N-GRAMS

Speech and Language Processing, chapter6

Presented by Louis Tsai
CSIE, NTNU
louis@csie.ntnu.edu.tw
2003/03/18

N-grams

- What word is likely to follow this sentence fragment?

  I'd like to make a collect…

  Probably most of you concluded that a very likely word is *call*, although it's possible the next word could be *telephone*, or *person-to-person* or *international*

N-grams

- Word prediction
  - speech recognition, hand-writing recognition, augmentative communication for the disabled, and spelling error detection
- In such tasks, word-identification is difficult because the input is very noisy and ambiguous.
- Looking at previous word can give us an important cue about the next ones are going to be

N-grams

- Example:*Take the Money and Run*
  sloppily written hold-up note “I have a gub”
- A speech recognition system (and a person) can avoid this problem by their knowledge of word sequences (“a gub” isn’t an English word sequence) and of their probabilities (especially in the context of a hold-up, “I have a gun” will have a much higher probability than “I have a gub” or even “I have a gull”)

N-grams

- Augmentative communication system for the disabled
- People who are unable to use speech of sign-language to communicate, use systems that speak for them, letting them choose words with simple hand movements, either by spelling them out, or by selecting from a menu of possible words
- Spelling is very slow, and a menu can't show all possible English words on one screen
- Thus it is important to be able to know which words the speaker is likely to want next, then put those on the menu

N-grams

- Detecting real-word spelling errors
  - They are leaving in about fifteen *minuets* to go to her house
  - The study was conducted mainly *be* John Black
  - Can they *lave* him my messages?
  - He is trying to *fine* out
- We can't find those errors by just looking for words that aren't in the dictionary
- Look for low probability combinations (they lave him, to fine out)

N-grams

- Probability of a sequence of words

  …all of a sudden I notice three guys standing on the sidewalk taking a very good long gander at me

  with the same set of words in a different order probably has a very low probability

  good all I of notice a taking sidewalk the me long three at sudden guys gander on standing a a the very

N-grams

- An *N*-gram model uses the previous *N*-1 words to predict the next one
- In speech recognition, it is traditional to use the term **language model** or **LM** for such statistical models of word sequences

Counting Words in Corpora

- Probabilities are based on counting thing
- For computing word probabilities, we will be counting words in a training corpus
- Brown Corpus, a 1 million word collection of samples from 500 written texts from different genres (newspaper, novels, etc), which was assembled at Brown University in 1963-64

Counting Words in Corpora

- He stepped out into the hall, was delighted to encounter a water brother. (6.1)
- (6.1) has 13 words if we don't count punctuation-marks as words, 15 if we count punctuation
- In natural language processing applications, question-marks are an important cue that someone has asked a question

Counting Words in Corpora

- Corpora of spoken language usually don't have punctuation
- I do uh main- mainly business data processing (6.2)
- Fragments: words that are broken off in the middle (main-)
- filled pauses : uh
- Should we consider there to be words?

Counting Words in Corpora

- We might want to strip out the fragments
- *uh*s and *um*s are in fact much more like words
- Generally speaking *um* is used when speakers are having major planning problems in producing an utterance, while *uh* is used when they know what they want to say, but are searching for the exact words to express it

Counting Words in Corpora

- Are *They* and *they* the same word?

- How should we deal with inflected forms like *cats* vs. *cat*?
- Wordform : cats and cat are treated as two words
- Lemma : cats and cat are the same word

Counting Words in Corpora

- How many word are there in English?
- Types : the number of distinct word in a corpus
- Tokens : the total number of running words

- They picnicked by the pool, then lay back on the grass and looked at the stars. (6.3)
- (6.3) has 16 word tokens and 14 word types (not counting punctuation)

Simple (Unsmoothed) N-grams

- The simplest possible model of word sequences would simply let any word of the language follow any other word

  If English had 100,000 words, the probability of any word following any other word would be 1/100,000 or .00001

- In a slightly more complex model of word sequences, any word could follow any other word, but the following word would appear with its normal frequency of occurrence

  the occurs 69,971 times in the Brown corpus of 1,000,000 words, 7% of the words in this particular corpus are *the*; *rabbit* occurs only 11 times in the Brown corpus

Simple (Unsmoothed) N-grams

- We can use the probability .07 for *the* and .00001 for *rabbit* to guess the next word
- But suppose we've just seen the following string:

  Just the, the white

  In this context, *rabbit* seems like a more reasonable word to follow *white* than *the* does
- $P(rabbit|white)$

Simple (Unsmoothed) N-grams

$$P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)...P(w_n \mid w_1^{n-1})$$

$$= \prod_{k=1}^{n} P(w_k \mid w_1^{k-1}) \qquad (6.5)$$

- But how can we compute probabilities like $P(w_n|w_1{}^{n-1})$?

  We don't know any easy way to compute the probability of a word given a long sequence of preceding words

Simple (Unsmoothed) N-grams

- We *approximate* the probability of a word given all the previous words
- The probability of the word given the single previous word! → **bigram**
  用$P(w_n|w_{n-1})$來近似$P(w_n|w_1^{n-1})$
- P(rabbit | Just the other I day I saw a) (6.6)
  $\doteqdot$ P(rabbit | a) (6.7)
- This assumption that the probability of a word depends only on the previous word is called a **Markov** assumption

Simple (Unsmoothed) N-grams

- The general equation for the N-gram approximation to the conditional probability of the next word in a sequence is

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1}) \qquad (6.8)$$

- For a bigram grammar, we compute the probability of a complete string

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-1}) \qquad (6.9)$$

Simple (Unsmoothed) N-grams

- Berkeley Restaurant Project
  - I'm looking for Cantonese food.
  - I'd like to eat dinner someplace nearby.
  - Tell me about Chez Panisse.
  - Can you give me a listing of the kinds of food that are available?
  - I'm looking for a good place to eat breakfast.
  - I definitely do not want to have cheap Chinese food.
  - When is Caffe Venezia open during the day?
  - I don't wanna walk more than ten minutes.

Simple (Unsmoothed) N-grams

eat on	.16	eat Thai	.03
eat some	.06	eat breakfast	.03
eat lunch	.06	eat in	.02
eat dinner	.05	eat Chinese	.02
eat at	.04	eat Mexican	.02
eat a	.04	eat tomorrow	.01
eat Indian	.04	eat dessert	.007
eat today	.03	eat British	.001

Figure 6.2 A fragment of a bigram grammar from the Berkeley Restaurant Project showing the most likely words to follow *eat*.

Simple (Unsmoothed) N-grams

<s>I	.25	I want	.32	want to	.65	to eat	.26	British food	.60
<s>I'd	.06	I would	.29	want a	.05	to have	.14	British restaurant	.15
<s>Tell	.04	I don't	.08	want some	.04	to spend	.09	British cuisine	.01
<s>I'm	.02	I have	.04	want thai	.01	to be	.02	British lunch	.01

Figure 6.3 More fragments from the bigram grammar from the Berkeley Restaurant Project.

- P(I want to eat British food)
  $= P(\text{I}|\text{<s>})\ P(\text{want}|\text{I})\ P(\text{to}|\text{want})\ P(\text{eat}|\text{to})$
  $P(\text{British}|\text{eat})\ P(\text{food}|\text{British})$
  $= .25 * .32 * .35 * .26 * .002 * .60$
  $= .000016$

Simple (Unsmoothed) N-grams

- Since probabilities are all less than 1, the product of many probabilities gets smaller the more probabilities we multiply → **logprob**
- A **trigram** model condition on the two previous words (e.g., *P*(*food* | *eat British*))
- First trigram : use two pseudo-words
  P(*I* | <*start1*><*start2*>)

Simple (Unsmoothed) N-grams

- **Normalizing** means dividing by some total count so that the resulting probabilities fall legally between 0 and 1

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \quad (6.10)$$

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \quad (6.11)$$

$$P(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})} \quad (6.12)$$

Simple (Unsmoothed) N-grams

	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
luhch	4	0	0	0	0	1	0

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

Simple (Unsmoothed) N-grams

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

Simple (Unsmoothed) N-grams

	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
luhch	.0087	0	0	0	0	.0022	0

Figure 6.5 Bigram probabilities for seven of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ≒10,000 sentences.

More on N-grams and Their Sensitivity to the Training Corpus

- Two important facts about N-grams:
- (1) The increasing accuracy of N-gram models as we increase the value of N
- (2) It is very strong dependency on their training corpus

- Let's train various N-grams and then use each to generate random sentences

Unigram approximation to Shakespeare

- (a) To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have.
- (b) Every enter now severally so, let
- (c) Hill he late speaks; or! A more to leg less first you enter
- (d)Will rash been and by I the me loves gentle me not slavish page, the and hour; ill let
- (e) Are where exeunt and sighs have rise excellency took of.. Sleep knave we, near; vile like

Bigram approximation to Shakespeare

- (a)What means, sir. I confess she? Then all sorts, he is trim, captain.
- (b)Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
- (c)What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?
- (d)Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff!! Exeunt
- (e)Thou whoreson chops. Consumption catch your dearest friend, well, and I know where many mouths upon my undoing all but be, how soon, then; we'll execute upon my love's bonds and we do you will?
- (f)The world shall- my lord!

Trigram approximation to Shakespeare

- (a)Sweet prince, Falstaff shall die. Harry of Monmouth's grave.
- (b)This shall forbid it should be branded, if renown made it empty.
- (c)What is't that cried?
- (d)Indeed the duke; and had a very good friend.
- (e)Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
- (f)The sweet! How many then shall posthumus end his miseries.

Quadrigram approximation to Shakespeare

- (a)King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
- (b)Will you not tell me who I am?
- (c)It cannot be but so.
- (d)Indeed the short and the long. Marry, 'tis a noble Lepidus
- (e)They say all lovers swear more performance than they are wont to keep obliged faith unforfeited!
- (f)Enter Leonato's brother Antonio, and the rest, but seek the weary beds of people sick.

More on N-grams and Their Sensitivity to the Training Corpus

- The longer the context on which we train the model, the more coherent the sentences
- In the unigram sentences, there is no coherent relation between words, and in fact none of the sentences end in a period or other sentence-final punctuation
- The bigram sentences can be seen to have very local word-to-word coherence
- The trigram and quadrigram sentences are beginning to look a lot like Shakespeare

Smoothing

- One major problem with standard *N*-gram models is that they must be trained from some corpus, and because any particular training corpus is finite, some perfectly acceptable English *N*-grams are bound to be missing from it
- Smoothing : reevaluating some of the zero-probability and low-probability *N*-grams, and assigning them non-zero values

Add-One Smoothing

- Add one to all the counts
- Unsmoothed MLE: dividing the count of the word by the total number of word token N

$$P(w_x) = \frac{c(w_x)}{\sum_i c(w_i)} = \frac{c(w_x)}{N}$$

- Add-one smoothing

$$c_i^* = (c_i + 1)\frac{N}{N+V} \qquad (6.13)$$

normalization factor $\frac{N}{N+V}$,where V is the total number of word types in the language

Add-One Smoothing

- Discounting
- discount d_c

$$d_c = \frac{c^*}{c}$$

- Counts can be turned into probabilities P_i^* by normalizing by N

$$P_i^* = \frac{c_i + 1}{N + V}$$

Add-One 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
luhch	5	1	1	1	1	2	1

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

Add-One Smoothing

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \quad (6.14)$$

$$p^*(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V} \quad (6.15)$$

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

$V = 1616$

Add-One Smoothing

	I	want	to	eat	Chinese	food	lunch
I	.0018	.22	.00020	.0028	.00020	.00020	.00020
want	.0014	.00035	.28	.00035	.0025	.0032	.0025
to	.00089	.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
luhch	.0024	.00048	.00048	.00048	.00048	.00096	.00048

Figure 6.7 Add-One Smoothed bigram probabilities for seven of the words (out of 1616 total word types) in the Berkeley Restaurant Project corpus of ≒10,000 sentences.

Witten-Bell Discounting

- **Key Concept—Things Seen Once:** Use the count of things you've seen once to help estimate the count of things you've never seen
- So we estimate the *total* probability mass of all the zero *N*-grams with the number of types divided by the number of tokens plus observed types:

$$\sum_{i:c_i=0} p_i^* = \frac{T}{N+T} \qquad (6.16)$$

Witten-Bell Discounting

- (6.16) gives the total "probability of unseen *N*-grams", we need to divide this up among all the zero *N*-grams
- We could just choose to divide it equally

$$Z = \sum_{i:c_i=0} 1 \qquad (6.17)$$

Z is the total number of *N*-grams with count zero

$$p_i^* = \frac{T}{Z(N+T)} \qquad (6.18)$$

Witten-Bell Discounting

$$p_i^* = \frac{c_i}{N+T} \text{if}(c_i > 0) \tag{6.19}$$

$$c_i^* = \begin{cases} \frac{T}{Z}\frac{N}{N+T}, & \text{if} \quad c_i = 0 \\ c_i \frac{N}{N+T}, & \text{if} \quad c_i > 0 \end{cases} \tag{6.20}$$

Witten-Bell Discounting

- For bigram

$$\sum_{i:c(w_x w_i)=0} p^*(w_i \mid w_x) = \frac{T(w_x)}{N(w_x)+T(w_x)} \qquad (6.21)$$

T: the number of bigram types, N: the number of bigram token

$$Z(w_x) = \sum_{i:c(w_x w_i)=0} 1 \qquad (6.22)$$

$$p^*(w_i \mid w_{i-1}) = \frac{T(w_{i-1})}{Z(w_{i-1})(N+T(w_{i-1}))} \quad \text{if}(c_{w_{i-1}w_i} = 0) \qquad (6.23)$$

$$\sum_{i:c(w_x w_i)>0} p^*(w_i \mid w_x) = \frac{c(w_x w_i)}{c(w_x)+T(w_x)} \qquad (6.24)$$

Witten-Bell Discounting

	$T(w)$	$Z(w) = V - T(w)$	V=1616
I	95	1521	
want	76	1540	
to	130	1486	
eat	124	1492	
Chinese	20	1596	
food	82	1534	
lunch	45	1571	

Witten-Bell Discounting

	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	872	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
luhch	4	.026	.026	.0026	.026	1	.026

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

Good-Turing Discounting

- 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
- N_0 is the number of bigrams b of count 0,
  N_1 is the number of bigrams with count 1, and so on:

$$N_c = \sum_{b:c(b)=c} 1 \qquad (6.26)$$

$$c^* = (c+1)\frac{N_{c+1}}{N_c} \qquad (6.27)$$

Good-Turing Discounting

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).

Good-Turing Discounting

$$c^* = c \quad \text{for} \quad c > k$$

Katz (1987) suggests setting k at 5

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}} \qquad (6.29)$$

Backoff

$$\hat{P}(w_i \mid w_{i-2}w_{i-1}) = \begin{cases} P(w_i \mid w_{i-2}w_{i-1}) & \text{if } C(w_{i-2}w_{i-1}w_i)>0 \\ \alpha_1 P(w_i \mid w_{i-1}) & \text{if } C(w_{i-2}w_{i-1}w_i)=0 \text{ and } C(w_{i-1}w_i)>0 \\ \alpha_2 P(w_i) & \text{Otherwise.} \end{cases} \quad (6.30)$$

$$\hat{P}(w_n \mid w_{n-N+1}^{n-1}) = \tilde{P}(w_n \mid w_{n-N+1}^{n-1}) + \theta(P(w_n \mid w_{n-N+1}^{n-1}))\alpha\hat{P}(w_n \mid w_{n-N+2}^{n-1}) \quad (6.31)$$

$$\theta(x) = \begin{cases} 1, \text{if } x = 0 \\ 0, \text{otherwise.} \end{cases} \quad (6.32)$$

Combining Backoff with Discounting

- Discounting：how much total probability mass to set aside for all the events we haven't seen
- Backoff：how to distribute this probability in a clever way

Combining Backoff with Discounting

$$\hat{P}(w_n \mid w_{n-N+1}^{n-1}) = \tilde{P}(w_n \mid w_{n-N+1}^{n-1}) + \theta(P(w_n \mid w_{n-N+1}^{n-1})) \bullet \alpha(w_{n-N+1}^{n-1})\hat{P}(w_n \mid w_{n-N+2}^{n-1}) \quad (6.34)$$

- The $\tilde{P}$ comes form our need to discount the MLE probabilities to save some probability mass for the lower order *N*-grams
- The α is used to ensure that the probability mass from all the lower order *N*-grams sums up to exactly the amount that we saved by discounting the higher-order *N*-grams

Combining Backoff with Discounting

$$\widetilde{P}(w_n \mid w_{n-N+1}^{n-1}) = \frac{c^*(w_{n-N+1}^{n})}{c(w_{n-N+1}^{n-1})} \qquad (6.35)$$

- This probability $\widetilde{P}$ will be slightly less than the MLE estimate

$$\frac{c(w_{n-N+1}^{n})}{c(w_{n-N+1}^{n-1})}$$

this will leave some probability mass for the lower order N-grams

Combining Backoff with Discounting

- Let's represent the total amount of left-over probability mass by the function β, a function of the *N*-1 gram context

$$\beta(w_{n-N+1}^{n-1}) = 1 - \sum_{w_n : c(w_{n\_N+1}^{n})>0} \tilde{P}(w_n \mid w_{n-N+1}^{n-1}) \qquad (6.36)$$

this gives us the total probability mass that we are ready to distribute to all *N*-1-gram (e.g., bigrams if our original model was a trigramm)

Combining Backoff with Discounting

- How much probability mass to distribute from an *N*-gram to an *N*-1-gram is represented by the function α:

$$\alpha(w_{n-N+1}^{n-1}) = \frac{1 - \sum_{w_n : c(w_{n-N+1}^{n}) > 0} \tilde{P}(w_n \mid w_{n-N+1}^{n-1})}{1 - \sum_{w_n : c(w_{n-N+1}^{n}) > 0} \tilde{P}(w_n \mid w_{n-N+2}^{n-1})} \tag{6.37}$$

Combining Backoff with Discounting

- When the counts of an N-1-gram context are 0, (i.e., when $c(w_{n-N+1}^{n-1}) = 0$)

$$P(w_n \mid w_{n-N+1}^{n-1}) = 0 \qquad (6.38)$$

$$\widetilde{P}(w_n \mid w_{n-N+1}^{n-1}) = 0 \qquad (6.39)$$

$$\beta(w_{n-N+1}^{n-1}) = 1 \qquad (6.40)$$

Combining Backoff with Discounting

- backoff model in trigram version:

$$\hat{P}(w_i \mid w_{i-2}w_{i-1}) = \begin{cases} \widetilde{P}(w_i \mid w_{i-2}w_{i-1}), & \text{if } c(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha(w_{n-2}^{n-1})\widetilde{P}(w_i \mid w_{i-1}), & \text{if } c(w_{i-2}w_{i-1}w_i) = 0 \\ & \text{and } c(w_{i-1}w_i) > 0 \\ \alpha(w_{n-1})\widetilde{P}(w_i), & \text{otherwise.} \end{cases}$$

- In practice, when discounting, we usually ingore counts of 1, that is, we treat *N*-grams with a count 1 as if they never occurred

Deleted Interpolation

- Combines different *N*-gram orders by linearly interpolating all tree models whenever we are computing any trigram

$$\begin{aligned}\hat{P}(w_n \mid w_{n-2}w_{n-1}) = \lambda_1 P(w_n \mid w_{n-2}w_{n-1}) \\ + \lambda_2 P(w_n \mid w_{n-1}) \\ + \lambda_3 P(w_n)\end{aligned} \tag{6.41}$$

$$\sum_i \lambda_i = 1 \tag{6.42}$$

Deleted Interpolation

- If we have particularly accurate counts for a particular bigram, we assume that the counts of the trigrams based on this bigram will be more trustworthy, and so we can make the lambdas for those trigrams higher and thus give that trigram more weight in the interpolation

$$\begin{aligned}\hat{P}(w_n \mid w_{n-2}w_{n-1}) = {} & \lambda_1(w_{n-2}^{n-1})P(w_n \mid w_{n-2}w_{n-1}) \\ & + \lambda_2(w_{n-2}^{n-1})P(w_n \mid w_{n-1}) \\ & + \lambda_3(w_{n-2}^{n-1})P(w_n)\end{aligned} \tag{6.43}$$

Context-Sensitive Spelling Error Correction

- Detecting Spelling errors by looking for words that are not in a dictionary, are not generated by some finite-state model of English word-formation, or have low probability
- Typographical errors (insertion, deletion, transposition) accidentally produce a real word (e.g., *there* for *three*)
- Writer substituted the wrong spelling of a homophone or near-homophone (e.g., *dessert* for *desert*, or *piece* for *peace*)
- The task of correcting these error is called **context-sensitive spelling error correction**

Context-Sensitive Spelling Error Correction

- How important are these errors?

  Single typographical errors (single insertions, deletions, substitutions, or transpositions), Peterson (1986) estimates that 15% of such spelling errors produce valid English words (given a very large list of 350,000 words)

  Kukich (1992) summarizes a number of other analyses based on empirical studies of corpora, which give figures between of 25% and 40% for the percentage of errors that are valid English words

Context-Sensitive Spelling Error Correction

- Local errors are those that are probably detectable from the immediate surrounding words
- Global errors are ones in which error detection requires examination of a large context

Context-Sensitive Spelling Error Correction

Local Errors
The study war conducted mainly *be* John Black. They are leaving in about fifteen *minuets* to go to her house. The design *an* construction of the system will take more that a year. Hopefully, all *with* continue smoothly in my absence. Can they *lave* him my messages? I need to *notified* the bank of [this problem.] He *need* to go there right *no w*. He is trying to *fine* out.
Global Errors
Won't they *heave if* next Monday at that time? This thesis is supported by the fact that since 1989 the system has been operating *system* with all four units on-line, but…
Figure 6.11 Some attested real-word spelling errors from Kukich (1992), broken down into **local** and **global** errors.

Context-Sensitive Spelling Error Correction

- Based on *N*-grams
  to generate every possible misspelling of each word in a sentence either by typographical modifications, or by including homophones, and then choosing the spelling that gives the sentence the highest prior probability
- Given a sentence $W=\{w_1,w_2,\ldots,w_k,\ldots,w_n\}$, where w_k has alternative spelling w_k', w_k'', etc., we choose the speeling among these possible spellings that maximizes $P(W)$, using the *N*-gram grammar to compute $P(W)$

Entropy

- Computing entropy requires that we establish a random variable X that ranges over whatever we are predicting (words, letters, parts of speech, the set of which we'll call χ), and that has a particular probability function, call it $p(x)$. The entropy of this random variable X is then

$$H(X) = -\sum_{x \in \chi} p(x) \log_2 p(x) \qquad (6.44)$$

- The log can in principle be computed in any base; we use log base 2, the result of this is that the entropy is measured in **bits**

Entropy

- Thinking of the entropy as a lower bound on the number of bits it would take to encode a certain decision or piece of information in the optimal coding scheme
- Imagine that we want to place a bet on a house race but it is too far to go all the way to Yonkers Racetrack, and we'd like to send a short message to the bookie to tell him which horse to bet on. Suppose there are *eight* horses in this particular race
- One way to encode this message is just to use the binary representation of the horse's number as the code; thus horse 1 would be 001, horse 2 010, and so on, with horse 8 coded as 000. On the average we would be sending 3 bits per race

Entropy

- Can we do better?
  The prior probability of each horse as follows:

Horse 1	½	Horse 5	1/64
Horse 2	¼	Horse 6	1/64
Horse 3	1/8	Horse 7	1/64
Horse 4	1/16	Horse 8	1/64

- The entropy of the random variable X that ranges over horses gives us a lower bound on the number of bits, and is:

$$H(X) = -\sum_{i=1}^{i=8} p(i)\log p(i)$$

$$= -\tfrac{1}{2}\log\tfrac{1}{2} - \tfrac{1}{4}\log\tfrac{1}{4} - \tfrac{1}{8}\log\tfrac{1}{8} - \tfrac{1}{16}\log\tfrac{1}{16} - 4(\tfrac{1}{64}\log\tfrac{1}{64})$$

$$= 2\text{bits} \qquad (6.45)$$

Entropy

- A code that averages 2 bits per race can be built by using short encodings for more probable horses, and longer encodings for less probable horses

- What if the horses are equally likely?

$$H(x) = -\sum_{i=1}^{i=8} \tfrac{1}{8} \log \tfrac{1}{8} = -\log \tfrac{1}{8} = 3\text{bits} \qquad (6.46)$$

Entropy

- The value 2^H is called the **perplexity**
- Perplexity can be intuitively thought of as the weighted average number of choices a random variable has to make

  H=3 bits, the perplexity is 2^3 or 8
  H=2 bits, the perplexity is 2^2 or 4

Entropy

- Compute the entropy of some sequence of words $W=\{\ldots w_0,w_1,w_2,\ldots,w_n\}$:
  we can computer the entropy of a random variable that ranges over all finite sequences of words of length b in some language L as follows:

$$H(w_1, w_2, ..., w_n) = -\sum_{W_1^n \in L} p(W_1^n) \log p(W_1^n) \qquad (6.47)$$

- We could define the **entropy rate** (**per-word entropy**)

$$\frac{1}{n} H(W_1^n) = -\frac{1}{n} \sum_{W_1^n \in L} p(W_1^n) \log p(W_1^n) \qquad (6.48)$$

Entropy

- But to measure the true entropy of a language, we need to consider sequences of infinite length. The entropy rate $H(L)$ is defined as:

$$H(L) = \lim_{n\to\infty} \frac{1}{n} H(w_1, w_2, ..., w_n)$$

$$= \lim_{n\to\infty} -\frac{1}{n} \sum_{W_1^n \in L} p(w_1, ..., w_n) \log p(w_1, ..., w_n) \qquad (6.49)$$

- The Shannon-McMillan-Breiman theorem states that if the language is regular in certain ways (stationary and ergodic)

$$H(L) = \lim_{n\to\infty} -\frac{1}{n} \log p(w_1, ..., w_n) \qquad (6.50)$$

Entropy

- That is, we can take a single sequence that is long enough instead of summing over all possible sequences
- The intuition of the Shannon-McMillan-Breiman theorem is that a long enough sequence of words will contain in it many other shorter sequences, and that each of these shorter sequences will reoccur in the longer sequence according to their probabilities
- A stochastic process is said to be **stationary** if the probabilities it assigns to a sequence are invariant wit respect to shifts in the time index
  → Markov models and *N*-grams are stationary
  in a bigram, P_i is dependent only on P_{i-1}, if we shift time index by x, P_{i+x} is still dependent on P_{i+x-1}

Entropy

- Natural language is not stationary, the probability of upcoming words can be dependent on events that were arbitrarily distant and time dependent

Cross Entropy for Comparing Models

- **Cross entropy**
  When we don't know the actual probability distribution p that generated some data. It allow us to use some m, which is a model of p (i.e., an approximation to p). The cross-entropy of m on p is defined by:

$$H(p,m) = \lim_{n\to\infty} -\frac{1}{n} \sum_{W \in L} p(w_1,...,w_n) \log m(w_1,...,w_n) \qquad (6.51)$$

- That is we draw sequences according to the probability distribution p, bus sum the log of their probability according to m

Cross Entropy for Comparing Models

- Following the Shannon-McMillan-Beriman theorem, for a stationary ergodic process:

$$H(p,m) = \lim_{n\to\infty} -\frac{1}{n}\log m(w_1, w_2, ..., w_n) \qquad (6.52)$$

- Cross entropy $H(p,m)$ is an upper bound on the entropy $H(p)$. For any model m:

$$H(p) \leqq H(p,m) \qquad (6.53)$$

Cross Entropy for Comparing Models

- The more accurate m is, the closer the cross entropy $H(p,m)$ will be to the true entropy $H(p)$
- The difference between $H(p,m)$ and $H(p)$ is a measure of how accurate a model is
- Between two models m_1 and m_2, the more accurate model will be the one with the lower cross-entropy
- The cross-entropy can never be lower than the true entropy, so a model cannot err by underestimating the true entropy

The Entropy of English

- Shannon's (1951) idea was to use human subjects, and to construct a psychological experiment that requires them to guess strings of letters; by looking at how many guesses it takes them to guess letters correctly we can estimate the probability of the letters, and hence the entropy of the sequence
- We record the number of guesses it takes for the subject to guess correctly
- Shannon's insight was that the entropy of the number-of-guesses sequence is the same as the entropy of English
- Shannon reported an entropy of 1.3 bots (for 27 characters (26 letters plus space))

The Entropy of English

$$H(\text{English}) \leq \lim_{n\to\infty} -\frac{1}{n}\log m(w_1 w_2 \ldots w_n) \qquad (6.54)$$

- Brown et al. (1992) trained a trigram language model on 583 million words of English, (293,181 different types) and used it to compute the probability of the entire Brown corpus (1,014,312 tokens)
- They obtained an entropy of 1.75 bits per character (where the set of characters included all the 95 printable ASCII characters)

The Entropy of English

- The average length of English written words (including space) has been reported at 5.5 letters (Nadas, 1984)
- If this is correct, it means that the Shannon estimate of 1.3 bets per letter corresponds to a per-word perplexity of 142 for general English

$$2^{1.3\times 5.5} = 142$$