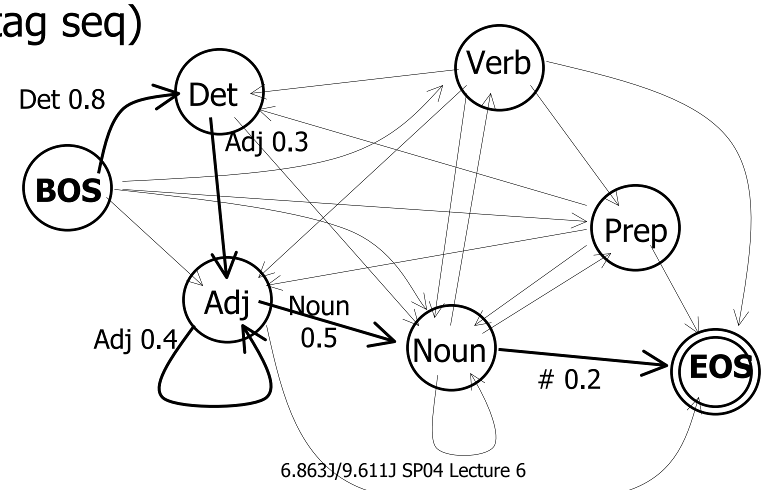
Unigram replacement model
P(word| tag)



Compose



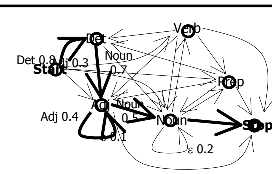
p(tag seq)

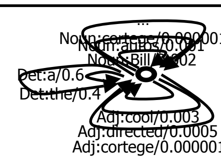


Now we compose (multiply) the nets

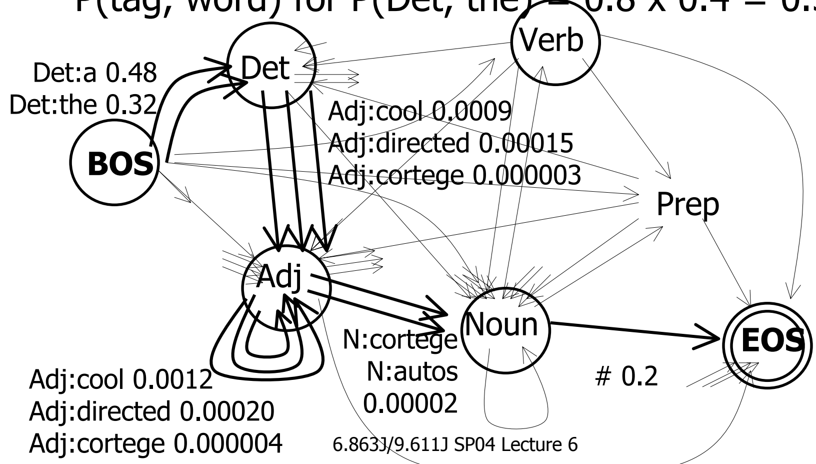
- Compose P(tag sequence) with P(word|tag)
- Result: P(tag, word)

Compose

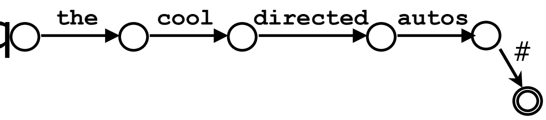




P(tag seq) * P(word seq | tag seq) e.g.,
P(tag, word) for P(Det, the) = 0.8 x 0.4 = 0.32



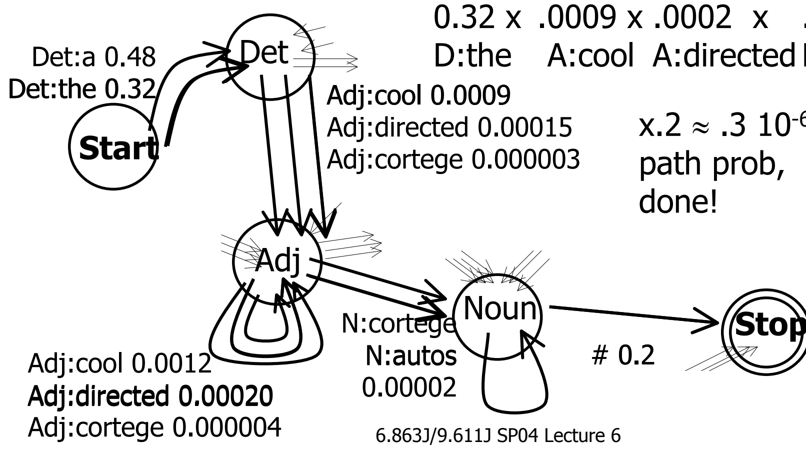
Compose with actual word seq



p(word seq, tag seq) = p(tag seq) * p(word seq | tag seq)

0.32 x .0009 x .0002 x .00002 x
D:the A:cool A:directed N:autos

x.2 ≈ .3 10^{-6} total path prob, done!



Well, we are almost done!

- The Pr of a <u>sequence</u> is just found by multiplying through as we go from start to stop
- <u>Given</u> the actual words in the sentence, trace through and find the <u>highest</u> value Pr – this will give the <u>most likely</u> tag sequence, word sequence combination
- (What have we wrought?)

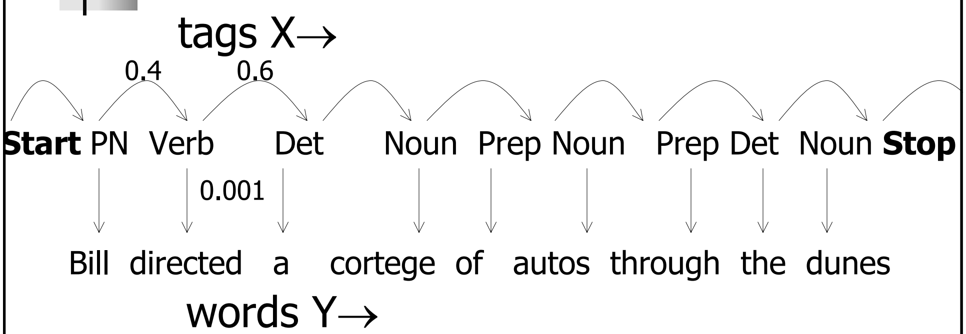
This is an Hidden Markov model for tagging

- Each hidden tag state produces a word in the sentence
- Each word is
  - Uncorrelated with all the other words and their tags
  - Probabilistic depending on the N previous tags only

The statistical view, in short:

- We are modeling p(word seq, tag seq)
- The tags are *hidden*, but we see the words
- Q: What is the most likely tag sequence?
- Use a finite-state automaton, that can emit the observed words
- FSA has limited memory
- Note that given words, in general, there could be more than 1 underlying state sequence corresponding to the words

Punchline – ok, where do the pr numbers come from?

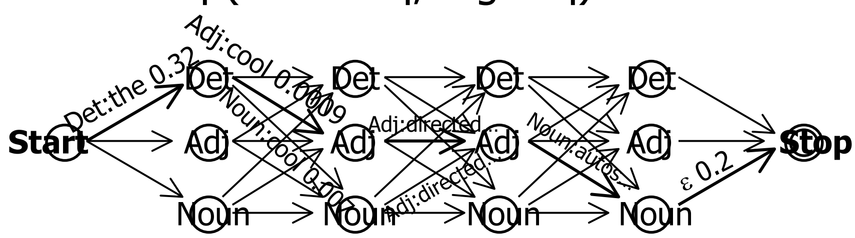


The tags are not observable & they are states of some fsa
We estimate transition probabilities between states
We also have 'emission' pr's from states
En tout: a <u>Hidden Markov Model</u> (HMM)

But…how do we find this 'best' path???

Unroll the fsa - All paths together form 'trellis'

p(word seq, tag seq)



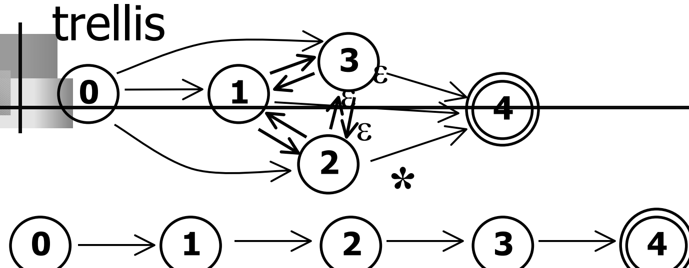
WHY?

The best path:

BOS Det Adj Adj Noun **EOS** = 0.32 * 0.0009 …

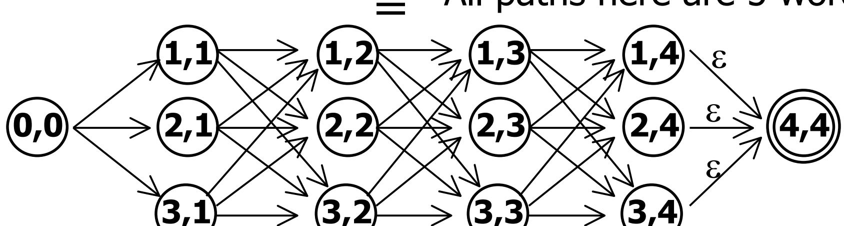
the cool directed autos

Cross-product construction forms trellis



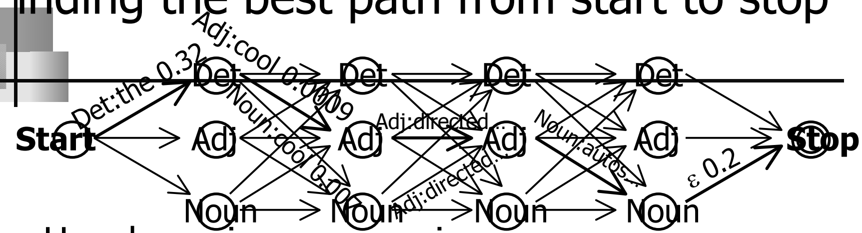
=

All paths here are 5 words



So all paths here must have 5 words on output side

Finding the best path from start to stop



- Use dynamic programming
- What is best path from Start to *each* node?
  - Work from left to right
  - Each node stores its best path from Start (as probability plus one backpointer)
- Special acyclic case of Dijkstra's shortest-path algorithm
- Faster if some arcs/states are absent

Method: Viterbi algorithm

- For each path reaching state s at step (word) t, we compute a path probability. We call the max of these viterbi(s,t)
- [Base step] Compute viterbi(0,0)=1
- [Induction step] Compute viterbi(s',t+1), assuming we know viterbi(s,t) for all s

Viterbi recursion

path-prob(s'|s,t) = viterbi(s,t) * a[s,s']

probability of path to s' through s = max path score for state s at time t * transition p $s \rightarrow s'$

viterbi(s',t+1) = $\max_{s \text{ in STATES}}$ path-prob(s' | s,t)

Method…

- This is *almost* correct…but again, we need to factor in the *unigram* prob of a state s' given an observed surface word *w*
- So the correct formula for the path prob is:
  path-prob(s'|s,t) = viterbi(s,t) * a[s,s'] * $b_{s'}(o_t)$

  bigram unigram

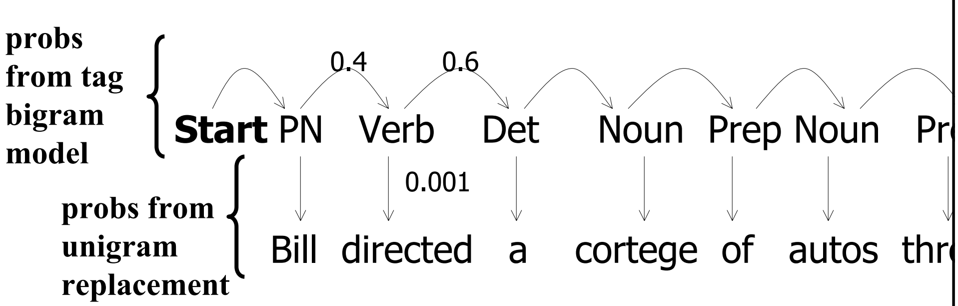
Or as in your text…p. 179

```
function VITERBI(observations of len T,state-graph) returns best-path

  num-states ← NUM-OF-STATES(state-graph)
  Create a path probability matrix viterbi[num-states+2,T+2]
  viterbi[0,0] ← 1.0
  for each time step t from 0 to T do
    for each state s from 0 to num-states do
      for each transition s′ from s specified by state-graph
        new-score ← viterbi[s, t] * a[s,s′] * b_s′(o_t)
        if ((viterbi[s′,t+1] = 0) || (new-score > viterbi[s′, t+1]))
          then
              viterbi[s′, t+1] ← new-score
              back-pointer[s′, t+1] ← s
  Backtrace from highest probability state in the final column of viterbi[] and
return path
```


Summary

- We are modeling p(word seq, tag seq)
- The tags are hidden, but we see the words
- Is tag sequence X likely with these words?
- Model is a “Hidden Markov Model”:



- Find X that maximizes probability product