

N-GRAMS

Date : 2004/04/16

Present : Yao-Min Huang

Reference : Speech and Language Processing ch6

Outline

- N-grams
- Simple (Unsmoothed) N-grams
- Smoothing
- Backoff
- Interpolation
- Context-Sensitive Spelling Error Correction
- Entropy

N-grams

- Intuition
 - What word is likely to follow this sentence fragment?
Ex : I'd like to make a collect...
- Application
 - Word Prediction
 - In such tasks, word-identification is difficult because the input is very noisy and ambiguous.
 - Augmentative communication system for the disabled
 - It is important to be able to know which words the speaker is likely to want next, then put those on the menu
 - Detecting real-word spelling errors
 - We can't find those errors by just looking for words that aren't in the dictionary

N-grams

- Definition
 - 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

N-grams

- 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

N-grams

- Counting Words in Corpora
 - count punctuation-marks as words ?
 - *uhs* and *ums* are 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
 - Are *They* and *they* the same word?
 - inflected forms like *cats* vs. *cat*
 - **Wordform** : cats and cat are treated as two words
 - **Lemma** : cats and cat are the same word

N-grams

- 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
 - Ex :

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

- Model
 - let any word of the language follow any other word
 - any word could follow any other word, but the following word would appear with its normal frequency of occurrence
- Condition Probability Model
 - Model : $P(w_1^n) = P(w_1)P(w_2 | w_1)P(w_3 | w_1^2) \dots P(w_n | w_1^{n-1})$
$$= \prod_{k=1}^n P(w_k | w_1^{k-1})$$
 - Problem : $P(w_n | w_1^{n-1}) \rightarrow$ hard to compute
 - Solution : $P(w_n | w_{n-1}) \rightarrow$ approximate the probability of a word given all the previous words

Simple (Unsmoothed) N-grams

- **Markov assumption**

- assumption that the probability of a word depends only on the previous word

- The general equation for the N-gram

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

- For a Bigram grammar

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

- Calculation skill

- **Logprob**

- Since probabilities are all less than 1, the product of many probabilities gets smaller the more probabilities we multiply

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.

- *Bigram Ex* :

$P(\text{I want to eat British food})$

$= P(\text{I}|\text{<s>}) P(\text{want}|\text{I}) P(\text{to}|\text{want}) P(\text{eat}|\text{to})P(\text{British}|\text{eat}) P(\text{food}|\text{British})$

$= .25 * .32 * .35 * .26 * .002 * .60$

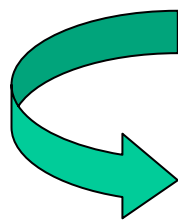
$= .000016$

- A **trigram** model condition on the two previous words (e.g., $P(\text{food} | \text{eat British})$)

Simple (Unsmoothed) N-grams

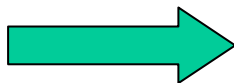
- N-gram models can be trained by counting and normalizing
 - **Normalizing** means dividing by some total count so that the resulting probabilities fall legally between 0 and 1

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



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

Simplify



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

Generalization

Simple (Unsmoothed) N-grams

- Two important facts about N-grams
 - The increasing accuracy of N-gram models as we increase the value of N (have a plateau)
 - It is very strong dependency on their training corpus
- Observation : various N-grams & use each to generate random sentences
 - 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
 - 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 (original content)

Smoothing

- Intuition
 - Problem :
 - trained from some corpus → corpus is finite → missing from it
 - Solution : Smoothing
 - reevaluating some of the zero-probability and low-probability N -grams, and assigning them non-zero values
 - In the following , we introduce three method
 - Add-one smoothing
 - Witten-Bell Discounting
 - Good Turning Discounting

Add-One Smoothing

N : tokens


V : types

c_i^* : count of word c

- Add one to all the counts

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

– normalization factor $\frac{N}{N + V}$



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

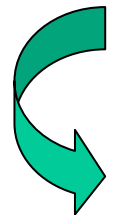
- $P_i^* = \frac{c_i + 1}{N + V}$ (Add-One Smoothing MLE)

- Alternative way to view a smoothing

– Discounting $d_c = \frac{c^*}{c}$

Add-One Smoothing

- For Bigram


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

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

- Gale and Church (1994)
 - Add-one is much worse at predicting the actual probability for bigram with zero counts than other methods like the Good-Turning.

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
- Estimate the probability of the zero N -grams

N : tokens

T : observed types

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

$$Z = \sum_{i:c_i=0} 1 \quad (6.17) \quad \leftarrow \text{All the zero } N\text{-grams}$$

Divide it equally

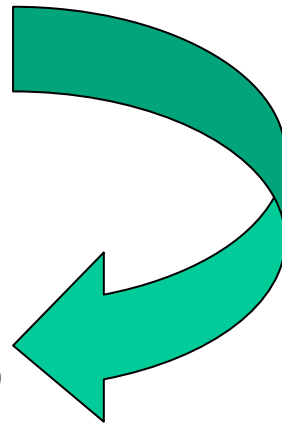
$$p_i^* = \frac{T}{Z(N+T)} \quad (6.18)$$

Witten-Bell Discounting

-

$$p_i^* = \begin{cases} \frac{T}{Z(N+T)} & , \text{ if } c_i = 0 \\ \frac{c_i}{N+T} & , \text{ if } c_i > 0 \end{cases} \quad (6.19)$$

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



$$c_i^* = p_i^* \times N$$

Witten-Bell Discounting


N : bigram tokens

T : observed bigram types

- For bigram

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

$$p^*(w_i | w_x) = \begin{cases} \frac{T(w_x)}{Z(w_x)(N + T(w_x))} & , \text{ if } c_{w_x w_i} = 0 \quad (6.23) \\ \frac{c(w_x w_i)}{c(w_x) + T(w_x)} & , \text{ if } c_{w_x w_i} > 0 \quad (6.24) \end{cases}$$



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

$$Z(w) = V - T(w) \quad (6.25)$$

Good-Turing Discounting

- Intuition
 - 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](#)

- *For bigram*

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

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

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

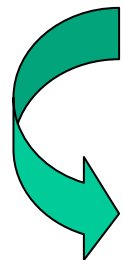
Good-Turing Discounting

- Large counts are assumed to be reliable.
 - Katz (1987) suggests setting k at 5
 - $c^* = c$ for $c > k$ (6.28)

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

Backoff

- If we have no example of a particular trigram, we can estimate its probability by using bigram probability.
- The trigram version



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

$$\hat{P}(w_n | w_{n-N+1}^{n-1}) = \tilde{P}(w_n | w_{n-N+1}^{n-1})$$

Generalization (recursive)

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

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

Combining Backoff with Discounting

- Combination
 - 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
- Why did we need α ?
 - If we backoff a lower order model when the prob. is zero \rightarrow total prob. of a word will greater than 1 \rightarrow
Any backoff language model must also be discounting

$$\therefore \sum_{i,j} P(w_n | w_i w_j) = 1 \quad (6.33)$$

Combining Backoff with Discounting

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

$$\bullet \alpha(w_{n-N+1}^{n-1}) \hat{P}(w_n | w_{n-N+2}^{n-1})$$

- The \tilde{P} comes from 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

- This probability \tilde{P} will be slightly less than the MLE estimate & leave some probability mass for the lower order N -grams

$$\tilde{P}(w_n | w_{n-N+1}^{n-1}) = \frac{c^*(w_{n-N+1}^n)}{c(w_{n-N+1}^{n-1})} \quad (6.35)$$

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

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

- The total probability mass that we are ready to distribute to all $N-1$ -gram
- Each individual $N-1$ gram will only get a fraction of this mass \rightarrow
normalize β by the total prob. of all $N-1$ grams $\rightarrow \alpha(w_{n-N+1}^{n-1})$

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

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

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

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

$$\tilde{P}(w_n | w_{n-N+1}^{n-1}) = 0 \quad (6.39)$$

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

Combining Backoff with Discounting

- For trigram

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

- In practice, when discounting, we usually ignore 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

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

$$\sum_i \lambda_i = 1 \quad (6.42)$$

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
- Training Method
 - For example : EM algorithm

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

Context-Sensitive Spelling Error Correction

- **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* → *three*)
 - Writer substituted the wrong spelling of a homophone or near-homophone (e.g., *dessert* for *desert*, or *piece* for *peace*)

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

- Error
 - **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
- One method
 - 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**
 - That is, given a sentence $W = \{w_1, w_2, \dots, w_k, \dots, w_n\}$, where w_k has alternative spelling w_k' , w_k'' , etc., we choose the spelling among these possible spellings that maximizes $P(W)$, using the N -gram grammar to compute $P(W)$

Entropy

- Recall

- The **entropy** of this random variable X

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

- The **perplexity**
 - 2^H

- Compute the entropy of some sequence of words

$$W = \{\dots w_0, w_1, w_2, \dots, w_n\}$$

- we can compute the entropy of a random variable that ranges over all finite sequences of words of length b in some language L as follows

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

Entropy

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

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

- 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:

$$H(L) = \lim_{n \rightarrow \infty} \frac{1}{n} H(w_1, w_2, \dots, w_n) \quad (6.49)$$

$$= \lim_{n \rightarrow \infty} -\frac{1}{n} \sum_{W_1^n \in L} p(w_1, \dots, w_n) \log p(w_1, \dots, w_n)$$

- The Shannon-McMillan-Breiman theorem states that if the language is regular in certain ways

$$H(L) = \lim_{n \rightarrow \infty} -\frac{1}{n} \log p(w_1, \dots, w_n) \quad (6.50)$$

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 with 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
 - ➔ Our statistical models only give an approximation to the correct distributions and entropy of natural language

Cross Entropy for Comparing Models

- **Recall**

- **Cross entropy**

- The actual probability distribution p , an approximation probability m

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

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

Cross Entropy for Comparing Models

- Apply Shannon-McMillan-Beriman theorem again

$$H(p, m) = \lim_{n \rightarrow \infty} -\frac{1}{n} \log m(w_1, w_2, \dots, w_n) \quad (6.52)$$

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

$$H(p) \leq H(p, m) \quad (6.53)$$

- The more accurate m is, the closer the cross entropy $H(p, m)$ (lower cross-entropy) will be to the true entropy $H(p)$
- The cross-entropy can never be lower than the true entropy, so a model cannot err by underestimating the true entropy

Thanks for your attention!