
Empirical Methods in Natural Language Processing

Lecture 3

N-gram Language Models

(most slides from Sharon Goldwater; some adapted from Alex Lascarides)

15 January 2020



Recap

- Previously, we talked about corpus data and some of the information we can get from it, like word frequencies.
- For some tasks, like sentiment analysis, word frequencies alone can work pretty well (though can certainly be improved on).
- For other tasks, we need more.
- Today we consider **sentence probabilities**: what are they, why are they useful, and how might we compute them?

Review: Word-based sentiment

- Recall that we can predict sentiment for a document based on counting positive and negative words.
- Do you think the following words would be positive or negative in a movie review?
 - OK
 - Action
 - Star

N-grams

- In some cases, looking at more than one word at a time might be more informative.
 - action movie vs. action packed
 - Star Wars vs. star studded
- An n -gram is a word sequence of length n .
 - 1-gram or **unigram**: action
 - 2-gram or **bigram**: action packed
 - 3-gram or **trigram**: action packed adventure
 - 4-gram: action packed adventure film

N-grams

The Force Awakens brings back the Old Trilogy 's heart , humor , mystery , and fun .

How many:

- Unigrams?
- Bigrams?
- Trigrams?

Character N-grams

- A **character n-gram** applies the same idea to characters rather than words.
- E.g. `unnatural` has character bigrams `un`, `nn`, `na`, . . . , `al`
- Why might this concept be useful for NLP?

Towards Sentence Probabilities

- “Probability of a sentence” = how likely is it to occur in natural language
 - Consider only a specific language (English)

$P(\text{the cat slept peacefully}) > P(\text{slept the peacefully cat})$

$P(\text{she studies morphosyntax}) > P(\text{she studies more faux syntax})$

Language models in NLP

- It's very difficult to know the true probability of an arbitrary sequence of words.
- But we can define a **language model** that will give us good approximations.
- Like all models, language models will be good at capturing some things and less good for others.
 - We might want different models for different tasks.
 - Today, one type of language model: an **N-gram model**.

Spelling correction

Sentence probabilities help decide correct spelling.

mis-spelled text

no much effort



(Error model)

possible outputs

no much effect

so much effort

no much effort

not much effort

...



(Language model)

best-guess output

not much effort

Automatic speech recognition

Sentence probabilities help decide between similar-sounding options.

speech input



(Acoustic model)

possible outputs

She studies morphosyntax
She studies more faux syntax
She's studies morph or syntax

...



(Language model)

best-guess output

She studies morphosyntax

Machine translation

Sentence probabilities help decide word choice and word order.

non-English input



(Translation model)

possible outputs

She is going home

She is going house

She is traveling to home

To home she is going

...



(Language model)

best-guess output

She is going home

LMs for prediction

- LMs can be used for **prediction** as well as correction.
- Ex: predictive text correction/completion on your mobile phone.
 - Keyboard is tiny, easy to touch a spot slightly off from the letter you meant.
 - Want to correct such errors as you go, and also provide possible completions.
Predict as as you are typing: **ineff...**
- In this case, LM may be defined over sequences of *characters* instead of (or in addition to) sequences of words.

But how to estimate these probabilities?

- We want to know the probability of word sequence $\vec{w} = w_1 \dots w_n$ occurring in English.
- Assume we have some **training data**: large corpus of general English text.
- We can use this data to **estimate** the probability of \vec{w} (even if we never see it in the corpus!)

Probability theory vs estimation

- Probability theory can solve problems like:
 - I have a jar with 6 blue marbles and 4 red ones.
 - If I choose a marble uniformly at random, what's the probability it's red?

Probability theory vs estimation

- Probability theory can solve problems like:
 - I have a jar with 6 blue marbles and 4 red ones.
 - If I choose a marble uniformly at random, what's the probability it's red?
- But often we don't know the true probabilities, only have data:
 - I have a jar of marbles.
 - I repeatedly choose a marble uniformly at random and then replace it before choosing again.
 - In ten draws, I get 6 blue marbles and 4 red ones.
 - On the next draw, what's the probability I get a red marble?
- The latter also requires estimation theory.

Notation

- I will often omit the random variable in writing probabilities, using $P(x)$ to mean $P(X = x)$.
- When the distinction is important, I will use
 - $P(x)$ for *true* probabilities
 - $\hat{P}(x)$ for *estimated* probabilities
 - $P_E(x)$ for estimated probabilities using a particular estimation method E .
- But since we almost always mean estimated probabilities, may get lazy later and use $P(x)$ for those too.

Example estimation: M&M colors

What is the proportion of each color of M&M?

- In 48 packages, I find¹ 2620 M&Ms, as follows:

Red	Orange	Yellow	Green	Blue	Brown
372	544	369	483	481	371

- How to estimate probability of each color from this data?

¹Actually, data from: <https://joshmadison.com/2007/12/02/mms-color-distribution-analysis/>

Relative frequency estimation

- Intuitive way to estimate discrete probabilities:

$$P_{\text{RF}}(x) = \frac{C(x)}{N}$$

where $C(x)$ is the count of x in a large dataset, and $N = \sum_{x'} C(x')$ is the total number of items in the dataset.

Relative frequency estimation

- Intuitive way to estimate discrete probabilities:

$$P_{\text{RF}}(x) = \frac{C(x)}{N}$$

where $C(x)$ is the count of x in a large dataset, and $N = \sum_{x'} C(x')$ is the total number of items in the dataset.

- M&M example: $P_{\text{RF}}(\text{red}) = \frac{372}{2620} = .142$
- This method is also known as **maximum-likelihood estimation** (MLE) for reasons we'll get back to.

MLE for sentences?

Can we use MLE to estimate the probability of \vec{w} as a sentence of English? That is, the prob that some sentence S has words \vec{w} ?

$$P_{\text{MLE}}(S = \vec{w}) = \frac{C(\vec{w})}{N}$$

where $C(\vec{w})$ is the count of \vec{w} in a large dataset, and N is the total number of sentences in the dataset.

Sentences that have never occurred

the Archaeopteryx soared jaggedly amidst foliage

VS

jaggedly trees the on flew

- Neither ever occurred in a corpus (until I wrote these slides).
⇒ $C(\vec{w}) = 0$ in both cases: MLE assigns both zero probability.
- But one is grammatical (and meaningful), the other not.
⇒ Using MLE on full sentences doesn't work well for language model estimation.

The problem with MLE

- MLE thinks anything that hasn't occurred will never occur ($P=0$).
- Clearly not true! Such things can have differing, and non-zero, probabilities:
 - My hair turns blue
 - I injure myself in a skiing accident
 - I travel to Finland
- And similarly for word sequences that have never occurred.

Sparse data

- In fact, even things that occur once or twice in our training data are a problem. Remember these words from Europarl?

cornflakes, mathematicians, pseudo-rapporteur, lobby-ridden, Lycketoft, UNCITRAL, policyfor, Commissioneris, 145.95

All occurred once. Is it safe to assume all have equal probability?

- This is a **sparse data** problem: not enough observations to estimate probabilities well. (Unlike the M&Ms, where we had large counts for all colours!)
- For sentences, many (most!) will occur rarely if ever in our training data. So we need to do something smarter.

Towards better LM probabilities

- One way to try to fix the problem: estimate $P(\vec{w})$ by combining the probabilities of smaller parts of the sentence, which will occur more frequently.
- This is the intuition behind **N-gram language models**.

Deriving an N-gram model

- We want to estimate $P(S = w_1 \dots w_n)$.
 - Ex: $P(S = \text{the cat slept quietly})$.
- This is really a joint probability over the words in S :
 $P(W_1 = \text{the}, W_2 = \text{cat}, W_3 = \text{slept}, \dots W_4 = \text{quietly})$.
- Concisely, $P(\text{the, cat, slept, quietly})$ or $P(w_1, \dots w_n)$.

Deriving an N-gram model

- We want to estimate $P(S = w_1 \dots w_n)$.
 - Ex: $P(S = \text{the cat slept quietly})$.
- This is really a joint probability over the words in S :
 $P(W_1 = \text{the}, W_2 = \text{cat}, W_3 = \text{slept}, \dots W_4 = \text{quietly})$.
- Concisely, $P(\text{the, cat, slept, quietly})$ or $P(w_1, \dots w_n)$.
- Recall that for a joint probability, $P(X, Y) = P(Y|X)P(X)$. So,
$$\begin{aligned} P(\text{the, cat, slept, quietly}) &= P(\text{quietly}|\text{the, cat, slept})P(\text{the, cat, slept}) \\ &= P(\text{quietly}|\text{the, cat, slept})P(\text{slept}|\text{the, cat})P(\text{the, cat}) \\ &= P(\text{quietly}|\text{the, cat, slept})P(\text{slept}|\text{the, cat})P(\text{cat}|\text{the})P(\text{the}) \end{aligned}$$

Deriving an N-gram model

- More generally, the chain rule gives us:

$$P(w_1, \dots, w_n) = \prod_{i=1}^n P(w_i | w_1, w_2, \dots, w_{i-1})$$

- But many of these conditional probs are just as sparse!
 - If we want $P(\text{I spent three years before the mast})\dots$
 - we still need $P(\text{mast} | \text{I spent three years before the})$.

Example due to Alex Lascarides/Henry Thompson

Deriving an N-gram model

- So we make an **independence assumption**: the probability of a word only depends on a fixed number of previous words (**history**).
 - **trigram model**: $P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i|w_{i-2}, w_{i-1})$
 - **bigram model**: $P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i|w_{i-1})$
 - **unigram model**: $P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i)$
- In our example, a trigram model says
 - $P(\text{mast}|\text{I spent three years before the}) \approx P(\text{mast}|\text{before the})$

Trigram independence assumption

- Put another way, trigram model assumes these are all equal:
 - $P(\text{mast}|\text{I spent three years before the})$
 - $P(\text{mast}|\text{I went home before the})$
 - $P(\text{mast}|\text{I saw the sail before the})$
 - $P(\text{mast}|\text{I revised all week before the})$

because all are estimated as $P(\text{mast}|\text{before the})$

- Also called a **Markov assumption**



Andrey Markov →

- Not always a good assumption! But it does reduce the sparse data problem.

Estimating trigram conditional probs

- We still need to estimate $P(\text{mast}|\text{before, the})$: the probability of `mast` given the two-word history `before, the`.
- If we use relative frequencies (MLE), we consider:
 - Out of all cases where we saw `before, the` as the first two words of a trigram,
 - how many had `mast` as the third word?

Estimating trigram conditional probs

- So, in our example, we'd estimate

$$P_{MLE}(\text{mast}|\text{before, the}) = \frac{C(\text{before, the, mast})}{C(\text{before, the})}$$

where $C(x)$ is the count of x in our training data.

- More generally, for any trigram we have

$$P_{MLE}(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}$$

Example from *Moby Dick* corpus

$$\begin{aligned} C(\textit{before, the}) &= 55 & \frac{C(\textit{before, the, mast})}{C(\textit{before, the})} &= 0.0727 \\ C(\textit{before, the, mast}) &= 4 \end{aligned}$$

- *mast* is the second most common word to come after *before the* in *Moby Dick*; *wind* is the most frequent word.
- $P_{MLE}(\textit{mast})$ is 0.00049, and $P_{MLE}(\textit{mast|the})$ is 0.0029.
- So seeing *before the* vastly increases the probability of seeing *mast* next.

Trigram model: summary

- To estimate $P(\vec{w})$, use chain rule and make an indep. assumption:

$$P(w_1, \dots, w_n) = \prod_{i=1}^n P(w_i | w_1, w_2, \dots, w_{i-1})$$
$$\approx P(w_1)P(w_2 | w_1) \prod_{i=3}^n P(w_i | w_{i-2}, w_{i-1})$$

- Then estimate each trigram prob from data (here, using MLE):

$$P_{MLE}(w_i | w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}$$

- On your own: work out the equations for other N -grams (e.g., bigram, unigram).

Practical details (1)

- Trigram model assumes two word history:

$$P(\vec{w}) = P(w_1)P(w_2|w_1) \prod_{i=3}^n P(w_i|w_{i-2}, w_{i-1})$$

- But consider these sentences:

	w_1	w_2	w_3	w_4
(1)	he	saw	the	yellow
(2)	feeds	the	cats	daily

- What's wrong? Does the model capture these problems?

Beginning/end of sequence

- To capture behaviour at beginning/end of sequences, we can augment the input:

	w_{-1}	w_0	w_1	w_2	w_3	w_4	w_5
(1)	<s>	<s>	he	saw	the	yellow	</s>
(2)	<s>	<s>	feeds	the	cats	daily	</s>

- That is, assume $w_{-1} = w_0 = \text{<s>}$ and $w_{n+1} = \text{</s>}$ so:

$$P(\vec{w}) = \prod_{i=1}^{n+1} P(w_i | w_{i-2}, w_{i-1})$$

- Now, $P(\text{</s>} | \text{the, yellow})$ is low, indicating this is not a good sentence.

Beginning/end of sequence

- Alternatively, we could model all sentences as one (very long) sequence, including punctuation:

two cats live in sam 's barn . sam feeds the cats daily . yesterday , he
saw the yellow cat catch a mouse . [...]

- Now, trigrams like $P(.|cats\ daily)$ and $P(,|. yesterday)$ tell us about behavior at sentence edges.
- Here, all tokens are lowercased. What are the pros/cons of *not* doing that?

Practical details (2)

- Word probabilities are typically very small.
- Multiplying lots of small probabilities quickly gets so tiny we can't represent the numbers accurately, even with double precision floating point.
- So in practice, we typically use **negative log probabilities** (sometimes called **costs**):
 - Since probabilities range from 0 to 1, negative log probs range from 0 to ∞ :
lower cost = higher probability.
 - Instead of *multiplying* probabilities, we *add* neg log probabilities.

Interim Summary: N -gram probabilities

- “Probability of a sentence”: how likely is it to occur in natural language? Useful in many natural language applications.
- We can never know the true probability, but we may be able to estimate it from corpus data.
- N -gram models are one way to do this:
 - To alleviate sparse data, assume word probs depend only on short history.
 - Tradeoff: longer histories may capture more, but are also more sparse.
 - So far, we estimated N -gram probabilities using MLE.

Interim Summary: Language models

- **Language models** tell us $P(\vec{w}) = P(w_1 \dots w_n)$: *How likely to occur is this sequence of words?*

Roughly: *Is this sequence of words a “good” one in my language?*

- LMs are used as a component in applications such as speech recognition, machine translation, and predictive text completion.
- To reduce sparse data, N-gram LMs assume words depend only on a fixed-length history, even though we know this isn't true.

Coming up next:

- Weaknesses of MLE and ways to address them (more issues with sparse data).
- How to evaluate a language model: are we estimating sentence probabilities accurately?

Evaluating a language model

- Intuitively, a trigram model captures more context than a bigram model, so should be a “better” model.
- That is, it should more accurately predict the probabilities of sentences.
- But how can we measure this?

Two types of evaluation in NLP

- **Extrinsic**: measure performance on a downstream application.
 - For LM, plug it into a machine translation/ASR/etc system.
 - The most reliable evaluation, but can be time-consuming.
 - And of course, we still need an evaluation measure for the downstream system!
- **Intrinsic**: design a measure that is inherent to the current task.
 - Can be much quicker/easier during development cycle.
 - But not always easy to figure out what the right measure is: ideally, one that correlates well with extrinsic measures.

Let's consider how to define an intrinsic measure for LMs.

Entropy

- Definition of the **entropy** of a random variable X :

$$H(X) = \sum_x -P(x) \log_2 P(x)$$

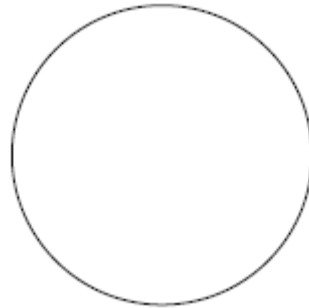
- Intuitively: a measure of uncertainty/disorder
- Also: the expected value of $-\log_2 P(X)$

Entropy Example

One event (outcome)

$$P(a) = 1$$

$$\begin{aligned} H(X) &= -1 \log_2 1 \\ &= 0 \end{aligned}$$



Entropy Example

2 equally likely events:

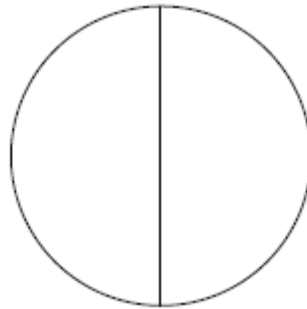
$$P(a) = 0.5$$

$$P(b) = 0.5$$

$$H(X) = -0.5 \log_2 0.5 - 0.5 \log_2 0.5$$

$$= -\log_2 0.5$$

$$= 1$$



Entropy Example

4 equally likely events:

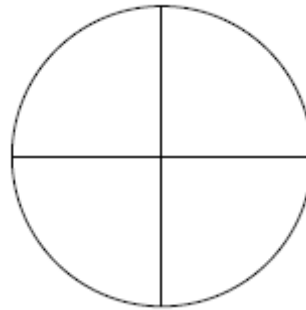
$$P(a) = 0.25$$

$$P(b) = 0.25$$

$$P(c) = 0.25$$

$$P(d) = 0.25$$

$$\begin{aligned} H(X) &= -0.25 \log_2 0.25 - 0.25 \log_2 0.25 \\ &\quad - 0.25 \log_2 0.25 - 0.25 \log_2 0.25 \\ &= -\log_2 0.25 \\ &= 2 \end{aligned}$$



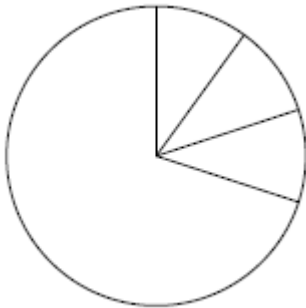
Entropy Example

$$P(a) = 0.7$$

$$P(b) = 0.1$$

$$P(c) = 0.1$$

$$P(d) = 0.1$$



3 equally likely events and one more likely than the others:

$$\begin{aligned} H(X) &= -0.7 \log_2 0.7 - 0.1 \log_2 0.1 \\ &\quad - 0.1 \log_2 0.1 - 0.1 \log_2 0.1 \\ &= -0.7 \log_2 0.7 - 0.3 \log_2 0.1 \\ &= -(0.7)(-0.5146) - (0.3)(-3.3219) \\ &= 0.36020 + 0.99658 \\ &= 1.35678 \end{aligned}$$

Entropy Example

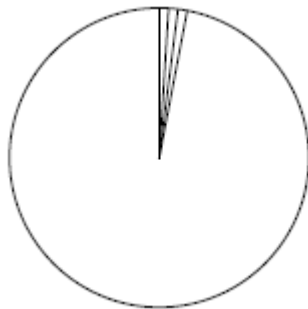
3 equally likely events and one much more likely than the others:

$$P(a) = 0.97$$

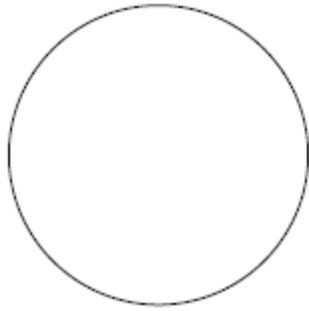
$$P(b) = 0.01$$

$$P(c) = 0.01$$

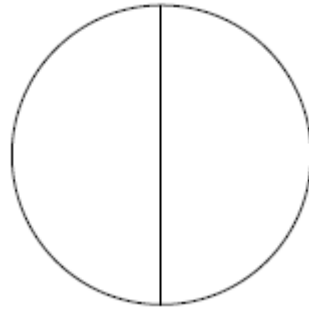
$$P(d) = 0.01$$



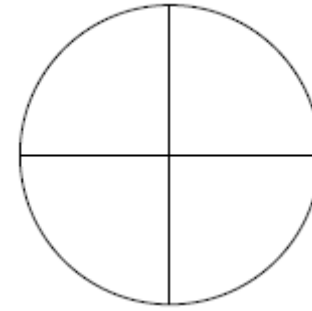
$$\begin{aligned} H(X) &= -0.97 \log_2 0.97 - 0.01 \log_2 0.01 \\ &\quad - 0.01 \log_2 0.01 - 0.01 \log_2 0.01 \\ &= -0.97 \log_2 0.97 - 0.03 \log_2 0.01 \\ &= -(0.97)(-0.04394) - (0.03)(-6.6439) \\ &= 0.04262 + 0.19932 \\ &= 0.24194 \end{aligned}$$



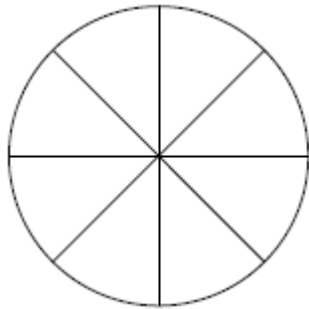
$$H(X) = 0$$



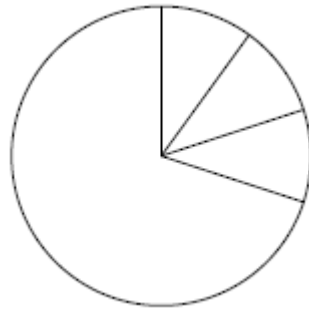
$$H(X) = 1$$



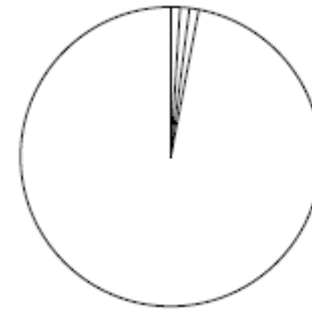
$$H(X) = 2$$



$$H(X) = 3$$



$$H(X) = 1.35678$$



$$H(X) = 0.24194$$

Entropy as y/n questions

How many yes-no questions (bits) do we need to find out the outcome?

- Uniform distribution with 2^n outcomes: n q's.
- Other cases: entropy is the average number of questions per outcome in a (very) long sequence of outcomes, where questions can consider multiple outcomes at once.

Entropy as encoding sequences

- Assume that we want to encode a sequence of events X .
- Each event is encoded by a sequence of bits, we want to use as few bits as possible.
- For example
 - Coin flip: heads = 0, tails = 1
 - 4 equally likely events: $a = 00$, $b = 01$, $c = 10$, $d = 11$
 - 3 events, one more likely than others: $a = 0$, $b = 10$, $c = 11$
 - Morse code: e has shorter code than q
- Average number of bits needed to encode $X \geq$ entropy of X

The Entropy of English

- Given the start of a text, can we guess the next word?
- For humans, the measured entropy is only about 1.3.
 - Meaning: on average, given the preceding context, a human would need only 1.3 y/n questions to determine the next word.
 - This is an upper bound on the true entropy, which we can never know (because we don't know the true probability distribution).
- But what about N -gram models?

Cross-entropy

- Our LM estimates the probability of word sequences.
- A good model assigns high probability to sequences that actually have high probability (and low probability to others).
- Put another way, our model should have low uncertainty (entropy) about which word comes next.
- We can measure this using **cross-entropy**.
- Note that **cross-entropy** \geq **entropy**: our model's uncertainty can be no less than the true uncertainty.

Computing cross-entropy

- For $w_1 \dots w_n$ with large n , per-word cross-entropy is well approximated by:

$$H_M(w_1 \dots w_n) = -\frac{1}{n} \log_2 P_M(w_1 \dots w_n)$$

- This is just the average negative log prob our model assigns to each word in the sequence. (i.e., normalized for sequence length).
- Lower cross-entropy \Rightarrow model is better at predicting next word.

Cross-entropy example

Using a bigram model from Moby Dick, compute per-word cross-entropy of *I spent three years before the mast* (here, without using end-of sentence padding):

$$\begin{aligned} & -\frac{1}{7} (\lg_2(P(I)) + \lg_2(P(\textit{spent}|I)) + \lg_2(P(\textit{three}|\textit{spent})) + \lg_2(P(\textit{years}|\textit{three})) \\ & \quad + \lg_2(P(\textit{before}|\textit{years})) + \lg_2(P(\textit{the}|\textit{before})) + \lg_2(P(\textit{mast}|\textit{the}))) \\ = & -\frac{1}{7} (-6.9381 - 11.0546 - 3.1699 - 4.2362 - 5.0 - 2.4426 - 8.4246) \\ = & -\frac{1}{7} (41.2660) \\ \approx & 6 \end{aligned}$$

- Per-word cross-entropy of the *unigram* model is about 11.
- So, unigram model has about 5 bits more uncertainty per word than bigram model. But, what does that mean?

Data compression

- If we designed an optimal code based on our bigram model, we could encode the entire sentence in about 42 bits.
- A code based on our unigram model would require about 77 bits.
- ASCII uses an average of 24 bits per word (168 bits total)!
- So better language models can also give us better data compression: as elaborated by the field of **information theory**.

Perplexity

- LM performance is often reported as **perplexity** rather than cross-entropy.
- Perplexity is simply $2^{\text{cross-entropy}}$
- The average branching factor at each decision point, if our distribution were uniform.
- So, 6 bits cross-entropy means our model perplexity is $2^6 = 64$: equivalent uncertainty to a uniform distribution over 64 outcomes.

Interpreting these measures

I measure the cross-entropy of my LM on some corpus as 5.2.
Is that good?

Interpreting these measures

I measure the cross-entropy of my LM on some corpus as 5.2.
Is that good?

- No way to tell! Cross-entropy depends on both the model and the corpus.
 - Some language is simply more predictable (e.g. casual speech vs academic writing).
 - So lower cross-entropy could mean the corpus is “easy”, or the model is good.
- We can only compare different models on the same corpus.
- Should we measure on training data or held-out data? Why?

Sparse data, again

Suppose now we build a *trigram* model from Moby Dick and evaluate the same sentence.

- But *I spent three* never occurs, so $P_{MLE}(\text{three} \mid \text{I spent}) = 0$
- which means the cross-entropy is infinite.
- Basically right: our model says *I spent three* should never occur, so our model is infinitely wrong/surprised when it does!
- Even with a unigram model, we will run into words we never saw before. So even with short N -grams, we need better ways to estimate probabilities from sparse data.

Smoothing

- The flaw of MLE: it estimates probabilities that make the training data maximally probable, by making everything else (unseen data) minimally probable.
- **Smoothing** methods address the problem by stealing probability mass from seen events and reallocating it to unseen events.
- Lots of different methods, based on different kinds of assumptions. We will discuss just a few.

Add-One (Laplace) Smoothing

- Just pretend we saw everything one more time than we did.

$$P_{\text{ML}}(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}$$

$$\Rightarrow P_{+1}(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i) + 1}{C(w_{i-2}, w_{i-1})} \quad ?$$

Add-One (Laplace) Smoothing

- Just pretend we saw everything one more time than we did.

$$P_{\text{ML}}(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}$$

$$\Rightarrow P_{+1}(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i) + 1}{C(w_{i-2}, w_{i-1})} \quad ?$$

- NO! Sum over possible w_i (in vocabulary V) must equal 1:

$$\sum_{w_i \in V} P(w_i|w_{i-2}, w_{i-1}) = 1$$

- If increasing the numerator, must change denominator too.

Add-one Smoothing: normalization

- We want:
$$\sum_{w_i \in V} \frac{C(w_{i-2}, w_{i-1}, w_i) + 1}{C(w_{i-2}, w_{i-1}) + x} = 1$$

- Solve for x :

$$\begin{aligned} \sum_{w_i \in V} (C(w_{i-2}, w_{i-1}, w_i) + 1) &= C(w_{i-2}, w_{i-1}) + x \\ \sum_{w_i \in V} C(w_{i-2}, w_{i-1}, w_i) + \sum_{w_i \in V} 1 &= C(w_{i-2}, w_{i-1}) + x \\ C(w_{i-2}, w_{i-1}) + v &= C(w_{i-2}, w_{i-1}) + x \\ v &= x \end{aligned}$$

where v = vocabulary size.

Add-one example (1)

- *Moby Dick* has one trigram that begins **I spent** (it's **I spent in**) and the vocabulary size is 17231.
- Comparison of MLE vs Add-one probability estimates:

	MLE	+1
$\hat{P}(\text{three} \mid \text{I spent})$	0	0.00006
$\hat{P}(\text{in} \mid \text{I spent})$	1	0.0001

- $\hat{P}(\text{in} \mid \text{I spent})$ seems very low, especially since **in** is a very common word. But can we find better evidence that this method is flawed?

Add-one example (2)

- Suppose we have a more common bigram w_1, w_2 that occurs 100 times, 10 of which are followed by w_3 .

	MLE	+1
$\hat{P}(w_3 w_1, w_2)$	$\frac{10}{100}$	$\frac{11}{17331}$
		≈ 0.0006

- Shows that the very large vocabulary size makes add-one smoothing steal way too much from seen events.
- In fact, MLE is pretty good for frequent events, so we shouldn't want to change these much.

Add- α (Lidstone) Smoothing

- We can improve things by adding $\alpha < 1$.

$$P_{+\alpha}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha v}$$

- Like Laplace, assumes we know the vocabulary size in advance.
- But if we don't, can just add a single "unknown" (UNK) item to the vocabulary, and use this for all unknown words during testing.
- Then: how to choose α ?

Optimizing α (and other model choices)

- Use a three-way data split: **training** set (80-90%), **held-out** (or **development**) set (5-10%), and **test** set (5-10%)
 - Train model (estimate probabilities) on training set with different values of α
 - Choose the α that minimizes cross-entropy on development set
 - Report final results on test set.
- More generally, use dev set for evaluating different models, debugging, and optimizing choices. Test set simulates deployment, use it only once!
- Avoids overfitting to the training set and even to the test set.

Summary

- We can measure the relative goodness of LMs on the same corpus using cross-entropy: how well does the model predict the next word?
- We need smoothing to deal with unseen N -grams.
- Add-1 and Add- α are simple, but not very good.

Postscript

- There are better smoothing methods for N -gram language models, including one called **Kneser-Ney smoothing**.
- But neural network language models can do even better without a Markov assumption. (Later in the course.)