Carnegie Mellon University
Language
Technologies
Institute

11-411/11-611 Natural Language Processing

Introduction to Language Modeling: Ngram LMs

David R. Mortensen

September 27, 2022

Language Technologies Institute

Introduction to Language Models

Language Models Estimate the Probability of Sequences

Which of these sentences would you be more likely to observe in an English corpus?

- I hugged my big brother.
- Hugged I big brother my.
- I hugged my large brother.

Which of following word would be most likely to come after “David hates visiting New...”

- York
- California
- giggled

These are actually instances of the same problem: the language modeling problem!

Language Modeling is Tremendously Useful

LMs (language models) are at the center of NLP today and have many different applications

- **Machine Translation**
  P(high winds tonight) > P(large winds tonight)
- **Spelling Correction**
  P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
- **Text Input Methods**
  P(i cant believe how hot you **are**) > P(i cant believe how hot you **art**)
- **Speech Recognition**
  P(recognize speech) > P(wreck a nice beach)

The Goal of Language Modeling

Compute the probability of a sequence of words/tokens/characters:

$$P(\boldsymbol{w}) = P(w_1, w_2, w_3, w_5, \ldots, w_n) \tag{1}$$

$$P(\text{I, hugged, my, big, brother}) \tag{2}$$

This is related to next-word prediction:

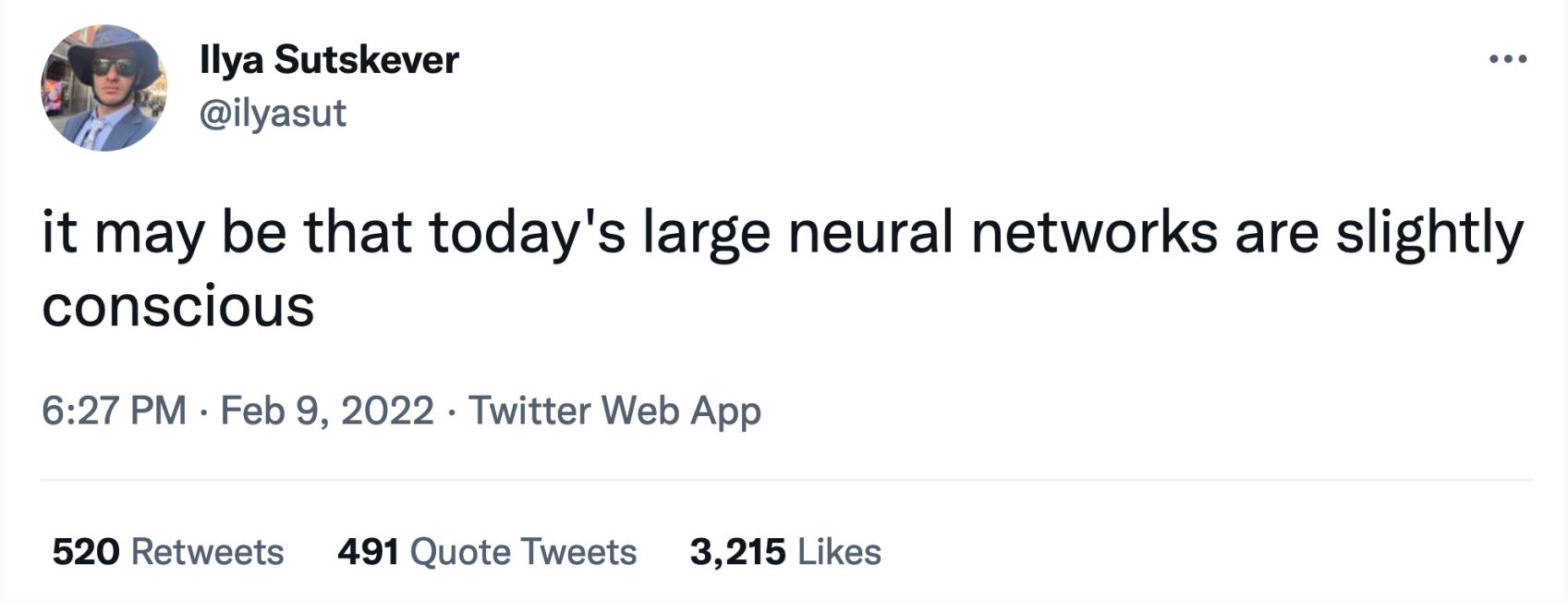
$$P(w_t | w_1 w_2 \ldots w_{t-1}) \tag{3}$$

$$P(\text{York}|\text{David, hates, going, to, New}) \tag{4}$$

Do you compute either of these? Then you're in luck:

You are a language model!

A Recent Tweet about Large Language Models by OpenAI Scientist Ilya Sutskever



While it is widely agreed (by people who don't work for OpenAI) that LMs cannot do everything that humans can do, it does seem that humans do many of the things that LMs do.

Introducing N-Gram Language Models

The Chain Rule Helps Us Compute Joint Probabilities

The definition of conditional probability is

$$P(B|A) = \frac{P(A,B)}{P(A)} \tag{5}$$

which can be rewritten as

$$P(A,B) = P(A)P(B|A) \tag{6}$$

If we add more variables, we see the following pattern:

$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C) \tag{7}$$

which can be generalized as

$$P(x_1, x_2, x_3, \ldots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)\ldots P(x_n|x_1, \ldots, x_{n-1}) \tag{8}$$

The Chain Rule!

We Can Use the Chain Rule to Compute the Joint Probability of Words in a Sentence

$$P(w_1, w_2, w_3, \ldots, w_n) = \prod_i^n P(w_i | w_1 w_2 \ldots w_{i-1}) \tag{9}$$

$$P(\text{now is the winter of our discontent}) =$$
$$P(\text{now}) \times P(\text{is}|\text{now}) \times P(\text{winter}|\text{is now}) \times$$
$$P(\text{of}|\text{now is the winter}) \times P(\text{our}|\text{now is the winter of}) \times$$
$$P(\text{discontent}|\text{now is the winter of our})$$

This May not Seem Very Helpful

Is $P(\text{discontent}|\text{now is the winter of our})$ really easier to compute than $P(\text{now is the winter of our discontent})$?

How is the chain rule helping us? A peak back at Naïve Bayes may provide a hint: cheat.

How Are We Estimating these Probabilities?

Could we just count and divide?

$$P(\text{discontent}|\text{now is the winter of our}) = \frac{\text{Count}(\text{now is the winter of our discontent})}{\text{Count}(\text{now is the winter of our})}$$

But this can't be a valid estimate! How many times in a corpus are either "now is the winter of our" or "now is the winter of our discontent" going to occur? This cannot be an estimate of their true probability.

Enter a Hero: Andrei Markov

Born	20 December 1978 (age 43) Voskresensk, Russian SFSR, Soviet Union
Height	6 ft 0 in (183 cm)
Weight	203 lb (92 kg; 14 st 7 lb)
Position	Defence
Played for	Khimik Voskresensk Dynamo Moscow Montreal Canadiens Vityaz Chekhov Ak Bars Kazan Lokomotiv Yaroslavl
Playing career	1995–2020

Or, Rather, Andrey Markov

Born	14 June 1856 N.S. Ryazan, Russian Empire
Died	20 July 1922 (aged 66) Petrograd, Russian SFSR
Known for	Markov chains; Markov processes; stochastic processes
Fields	Mathematics, specifically probability theory and statistics
Doctoral advisor	Pafnuty Chebyshev

Markov Did a Computational Linguistics

Interestingly, Markov's first application of his idea of **Markov Chains** was to language, specifically to modeling alliteration and rhyme in Russian poetry.

As such, he can be seen not only as a great mathematician and statistician, but also one of the forerunners of **computational linguistics** and **computational humanities**.

Markov Showed that You Could Make a Simplifying Assumption

One can approximate

$$P(\text{discontent}|\text{now is the winter of our})$$

by computing

$$P(\text{discontent}|\text{our})$$

or perhaps

$$P(\text{discontent}|\text{of our})$$

We only get an estimate this way, but we can obtain it by only counting “our disconent” and “disconent” (on in the second case, “of our disconent” and “of our”). Ngram language modeling is a generalization of this observation.

This Assumptions is the Markov Assumption

$$P(w_1, w_2, \ldots, w_n) \approx \prod_i P(w_i | w_{i-k} w_{i-i}) \tag{10}$$

In other words, we approximate each component in the product:

$$P(w_i | w_1, w_2, \ldots, w_{i-1}) \approx P(w_i | w_{i-k} \ldots w_{i-1}) \tag{11}$$

We will now walk through what this looks like for different values of k.

The Unigram Model ($k = 1$)

$$P(w_1 w_2 \ldots w_i) \approx \prod_i P(w_i) \tag{12}$$

The probability of a sequence is approximately the product of the probabilities of the individual words.

Some automatically generated sequences from a unigram model:

- fifth, an, of, futures, the, an, incorporated, a, a, the, inflation, most, dollars, quarter, in, is, mass
- thrift, did, eighty, said, hard, 'm, july, bullish
- that, or, limited, the

What do you notice about them?

The Bigram Model ($k = 2$)

If you condition on the previous word, you get the following:

$$P(w_1|w_1 w_2 \dots w_{i-1}) \approx P(w_i|w_{i-1}) \tag{13}$$

Some examples generated by a bigram model:

- texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen
- outside, new, car, parking, lot, of, the, agreement, reached
- this, would, be, a, record, november

Are these better?

The Trigram Model

The trigram model is just like the bigram model, only with a larger k:

$$P(w_1|w_1 w_2 \dots w_{i-1}) \approx P(w_i|w_{i-2} w_{i-1}) \tag{14}$$

The output of a trigram language model is generally **much** better than that of a bigram model provided the training corpus is large enough. Why do you need a larger corpus to train a trigram corpus than a bigram or unigram corpus?

We Can Just Keep Increasing k

We can extent the same idea to 4-grams. 5-grams, and so on. Why not go with an arbitrarily large value of k? Sparcity! You cannot estimate ngram probabilities accurately if the chances of observing any particular token is too low.

Ngram Language Models Have Trouble with Long-Distance Dependencies

In general, n-gram models are very impoverished models of language. For example, language has relationships that span many words:

- The **students** who worked on the assignment for three hours straight ***is/are** finally resting.
- The **teacher** who might have suddenly and abruptly met students **is/*are** tall.
- Violins are easy to mistakenly think you can learn to play ***them/quickly**.
- Negative polarity: predict ”some” vs ”any”
  - *I want **any**.
  - I want **some**.
  - I **don’t** want **any**.
  - *I think you said he thought we told them that she wants **any**.
  - I think you said he thought we told them that she wants **some**.

Ngram LMs Are Often Adequate

Nevertheless, for many applications, ngram models are good enough (and they're super fast and efficient)

Estimating Ngram Probabilities

Estimating Bigram Probablities with the Maximum Likelihood Estimate

MLE for bigram probabilities can be computed as:

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})} \tag{15}$$

which we will sometimes represent as

$$P(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \tag{16}$$

An example

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

More examples:
Berkeley Restaurant Project sentences

can you tell me about any good cantonese restaurants close by

mid priced thai food is what i'm looking for

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

when is caffe venezia open during the day

Raw bigram counts

Out of 9222 sentences

	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

Raw bigram probabilities

Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Bigram estimates of sentence probabilities

P(<s> I want english food </s>) =

P(I|<s>)

× P(want|I)

× P(english|want)

× P(food|english)

× P(</s>|food)

= .000031

Multiplication Considered Harmful

In reality, as was the case with NB classification, we do all of our computation in log space

- **Avoid underflow** Multiplying small probabilities by small probabilities results in *very small* numbers, which is problematic
- **Optimzie computation** Addition is cheaper than multiplication

$$\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4 \tag{17}$$

There are High-Performance Toolkits for N-Gram Language Modeling

- SRILM `http://www.speech.sri.com/projects/srilm/`
- KenLM `https://kheafield.com/code/kenlm/`

Perplexity and Evaluating Language Models

The Evaluation Process for ML Models

The goal of LM evaluation:

- Does our model prefer good sentences to bad sentences?
- Specifically, does it assign higher probabilities to the good/grammatical/frequently observed ones and lower probabilities to the bad/ungrammatical/seldom observed ones?

In ML evaluation, we divide our data into three sets: `train`, `dev`, and `test`.

- We train the model's parameters on the `train` set
- We tune the model's parameters (if appropriate) on the `dev` set (which should not overlap with the `train` set
- We test the model on the `test` set, which should not overlap with `train` or `dev`

An evaluation metric tells us how well our model has done on `test`.

We Can Evaluate Models Intrinsically or Extrinsically

- **Extrinsic Evaluation** means asking how much the model contributes to a larger task or goal. We may evaluate an LM based on how much it improves machine translation over a BASELINE.
- **Intrinsic Evaluation** means measuring some property of the model directly. We may quantify the probability that an LM assigns to a corpus of text.

In general, EXTRINSIC EVALUATION is better, but more expensive and time-consuming.

Extrinsic Evaluation of LMs

Best evaluation for comparing models A and B

- Put each model in a task (spelling corrector, speech recognizer, MT system)
- Run the task, get an accuracy for A and for B
  - How many misspelled words corrected properly?
  - How many sentences translated correctly?
- Compare scores for A and B

This takes a lot of time to set up and can be expensive to carry out.

Perpelxity Is an Intrinsic Metric for Language Modeling

Perplexity evaluates the probability assigned by a model to a collection of text and is, thus, useful for evaluating LMs. Note:

- It is a rather crude instrument
- It sometimes correlates only weakly with performance on DOWNSTREAM tasks (the larger tasks that you are using the language model to perform)
- It's only useful for pilot experiments
- But it's cheap and easy to compute, so it's important to understand

Intuition of Perplexity

The **Shannon Game**:

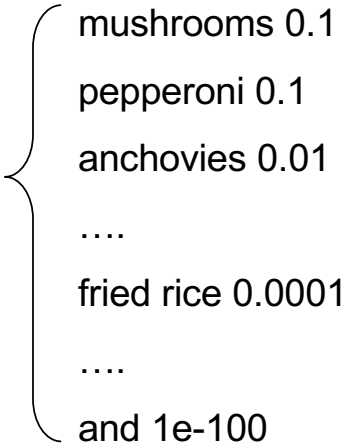
- How well can we predict the next word?

  I always order pizza with cheese and ____

  The 33rd President of the US was ____

  I saw a ____

- Unigrams are terrible at this game. (Why?)



A better model of a text

- is one which assigns a higher probability to the word that actually occurs

Deriving Perplexity for Bigrams

$$
\begin{aligned}
PP(\boldsymbol{w}) &= P(w_1 w_2 \ldots 2_n)^{-\frac{1}{n}} && \text{Definition} \\
&= \sqrt[n]{\frac{1}{P(w_1 w_2 \ldots w_n)}} \\
&= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1 w_2 \ldots W_{i-1})}} && \text{Chain Rule} \\
&= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_{i-1})}} && \text{For Bigrams}
\end{aligned}
$$

To minimize perplexity is to maximize probability!

The Shannon Game intuition for perplexity

From Josh Goodman

Perplexity is weighted equivalent branching factor

How hard is the task of recognizing digits ‘0,1,2,3,4,5,6,7,8,9’

- Perplexity 10

How hard is recognizing (30,000) names at Microsoft.

- Perplexity = 30,000

Let's imagine a call-routing phone system gets 120K calls and has to recognize

- "Operator" (let's say this occurs 1 in 4 calls)
- "Sales" (1in 4)
- "Technical Support" (1 in 4)
- 30,000 different names (each name occurring 1 time in the 120K calls)
- What is the perplexity? Next slide

The Shannon Game intuition for perplexity

Josh Goodman: imagine a call-routing phone system gets 120K calls and has to recognize

- "Operator" (let's say this occurs 1 in 4 calls)
- "Sales" (1in 4)
- "Technical Support" (1 in 4)
- 30,000 different names (each name occurring 1 time in the 120K calls)

We get the perplexity of this sequence of length 120Kby first multiplying 120K probabilities (90K of which are 1/4 and 30K of which are 1/120K), nd then taking the inverse 120,000th root:

$$\text{Perp} = (¼ * ¼ * ¼ * ¼ * ¼ * \ldots * 1/120K * 1/120K * \ldots)^{(-1/120K)}$$

But this can be arithmetically simplified to just N = 4: the operator (1/4), the sales (1/4), the tech support (1/4), and the 30,000 names (1/120,000):

$$\text{Perplexity} = ((¼ * ¼ * ¼ * 1/120K)^{(-1/4)} = 52.6$$

In general, a lower perplexity implies a better model.

The Problem of Zeros

Sometimes there Are Ngrams in the Test Set that Weren't in the Training Set

Suppose our bigram LM, trained on Twitter, reads a document by the philosopher Wittgenstein:

> *Whereof one cannot speak, thereof one must be silent.*

This contains the bigrams: whereof one, one cannot, cannot speak, speak [comma], [comma] thereof, thereof one, one must, must be, be silent.

Suppose "whereof one" never occurs in the training corpus (`train`) but whereof occurs 20 times. According to MLE, it's probability is

$$P(\text{one}|\text{whereof}) = \frac{c(\text{whereof, one})}{c(\text{whereof})} = \frac{0}{20} = 0 \tag{18}$$

The probability of the sentence is the **product** of the probabilities of the bigrams. What happens if one of the probabilities is zero? Compare, again, to Naïve Bayes.

The Perils of Overfitting

N-grams only work well for word prediction if the test corpus looks like the training corpus

- In real life, it often doesn't
- We need to train robust models that generalize!
  - One kind of generalization: Zeros!
  - Things that don't ever occur in the training set but occur in the test set

Zero Probability Bigrams Are an Obstacle

If a bigram in the test set has zero probability, we must assign 0 probability to the whole test set!

Hence, we cannot compute perplexity for the test set (since we cannot divide by zero)!

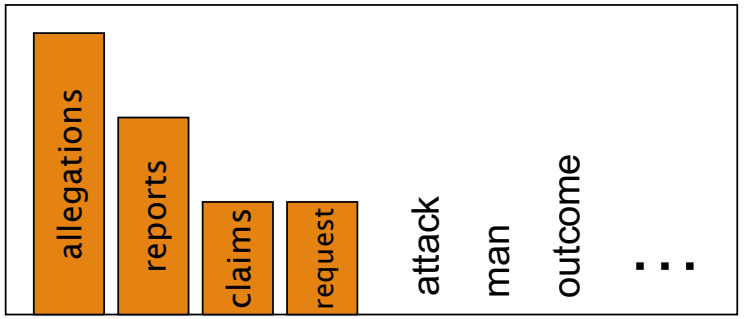
However, there is a way forward.

Laplace Smoothing

The intuition of smoothing (from Dan Klein)

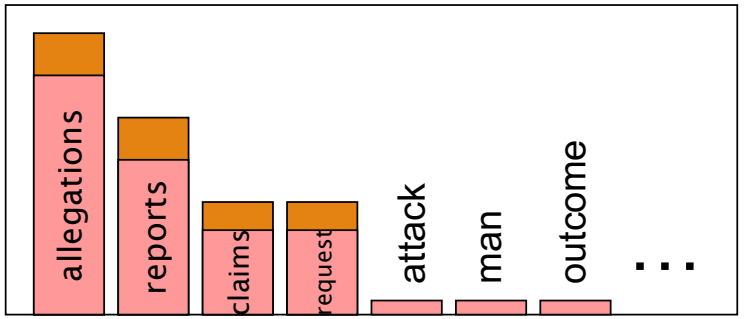
When we have sparse statistics:

P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total



Steal probability mass to generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



Pretending that We Saw Each Word Once More than We Did is Laplace Smoothing

This should already be familiar from the Naïve Bayes lecture and the third homework assignment.

$$\text{MLE estimate } P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$$\text{Add-1 estimate } P_{Add-1}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + |V|}$$

Where V is the vocabulary of the corpus.

Laplace Smooth Is too Blunt

It shifts too much probability mass away from attested ngrams and onto unattested ngrams, so it isn't used much for ngram LMs any more (there are better methods).

But remember that is does work:

- For text classification
- In domains where the number of zeros isn't as large as with ngrams

Interpolation and Backoff

Backoff and Interpolation Let You Use Less Context

Suppose you have a context you don't know much about (because you have seen few or no relevant ngrams). You can condition your probabilities for these contexts on shorter contexts you know more about.

Backoff Use trigram if you have good evidence, otherwise bigram, otherwise unigram.

Interpolation Mix unigrams, bigrams, and trigrams together in one (weighted) probability soup.

Interpolation works better; backoff is sometimes cheaper.

In Linear Interpolation, the Probabilities of Different Orders of Ngrams Are Taken into Account

The simplest way to do this is to **not** take context into account. The lambdas, in the following formula, are weighting factors:

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

where

$$\forall i\ \lambda_i \geq 0 \wedge \sum_i^n \lambda_i = 1$$

That is, the lambdas must sum to one.

Lambdas Are Tuned Using a Held-Out `dev` Set

Choose λs to maximize the probability of held-out data (`dev`):

- Fix the ngram probabilities (on `train`)
- Then search for λs that give the largest probability to `dev`:

$$\log P(w_1 \dots w_n | M(\lambda_1 \dots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \dots \lambda_k)}(w_i | w_{i-1}) \quad (19)$$

Unknown Words

If we know all the words in advanced

- Vocabulary V is fixed
- Closed vocabulary task

Often we don't know this

- Out Of Vocabulary = OOV words
- Open vocabulary task

Instead: create an unknown word token <UNK>

- Training of <UNK> probabilities
- Create a fixed lexicon L of size V
- At text normalization phase, any training word not in L changed to <UNK>
- Now we train its probabilities like a normal word
- At decoding time
- If text input: Use UNK probabilities for any word not in training

Web-Scale Ngrams

How to deal with, e.g., Google N-gram corpus Pruning

- Only store N-grams with count > threshold.
- Remove singletons of higher-order n-grams
- Entropy-based pruning

Efficiency

- Efficient data structures like tries
- Bloom filters: approximate language models
- Store words as indexes, not strings
- Use Huffman coding to fit large numbers of words into two bytes
- Quantize probabilities (4-8 bits instead of 8-byte float)

Stupid Backoff is Stupid but Efficient

No discounting, just use relative frequencies

$$S(w_i|w_{i-k+1}^{i-1}) = \begin{cases} \frac{c(w_{i-k+1}^{i})}{c(w_{i-k+1}^{i-1})} & \text{if } c(w_{i-k+1}^{i}) > 0 \\ 0.4S(w_i|w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases} \tag{20}$$

$$S(w_i) = \frac{c(w_i)}{N} \tag{21}$$

Kneser-Ney Smoothing (Appendix)

Absolute discounting: just subtract a little from each count

Suppose we wanted to subtract a little from a count of 4 to save probability mass for the zeros

How much to subtract ?

Church and Gale (1991)'s clever idea

Divide up 22 million words of AP Newswire

- Training and held-out set
- for each bigram in the training set
- see the actual count in the held-out set!

It sure looks like c* = (c - .75)

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Absolute Discounting Interpolation

Save ourselves some time and just subtract 0.75 (or some d)!

discounted bigram

Interpolation weight

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w)$$

unigram

- (Maybe keeping a couple extra values of d for counts 1 and 2)

But should we really just use the regular unigram P(w)?

Kneser-Ney Smoothing I

Better estimate for probabilities of lower-order unigrams!

- Shannon game: *I can't see without my reading*_____?
- "Kong" turns out to be more common than "glasses"
- … but "Kong" always follows "Hong"

The unigram is useful exactly when we haven't seen this bigram!

Instead of P(w): "How likely is w"

$P_{continuation}$(w): "How likely is w to appear as a novel continuation?

- For each word, count the number of bigram types it completes
- Every bigram type was a novel continuation the first time it was seen

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

Kneser-Ney Smoothing II

How many times does w appear as a novel continuation:

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

Normalized by the total number of word bigram types

$$\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|$$

$$P_{CONTINUATION}(w) = \frac{\left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|}{\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|}$$

Kneser-Ney Smoothing III

Alternative metaphor: The number of # of word types seen to precede w

$$|\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

normalized by the # of words preceding all words:

$$P_{CONTINUATION}(w) = \frac{\left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|}{\sum_{w'} \left|\{w'_{i-1} : c(w'_{i-1}, w') > 0\}\right|}$$

A frequent word (Kong) occurring in only one context (Hong) will have a low continuation probability

Kneser-Ney Smoothing IV

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i)$$

λ is a normalizing constant; the probability mass we've discounted

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{w : c(w_{i-1}, w) > 0\} \right|$$

the normalized discount

The number of word types that can follow w_{i-1}
= # of word types we discounted
= # of times we applied normalized discount

Kneser-Ney Smoothing: Recursive formulation

$$P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1}) P_{KN}(w_i \mid w_{i-n+2}^{i-1})$$

$$c_{KN}(\bullet) = \begin{cases} count(\bullet) & \text{for the highest order} \\ continuationcount(\bullet) & \text{for lower order} \end{cases}$$

Continuation count = Number of unique single word contexts for •

Questions?