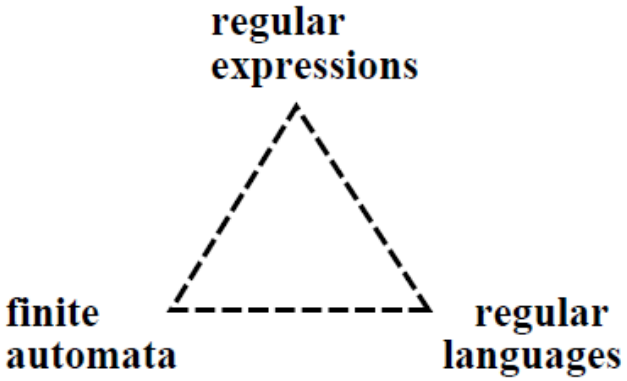
WORD LEVEL ANALYSIS

- Dr. Kiran Bhowmick
- Dr. Chetashri Bhadane

Unit 2 - Syllabus

2	**Word Level Analysis:** Morphology Analysis –Survey of English Morphology, Inflectional Morphology & Derivational Morphology, Lemmatization, Regular Expression, Finite Automata, Finite State Transducers (FST), Morphological Parsing With FST, Lexicon Free FST Porter Stemmer. N –Grams, Unigrams/Bigrams Language Models, Corpora, Computing the Probability Of Word Sequence, Training and Testing. Perplexity And Entropy: Smoothing and Backup, Zipf's Law, Add One Smoothing, Witten-Bell Discounting, Good Turing Discounting, Back Off Methods, Class Based Models, Google N-Gram Release.	**08**

Finite State Automata



- A regular expression is one way of describing a **finite-state automaton** (**FSA**).
- Any FSA regular expression can be implemented as a finite-state automaton (except regular expressions that use the memory feature).
- Symmetrically, any finite-state automaton can be described with a regular expression.
- A regular expression is one way of characterizing a particular kind of formal language called a **regular language**.
- Both regular expressions and finite-state automata can be used to described regular languages

FSA

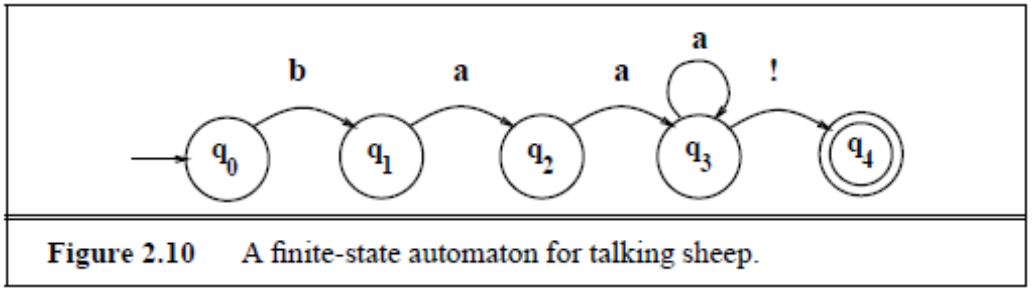


Figure 2.10 A finite-state automaton for talking sheep.

baa!
baaa!
baaaa!
baaaaa!
baaaaaa!
...

The automaton has five **state**s, which are represented by nodes in the graph. State 0 is the **start state** which we represent by the incoming arrow. State 4 is the **final state** or **accepting state**, which we represent by the double circle. It also has four **transitions**, which we represent by arcs in the graph.

Definition of FA

- Q: a finite set of N states $q_0, q_1, \ldots q_N$
- Σ: a finite input alphabet of symbols
- q_0: the start state
- F: the set of final states, $F \subseteq Q$
- $\delta(q, i)$: the transition function or transition matrix between states. Given a state $q \in Q$ and an input symbol $i \in \Sigma$, $\delta(q, i)$ returns a new state $q' \in Q$. δ is thus a relation from $Q \times \Sigma$ to Q;

For the sheeptalk automaton in Figure 2.10, $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b, !\}$, $F = \{q_4\}$, and $\delta(q, i)$ is defined by the transition table in Figure 2.12.

State-transition table

	Input		
State	b	a	!
0	1	0	0
1	0	2	0
2	0	3	0
3	0	3	4
4:	0	0	0

Deterministic and Non-deterministic FSA

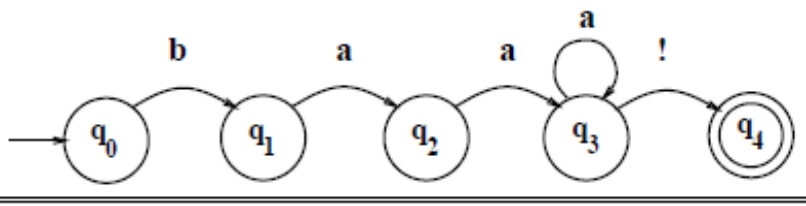


Figure 2.10 A finite-state automaton for talking sheep.

A deterministic automaton can be referred to as a **DFSA**

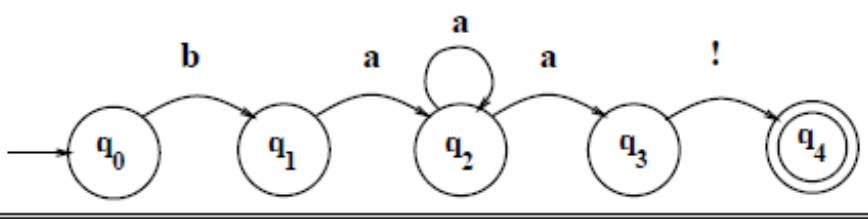


Figure 2.18 A non-deterministic finite-state automaton for talking sheep (NFSA #1). Compare with the deterministic automaton in Figure 2.10.

When we get to state 2, if we see an **a** we don't know whether to remain in state 2 or go on to state 3. Automata with decision points like this are called **non-deterministic FSA**s (or **NFSA**s).

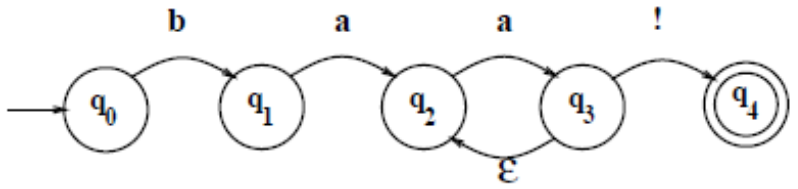


Figure 2.19 Another NFSA for the sheep language (NFSA #2). It differs from NFSA #1 in Figure 2.18 in having an ε-transition.

The automaton in Figure 2.19 defines the exact same language as the last one, or our first one, but it does it with an ε-transition.

We interpret this new arc as follows: if we are in state 3, we are allowed to move to state 2 *without* looking at the input, or advancing our input pointer. So this introduces another kind of non-determinism – we might not know whether to follow the e-transition or the **!** arc.

Regular languages and FSA

- The class of languages that are **definable** by regular expressions is exactly the same as the class of languages that are **characterizable** by finite-state automata (whether deterministic or non-deterministic).
- Because of this, we call these languages the **regular languages**

1. $\emptyset$ is a regular language
2. $\forall a \in \Sigma \cup \varepsilon$, $\{a\}$ is a regular language
3. If L_1 and L_2 are regular languages, then so are:
   (a) $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$, the **concatenation** of L_1 and L_2
   (b) $L_1 \cup L_2$, the **union** or **disjunction** of L_1 and L_2
   (c) L_1^*, the **Kleene closure** of L_1

All and only the sets of languages which meet the above properties are regular languages.
Operations of regular languages

- Concatenation
- Disjunction / union
- Kleene closure

Regular languages are also closed under the following operations:

- Intersection
- Difference
- Complementation
- Reversal

Prove that

Regular expressions are equivalent to finite-state automata

(hint: showing that an automaton can be built for each regular language, and conversely that a regular language can be built for each automaton. Use Mathematical Induction)

Morphology

- Morphology is the study of the way words are built from smaller meaningful units called **morphemes**.
- We can divide morphemes into two broad classes.
  - **Stems** – the core meaningful units, the root of the word.
  - **Affixes** – add additional meanings and grammatical functions to words.
- Affixes are further divided into:
  - **Prefixes** – precede the stem: do / undo
  - **Suffixes** – follow the stem: eat / eats
  - **Infixes** – are inserted inside the stem: taboo morphemes are inserted in the middle of the word
  - **Circumfixes** – precede and follow the stem: write / rewrites, believe / unbelievably
- English doesn't stack more affixes.
- But Turkish can have words with a lot of suffixes.
- Languages, such as Turkish, tend to string affixes together are called **agglutinative** languages.

Inflectional and Derivational Morphology

- There are two broad classes of morphology:
  - **Inflectional morphology**
  - **Derivational morphology**
- After a combination with an **inflectional morpheme**, the meaning and class of the actual stem usually do not change.
  - eat / eats
  - pencil / pencils
- After a combination with a **derivational morpheme**, the meaning and the class of the actual stem usually change.
  - compute / computer
  - do / undo
  - friend / friendly
- The irregular changes may happen with derivational affixes.

English Inflectional Morphology

- Only nouns, verbs and some adjectives can be inflected.
- Nouns have simple inflectional morphology.
  - Singular form : bare stem e.g. cat, dog, boy
  - Plural form:
    - Regular plural : *-s* e.g. cats / dogs / boys
    - Words ending with *-s* e.g. ibis/ibises
    - Words ending with *-sh* e.g. bush/bushes
    - Words ending with *-ch* e.g. finch/finches
    - Words ending with -z e.g. waltz/waltzes
    - Words ending with -x e.g. box/boxes
    - Words ending with -y e.g. butterfly/butterflies
  - Possessive -- John / John's

	Regular Nouns		Irregular Nouns	
Singular	cat	thrush	mouse	ox
Plural	cats	thrushes	mice	oxen

English Inflectional Morphology

- Verbs have slightly more complex inflectional, but still relatively simple inflectional morphology.
  - past form -- walk / walked
  - past participle form -- walk / walked
  - gerund -- walk / walking
  - singular third person -- walk / walks
- Verbs can be categorized as:
  - main verbs – eat, sleep, impeach
  - modal verbs -- can, will, should
  - primary verbs -- be, have, do
- Regular and irregular verbs: walk / walked -- go / went

Morphological Form Classes	Regularly Inflected Verbs			
stem	walk	merge	try	map
-s form	walks	merges	tries	maps
-ing participle	walking	merging	trying	mapping
Past form or *-ed* participle	walked	merged	tried	mapped

Morphological Form Classes	Irregularly Inflected Verbs		
stem	eat	catch	cut
-s form	eats	catches	cuts
-ing participle	eating	catching	cutting
Past form	ate	caught	cut
-ed participle	eaten	caught	cut

Note that an irregular verb can inflect in the past form (also called the **preterite**) by changing its vowel (*eat/ate*), or its vowel and some consonants (*catch/caught*), or with no ending at all (*cut/cut*).

English Inflectional Morphology

- The *-s* form is used in the 'habitual present' form to distinguish the 3rd-person singular ending
  - (*She jogs every Tuesday*) from the other choices of person and number (*I/you/we/they jog every Tuesday*).
- The stem form is used in the infinitive form, and also after certain other verbs
  - (*I'd rather walk home*, *I want to walk home*).
- The *-ing* participle is used when the verb is treated as a noun; this particular kind of nominal use of a verb is called a **gerund** use:
  - *Fishing is fine if you live near water.*
- The *-ed* participle is used in the **perfect** construction
  - (*He's eaten lunch already*)
- Or the **passive** construction
  - (*The verdict was overturned yesterday.*).
- In addition to noting which suffixes can be attached to which stems, we need to capture the fact that a number of regular spelling changes occur at these morpheme boundaries.
  - For example, a single consonant letter is doubled before adding the -ing and -ed suffixes (beg/begging/begged).
  - If the final letter is 'c', the doubling is spelled 'ck' (picnic/picnicking/picnicked).
  - If the base ends in a silent -e, it is deleted before adding -ing and -ed (merge/merging/merged).

English Derivational Morphology

- Derivation is a combination of word stem with a grammatical morpheme usually resulting in a word of a different class where the meaning is often hard to predict.
- Some English derivational affixes
  - -ation : transport / transportation
  - -er : kill / killer
  - -ness : fuzzy / fuzziness
  - -al : computation / computational
  - -able : break / breakable
  - -less : help / helpless
  - un : do / undo
  - re : try / retry

Nominalization: formation of new nouns from verbs or adjectives.

Suffix	Base Verb/Adjective	Derived Noun
-ation	computerize (V)	computerization
-ee	appoint (V)	appointee
-er	kill (V)	killer
-ness	fuzzy (A)	fuzziness

Adjectives can also be derived from nouns and verbs

Suffix	Base Noun/Verb	Derived Adjective
-al	computation (N)	computational
-able	embrace (V)	embraceable
-less	clue (N)	clueless

Lemmatization

Represent all words as their lemma, their shared root
= dictionary headword form:

- *am, are, is* → *be*
- *car, cars, car's, cars'* → *car*

- Lemmatization is done by Morphological Parsing

Morphological Parsing

- Morphological parsing is to find the lexical form of a word from its surface form.

Input	Morphological Parsed Output
cats	`cat +N +PL`
cat	`cat +N +SG`
cities	`city +N +PL`
geese	`goose +N +PL`
goose	`(goose +N +SG)` or `(goose +V)`
gooses	`goose +V +3SG`
merging	`merge +V +PRES-PART`
caught	`(catch +V +PAST-PART)` or `(catch +V +PAST)`

- There can be more than one lexical level representation for a given word. (ambiguity)

Parts of A Morphological Processor

- For a morphological processor, we need at least followings: Lexicon, Morphotactics, Rules
- **Lexicon** : The list of stems and affixes together with basic information about them such as their main categories (noun, verb, adjective, …) and their sub-categories (regular noun, irregular noun, …).
  - A lexicon is a repository for words (stems).
  - They are grouped according to their main categories.
    - noun, verb, adjective, adverb, …
  - They may be also divided into sub-categories.
    - regular-nouns, irregular-singular nouns, irregular-plural nouns, …
  - The simplest way to create a morphological parser, put all possible words (together with its inflections) into a lexicon.
    - We do not this because their numbers are huge (theoretically for Turkish, it is infinite)

- **Morphotactics** : Morphotactics is the model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word. For example, the rule that the English plural morpheme follows the noun rather than preceding it.
- Which morphemes can follow which morphemes.

- **Orthographic Rules (Spelling Rules)** : These spelling rules are used to model changes that occur in a word (normally when two morphemes combine).
- For example: y -> ie rule ….. city -> cities

Lexicon and Morphotactics – nominal inflection

Lexicon:

reg-noun	irreg-pl-noun	irreg-sg-noun	plural
fox	geese	goose	-s
cat	sheep	sheep	
dog	mice	mouse	
aardvark			

- Simple English Nominal Inflection (Morphotactic Rules)

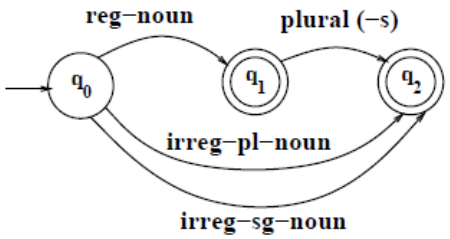


Figure 3.2 A finite-state automaton for English nominal inflection.

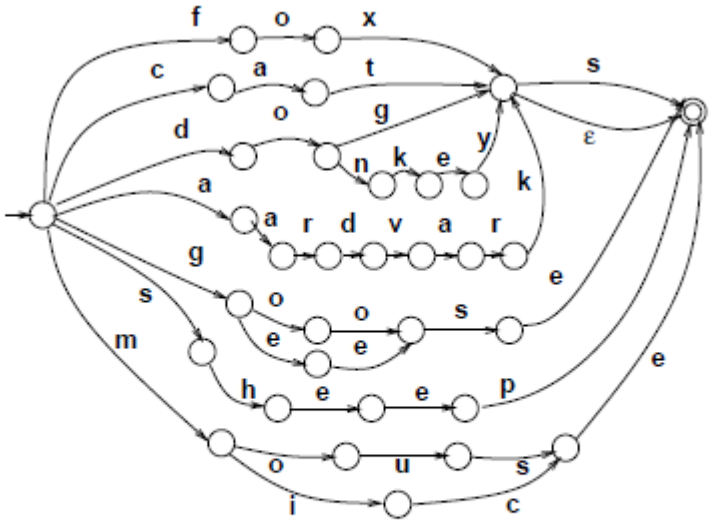


Figure 3.7 Compiled FSA for a few English nouns with their inflection. Note that this automaton will incorrectly accept the input *foxs*. We will see beginning on page 76 how to correctly deal with the inserted *e* in *foxes*.

Lexicon and Morphotactics – verbal inflection

- Lexicon: This lexicon has three stem classes (reg-verb-stem, irreg-verb-stem, and irreg-past-verb-form), plus 4 more affix classes (*-ed* past, *-ed* participle, *-ing* participle, and 3rd singular *-s*):

reg-verb-stem	irreg-verb-stem	irreg-past-verb	past	past-part	pres-part	3sg
walk fry talk impeach	cut speak sing sang cut spoken	caught ate eaten	-ed	-ed	-ing	-s

- Simple English Verbal Inflection (Morphotactic Rules)

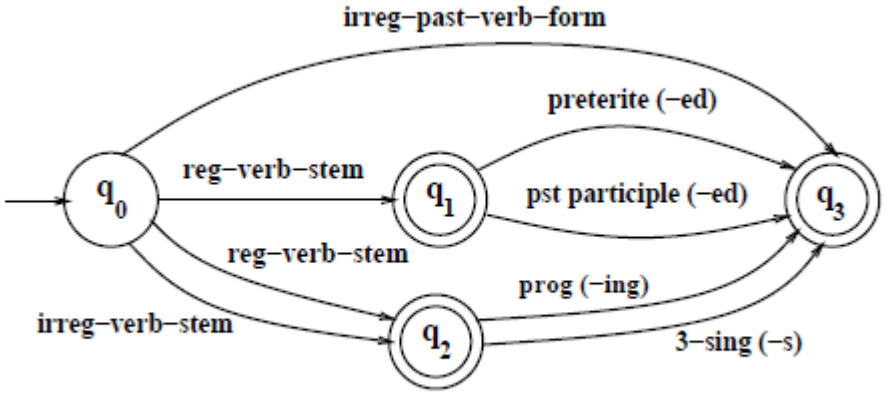


Figure 3.3 A finite-state automaton for English verbal inflection

Lexicon and Morphotactics – adjective inflection

big, bigger, biggest
cool, cooler, coolest, coolly
red, redder, reddest
clear, clearer, clearest, clearly, unclear, unclearly
happy, happier, happiest, happily
unhappy, unhappier, unhappiest, unhappily
real, unreal, really

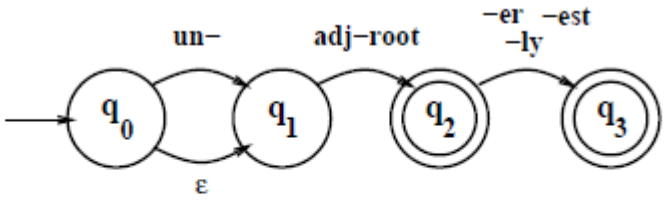


Figure 3.4 An FSA for a fragment of English adjective morphology: Antworth's Proposal #1.

also recognize ungrammatical forms like *unbig*, *redly*, and *realest*

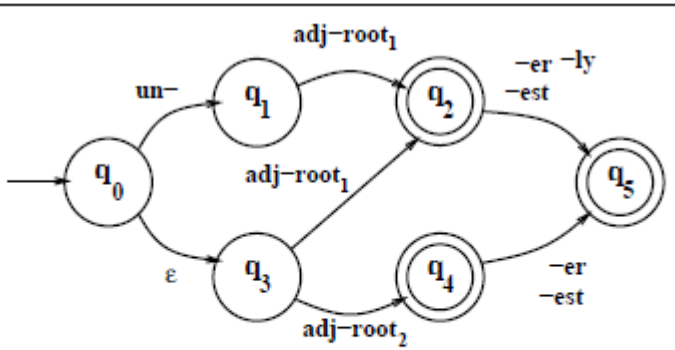


Figure 3.5 An FSA for a fragment of English adjective morphology: Antworth's Proposal #2.

We need to set up classes of roots and specify which can occur with which suffixes.

1. **adj-root**1 would include adjectives that can occur with *un-* and *-ly* (*clear*, *happy*, and *real*)
2. **adj-root**2 will include adjectives that can't occur (*big*, *cool*, and *red*).

Lexicon and Morphotactics – derivational morphology

- This FSA models a number of derivational facts, such as the well known generalization that any verb ending in *-ize* can be followed by the nominalizing suffix *–ation.*
- Thus since there is a word *fossilize*, we can predict the word *fossilization* by following states *q*0, *q*1, and *q*2.
- Similarly, adjectives ending in *-al* or *-able* at *q*5 (*equal*, *formal*, *realizable*) can take the suffix *-ity*, or sometimes the suffix *-ness* to state *q*6 (*naturalness*, *casualness*).

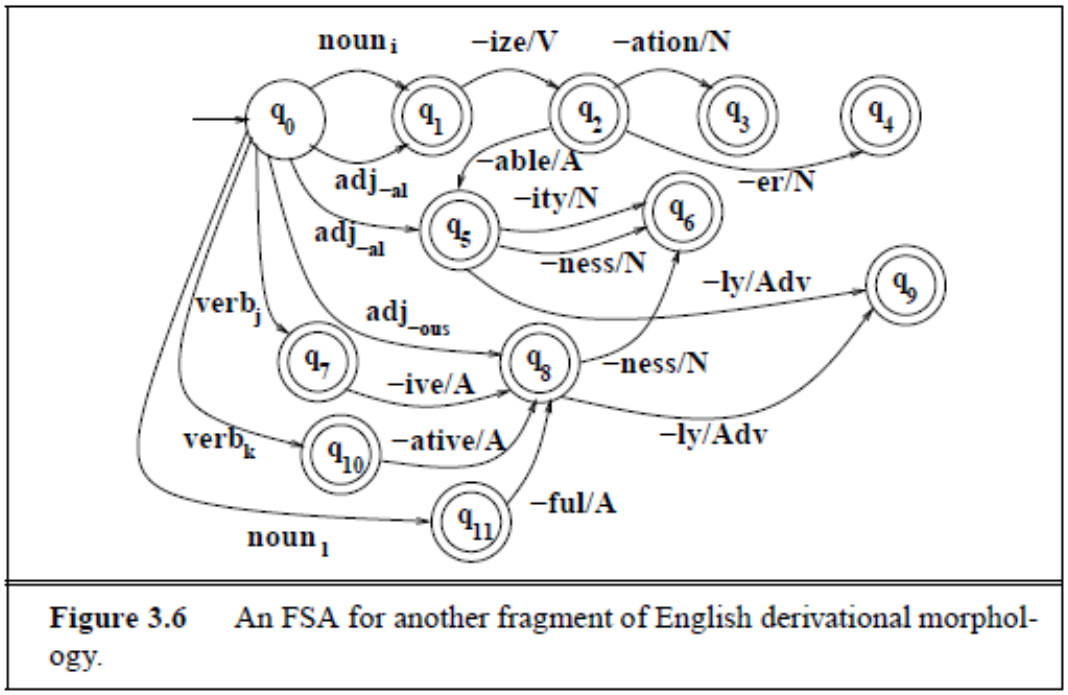


Figure 3.6 An FSA for another fragment of English derivational morphology.

Combine Lexicon and Morphotactics

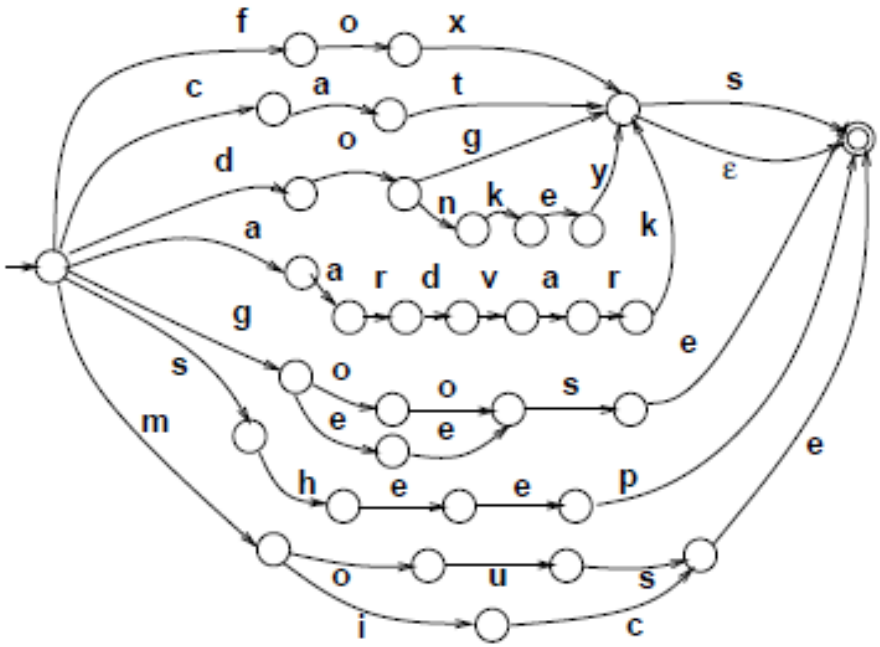
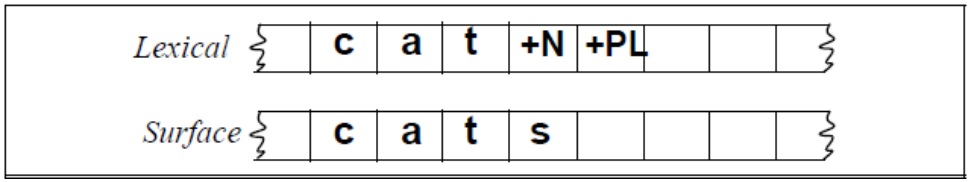


Figure 3.7 Compiled FSA for a few English nouns with their inflection. Note that this automaton will incorrectly accept the input *foxs*. We will see beginning on page 76 how to correctly deal with the inserted *e* in *foxes*.

Morphological parsing with FST

- Two-level morphology represents the correspondence between lexical and surface levels.



- We use a finite-state transducer to find mapping between these two levels.
- A FST is a two-tape automaton:
  - Reads from one tape, and writes to other one.
- For morphological processing, one tape holds lexical representation, the second one holds the surface form of a word.

FST as recognizer: a transducer that takes a pair of strings as input and outputs *accept* if the string-pair is in the string-pair language, and a *reject* if it is not.

FST as generator: machine that outputs pairs of strings of the language. Thus the output is a yes or no, and a pair of output strings.

FST as translator: a machine that reads a string and outputs another string.

FST as a set relater: a machine that computes relations between sets.

Formal Definition of FST (Mealey Machine)

Q: a finite set of N states $q_0, q_1, \ldots q_N$

Σ: a finite alphabet of complex symbols. Each complex symbol is composed of an input-output pair $i : o$; one symbol i from an input alphabet I, and one symbol o from an output alphabet O, thus $\Sigma \subseteq I \times O$. I and O may each also include the epsilon symbol ε.

q_0: the start state

F: the set of final states, $F \subseteq Q$

$\delta(q, i : o)$: the transition function or transition matrix between states. Given a state $q \in Q$ and complex symbol $i : o \in \Sigma$, $\delta(q, i : o)$ returns a new state $q' \in Q$. δ is thus a relation from $Q \times \Sigma$ to Q;

Where an FSA accepts a language stated over a finite alphabet of single symbols, such as the alphabet of our sheep language:

$$\Sigma = \{b, a, !\}$$

an FST accepts a language stated over *pairs* of symbols, as in:

$$\Sigma = \{a : a,\ b : b,\ ! : !,\ a : !,\ a : \varepsilon,\ \varepsilon : !\}$$

FST Properties

- FSTs are closed under: union, inversion, and composition.

- **union** : The union of two regular relations is also a regular relation.
- **inversion** : The inversion of a FST simply switches the input and output labels.
  - This means that the same FST can be used for both directions of a morphological processor.
- **composition** : If T_1 is a FST from I_1 to O_1 and T_2 is a FST from I_2 to O_2, then composition of T_1 and T_2 i.e. (T_1 o T_2) maps from I_1 to O_2.

- We use these properties of FSTs in the creation of the FST for a morphological processor.

A FST for Simple English Nominals

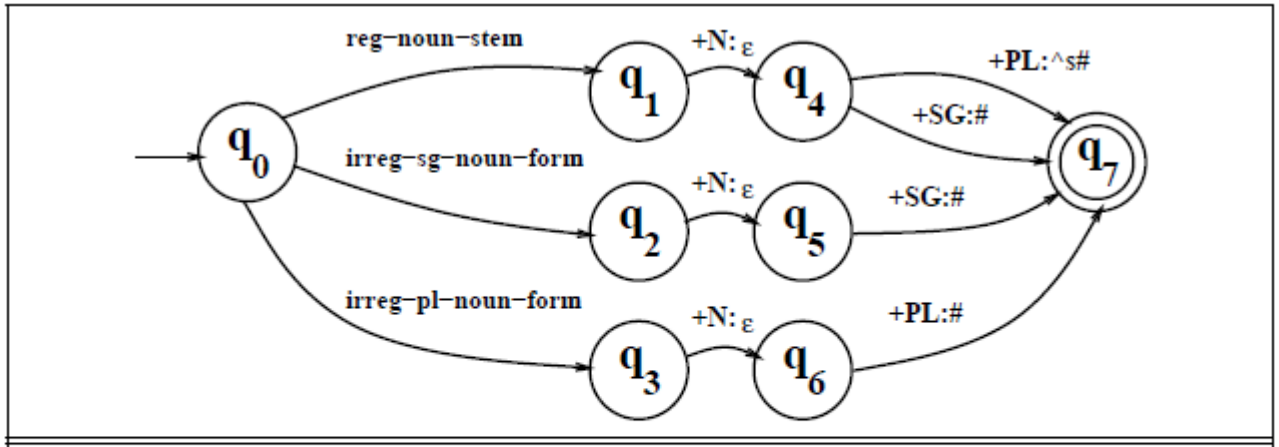


Figure 3.9 A transducer for English nominal number inflection T_{num}. Since both q_1 and q_2 are accepting states, regular nouns can have the plural suffix or not. The morpheme-boundary symbol ^ and word-boundary marker # will be discussed below.

reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e s e	goose
cat	sheep	sheep
dog	m o:i u:ɛ s:c e	mouse
aardvark		

FST for stems

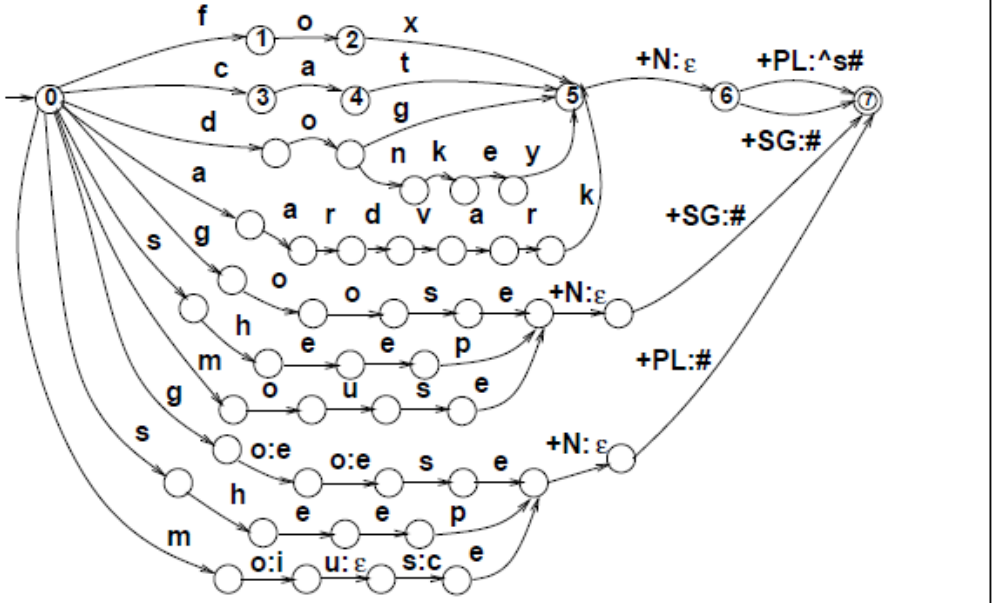


Figure 3.11 A fleshed-out English nominal inflection FST $T_{lex} = T_{num} \circ T_{stems}$.

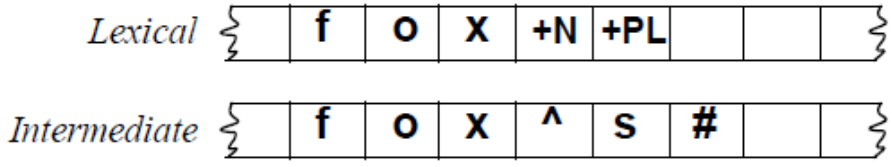
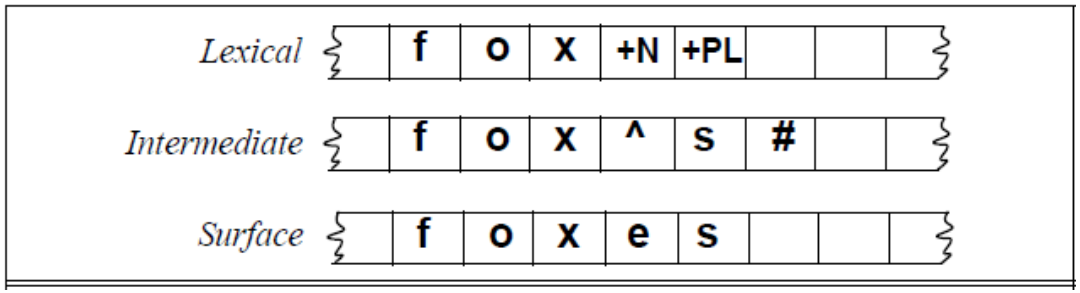


Figure 3.12 An example of the lexical and intermediate tapes.

Lexical to Intermediate FST



We represent these rules using two-level morphology rules: Context Sensitive rules

Rule Notation

$a \rightarrow b / c\_d$: "rewrite a as b when it occurs between c and d."

Eg. c a d → c b d

Apply these rules to go from intermediate level → surface level

$$\varepsilon \rightarrow e / \left\{ \begin{matrix} x \\ s \\ z \end{matrix} \right\} \hat{} \_\_ s\#$$

insert an *e* after a morpheme - final *x*, *s*, or *z*, and before the morpheme *s*

Generating or Parsing with FST Lexicon and Rules

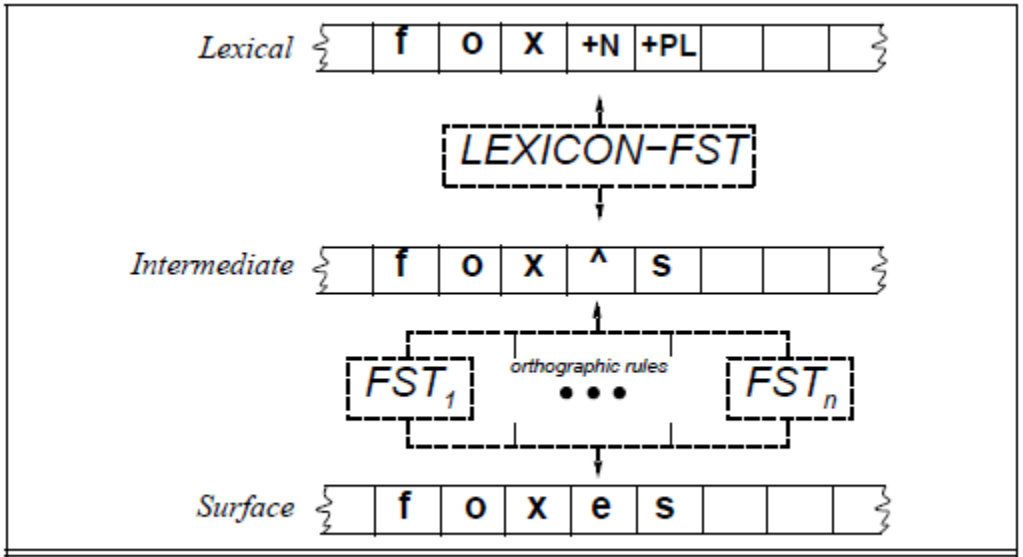


Figure 3.16 Generating or Parsing with FST lexicon and rules

The cascade in Figure 3.16 has two transducers in series: the transducer mapping from the lexical to the intermediate levels, and the collection of parallel transducers mapping from the intermediate to the surface level. The cascade can be run top-down to generate a string, or bottom-up to parse it;

Porter Stemming

- The Porter Stemming algorithm is a process for removing suffixes in English words

- Some applications (some informational retrieval applications) do not the whole morphological processor.
- They only need the stem of the word.
- A stemming algorithm (Port Stemming algorithm) is a lexicon-free FST.
- It is just a cascaded rewrite rules.
- Stemming algorithms are efficient but they may introduce errors because they do not use a lexicon.

Porter Stemming

- Notation

v **vowel(s)**

A, E, I, O, and U, and other than Y preceded by a consonant

c **consonant(s)**

e.g. TOY T and Y are consonants

SSYGY Y is a vowel as it is preceded by a consonant

$$(C)(VC)^m(V)$$

Any word can be written:

$(vc)_m$ vowel(s) followed by constant(s), repeated m times

For example the word *troubles* maps to the following sequence:

```
troubles
C V C VC
```

Porter Stemming

$$(C)(VC)^m(V)$$

- m is called the measure of the word

m=0	TR, EE, TREE, Y, BY
m=1	TROUBLE, OATS, TREES, IVY
m=2	TROUBLES, PRIVATE, OATEN, ORRERY

The rules that we will present below will all be in the following format:

(condition) S1 → S2

meaning "if a word ends with the suffix S1, and the stem before S1 satisfies the condition, S1 is replaced by S2". Conditions include the following and any boolean combinations of them:

Porter Stemming

Step 1: Plural Nouns and Third Person Singular Verbs

The rules in this set do not have conditions:

SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
			ties	→	ti
SS	→	SS	caress	→	caress
S	→	ε	cats	→	cat

Step 2a: Verbal Past Tense and Progressive Forms

(m> 1)	EED	→	EE	feed	→	feed
				agreed	→	agree
(*v*)	ED	→	ε	plastered	→	plaster
				bled	→	bled
(*v*)	ING	→	ε	motoring	→	motor
				sing	→	sing

Porter Stemming

Step 2b: Cleanup

If the second or third of the rules in 2a is successful, we run the following rules (that remove double letters and put the E back on -ATE/-BLE)

	AT	→	ATE	conflat(ed)	→ conflate
	BL	→	BLE	troubl(ing)	→ trouble
	IZ	→	IZE	siz(ed)	→ size
(*d & !(*L or *S or *Z))		→	single letter	hopp(ing) tann(ed) fall(ing) hiss(ing) fizz(ed)	→ hop → tan → fall → hiss → fizz
(m=1 & *o)		→	E	fail(ing) fil(ing)	→ fail → file

Step 3: Y → I

(*v*)	Y	→	I	happy sky	→ happi → sky

Porter Stemming

Step 4: Derivational Morphology I: Multiple suffixes

(m > 0)	ATIONAL	→	ATE	relational	→	relate
(m > 0)	TIONAL	→	TION	conditional	→	condition
				rational	→	rational
(m > 0)	ENCI	→	ENCE	valenci	→	valence
(m > 0)	ANCI	→	ANCE	hesitanci	→	hesitance
(m > 0)	IZER	→	IZE	digitizer	→	digitize
(m > 0)	ABLI	→	ABLE	conformabli	→	conformable
(m > 0)	ALLI	→	AL	radicalli	→	radical
(m > 0)	ENTLI	→	ENT	differentli	→	different
(m > 0)	ELI	→	E	vileli	→	vile
(m > 0)	OUSLI	→	OUS	analogousli	→	analogous
(m > 0)	IZATION	→	IZE	vietnamization	→	vietnamize
(m > 0)	ATION	→	ATE	predication	→	predicate
(m > 0)	ATOR	→	ATE	operator	→	operate
(m > 0)	ALISM	→	AL	feudalism	→	feudal
(m > 0)	IVENESS	→	IVE	decisiveness	→	decisive
(m > 0)	FULNESS	→	FUL	hopefulness	→	hopeful
(m > 0)	OUSNESS	→	OUS	callousness	→	callous
(m > 0)	ALITI	→	AL	formaliti	→	formal
(m > 0)	IVITI	→	IVE	sensitiviti	→	sensitive
(m > 0)	BILITI	→	BLE	sensibiliti	→	sensible

Porter Stemming

Step 5: Derivational Morphology II: More multiple suffixes

(m > 0)	ICATE	→	IC	triplicate	→	triplic
(m > 0)	ATIVE	→	ε	formative	→	form
(m > 0)	ALIZE	→	AL	formalize	→	formal
(m > 0)	ICITI	→	IC	electriciti	→	electric
(m > 0)	FUL	→	ε	hopeful	→	hope
(m > 0)	NESS	→	ε	goodness	→	good

Step 6: Derivational Morphology III: single suffixes

(m > 1)	AL	→	ε	revival	→	reviv
(m > 1)	ANCE	→	ε	allowance	→	allow
(m > 1)	ENCE	→	ε	inference	→	infer
(m > 1)	ER	→	ε	airliner	→	airlin
(m > 1)	IC	→	ε	gyroscopic	→	gyroscop
(m > 1)	ABLE	→	ε	defensible	→	defens
(m > 1)	ANT	→	ε	irritant	→	irrit
(m > 1)	EMENT	→	ε	replacement	→	replac
(m > 1)	MENT	→	ε	adjustment	→	adjust
(m > 1)	ENT	→	ε	dependent	→	depend
(m > 1)	(*S or *T) & ION	→	ε	adoption	→	adopt
(m > 1)	OU	→	ε	homologou	→	homolog
(m > 1)	ISM	→	ε	communism	→	commun
(m > 1)	ATE	→	ε	activate	→	activ
(m > 1)	ITI	→	ε	angulariti	→	angular
(m > 1)	OUS	→	ε	homologous	→	homolog
(m > 1)	IVE	→	ε	effective	→	effect
(m > 1)	IZE	→	ε	bowdlerize	→	bowdler

Porter Stemming

Step 7a: Cleanup

(m > 1) E → ε	probate → probat	
	rate → rate	
(m = 1 & ! *o) E → ε	cease → ceas	

Step 7b: Cleanup

(m > 1 & *d *L) → [single letter]	controll → control	
	roll → roll	

Porter Stemming Example

Apply porter stemmer for the following example:
Tapping
Smiling
Computerization

N-grams

Unigram, Bi-gram, Tri-gram, ….

Language models

- Models that assign probabilities to sequences of words are called **language models** or **LMs**
- The simplest model that assigns probabilities to sentences and sequences of words, is called the **n-gram**.
- An n-gram is a contiguous sequence of *n* items from a given text or speech:

Unigram (1-gram): • is a single word sequence. • No history used.	Bi-gram (2-gram): • is a two-word sequence of words • One word history
Tri-gram (3-gram): • is a three-word sequence of words • Two words history	Four-gram (4-gram): • is a four-word sequence of words • Three words history

Corpora (Corpus)

- Statistical processing of natural language is based on **corpora** (singular **corpus**), online collections of text and speech.
- For computing word probabilities, we will be counting words in a training corpus.
- How many words are in this sentence?

  He stepped out into the hall, was delighted to encounter a water brother.

- Sentence has 13 words without counting punctuation-marks as words, 15 if we count punctuation.
- **Counting words in corpus**

Types - mean the number of distinct words in a corpus, i.e. the size of the vocabulary,

Tokens to mean the total number of running words.

They picnicked by the pool, then lay back on the grass and looked at the stars.

16 word tokens and 14 word types

Some important formulas

- Conditional probability: $P(B|A) = \frac{P(A,B)}{P(A)}$

$$P(A,B) = P(A)P(B|A)$$

- **More variables:**

$$P(A,B,C,D) = P(A)\,P(B|A)\,P(C|A,B)\,P(D|A,B,C)$$

- **Chain rule:**

$$P(w_1,w_2,\ldots..,w_n) = P(w_1)\,P(w_2|w_1)\,P(w_3|w_1,w_2)\,\ldots\ldots\,P(w_n|w_1,w_2\,\ldots\ldots,w_{n-1})$$

- **Probability of word in a sentence:**

$$P(w_1,w_2,\ldots..,w_n) = \prod_i P(w_i|w_1,w_2\,\ldots\ldots,w_{i-1})$$

P(I like Chinese food) = P(I) x P(like | I) x P(Chinese | I like) x P(food | I like Chinese)

N-gram models again

- **Unigram : no history used**
- "I like Italian ……."
- Lets assume food has highest probability in the corpus
- Unigram takes into account the probabilities with the previous words viz, like Italian and predict food.
- **Bigram : one word history**
- $P(w_1, w_2) = \prod_{i=2} P(w_i|w_1, w_2)$ $\quad P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$
- "I like Italian ……."
- Assumption: Next word may be food, music
- Unigram takes into account the probabilities with the previous words viz, like Italian and predict food.

N-gram models again

- **Bigram : one word history**
- $P(w_1, w_2) = \prod_{i=2} P(w_2|w_1)$ $\quad P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$
- "I like Italian ……."
- Assumption: Next word may be food, music

$$P(\text{food} \mid \text{I like Italian}) = \frac{count(\text{I like Italian food})}{count(\text{I like Italian})}$$

$$P(\text{music} \mid \text{I like Italian}) = \frac{count(\text{I like Italian music})}{count(\text{I like Italian})}$$

N-gram models again

- **Bigram : one word history**
- "I like Italian ……."

$$count(\text{I like Italian}) = P(I \mid <S>) \text{ x P(like | I) x P(Italian | like)}$$

$count(\text{I like Italian food}) = P(I \mid <S>)$ x P(like | I) x P(Italian | like) x P(food | Italian)

$count(\text{I like Italian music}) = P(I \mid <S>)$ x P(like | I) x P(Italian | like) x P(music | Italian)

On substituting in previous eqn, we get

$$P(\text{food} \mid \text{I like Italian}) = \frac{count(\text{I like Italian food})}{count(\text{I like Italian})} = P(food \mid Italian)$$

$$P(\text{music} \mid \text{I like Italian}) = \frac{count(\text{I like Italian music})}{count(\text{I like Italian})} = P(music \mid Italian)$$

N-gram models again

- **Trigram : two word history**

- $P(w_1, w_2, w_3) = \prod_{i=3} P(w_i|w_1, w_2)$ $\qquad P(w_i|w_{i-1}, w_{i-2}) = \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-2}, w_{i-1})}$

$count(\text{I like Italian}) = P(like \mid < S >, I) \text{x } P(Italian \mid I, like)$

$count(\text{I like Italian food}) = P(like \mid < S > I) \text{x } P(Italian \mid I\ like) \text{ x } P(food \mid like\ Italian)$

$count(\text{I like Italian } music) = P(like \mid < S > I) \text{x } P(Italian \mid I\ like)$

x $P(music \mid like\ Italian)$

On substituting in previous eqn, we get

$$P(\text{food} \mid \text{I like Italian}) = \frac{count(\text{I like Italian food})}{count(\text{I like Italian})} = P(food \mid like\ Italian)$$

$$P(\text{music} \mid \text{I like Italian}) = \frac{count(\text{I like Italian music})}{count(\text{I like Italian})} = P(music \mid like\ Italian)$$

N-gram models again

- **Four-gram : three word history**
- $P(w_1, w_2, w_3, w_4) = \prod_{i=4} P(w_i | w_1, w_2, w_3)$
- $P(w_i | w_{i-1}, w_{i-2}, w_{i-3}) = \frac{count(w_{i-3}, w_{i-2}, w_{i-1}, w_i)}{count(w_{i-3}, w_{i-2}, w_{i-1})}$

count(I like Italian) = P(Italian | < S > I like)

count(I like Italian food) = P(Italian | < S > I like) x P(food | I like Italian)

count(I like Italian music) = P(Italian | < S > I like) X P(music | I like Italian)

On substituting in previous eqn, we get

$$\text{P(food | I like Italian)} = \frac{\text{count(I like Italian food)}}{\text{count(I like Italian)}} = \text{P(food |I like Italian)}$$

$$\text{P(music | I like Italian)} = \frac{\text{count(I like Italian music)}}{\text{count(I like Italian)}} = \text{P(music |I like Italian)}$$

Sample – computing probabilities of word sequences (Sentences)

- The man from Jupiter came
- Unigram P(<s>The man from Jupiter came </s>) =
- Bigram P(<s>The man from Jupiter came </s>) =
- Trigram P(<s>The man from Jupiter came </s>) =

Exercise 1

What is the next probable word predicted by the model for the following word sequence?

Given Corpus

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Exercise: Using bi-gram

1) <S> Do ?

Next word prediction probability W_{i-1} = do

Word	Frequency
P(</S> | do)	0/4
P(I | do)	2/4
P(am | do)	0/4
P(Henry | do)	1/4
P(like | do)	1/4
P(college | do)	0/4
P(do | do)	0/4

I is more probable

Given Corpus

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Exercise: Using bi-gram

2) <S> I like Henry ?

Next word prediction probability W_{i-1} = Henry

Word	Frequency
P(</S> | Henry)	3/5
P(I | Henry)	1/5
P(am | Henry)	0
P(Henry | Henry)	0
P(like | Henry)	1/5
P(college | Henry)	0
P(do | Henry)	0

</S> is more probable

Given Corpus

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Exercise: Using tri-gram

1) <S> Do I like ?

Next word prediction probability W_{i-2} = I W_{i-1} = like

Word	Frequency
P(</S> | I like)	0/3
P(I | I like)	0/3
P(am | I like)	0/3
P(Henry | I like)	1/3
P(like | I like)	0/3
P(college | I like)	2/3
P(do | I like)	0/3

college is more probable

Given Corpus

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Exercise 2

Which of the following sentence is better i.e gets higher probability with bi-gram model?

<S> I like college </S>

<S> Do I like Henry </S>

Given Corpus

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

1. <S> I like college </S>

= P(I | <S>) x P(like | I) x P(college | like) x P (</S> | college) = 3/7 x 3/6 x 3/5 x 3/3 = 9/70 = 0.13

2. <S> Do I like Henry </S>

= P(Do | <S>) x P(I | Do) x P(like | I) x P(Henry | like) x P(</S> | Henry) = 3/7 x 2/4 x 3/6 x 2/5 x 3/5 = 9/350 = 0.0257

First statement is more probable

Exercise 3

Consider the corpus given below. What is the most probable next word predicted by the model for the following word sequence.

<S> I really ? (Use bi-gram model)

<S> Thank you so ? (Use tri-gram model)

Given Corpus

<s> Thank you so much for your help. </s>

<s> I really appreciate your help. </s>

<s> Excuse me, do you know what time it is? </s>

<s> I'm really sorry for not inviting you. </s>

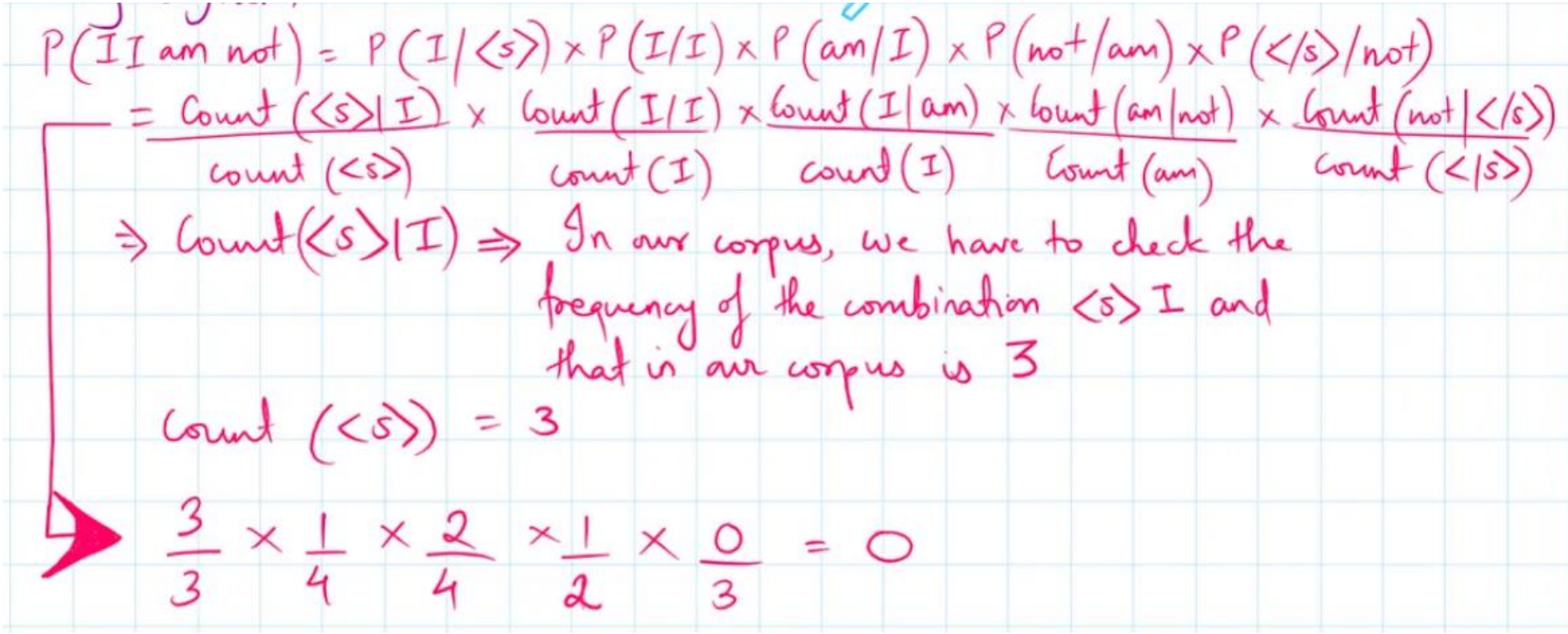
<s> I really like your watch. </s>

Exercise 4

Consider the corpus given below. Check the probability of

I I am not

Using bigram



Smoothing and discounting

Overcoming N-gram problems

N-grams Advantages and disadvantages

Generally in practical applications, bi-gram, tri-gram and four-gram models are used.

As no. of previous state (history) increases, it is very difficult to match that set of words in corpus.

Advantages:

- Easy to understand, implement
- Can be easily converted to any gram

Disadvantages:

- Underflow due to multiplication of probabilities
- **Solution:** Use log. Add probabilities
- Zero probability problem
- **Solution:** Use Laplace smoothing.

Use log to avoid underflow

Which of the following sentence is better i.e gets higher probability with bi-gram model?

1. <S> I like college </S>

= P(I | <S>) x P(like | I) x P(college | like) x P (</S> | college)

= 3/7 x 3/6 x 3/5 x 3/3 = 9/70 = **0.13**

= log(3/7) + log(3/6) + log(3/5) + log(3/3) = **-2.0513**

2. <S> Do I like Henry </S>

= P(Do | <S>) x P(I | Do) x P(like | I) x P(Henry | like) x P(</S> | Henry)

= 3/7 x 2/4 x 3/6 x 2/5 x 3/5 = 9/350 = **0.0257**

= log(3/7) + log(2/4) + log(3/6) + log(2/5) + log(3/5) **= -3.6607**

First statement is more probable

Laplace smoothing

Which of the following sentence is better i.e gets higher probability with bi-gram model?

Given Corpus

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

1. <S> like college </S>

= P(like | <S>) x P(college | like) x P (</S> | college) = 0/7 x 3/5 x 3/3 **= 0**

2. <S> Do I like Henry </S>

= P(Do | <S>) x P(I | Do) x P(like | I) x P(Henry | like) x P(</S> | Henry) = 3/7 x 2/4 x 3/6 x 2/5 x 3/5 = 9/350 = **0.0257**

Second statement is more probable

Laplace smoothing

Given Corpus

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Unique words are :<S>, </S>, I, Henry, do, like, am, college

Total unique words: 8

Excluding <S> as it never comes in bi-gram calculations

Total unique words: 7

Laplace smoothing

Give the estimation for the statements using Laplace smoothing

Given Corpus

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

1. <S> like college </S>

= P(like | <S>) x P(college | like) x P (</S> | college)

= (0+1)/(7+7) x (3+1)/(5+7) x (3+1)/(3+7)

= 1/14 x 4/12 x 4/10 = 0.0095

First statement is more probable

2. <S> Do I like Henry </S>

= P(Do | <S>) x P(I | Do) x P(like | I) x P(Henry | like) x P(</S> | Henry)

= (3+1)/(7+7) x (2+1)/(4+7) x (3+1)/(6+7) x (2+1)/(5+7) x (3+1)/(5+7)

= 4/14 x 3/11 x 4/13 x 3/12 x 4/12

= **0.0020**

Laplace smoothing

	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

Figure 6.4 Bigram counts for 7 of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ˜10,000 sentences.

I	3437
want	1215
to	3256
eat	938
Chinese	213
food	1506
lunch	459

	I	want	to	eat	Chinese	food	lunch
I	9	1088	1	14	1	1	1
want	4	1	787	1	7	9	7
to	4	1	11	861	4	1	13
eat	1	1	3	1	20	3	53
Chinese	3	1	1	1	1	121	2
food	20	1	18	1	1	1	1
lunch	5	1	1	1	1	2	1

Figure 6.6 Add-one Smoothed Bigram counts for 7 of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ˜10,000 sentences.

I	3437+1616	=	5053
want	1215+1616	=	2931
to	3256+1616	=	4872
eat	938+1616	=	2554
Chinese	213+1616	=	1829
food	1506+1616	=	3122
lunch	459+1616	=	2075

Laplace smoothing

	I	want	to	eat	Chinese	food	lunch
I	9	1088	1	14	1	1	1
want	4	1	787	1	7	9	7
to	4	1	11	861	4	1	13
eat	1	1	3	1	20	3	53
Chinese	3	1	1	1	1	121	2
food	20	1	18	1	1	1	1
lunch	5	1	1	1	1	2	1

Figure 6.6 Add-one Smoothed Bigram counts for 7 of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ˜10,000 sentences.

I	3437+1616	=	5053
want	1215+1616	=	2931
to	3256+1616	=	4872
eat	938+1616	=	2554
Chinese	213+1616	=	1829
food	1506+1616	=	3122
lunch	459+1616	=	2075

	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

Figure 6.7 Add-one smoothed bigram probabilities for 7 of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ˜10,000 sentences.

Witten bell smoothing

- Witten-Bell discounting is based on a simple but clever intuition about zero-frequency events.
- A zero-frequency word or N-gram is one that just hasn’t happened yet. When it does happen, it will be the first time we see this new N-gram.
- So the probability of seeing a zero-frequency N-gram can be modeled by the probability of seeing an N-gram for the first time
- We compute the probability of seeing an N-gram for the first time by counting the number of N-gram ***types*** we saw in the data.

Witten bell smoothing

- The total probability mass of all the zero N-grams is computed as the *number of types* divided by the *number of tokens plus observed types*

$$\sum_{i:c_i=0} p_i^* = \frac{T}{N+T}$$

Where,
T- no. of observed types
N- no of n-gram tokens
V – Vocabulary size / total types

- Now, we divide the above probability equally among all the zero N-grams.

$$Z = \sum_{i:c_i=0} 1$$

$$p_i^* = \frac{T}{Z(N+T)}$$

Where Z = total N-grams with zero probability

- Probability of all the seen N-grams as follows:

$$p_i^* = \frac{c_i}{N+T} \text{if } (c_i > 0)$$

- The smoothed counts for unigram by Witten-Bell discounting is then given by

$$c_i^* = \begin{cases} \frac{T}{Z}\frac{N}{N+T}, & \text{if } c_i = 0 \\ c_i\frac{N}{N+T}, & \text{if } c_i > 0 \end{cases}$$

Witten bell smoothing

- For bi-gram models, to compute the probability of a bigram w_{n-1} w_{n-2} we haven't seen, we use 'the probability of seeing a new bigram starting with w_{n-1}

$$Z(w_x) = \sum_{i:c(w_x w_i)=0} 1$$

$$p^*(w_i|w_{i-1}) = \frac{T(w_{i-1})}{Z(w_{i-1})(N + T(w_{i-1}))} \text{ if } (c_{w_{i-1}w_i} = 0)$$

$$\sum_{i:c(w_x w_i)>0} p^*(w_i|w_x) = \frac{c(w_x w_i)}{c(w_x) + T(w_x)}$$

Witten bell smoothing example

	a	b	c	d	...	Total = N(w1) nb seen tokens	T(w1) nb seen types	Z(w1) nb. unseen types
a	10	10	10	0		30	3	1
b	0	0	30	0		30	1	3
c	0	0	300	0		300	1	3
d								
...								

- all unseen bigrams starting with *a* will share a probability mass of

$$\frac{T(a)}{N(a)+T(a)} = \frac{3}{30+3} = 0.091$$

- each unseen bigrams starting with *a* will have an equal part of this

$$P(d|a) = \frac{1}{Z(a)} \times \frac{T(a)}{N(a)+T(a)} = \frac{1}{1} \times 0.091 = 0.091$$

Witten bell smoothing example

	a	b	c	d	...	Total = **N(w1)** nb seen tokens	**T(w1)** nb seen types	**Z(w1)** nb. unseen types
a	10	10	10	0		30	3	1
b	0	0	30	0		30	1	3
c	0	0	300	0		300	1	3
d								
...								

- all unseen bigrams starting with *b* will share a probability mass of

$$\frac{T(b)}{N(b)+T(b)} = \frac{1}{30+1} = 0.032$$

- each unseen bigrams starting with b will have an equal part of this

$$P(a|b) = \frac{1}{Z(b)} \times \frac{T(b)}{N(b)+T(b)} = \frac{1}{3} \times 0.032 = 0.011$$

$$P(b|b) = \frac{1}{Z(b)} \times \frac{T(b)}{N(b)+T(b)} = \frac{1}{3} \times 0.032 = 0.011$$

$$P(d|b) = \frac{1}{Z(b)} \times \frac{T(b)}{N(b)+T(b)} = \frac{1}{3} \times 0.032 = 0.011$$

Witten bell smoothing example

1st word	a	b	c	d	...	Total = N(w1) nb seen tokens	T(w1) nb seen types	Z(w1) nb. unseen types
a	10	10	10	0		30	3	1
b	0	0	30	0		30	1	3
c	0	0	300	0		300	1	3
d								
...								

- all unseen bigrams starting with c will share a probability mass of

$$\frac{T(c)}{N(c)+T(c)} = \frac{1}{300+1} = 0.0033$$

- each unseen bigrams starting with c will have an equal part of this

$$P(a|c) = \frac{1}{Z(c)} \times \frac{T(c)}{N(c)+T(c)} = \frac{1}{3} \times 0.0033 = 0.0011$$

$$P(b|c) = \frac{1}{Z(c)} \times \frac{T(c)}{N(c)+T(c)} = \frac{1}{3} \times 0.0033 = 0.0011$$

$$P(d|c) = \frac{1}{Z(c)} \times \frac{T(c)}{N(c)+T(c)} = \frac{1}{3} \times 0.0033 = 0.0011$$

Witten bell smoothing example

- For Seen bigrams:
- Since we added probability mass to unseen bigrams, we must decrease(**discount)** the probability mass of seen event (so that Total =1)
- We increased probability mass of all (not each) unseen events by a factor so we must discount by the same factor,

$$\frac{T}{N+T}$$

$$newProb(w_2 \mid w_1) = originalProb(w_2 \mid w_1) \times \left(1 - \frac{T(w_1)}{N(w_1) + T(w_1)}\right)$$

$$C(w_2 \mid w_1) = newProb(w_2 \mid w_1) \times N(w_1)$$

$$= originalProb(w_2 \mid w_1) \times \left(1 - \frac{T(w_1)}{N(w_1) + T(w_1)}\right) \times N(w_1)$$

$$= originalCount(w_2 \mid w_1) \times \frac{N(w_1) + T(w_1) - T(w_1)}{N(w_1) + T(w_1)}$$

$$= originalCount(w_2 \mid w_1) \times \frac{N(w_1)}{N(w_1) + T(w_1)} \quad // \text{ for seen}$$

Witten bell smoothing example

	I	want	to	eat	Chinese	food	lunch	...	N(w) seen bigram tokens	T(w) seen bigram types	Z(w) unseen bigram types
I	8	1087	0	13	0	0	0		3437	95	1521
want	3	0	786	0	6	8	6		1215	76	1540
to	3	0	10	860	3	0	12		3256	130	1486
eat	0	0	2	0	19	2	52		938	124	1492
Chinese	2	0	0	0	0	120	1		213	20	1592
food	19	0	17	0	0	0	0		1506	82	534
lunch	4	0	0	0	0	1	0		459	45	1571

- T(w)= number of different seen bigrams types starting with w
- we have a vocabulary of 1616 words, so we can compute
- Z(w)= number of unseen bigrams types starting with w

  $Z(w) = 1616 - T(w)$

- N(w) = number of bigrams tokens starting with w

Witten bell smoothing example

- the count of the unseen bigram "*I lunch*"

$$\frac{T(I)}{Z(I)} \times \frac{N(I)}{N(I)+T(I)} = \frac{95}{1521} \times \frac{3437}{3437+95} = 0.06$$

- the count of the seen bigram "*want to*"

$$\text{count(want to)} \times \frac{N(\text{want})}{N(\text{want})+T(\text{want})} = 786 \times \frac{1215}{1215+76} = 739.73$$

Witten-Bell smoothed bigram counts:

	I	*want*	*to*	*eat*	*Chinese*	*food*	*lunch*	...	**Total**
I	7.78	1057.76	.061	12.65	.06	.06	.06		**3437**
want	2.82	.05	739.73	.05	5.65	7.53	5.65		**1215**
to	2.88	.08	9.62	826.98	2.88	.08	12.50		**3256**
eat	.07	.07	19.43	.07	16.78	1.77	45.93		**938**
Chinese	1.74	.01	.01	.01	.01	109.70	.91		**213**
food	18.02	.05	16.12	.05	.05	.05	.05		**1506**
lunch	3.64	.03	.03	.03	.03	0.91	.03		**459**

Witten bell smoothing example

Witten-Bell conditional bigram probabilities - $P(w_2|w_1)$

	I	*want*	*to*	*eat*	*Chinese*	*food*	*lunch*	...	**Total**
I	.0022 (7.78/3437)	.3078	.000002	.0037	.000002	.000002	.000002		1
want	.00230	.00004	.6088	.00004	.0047	.0062	.0047		1
to	.00009	.00003	.0030	.2540	.00009	.00003	.0038		1
eat	.00008	.00008	.0021	.00008	.0179	.0019	.0490		1
Chinese	.00812	.00005	.00005	.00005	.00005	.5150	.0042		1
food	.0120	.00004	.0107	.00004	.00004	.00004	.00004		1
lunch	.0079	.00006	.00006	.00006	.00006	.0020	.00006		1

Witten bell smoothing example

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Witten bell smoothing

- Let the bi-gram types be

I	95
want	76
to	130
eat	124
Chinese	20
food	82
lunch	45

and

$$Z(w) = V - T(w)$$

Here are those Z values:

I	1,521
want	1,540
to	1,486
eat	1,492
Chinese	1,596
food	1,534
lunch	1,571

	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

Figure 6.4 Bigram counts for 7 of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ˜10,000 sentences.

	I	want	to	eat	Chinese	food	lunch
I	8	1060	.062	13	.062	.062	.062
want	3	.046	740	.046	6	8	6
to	3	.085	10	827	3	.085	12
eat	.075	.075	2	.075	17	2	46
Chinese	2	.012	.012	.012	.012	109	1
food	18	.059	16	.059	.059	.059	.059
lunch	4	.026	.026	.026	.026	1	.026

Figure 6.9 Witten-Bell smoothed bigram counts for 7 of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ˜10,000 sentences.

Good Turing Discounting

- To re-estimate the amount of probability mass to assign to N-grams with zero or low counts by looking at the number of N-grams with higher counts.

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

- Where c* based on the set of Nc for all c

c (MLE)	N_c	c^* (GT)
0	74,671,100,000	0.0000270
1	2,018,046	0.446
2	449,721	1.26
3	188,933	2.24
4	105,668	3.24
5	68,379	4.22
6	48,190	5.19
7	35,709	6.21
8	27,710	7.24
9	22,280	8.25

Figure 6.10 Bigram 'frequencies of frequencies' from 22 million AP bigrams, and Good-Turing re-estimations after Church and Gale (1991)

The first column shows the count *c*, i.e. the number of observed instances of a bigram.
The second column shows the number of bigrams that had this count. Thus 449,721 bigrams has a count of 2.
The third column shows the Good-Turing estimation

- the revised count for the bigrams that never occurred (c0) is estimating by dividing the number of bigrams that occurred once by the number of bigrams that SINGLETON never occurred (N0)

Backoff

- The discounting we have been discussing so far can help solve the problem of zero frequency n-grams.
- If we have no examples of a particular trigram $w_{n-2}w_{n-1}w_n$ to help us compute $P(w_n|w_{n-1}w_{n-2})$ we can estimate its probability by using the bigram probability $P(w_n|w_{n-1})$.
- Similarly, if we don't have counts to compute $P(w_n|w_{n-1})$ we can look to the unigram $P(w_n)$.
- There are two ways to rely on this N-gram 'hierarchy' viz., deleted interpolation and backoff.
- Backoff N-gram modeling is a nonlinear method introduced by Katz (1987).
- In the backoff model, like the deleted interpolation model, we build an N-gram model based on an (N-1)-gram model.
- The difference is that in backoff, if we have non-zero trigram counts, we rely solely on the trigram counts and don't interpolate the bigram and unigram counts at all.
- We only 'back off' to a lower-order N-gram if we have zero evidence for a higher-order N-gram.
- The trigram backoff version may be be represented as follows

$$\hat{P}(w_i|w_{i-2}w_{i-1}) = \begin{cases} P(w_i|w_{i-2}w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha_1 P(w_i|w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) = 0 \\ & \text{and } C(w_{i-1}w_i) > 0 \\ \alpha_2 P(w_i), & \text{otherwise.} \end{cases}$$

Deleted interpolation

- The deleted interpolation algorithm, due to Jelinek and Mercer (1980), combines different N-gram orders by linearly interpolating all three models whenever we are computing any trigram.
- We estimate the probability $P(w_n|w_{n-1}w_{n-2})$ by mixing together the unigram, bigram, and trigram probabilities.
- Each value is weighted with lambda

$$\begin{aligned}\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1 P(w_n|w_{n-1}w_{n-2}) \\ +\lambda_2 P(w_n|w_{n-1}) \\ +\lambda_3 P(w_n)\end{aligned}$$

such that the λs sum to 1:

$$\sum_i \lambda_i = 1$$

Evaluation and Perplexity

Entropy

- **Entropy** and **perplexity** are the most common metrics used to evaluate *N*-gram systems.
- **Entropy** is a measure of information, and is invaluable in natural language processing, speech recognition, and computational linguistics.
- It can be used as a metric for how much information there is in a particular grammar, for how well a given grammar matches a given language, for how predictive a given *N*-gram grammar is about what the next word could be.
- Given two grammars and a corpus, we can use entropy to tell us which grammar better matches the corpus.
- We can also use entropy to compare how difficult two speech recognition tasks are, and also to measure how well a given probabilistic grammar matches human grammars.

Perplexity

Perplexity is the inverse probability of the test set, normalized by the number of words.

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

Bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

Probability range is [0,1], perplexity range is [1,∞]

Minimizing perplexity is the same as maximizing probability

Perplexity

WSJ Corpus **Training:** 38 million words **Test:** 1.5 million words

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Perplexity for Bigram <S> I like college </S>

$=P(I \mid <S>) \times P(like \mid I) \times P(college \mid like) \times P(</S> \mid college)$

$=3/7 \times 3/6 \times 3/5 \times 3/3 = 9/70 = \mathbf{0.13}$

$\mathbf{PP(w) = (1/0.13)^{1/4} = 1.67}$

Perplexity for Trigram <S> I like college </S>

$P(w)=P(like \mid <S> I) \times P(college \mid I\ like) \times P(</S> \mid like\ college)$

$P(w) = 1/3 \times 2/3 \times 3/3 = 2/9 = \mathbf{0.22}$

$\mathbf{PP(w) = (1/0.22)^{1/3} = 1.66}$

Class based N-Grams

- **Class-based N-grams or cluster N-gram** is a variant of N-gram that uses information about word classes or clusters.
- **Class-based N-grams** are useful for dealing with sparsity in the training data.
- Suppose for a flight reservation system we want to compute the probability of the bigram to Shanghai, but this bigram never occurs in the training set. Instead, our training data has to London, to Beijing, and to Denver. If we knew that these were all cities, and assuming Shanghai does appear in the training set in other contexts, we could predict the likelihood of a city following from.
- There are many variants of cluster N-grams. The simplest one is sometimes known IBM clustering as IBM clustering, after its originators (Brown et al., 1992).
- IBM clustering is a kind of hard clustering, in which each word can belong to only one class. The model estimates the conditional probability of a word w_i by multiplying two factors: the probability of the word's class ci given the preceding classes (based on an N-gram of classes), and the probability of wi given ci. Here is the IBM model in bigram form:
- $P(w_i|w_{i-1}) \approx P(c_i|c_{i-1}) \times P(w_i|c_i)$

Class based N-Grams

- If we had a training corpus in which we knew the class for each word, the maximum likelihood estimate (MLE) of the probability of the word given the class and the probability of the class given the previous class could be computed as follows:
  - $$P(w|c) = \frac{C(w)}{C(c)}$$
  - $$P(c_i|c_{i-1}) = \frac{C(c_{i-1}c_i)}{\sum_c C(c_{i-1}c_i)}$$
- Cluster N-grams are used in two ways:
- In dialog systems, we often hand-design domain-specific word classes. Thus for an airline information system, we might use classes like CITYNAME, AIRLINE, DAYOFWEEK, or MONTH.
- In other cases, we can automatically induce the classes by clustering words in a corpus. Syntactic categories like part-of-speech tags don't seem to work well as classes.
- Whether automatically induced or hand-designed, cluster N-grams are generally mixed with regular word-based N-grams

Google N-gram Release

- Recent research in language modeling has focused on adaptation, on the use of sophisticated linguistic structures based on syntactic and dialogue structure, and on very very large N-grams.
- In 2006, Google publicly released a very large set of N-grams that is a useful research resource, consisting of all the five-word sequences that appear at least 40 times from 1,024,908,267,229 words of running text;
- There are 1,176,470,663 five-word sequences using over 13 million unique words types.
- Large language models generally need to be pruned to be practical, using techniques such as Stolcke (1998) and Church et al. (2007)

Training and Test Sets

- The probabilities of an N-gram model come from the corpus it is trained on. In general, the parameters of a statistical model are trained on some set of data, and then we apply the models to some new data in some task (such as speech recognition) and see how well they work.
- This training-and-testing paradigm can also be used to evaluate different N-gram architectures.
- If we want to compare different language models (such as those based on N-grams of different orders N, or using the different smoothing algorithms, we can begin by dividing the corpus into training and test set. Then we train the two different N-gram models on the training set and see which one better models the test set. **Perplexity** metric can be computed and whichever model has higher probability for Perplexity is a better model.
- **Training on the test set :** While computing probability on a test sentence, If our test sentence is part of the training corpus, we will mistakenly assign it an artificially high probability when it occurs in the test set. We call this situation **training on the test set**.
- Training on the test set introduces a bias that makes the probabilities all look too high and causes huge inaccuracies in perplexity.
- **Held-out set:** Sometimes we need an extra source of data to augment the training set. Such extra data is called a **held-out set** because we hold it out from our training set when we train our N-gram counts. The held-out corpus is then used to set some other parameters; for example we will see the use of held-out data to set interpolation weights in interpolated N-gram models.

Training and Test Sets

- **Development test:** Finally, sometimes we need to have multiple test sets. This happens because we might use a particular test set so often that we implicitly tune to its characteristics. Then we would definitely need a fresh test set which is truly unseen. In such cases, we call the initial test set the **development test set** or, **devset.**
- We want our test set to be as large as possible and a small test set may be accidentally unrepresentative. On the other hand, we want as much training data as possible.
- At the minimum, we would want to pick the smallest test set that gives us enough statistical power to measure a statistically significant difference between two potential models.
- In practice, we often just divide our data into 80% training, 10% development, and 10% test.
- Given a large corpus that we want to divide into training and test, test data can either be taken from some continuous sequence of text inside the corpus, or we can remove smaller “stripes” of text from randomly selected parts of our corpus and combine them into test set.