6.863J Natural Language Processing Lecture 4: From finite state machines to part-of-speech tagging

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

The Menu Bar

- Administrivia:
  - Schedule alert: Lab1 due next *Monday* (Feb 24)
  - Lab 2, handed out Feb 24; due the Weds after this – March 5
- *Agenda:*
- Kimmo – its use and abuse
- Part of speech ‘tagging’ (with sneaky intro to probability theory that we need)
- Ch. 6 & 8 in Jurafsky

What Kimmo is good for

- Ideally: locally, purely concatenative phenomena (obviously, because fsa's)
- FSAs are based *purely* on an associative concatenation operation over strings (i.e., ((a+b)+c) = (a+(b+c)) where =+ concat
- Turkish word: uygarlas,tiramadiklarimizdanmis,sinizcasina =

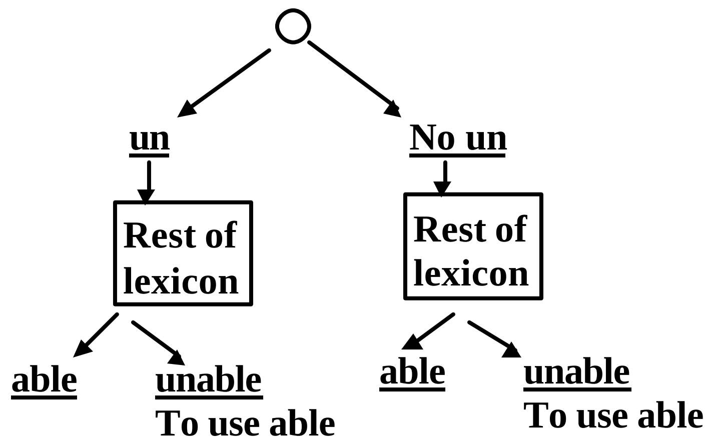
  uygar+las,+tir+ama+dik+lar+imiz+dan+mis,+siniz+casina

(behaving) as if you are among those whom we could not cause to become civilized

What Kimmo is not good for

- So, this lets us think what the system *might not* be good for... let's look at English first....
- There seem to be some kinds of 'long distance' constraints...
- Prefix/suffix links: only some prefixes tied to some suffixes
  - Un---------able
  - Undoable, uncanny, ?uncannyable, unthinkable, thinkable, readable, unreadable, unkind, *unkindable
- So, we have to 'keep track' that the *un* is first or *not* – what does lexicon look like?

Lexicon must be (grotesquely) duplicated



This kind of duplication is a litmus test of something wrong

- Duplication: no relation between the two lexicons, but we know they're identical
- Principle *AWP*
- We will see this again and again
- Usually means we haven't carved (factored) the knowledge at the right 'joints'
- Solution? Usually more powerful machinery 'overlay' representations

Not *all* long distance effects are a barrier...

- Phenomena: *Vowel harmony*
  - yourgun + sInIz → yorgunsunuz
  - Round vowels assimilate to round vowels; back vowels to back, etc. - all the way from left to right
- Can Kimmo do it? What would be your model?

Parsing words with Kimmo is computationally intractable

- Intuition: what if the characters on the surface don't give any clues as to what 'features' they ought to have underlyingly? (e.g., whether a Noun or a Verb, as in *police police police*)
- This seems awfully close to the famous 3-SAT problem: is there an assignment of T(rue), F(alse) to the literals of an arbitray Boolean formula in 3-conjunctive normal form s.t. the formula evaluates to *true*?
- In fact, we can simulate this problem using Kimmo

3-Sat

- *Given* (arb) cnf formula, e.g.,

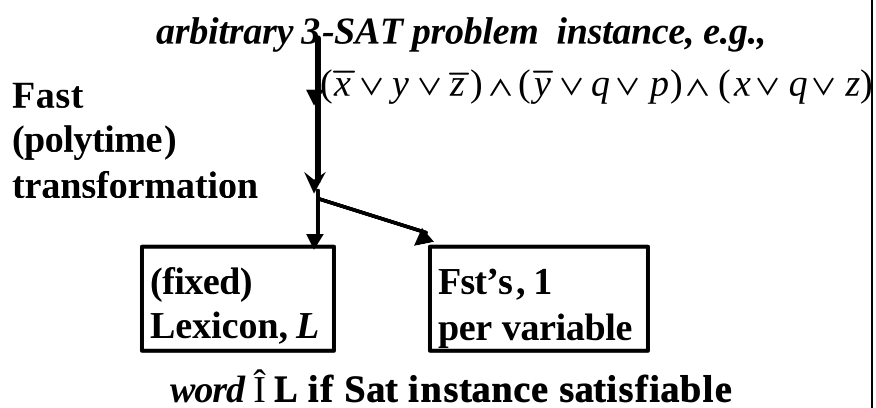
$$(\overline{x} \vee y \vee \overline{z}) \wedge (\overline{y} \vee q \vee p) \wedge (x \vee q \vee z)$$

- We can't figure out quickly (in deterministic polynomial time) whether there is an assignment of *true* or *false* to literals *x,y, z* in order to make the formula eval to true <u>just</u> by inspecting the local surface string
- We could guess this in polynomial time – i.e., Nondeterministic Polynomial, or NP time (time measured in length of the formula)

Reduction of 3-Sat to Kimmo recognition problem

- For every 3-Sat problem, we can find (in poly time) a corresponding Kimmo word recognition problem where there's a valid word if the 3-Sat problem was satisfiable
- If Kimmo recognition could be done in det poly time (P) then so could 3-SAT

The reduction



If **we could solve Kimmo recognition easily,**
Then **we could solve 3-Sat easily**

Why should we care?

- This is *typical* of a combination of 'agreement and ambiguity' that trickles through all of natural language
- The agreement part – like Turkish vowel harmony
- The ambiguity part – like the *police police police* example
- Suggests that speed won't come from the formalism <u>*all by itself*</u>

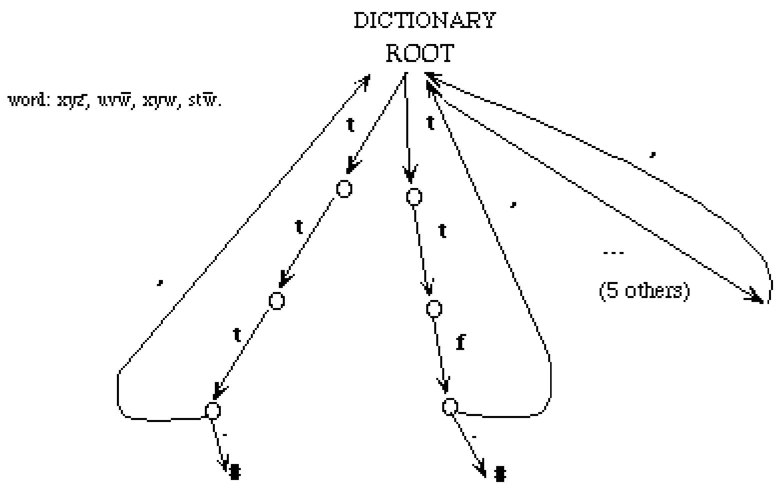
Two components to 3-Sat

- The fact that an *x* that has a truth assignment in one place, must have the same truth assignment everywhere - what morphological process is that like?
- The fact that every triple must have at least 1 ‘T’ underlyingly (so that the triple is true) - what morphological process is that like?

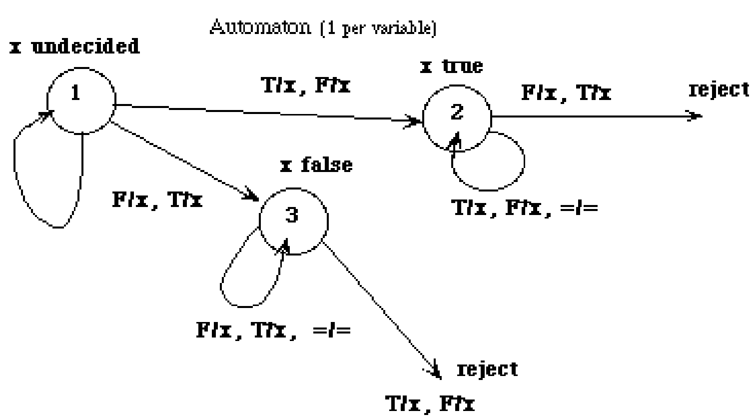
Two components

- Agreement: vowel harmony (if round at some point, round everywhere)
- Ambiguity: we can’t tell what the underlying value of *x* is from the surface, but if there’s at least one “t” per ‘part of word’, then we can spell out this constraint in dictionary
- Note that words (like Sat formulas) must be arbitrarily long... (pas de probleme)
- Dictionary is fixed...
- # of Vowel harmony processes corresponds to # of distinct literals

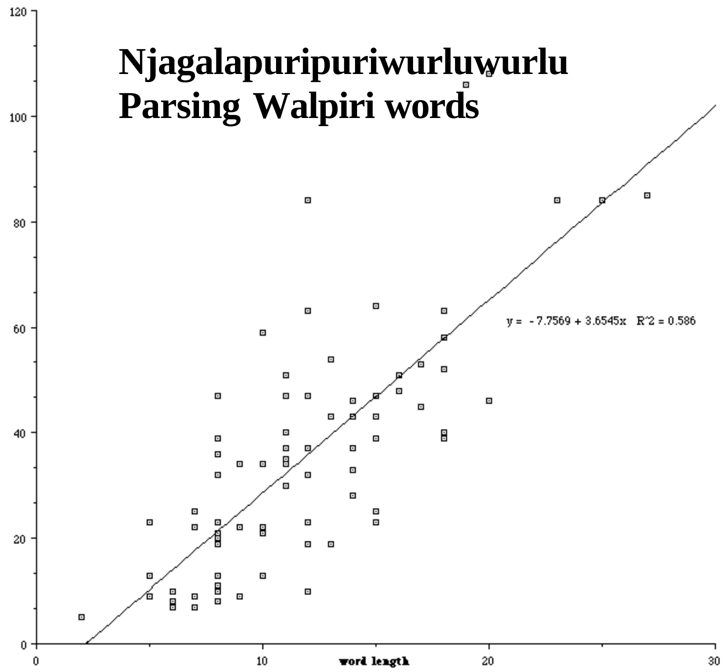
Reduce until done – formula must eval to *true*



Reduce until done: assignment consistency



Njagalapuripuriwurluwurlu
Parsing Walpiri words



Then can be *indescribable* words (for an fst)

- Can we even do all natural languages?
- Example: Bambarra (African language in Mali)
- Words in form *Noun+o+Noun,* as in *wuluowulo* =‘whichever dog’
- Also have repeated endings (like anti-anti…)
  wulu+nyini+la =‘dog searcher’
  wulunyinina+ nyini+la =‘one who searches for dog searchers’
- Fatal bite: combine with word *o* formation:
  wulunyininanyinila o wulunyininanyinila (arbitrarily long!)

Paradigmatic example for NLP

- Morphophonemic parsing
- Given surface form, recover underlying form:

  morpho-phonem-ic

Two ways

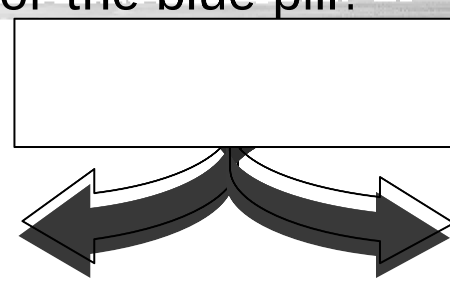
- Generative model – concatenate then fix up joints
  - stop + -ing = stopping,   fly + s = flies
  - Use a cascade of transducers to handle all the fixups
- Probabilistic model - some constraints on morpheme sequences using prob of one character appearing before/after another

  prob(ing | stop) vs. prob(ly| stop)
- (much more about prob in just one moment)

Two ways of looking at language & the Great Divide

- Text understanding vs. Information Retrieval (IR)
- Info retrieval example: name extraction; how does Google correct “Britney Speers”

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”

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

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 ./.

(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] ./.

What's it good for?

- Tags = parts-of-speech (but see later)
- Uses:
  - text-to-speech (how do we pronounce "lead"?)
  - can write regexps like **`Det Adj* N*`** over the output
  - preprocessing to speed up parser (but a little dangerous)
  - if you know the tag, you can back off to it in other tasks
  - Back-off: trim the info you know at that point

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 <u>books</u> the violin concert tickets early.
    - books → {(book-1,Noun,Pl,-,-),(book-2,Verb,Sg,Pres,3)}
    - tagging (disambiguation): ... → (Verb,Sg,Pres,3)
  - ...was pretty good. <u>However</u>, she did not realize...
    - However → {(however-1,Conj/coord,-,-,-),
      (however- 2,Adv,-,-,-)}
    - tagging: ... → (Conj/coord,-,-,-)

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!
- Sneaky: Introduce probabilistic models – paradigmatic contrast investigated in Lab 2.

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 X likely with these words?

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 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 performance Homer Simpson algorithm:
    - Tag every word with its most frequent tag
    - 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 **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 ...

Ok, what *should* we look at?

correct tags

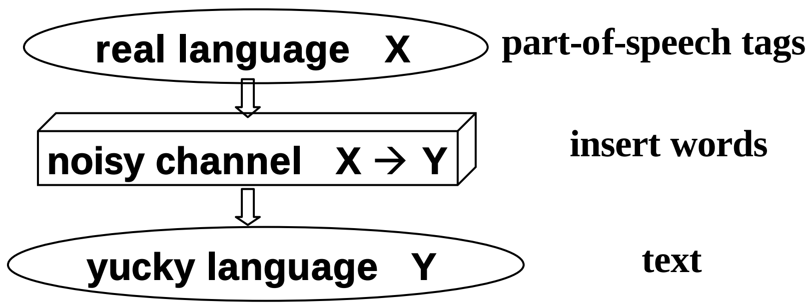
PN Verb Det Noun Prep Noun Prep Det Noun
Bill directed a cortege of autos through the dunes
PN Adj Det Noun Prep Noun Prep Det Noun
Verb Verb Noun Verb
Adj
Prep
...?

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 ...

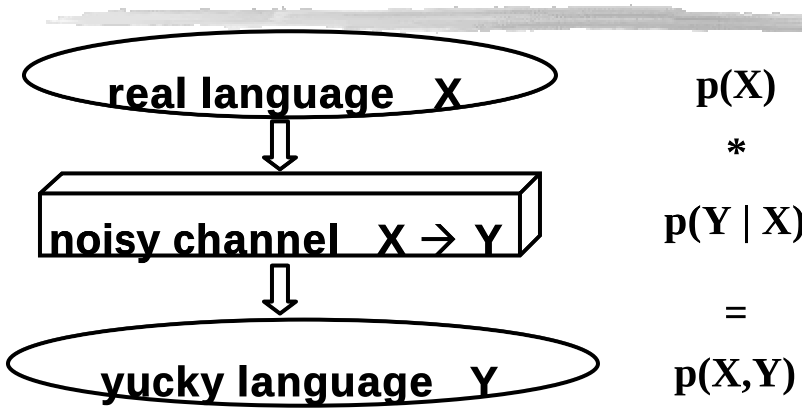
Finite-state approaches

- Noishy Chunnel Muddle (statistical)



want to recover X from Y

Noisy channel – and prob intro

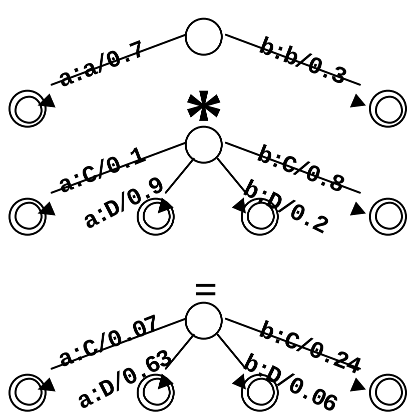


choose sequence of tags X that *maximizes* p(X | Y)
[oops… this isn't *quite* correct… need 1 more step]

Noisy channel maps well to our fsa/fst notions

- What's p(X)?
- Ans: p(tag sequence) – i.e., some finite state automaton
- What's p(Y|X)?
- Ans: transducer that takes tags→words
- What's P(X,Y)?
- The joint probability of the tag sequence, given the words (well, gulp, almost... we will need one more twist – why? What is Y?)

The plan modeled as composition (x-product) of finite-state machines



p(X)

*****

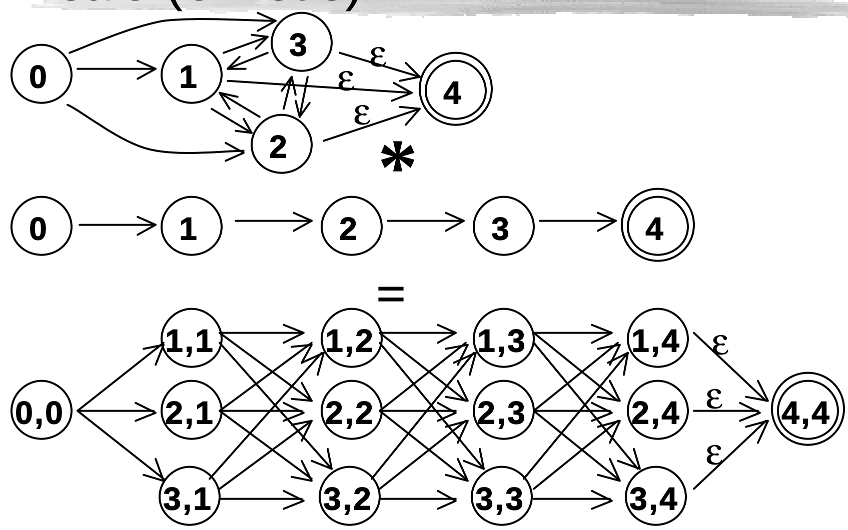
p(Y | X)

=

p(X,Y)

Note p(x,y) sums to 1.

Cross-product construction for fsa's (or fst's)



Pulled a bit of a fast one here...

- So far, we have a plan to compute P(X,Y) – but is this correct?
- Y= all the words in the world
- X= all the tags in the world (well, for English)
- What we get to see as input is **y Î Y** *not* Y!
- What we want to compute is REALLY this:

  want to recover xÎ X from yÎ Y
  choose x that <u>maximizes</u> p(X | y) so...

The real plan...

$$p(X)$$

$$*$$

$$p(Y \mid X)$$

$$*$$

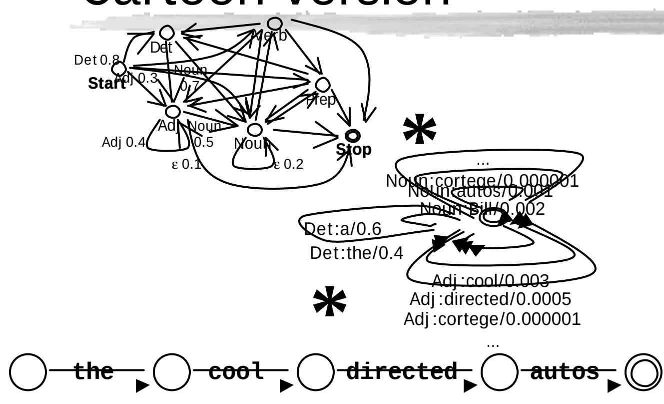
$$p(y \mid Y)$$

$$=$$

$$p(X, y)$$

Find x that maximizes this quantity

Cartoon version



$$p(X)$$

$$*$$

$$p(Y \mid X)$$

$$*$$

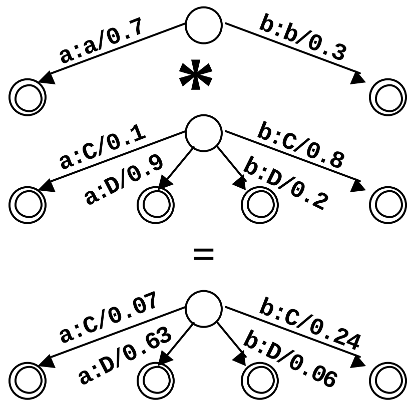
$$p(y \mid Y)$$

$$=$$

$$p(X, y)$$

transducer: scores candidate tag seqs on their joint probability with obs words; we should pick best path

The plan modeled as composition (product) of finite-state machines



p(X)

*****

p(Y | X)

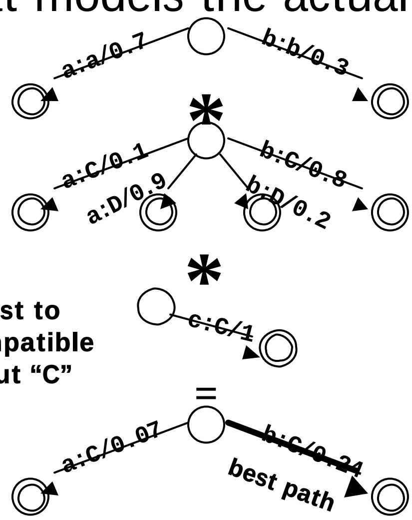
=

p(X,Y)

Note p(x,y) sums to 1.
Suppose y="C"; what is best "x"?

We need to factor in one more machine that models the actual word sequence, y

restrict just to paths compatible with output "C"



p(X)

*****

p(Y | X)

*****

p(y | Y)

find x to =
maximize p(X, y)

The statistical view, in short:

- We are modeling p(word seq, tag seq)
- The tags are *hidden*, but we see the words
- What is the most likely tag sequence?
- Use a finite-state automaton, that can emit the observed words
- FSA has limited memory
- AKA this Noisy channel model is a "<u>Hidden Markov Model</u>" -

Put the punchline before the joke

Bill directed a cortege of autos through the dunes

Recover tags

Punchline – recovering (words, tags)

tags X→

Start PN Verb Det Noun Prep Noun Prep Det Noun **Stop**

Bill directed a cortege of autos through the dunes

words Y→

Find tag sequence X that maximizes probability product

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

tags X→

0.4 0.6

Start PN Verb Det Noun Prep Noun Prep Det Noun **Stop**

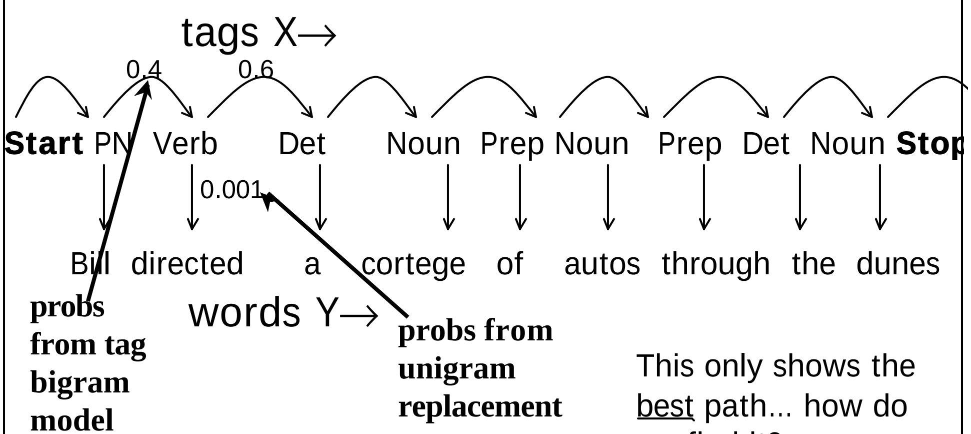
0.001

Bill directed a cortege of autos through the dunes

words Y→

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)

Our model uses <u>both</u> *bigrams* & *unigrams*:



This only shows the <u>best</u> path... how do we find it?

What are unigrams and bigrams?

- Letter or word frequencies: 1-grams
  - useful in solving cryptograms: ETAOINSHRDLU...
- If you know the previous letter: 2-grams
  - “h” is rare in English (4%; 4 points in Scrabble)
  - but “h” is common after “t” (20%)
- If you know the previous 2 letters: 3-grams
  - “h” is really common after “ ” “t”
    etc. ...

In our case

- Most likely word? Most likely tag *t* given a word *w?* = P(tag|word)
- Task of predicting the next word
- Woody Allen:

  "I have a gub"

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

Homer Simpson: just use the current word (don't look at context) = unigram (1-gram)

How far should we go?

- "*long distance*___"
- Next word? *Call?*
- $p(w_n|w$
- Consider special case above
- Approximation says that

  | long distance call|/|distance call| ≈ |distance call|/|distance|
- 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*_____

3-gram

[Genmetheyesse orils of Ted you doorder [6], the Grily Capiduatent pildred and For thy werarme: nomiterst halt i, what production the Covers, in calt cations on wile ars, was name conch rom the exce of the man, Winetwentagaint up, and and Al1. And of Ther so i hundal panite days th the res of th rand ung into the forD six es, wheralf the hie soulsee, frelatche rigat. And the LOperact camen unismelight fammedied: and nople,

4-gram

[1] By the returall benefit han every familitant of all thou go? And At the eld to parises of the nursed by thy way of all histantly be the ~aciedfag . to the narre gread abrasa of thing, and vas these conwuning clann com to one language; all Lah, which for the greath othey die. -

5-gram

```
[Gen 3:1] In the called up history of its opposition of
bourgeOIS AND Adam to rest, that the existing of heaven; and
land the bourgeoiS ANger anything but concealed, the land
whethere had doth know ther: bury thy didst of Terature their
faces which went masses the old society [2] is the breaks out
of oppressor of all which, the prolETARiat goest, unto German
pleast twelves applied in manner with these, first of this
polities have all
```

3-word-gram

```
[Gen 4:25] And Adam gave naines to ail feudal,
patriarchal, idyllic relations. It bas but –established
new classes, new conditions of oppression, new forme of
struggle in place of the West? The bourgeoisie keeps
more and more splitting up into two great lights;
the greater light to rule the day of my house is this
Eliezer of Damascus.
```

How far can we go??

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
- Use *backoff* and *smoothing*
- So we can't go very far...

Where do these probability info estimates come from?

- Use tagged corpus e.g. "Brown corpus" 1M words (fewer token *instances*); many others – Celex 16M words
- Use counts (relative frequencies) as estimates for probabilities (various issues w/ this, these so-called Maximum-Likelihood estimates – don't work well for low numbers)
- Train on texts to get estimates – use on new texts

Bigrams, fsa's, and Markov models – take two

- We *approximate* p(tag| all previous tags) Instead of

  p(rabbit|Just then the white...) we use:

  P(rabbit|white)
- This is a *Markov assumption* where past memory is limited to immediately previous state – just 1 state corresponding to the previous word or tag

Smoothing

- We don't see many of the words in English (uniqram)
- We don't see the huge majority of bigrams in English
- We see only a tiny sliver of the possible trigrams
- So: most of the time, bigram model assigns p(0) to bigram:

  p(food|want) = |want food| /|want| = 0/whatever

But means event can't happen – we aren't warranted to conclude this... therefore, we must adjust...how?

Simplest idea: add-1 smoothing

- Add 1 to every cell of
- P(food | want) = |want to| ÷ |want| = 1 ÷ 2931 = .0003

Initial counts – Berkeley restaurant project

	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

	I	want	to	eat	Chinese	food	lunch
I	.0023	.32	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	.0049
to	.00092	0	.0031	.26	.00092	0	.0037
eat	0	0	.0021	0	.020	.0021	.055
Chinese	.0094	0	0	0	0	.56	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

Old vs.New table

	I	want	to	eat	Chinese	food	lunch
I	.0023	.32	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	.0049
to	.00092	0	.0031	.26	.00092	0	.0037
eat	0	0	.0021	0	.020	.0021	.055
Chinese	.0094	0	0	0	0	.56	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

	I	want	to	eat	Chinese	food	lunch
I	.0018	.22	.00020	.0028	.00020	.00020	.00020
want	.0014	.00035	.28	.00035	.0025	.0032	.0025
to	.00082	.00021	.0023	.18	.00082	.00021	.0027
eat	.00039	.00039	.0012	.00039	.0078	.0012	.021
Chinese	.0016	.00055	.00055	.00055	.00055	.066	.0011
food	.0064	.00032	.0058	.00032	.00032	.00032	.00032
lunch	.0024	.00048	.00048	.00048	.00048	.00096	.00048

Changes

- All non-zero probs went *down*
- Sometimes probs don’t change much
- Some predictable events become less predictable (P(to|want) [0.65 to 0.22])
- Other probs change by large factors (P(lunch|Chinese) [0.0047 to 0.001]
- Conclusion: generally good idea, but effect on nonzeroes not always good – blur original model – too much prob to the zeros, we want less ‘weight’ assigned to them (zero-sum game, ‘cause probs always sum to 0)

Submenu for probability theory – redo n-grams a bit more formally

- Define all this p(X), p(Y|X), P(X,Y) notation
- p, event space, conditional probability & chain rule;
- Bayes' Law
- (Eventually) how do we estimate all these probabilities from (limited) text? (Backoff & Smoothing)

Rush intro to probability

p(Paul Revere wins | weather's clear) = 0.9

What's this mean?

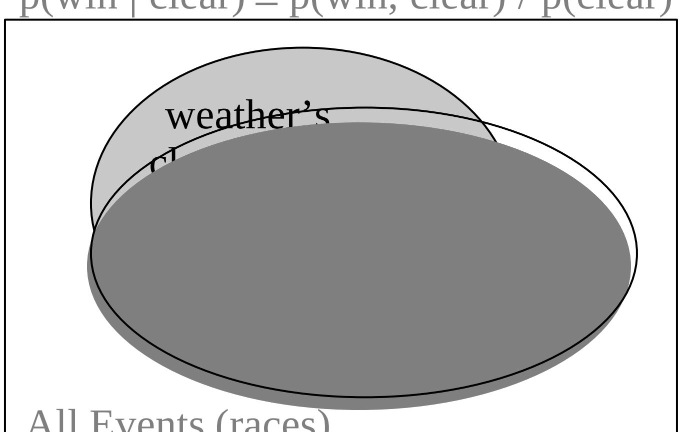
p(Paul Revere wins | weather's clear) = 0.9

- Past performance?
  - Revere's won 90% of races with clear weather
- Hypothetical performance?
  - If he ran the race in many parallel universes …
- Subjective strength of belief?
  - Would pay up to 90 cents for chance to win $1
- Output of some computable formula?
  - But then which formulas should we trust?

    p(X | Y) versus q(X | Y)

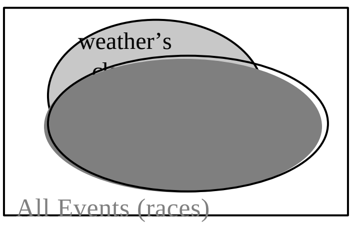
p is a function on event sets

p(win | clear) ≡ p(win, clear) / p(clear)



p is a function on event sets

$$\underbrace{p(\text{win} \mid \text{clear})}_{\text{syntactic sugar}} \equiv p(\underbrace{\text{win, clear}}_{\text{logical conjunction of predicates}}) \;/\; p(\underbrace{\text{clear}}_{\text{predicate selecting races where weather's clear}})$$



p measures total probability of a set of events.

Commas in p(x,y) mean conjunction – on the left…

p(Paul Revere wins, Valentine places, Epitaph shows | weather's clear)

what happens as we add conjuncts to <u>left</u> of bar ?

- probability can only decrease
- numerator of historical estimate likely to go to zero:

$$\frac{\text{\# times Revere wins AND Val places... AND weather's clear}}{\text{\# times weather's clear}}$$

Commas in p(x,y)...on the right

p(Paul Revere wins | weather’s clear, ground is dry, jockey getting over sprain, Epitaph also in race, Epitaph was recently bought by Gonzalez, race is on May 17, ...)

what happens as we add conjuncts to right of bar ?

- probability could increase or decrease
- probability gets more relevant to our case (less ***bias***)
- probability *estimate* gets less reliable (more ***variance*)**

$$\frac{\text{\# times Revere wins AND weather clear AND … it’s May 17}}{\text{\# times weather clear AND … it’s May 17}}$$

Backing off: simplifying the right-hand side...

p(Paul Revere wins | weather’s clear, ~~ground is dry, jockey getting over sprain, Epitaph~~ ~~also in race, Epitaph was recently bought by Gonzalez,~~ ~~race is on May 17~~, ...)

not exactly what we want but at least we can get a reasonable estimate of it!

try to *keep* the conditions that we suspect will have the most influence on whether Paul Revere wins

Recall ‘backing off’ in using just p(rabbit|white) instead of p(rabbit|Just then a white) – so this is a general method

What about simplifying the left-hand side?

p(Paul Revere wins~~, Valentine places,~~
~~Epitaph shows~~ | weather's clear)

NOT ALLOWED!
but we can do something similar to help …
We can FACTOR this information – the so-called
"Chain Rule"

Chain rule: factoring lhs

p(Revere, Valentine, Epitaph | weather's clear)	RVEW/W
= p(Revere | Valentine, Epitaph, weather's clear)	= RVEW/VEW
* p(Valentine | Epitaph, weather's clear)	* VEW/EW
* p(Epitaph | weather's clear)	* EW/W

True because numerators cancel against denominators
Makes perfect sense when read from bottom to top
Moves material to right of bar so it can be ignored

If this prob is unchanged by backoff, we say Revere was CONDITIONALLY INDEPENDENT of Valentine and Epitaph (conditioned on the weather's being clear). Often we just ASSUME conditional independence to get the nice product above.

The plan: summary so far

automaton: p(tag sequence) p(X)

* *

"Markov Model"

transducer: tags → words p(Y | X)

"Unigram Replacement"

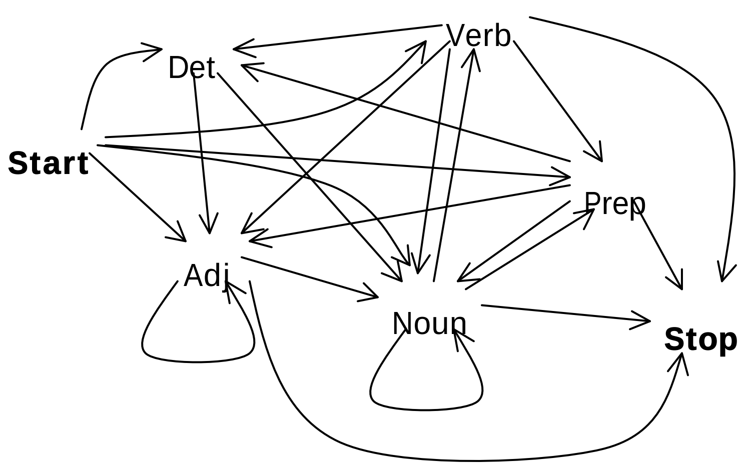
* *

automaton: the observed words p(y | Y)

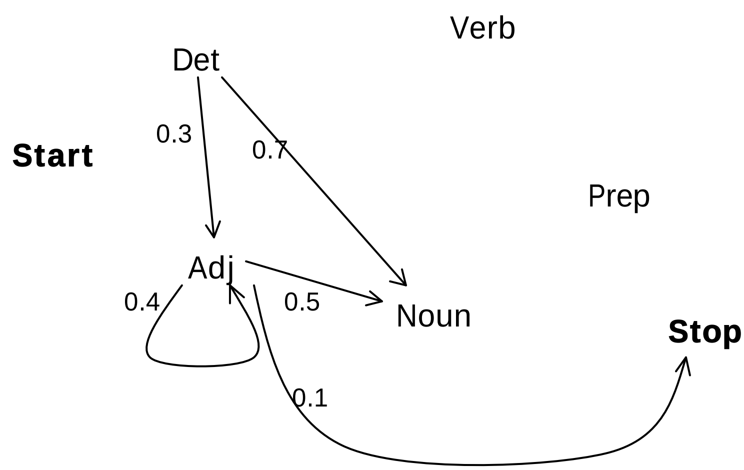
"straight line" = =

transducer: scores candidate tag seqs on their joint probability with obs words; pick best path p(X, y)

First-order Markov (bigram) model as fsa



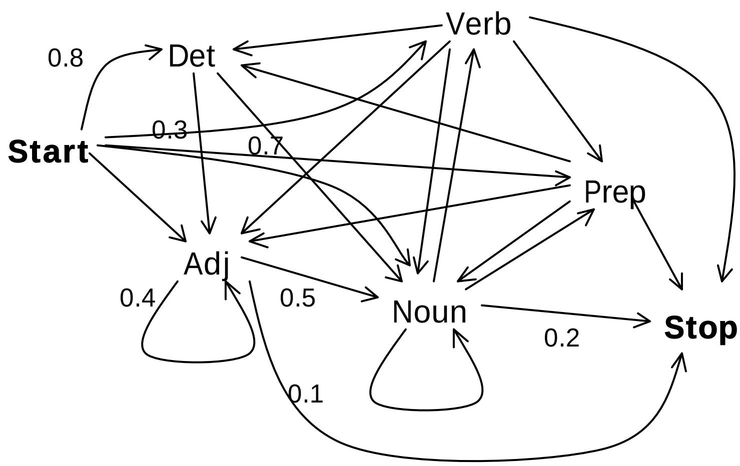
Add in transition probs - sum to 1



Same as bigram

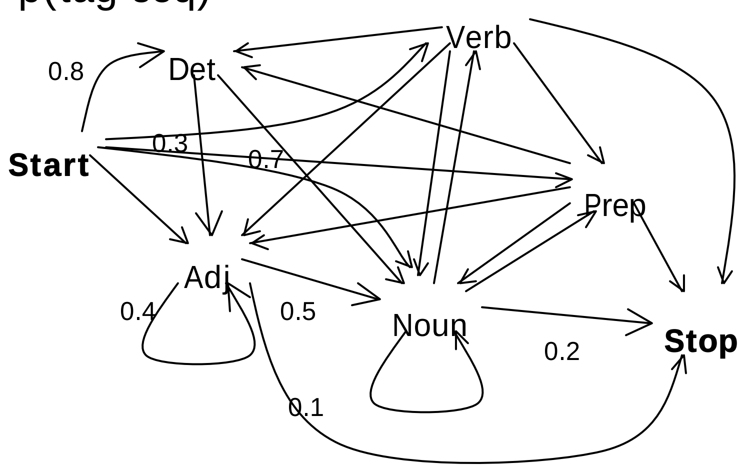
P(Noun|Det)=0.7 ≡

Add in start & etc.



Markov Model

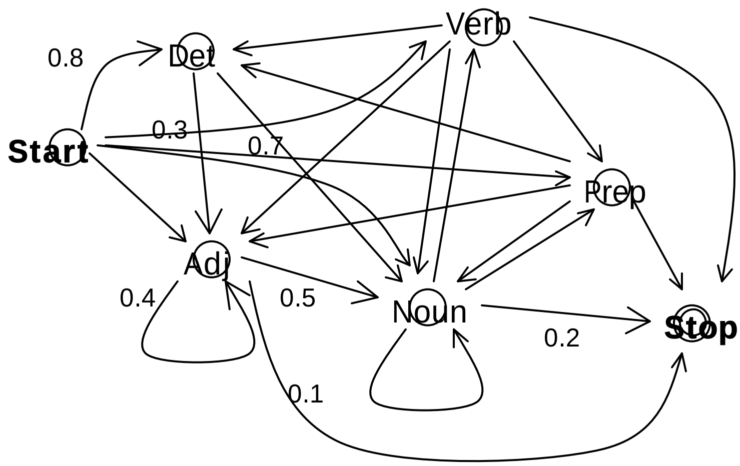
p(tag seq)



Start Det Adj Adj Noun **Stop** = 0.8 * 0.3 * 0.4 * 0.5 * 0.2

Markov model as fsa

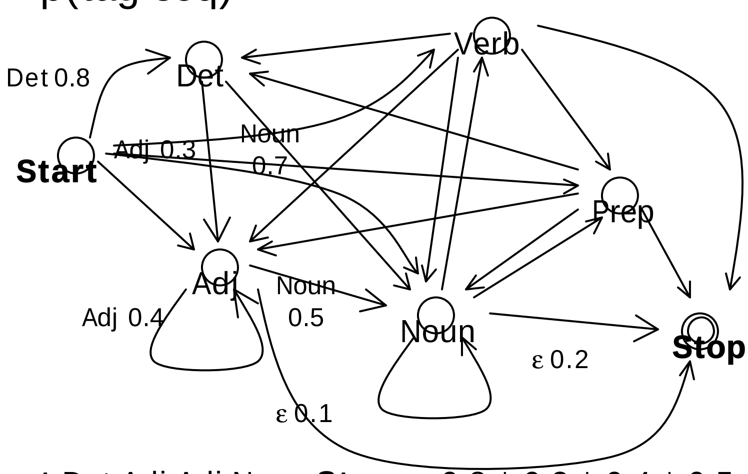
p(tag seq)



Start Det Adj Adj Noun **Stop** = 0.8 * 0.3 * 0.4 * 0.5 * 0.2

Add 'output tags' (transducer)

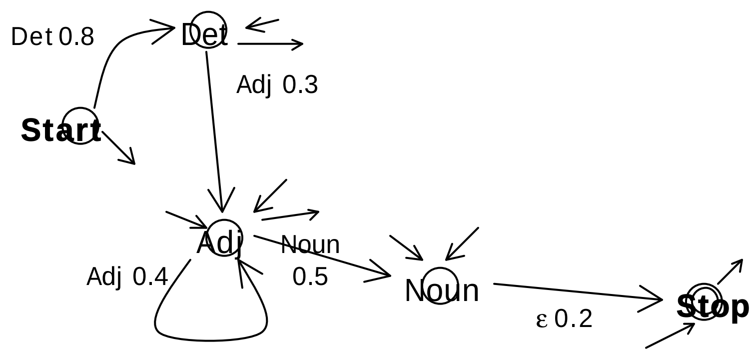
p(tag seq)



Start Det Adj Adj Noun **Stop** = 0.8 * 0.3 * 0.4 * 0.5 * 0.2

Tag bigram picture

p(tag seq)



Start Det Adj Adj Noun **Stop** = 0.8 * 0.3 * 0.4 * 0.5 * 0.2

Our plan

automaton: p(tag sequence)

"Markov Model"

*****

transducer: tags → words

"Unigram Replacement"

*****

automaton: the observed words

"straight line"

=

transducer: scores candidate tag seqs on their joint probability with obs words; pick best path

p(X)

*****

p(Y | X)

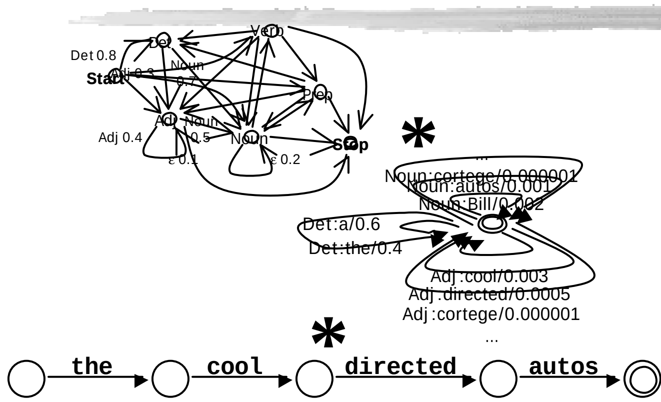
*****

p(y | Y)

=

p(X, y)

Cartoon form again



p(X)

*

p(Y | X)

*

p(y | Y)

=

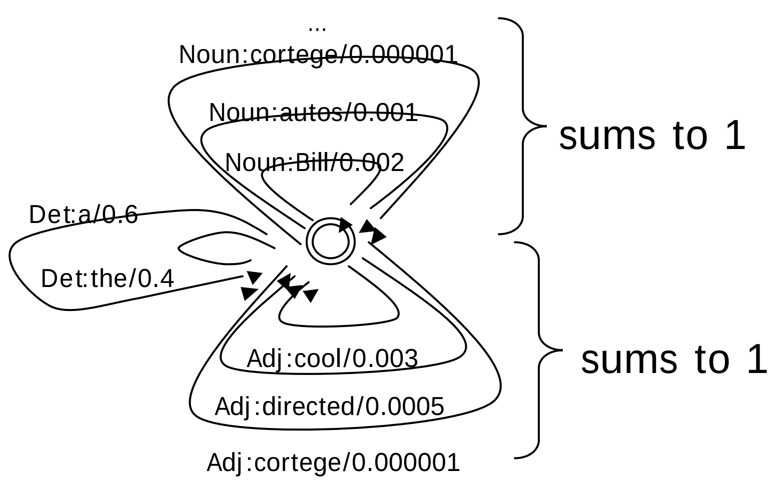
=

p(X, y)

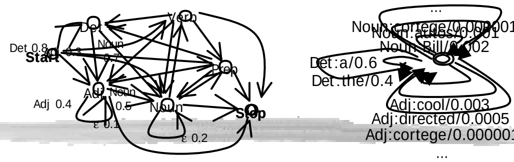
transducer: scores candidate tag seqs on their joint probability with obs words; we should pick best path

Next up: unigram replacement model

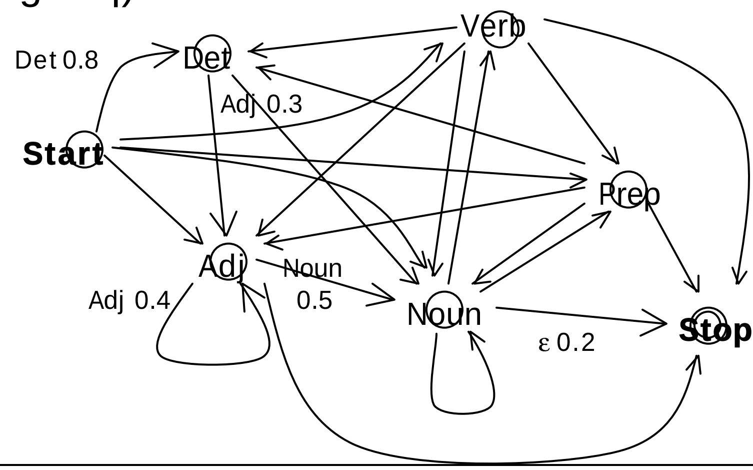
p(word seq | tag seq)



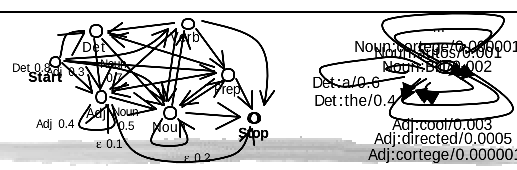
Compose



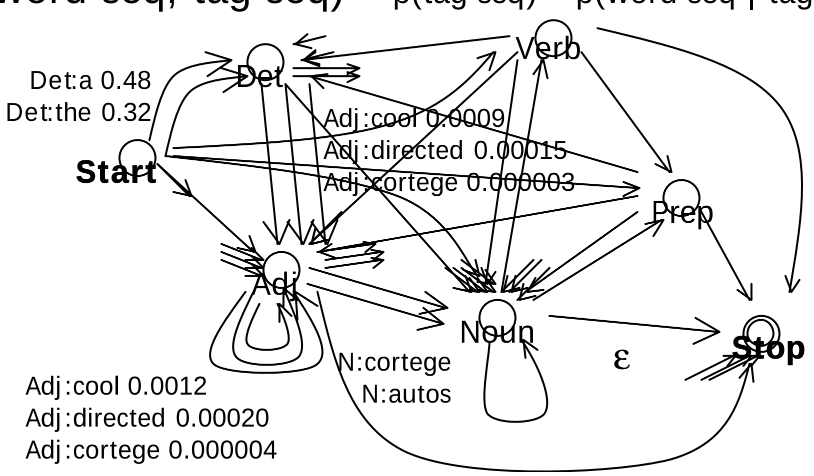
p(tag seq)



Compose

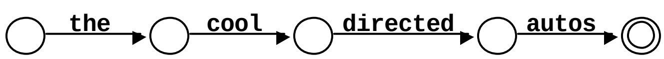


p(word seq, tag seq) = p(tag seq) * p(word seq | tag seq)



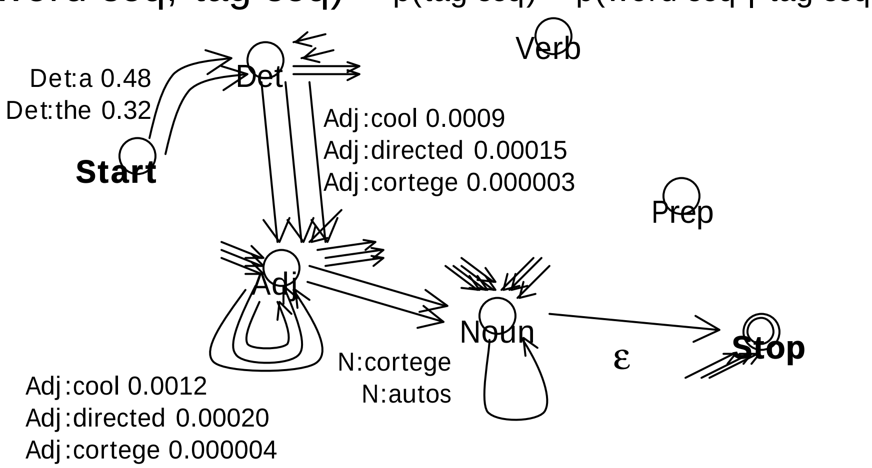
Observed words as straight-line fsa

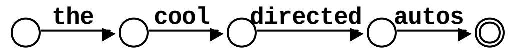
word seq



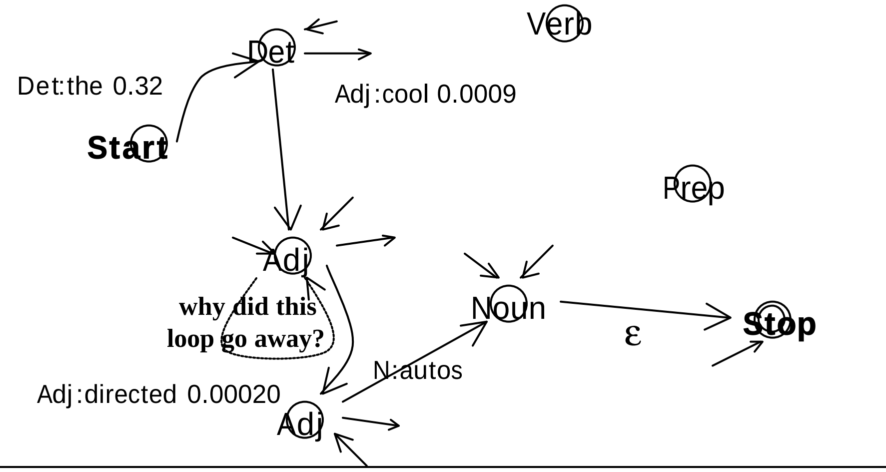
Compose with 

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



Compose with 

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

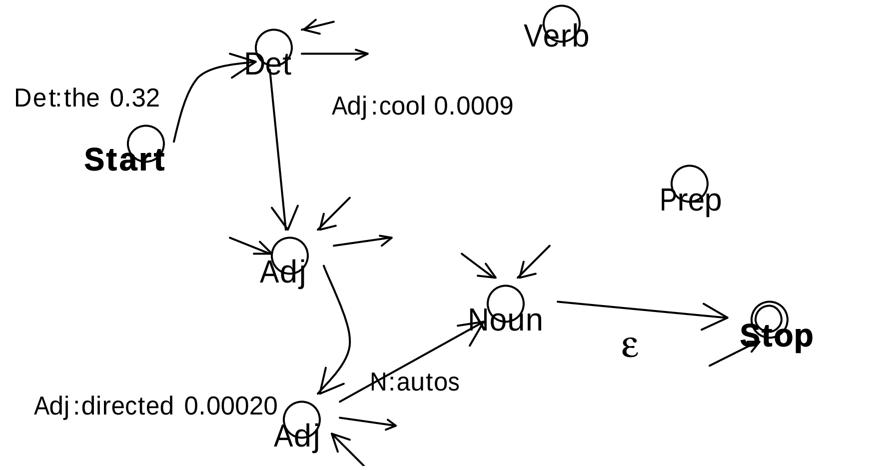


The best path:

Start Det Adj Adj Noun **Stop** = 0.32 * 0.0009*0.00020...

the cool directed autos

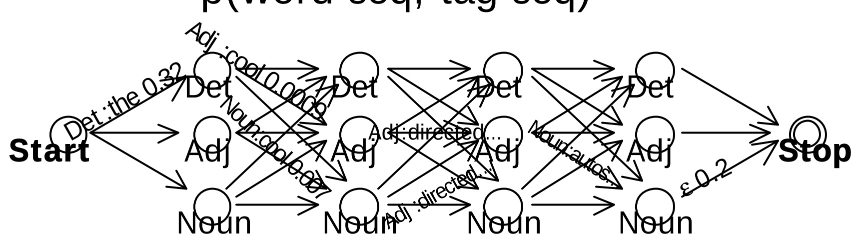
p(word seq, tag seq) = p(tag seq) * p(word seq | tag seq)



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

All paths together form 'trellis'

p(word seq, tag seq)

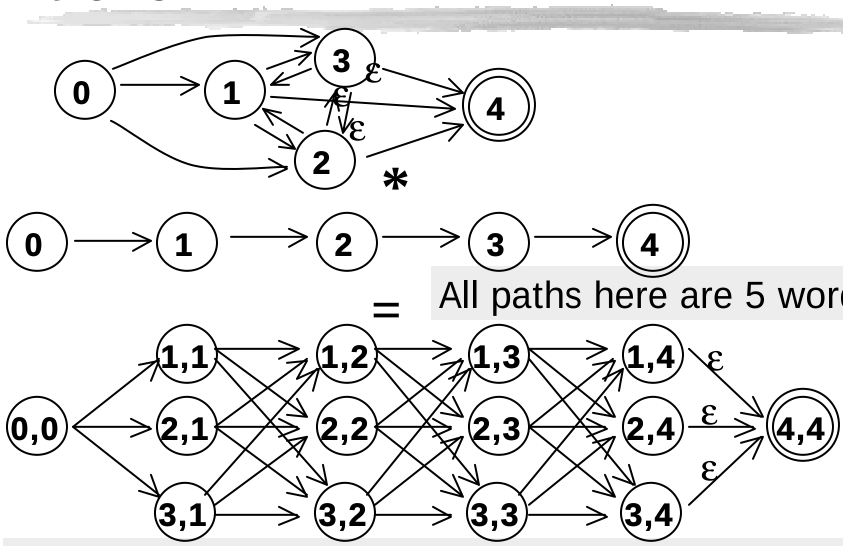


WHY?

The best path:

Start Det Adj Adj Noun **Stop** = 0.32 * 0.0009 ...
the cool directed autos

Cross-product construction forms trellis



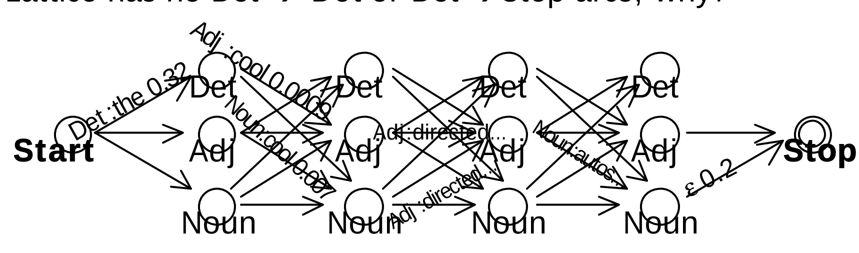
All paths here are 5 words

So all paths here must have 5 words on output side

Trellis isn’t complete

p(word seq, tag seq)

Lattice has no Det → Det or Det →Stop arcs; why?



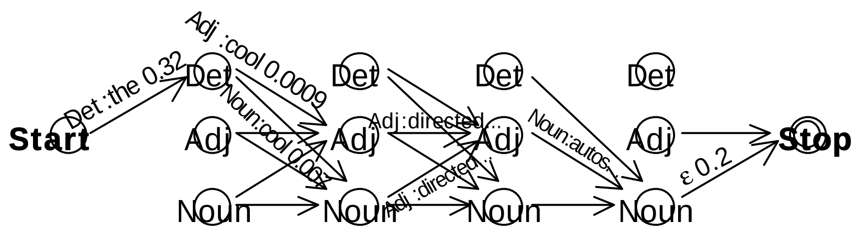
The best path:

Start Det Adj Adj Noun **Stop** = 0.32 * 0.0009 ...
the cool directed autos

Trellis incomplete

p(word seq, tag seq)

Lattice is missing some other arcs; why?



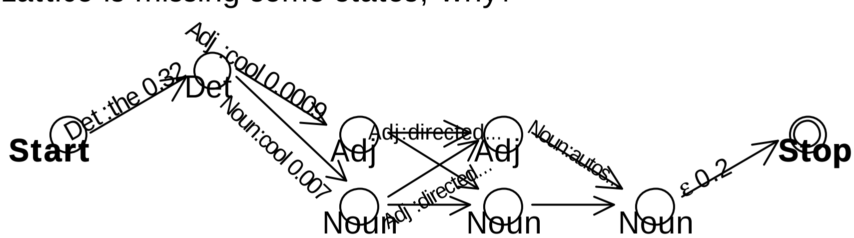
The best path:

Start Det Adj Adj Noun **Stop** = 0.32 * 0.0009 ...
the cool directed autos

And missing some states...

p(word seq, tag seq)

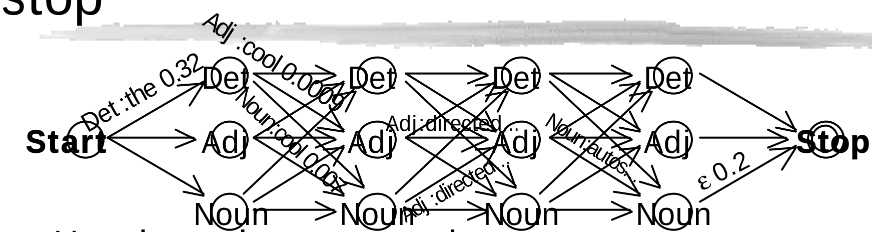
Lattice is missing some states; why?



The best path:

Start Det Adj Adj Noun **Stop** = 0.32 * 0.0009 ...
the cool directed autos

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

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 $_{\text{s in STATES}}$ 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) * $\underbrace{a[s,s']}_{\text{bigram}}$ * $\underbrace{b_{s'}(o_t)}_{\text{unigram}}$

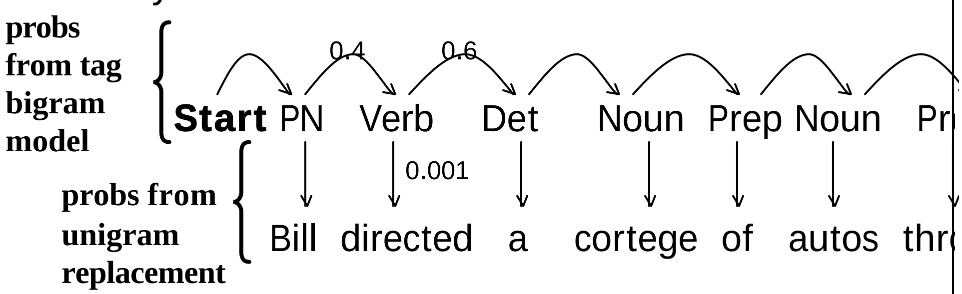
Or as in your text...p. 179

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?
- Noisy channel model is a “Hidden Markov Model”:



- Find X that maximizes probability product

Two finite-state approaches

1. Noisy Channel Model (statistical)

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

- PS: how do we *evaluate* taggers? (and such statistical models generally?)