N-gram Language Models

Speech and Language Processing – Chapter 3

Marion Di Marco

23. 04. 2024

N-Gram Language Models

- What could be the next word in the following sentence

```
Please turn your homework  ...  in
                                over
                                refrigerator
```

- **Language models**: assign a **probability** to upcoming words or sequences of words

- Assign a probability to sentences:

  `all of a sudden I notice three guys standing on the sidewalk`

  `on guys all I of notice sidewalk three a sudden standing the`

What can LMs be used for?

- Choose a better sentence or word
- Correct grammar or spelling

  Their are two midterms → There ...

  Everything has improve → ... improved
- Speech recognition

  I will be back soonish

  I will be bassoon dish
- Augmentative and Alternative communication

  Communication via eye gaze for people unable to speak physically: suggest word menu

Outline

Word Probabilities

- $P(w|h)$: the probability of the word w given some history h
- $P(the|its\ water\ is\ so\ transparent\ that)$
- Relative frequency counts based on a large corpus:
  $P(the|its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)}$
- Even a very large corpus cannot contain all possible sentences:
  "*the Neckar's water is so transparent that*": zero matches in Google
- Let's find a better method!

Chain Rule of Probability

- P(*the*) = probability of random variable X_i taking the value "*the*"
- Sequence of n words: $w_1 \ldots w_n$ or $w_{1:n}$

- Compute the probability of a word sequence like $P(w_1, w_2, \ldots, w_n)$
- Decompose the probability using the **chain rule of probability**

$$
\begin{aligned}
P(X_1 \ldots X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_{1:2}) \ldots P(X_n|X_{1:n-1}) \\
&= \prod_{k=1}^{n} P(X_k|X_{1:k-1}) \qquad (3.3)
\end{aligned}
$$

Applying the chain rule to words, we get

$$
\begin{aligned}
P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2}) \ldots P(w_n|w_{1:n-1}) \\
&= \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \qquad (3.4)
\end{aligned}
$$

- Problem: still cannot compute the exact probability of a word given a long sequence of preceding words $P(w_n|w_{1:n-1})$

N-Gram Models

- **N-gram model**: approximate the history by the last few words

- Bigram model: approximate the probability $P(w_n|w_{1:n-1})$ by the conditional probability of the previous word $P(w_n|w_{n-1})$

  $P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$

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

- Given the bigram assumption, compute the **probability of a sequence**

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}) \qquad (3.9)$$

Maximum Likelihood Estimation

- Estimate n-gram probabilities with *maximum likelihood estimation*: get counts from corpus; normalize such that they lie between 0 and 1

- Bigram probability: count of the bigram $C(w_{n-1}\, w_n)$ normalize with the sum of all bigrams sharing the first word w_{n-1}:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \tag{3.10}$$

- Simplify: the sum of all bigrams starting with w_{n-1} is equal to the unigram count of w_{n-1}

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{3.11}$$

N-gram Probabilities: Example

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \quad (3.11)$$

- Special symbol to denote beginning and end of a sentence: `<s>`, `</s>`

- Small example corpus:

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

- Some probabilities:

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

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

Maximum Likelihood Estimates

- The maximum likelihood estimate
  - of some parameter of a model M from a training set T
  - maximizes the likelihood of the training set T given the model M

- Suppose that “bagel” occurs 400 times in a corpus of a million words
- What is the probability that a random word from some other text will be “bagel”?
- MLE estimate is $400/1{,}000{,}000 = .0004$

- This may be a bad estimate for some other corpus
  - but it is the **estimate** that makes it **most likely** that “bagel will occur 400 times in a million word corpus

N-gram Models: Example

- Berkeley Restaurant Project Corpus (dialogue system)
- Sample user queries

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

N-gram Models: Example

- Bigram counts from Berkeley Restaurant Project
- Majority of the values are zero
- Samples are chosen to cohere with each other, a random set of words would be even more sparse

	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

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

N-gram Models: Example

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

	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

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

N-gram Models: Example

- Some more probabilities

$$P(\texttt{i}|\texttt{<s>}) = 0.25 \qquad P(\texttt{english}|\texttt{want}) = 0.0011$$
$$P(\texttt{food}|\texttt{english}) = 0.5 \qquad P(\texttt{</s>}|\texttt{food}) = 0.68$$

- Compute the probability for *"I want English food"*

$$\begin{aligned} & P(\texttt{<s> i want english food </s>}) \\ &\quad = P(\texttt{i}|\texttt{<s>})P(\texttt{want}|\texttt{i})P(\texttt{english}|\texttt{want}) \\ &\quad\qquad P(\texttt{food}|\texttt{english})P(\texttt{</s>}|\texttt{food}) \\ &\quad = .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\ &\quad = .000031 \end{aligned}$$

N-gram Models

- We can extend the n-gram size to trigrams, 4-grams, 5-grams
- In general, this is an insufficient model of language

  language has long-distance dependencies:

  The computer which I had just put into the machine room on the fifth floor crashed
- N-gram models often still work fine

Practical Issues

- Probabilities are less than 1
  → the more multiplications, the smaller the product becomes
  → risk of numerical underflow

- Represent language model probabilities as **log probabilities**

- Adding in log space is equivalent to multiplying in linear space

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4) \tag{3.13}$$

LM Toolkits and Resources

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

- All Our N-gram are Belong to You:
  `https://research.google/blog/all-our-n-gram-are-belong-to-you/`
- Google Book N-grams: `http://ngrams.googlelabs.com/`

- NLTK tools: `https://www.nltk.org/book/ch02.html`
  (Generating Random Text with Bigrams)

Outline

Evaluating LMs

- **Extrinsic evaluation**
  - embed the LM in application → measure improvement
  - for example machine translation: build MT systems incorporating the respective LMs, compare the results
  - In practice: often too expensive to train/run big NLP systems
  - Sidenote: measuring the quality of a translation (or some other NLP task) is often not trivial
- **Intrinsic evaluation**
  - measure the model's quality independent of another application
- Perplexity: standard intrinsic metric for LM performance

Training and Test Data

Three distinct data sets

- **Training set**
  - data set to learn parameters for the model
  - text corpus to get counts as basis for the n-grams probabilities
- **Test set**
  - held-out data set disjunct from training data
  - measure how well the model can handle *unknown* data
  - use test set to measure performance only for the final LM
- **Development set**
  - additional data to measure performance when working on the model

Training and Test Data

- The test set should reflect the type of language modeled in the LM
  - for example data of medical or chemical domain, hotel booking
  - general purpose: wide variety of texts
- "Fit of the model": the LM that has a tighter fit to the test set (= assigns a higher probability) is better
- Seeing test data during training: this is bad!
  - bias the model to the test set
  - artificially high probabilities, inaccurate perplexity
- Test too early on the test set: also bad!
  - tune the model to the test set's characteristics

Perplexity

- Perplexity: measures how well a model predicts a sample
  - a good model should not be "perplexed" or surprised when seeing a (valid) document

- Perplexity is the inverse probability of the test set, normalized by the number of words ("per-word-perplexity")
- For a test set $W = w_1 w_2 \ldots w_N$

$$\begin{aligned} \text{perplexity}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \end{aligned} \tag{3.14}$$

Or we can use the chain rule to expand the probability of W:

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}} \tag{3.15}$$

- Higher probability → lower perplexity

Perplexity

- Perplexity for a unigram language model

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i)}} \tag{3.16}$$

- Perplexity for a bigram language model

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \tag{3.17}$$

Perplexity: Example

- Training corpus for a unigram, bigram and trigram model:
  38 million words from Wall Street Journal, 19.979 word vocabulary
- Test corpus: 1.5 million words from Wall Street Journal

	Unigram	**Bigram**	**Trigram**
Perplexity	962	170	109

- Trigram model is less surprised than the unigram model
- Lower perplexity → better predictor of words in the test set

- (Intrinsic) improvement in perplexity: no guarantee for (extrinsic) improvement
- Perplexity often correlates with task improvements → convenient evaluation metric

Perplexity as Weighted Average Branching Factor

- Branching factor of a language: number of possible next words

- Assume a language of integer numbers with a vocabulary of 10 digits (0,1, ... , 9):
  branching factor = 10
- Each of the digits occurs with the same probability ($P = \frac{1}{10}$)
- Perplexity of a string pf length N:

$$
\begin{aligned}
\text{perplexity}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= (\frac{1}{10}^N)^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10 \qquad (3.18)
\end{aligned}
$$

Outline

Predicting Upcoming Words

- *The Shannon Game (1948)*: How well can we predict the next word?
  - `one upon a ___`
  - `for breakfast I ate ___`
  - `this is a picture of my ___`

$$\begin{cases} \texttt{time} & \texttt{0.2} \\ \texttt{midnight} & \texttt{0.1} \\ \texttt{and} & \texttt{0.3} \\ \texttt{...} & \\ \texttt{yellow} & \texttt{0.002} \end{cases}$$

- Unigram:

  `REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE.`

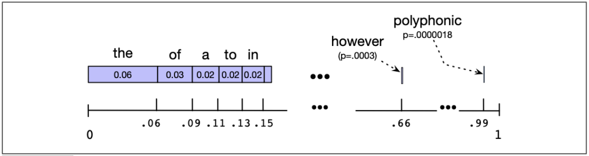
- Bigram:

  `THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED.`

Sampling Words from a Distribution

- Sampling from a distribution: choose a random point according to their likelihood

- Visualization for unigrams:



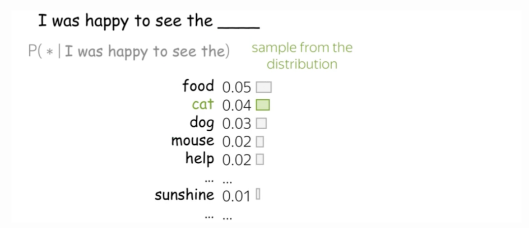
  - all words cover the probability space between 0 and 1
  - intervals in proportion to the relative frequency
  - cumulative probabilities in the bottom line

- Choose a random point between 0 and 1: find the word
- Continue until you encounter </s>

- Can also be applied to bigrams

Sampling

- Sampling from a language model: generate sentences according to the likelihood as defined by the model
- Intuition: a good LM prefers "real" sentences over "word salad"
- Sentences with a higher probability in the model are more likely



- There are many more sampling methods
  → often avoid words from the very tail of the distribution

  (for example: temperature sampling, tok-k sampling, top-p sampling)

Figure from https://lena-voita.github.io/nlp_course/language_modeling.html#generation_strategies_sampling

Outline

Context and Coherence

- More context is better: higher-order n-grams can capture more context
- More context $\rightarrow$ more coherent generated sentences

- Example: randomly generated sentences from Shakespeare

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; –It cannot be but so.

Context Size and Coherence

- Unigram: no coherent relation between words
- Bigram: some local coherence
- 3-gram and 4-gram: starts to ressemble Shakespeare

- The sequence *It cannot be but so* are directly from King John

- Comparatively small corpus: N = 884,647 and V = 29,066
  - n-gram probability matrices are very sparse
  - 300,000 out of $V^2 = 844$ million possible bigrams
  - 99.96% of the possible bigrams were never seen (= zero entry)

  - Once the 3- gram *It cannot be* is chosen: only seven possibilities for the next word: (*but, I, that, thus, this, and the period*)

Training Data

- Choosing the training data: use a training corpus that has a similar genre to the task

- Can you guess the original data?
  - `They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and gram Brazil on market conditions`
  - `‘‘You are uniformly charming!’’ cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.`

- N-grams work well if the training and test corpus are similar
- Even with a good training corpus: surprisal in the test set
- Thus: train robust models that are able to generalize

Data Sparsity

- Even in a large corpus: data sparsity problems
- For sufficiently observed n-grams: good estimate of probability
- But: some valid sequences do not occur in the corpus

- Example from Wall Street Journal corpus (40 million words)

denied the allegations	5
denied the speculation	2
denied the rumors	1
denied the report	1
denied the offer	–
denied the loan	–

- Thus, the LM will estimate that $P(\mathit{offer}|\mathit{denied\ the}) = 0$
  - under-estimate probability of valid sequences → harmful for task
  - if one word has probability of zero, test set has a probability of zero: perplexity is undefined

Unknown Words

- **Unknown words** or **out-of-vocabulary words (OOV)** :
  word in the test data that does not occur in the training data
- **OOV-rate**: percentage of OOVs in the test set

- Create an **open vocabulary system**: map unknown words to `<UNK>`
  - Choose a fixed vocabulary
  - Convert OOVs in the training data to the special token `<UNK>`
  - Estimate probabilities for `<UNK>` just as for regular words

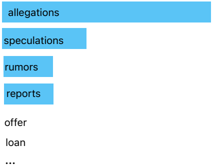
- **Closed vocabulary system**: there are no unknown words
  Most modern LMs: sub-word tokenization to segment words into smaller pieces (for example BPE)

Outline

Smoothing – Intuition

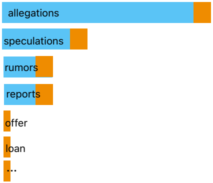
- Words that are in the vocabulary, but appear in an unseen context?

P(w | denied the)	
allegations	5
speculations	2
rumors	1
reports	1



- **Smoothing** or **discounting**: "steal" probability mass from more frequent events and give them to unseen events

P(w | denied the)	
allegations	4.5
speculations	1.5
rumors	0.5
reports	0.5
other	2



Laplace Smoothing

- Laplace smoothing or **add-one smoothing**
- Add one to all n-gram counts before normalizing into probabilities

- MLE estimate:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \qquad (3.23)$$

- For add-one smoothed bigram counts: augment the unigram count by the number of word types in the vocabulary V
- Add-1 estimate:

$$P_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w (C(w_{n-1}w)+1)} = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V} \qquad (3.24)$$

Laplace Smoothing: Berkeley Restaurant Corpus

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Figure 3.6 Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Previously-zero counts are in gray.

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Figure 3.7 Add-one smoothed bigram probabilities for eight of the words (out of $V = 1446$) in the BeRP corpus of 9332 sentences. Previously-zero probabilities are in gray.

Laplace Smoothing: Berkeley Restaurant Corpus

Reconstruct the count matrix:

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V} \quad (3.25)$$

	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

Figure 3.8 Add-one reconstituted counts for eight words (of $V = 1446$) in the BeRP corpus of 9332 sentences. Previously-zero counts are in gray.

	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

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

Laplace Smoothing: Berkeley Restaurant Corpus

- Add-1 smoothing can make a very big change to the counts
- For example, *C(want to)* changed from 608 to 238 and *C(Chinese food)* from 82 to 8.2
- Discount d: the ratio between new and old counts

- Sharp change in counts and probabilities: too much probability mass is moved to unseen events
- Add-1 is not used for n-grams, but for text classification or domains where the number of zeros is smaller

- Variant: add-k smoothing with a fractional count $k < 1$ to move less probability mass away from seen events.
  - requires a method to choose k (optimize on devset)
  - still doesn't work well for LMs

Backoff and Interpolation

- So far: target the problem of zero frequency n-grams

- Use **less context** to help the model generalize for contexts it has no knowledge about:

  to compute $P(w_n \mid w_{n-2}\ w_{n-1})$: if there are no examples of the trigram $\mathrm{w}_{n-2}\ \mathrm{w}_{n-1}\ \mathrm{w}_{n}$, use the bigram probability $P(w_n \mid w_{n-1})$ instead

- **Backoff:** use a lower-order n-gram if there is no evidence for a higher-order n-gram

- **Interpolation:** mix estimates from all n-gram orders using weights to comine them

  (Interpolation tends to be better)

Linear Interpolation

- Simple linear interpolation: combine unigram, bigram and trigram probabilities, each weighted with a λ

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) \;=\; & \lambda_1 P(w_n) \\ & +\lambda_2 P(w_n|w_{n-1}) \\ & +\lambda_3 P(w_n|w_{n-2}w_{n-1}) \qquad (3.27)\end{aligned}$$

- The λ_i must sum to 1 → weighted average

- Linear interpolation with context-conditioned weights

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) \;=\; & \lambda_1(w_{n-2:n-1})P(w_n) \\ & +\lambda_2(w_{n-2:n-1})P(w_n|w_{n-1}) \\ & +\lambda_3(w_{n-2:n-1})P(w_n|w_{n-2}w_{n-1}) \qquad (3.28)\end{aligned}$$

Linear Interpolation

- The λ values are learned from a **held out corpus**
  additional training data to learn hyperparameters λ

- Choose λs to maximize the probability of held-out data
  - fix the n-gram probabilities on the training data
  - search for λs that give the highest probability of the held-out set

- Various ways to find the optimal set of λs, for example the EM (expectation-maximization) algorithm

Katz Backoff

- Backoff: if an n-gram has zero counts, approximate with (n-1)-gram
- Discount higher-order n-grams to save some probability mass for lower-order n-grams:

  just replacing an n-gram which has zero probability with a lower order n-gram
  $\rightarrow$ adding to the total probability mass

- **Katz backoff**: backoff with discounting
- Discounted probability P^*
- α to distribute the probability mass to the lower-order n-grams

$$P_{\mathrm{BO}}(w_n|w_{n-N+1:n-1}) = \begin{cases} P^*(w_n|w_{n-N+1:n-1}), & \text{if } C(w_{n-N+1:n}) > 0 \\ \alpha(w_{n-N+1:n-1})P_{\mathrm{BO}}(w_n|w_{n-N+2:n-1}), & \text{otherwise.} \end{cases} \quad (3.29)$$

Outline

Absolute Discounting

- Consider an n-gram with count=4: How much should we discount?
- Church and Gale (1991): look at the count of n-grams with count=4 in held-out data
- Compute all bigrams from 22 million words, then check the counts of the bigrams in another 22 million words
- On average: a bigram with count=4 in C1 occurred 3.32 times in C2

Bigram count in training set	Bigram count in heldout set
0	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

Figure 3.9 For all bigrams in 22 million words of AP newswire of count 0, 1, 2,...,9, the counts of these bigrams in a held-out corpus also of 22 million words.

Absolute Discounting

- For counts > 1 the bigram counts in the held-out set can be estimated by subtracting 0.75 from the training set

- **Absolute discounting**: subtract a fixed discount *d* from each count
  - good estimates for high counts → small discount won't hurt
  - smaller counts: we don't necessarily trust the estimate
- Interpolated absolute discounting for bigrams:

$$P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{\sum_v C(w_{i-1}\,v)} + \lambda(w_{i-1})P(w_i) \qquad (3.31)$$

- First term: discounted bigram
  Second term: unigram with an interpolation weight λ
- Given Figure 3.9: set *d=0.75*, maybe *d=0.5* for bigrams with count=1
  (There are more complex ways to determine *d*)

Kneser-Ney Discounting

- More sophisticated way to handle lower-order unigram distribution
- Assume we are interpolating a bigram and unigram model

  `I can't see without my reading ___`

- *glasses* seems much more likely than *Francisco*
  → a unigram model should prefer *glasses*
- *San Francisco* is very frequent
  → *Francisco* is more common than *glasses*

- *Francisco* is frequent, but mainly occurs after *San*

  glasses has a wider distribution

- Words appearing in more contexts → more likely to appear in a new context

Kneser-Ney Discounting

- Unigram model $P_{CONTINUATION}$: how likely is w as a novel continuation?
- Base the estimation of $P_{CONTINUATION}$ on the number of different contexts w has appeared in (= number of bigram types it completes)

- Continuation probability associated with each unigram: proportional to the number of bigrams it completes

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

- Normalize by the total number of bigram types

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

- Frequent words appearing in very few contexts: low continuation probability

Interpolated Kneser-Ney

- The final equation for **Interpolated Kneser-Ney** smoothing for bigrams

$$P_{\mathrm{KN}}(w_i|w_{i-1}) = \frac{\max(C(w_{i-1}w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1}) P_{\mathrm{CONTINUATION}}(w_i) \quad (3.37)$$

- λ: normalizing constant

$$\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}v)} |\{w : C(w_{i-1}w) > 0\}| \quad (3.38)$$

- The first term: the normalized discount
- The second term: the number of word types that can follow w_{i-1} (= number of word types we discounted)

Interpolated Kneser-Ney

- Kneser-Ney evolved from absolute discounting interpolation (higher-order and lower-order n-grams, with some probability mass reallocated to unigrams)
- Kneser-Ney addresses the unigram part:
  - absolute discounting: simple unigram model
  - Kneser-Ney: continuation probability associated with each unigram
- Modified Kneser-Ney: instead of a fixed discount d, use different discounts d_1, d_2, d_{3+} for n-grams with counts of 1, 2 and3 or more

Outline

Huge LMs

- Using Web data or other enormous corpora → extremely large LMs
  - Web 1 Trillion 5-gram corpus released by Google: unigrams – 5-grams from 1,024,908,267,229 words (English)
  - Google Books Ngrams corpora: n-grams from 800 million tokens (Chinese, English, French, German, Hebrew, Italian, Russian, Spanish)

- Pruning
  - only store n-grams with count > threshold (→ Google corpus)
  - remove singletons of higher-order n-grams
  - entropy-based pruning to remove less important n-grams

- Efficiency
  - efficient data structures like tries
  - store words as indexes, not strings

Stupid Backoff

- With very large LMs, a simple smoothing strategy may be sufficient

- **Stupid backoff**: no probability distribution
  - no discounting of higher-order n-grams
  - backoff to lower-order n-gram if higher-order n-gram has a zero count
  - lower-order n-grams are weighted by a fixed weight

$$S(w_i|w_{i-N+1:i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-N+1:i})}{\text{count}(w_{i-N+1:i-1})} & \text{if count}(w_{i-N+1:i}) > 0 \\ \lambda S(w_i|w_{i-N+2:i-1}) & \text{otherwise} \end{cases} \quad (3.30)$$

The backoff terminates in the unigram, which has score $S(w) = \frac{count(w)}{N}$. Brants et al. (2007) find that a value of 0.4 worked well for λ.

Outline

Summary

- LMs: assign a probability to sentences or word sequences, and predict a word from preceding words
- n-grams are Markov models: estimate words from a fixed window of previous words
- n-gram probabilities: estimated from normalized counts in a corpus (maximum likelihood estimate)
- Evaluation
  - extrinsic evaluation on a task
  - intrinsic evaluation using perplexity
- Smoothing: more sophisticated way to estimate probabilities of n-grams
  - rely on lower-order n-grams through backoff or interpolation
  - require discounting to create a probability distribution

References

Speech and Language Processing
Dan Jurafsky and James H. Martin

Chapter 3: N-gram Language Models
`https://web.stanford.edu/~jurafsky/slp3/3.pdf`