



CMPT 413/713: Natural Language Processing

Language Models

Spring 2024
2024-01-10

Adapted from slides from Anoop Sarkar, Danqi Chen and Karthik Narasimhan

Consider

Today, in Vancouver, it is 31 F and red

vs

Today, in Vancouver, it is 31 F and snowing

- Both are grammatical
- But which is more likely?

What is Language Modeling?

- We want to be able to estimate the **probability** of a **sequence of words**
- How likely is a given phrase / sentence / paragraph / document?
- We want to be able to compute $P(s) = P(w_1, w_2, \dots, w_T)$

$$P(s) = P(w_1, \dots, w_T) = \prod_{t=1}^T P(w_t | w_{<t})$$

Why is this useful?

Applications

- Predicting words is important in many situations
 - Autocomplete
 - Machine translation

$$P(\text{a smooth finish}) > P(\text{a flat finish})$$

- Speech recognition/Spell checking

$$P(\text{high school principal}) > P(\text{high school principle})$$

- Information extraction, Question answering



Hypothesis scores for speech recognition

From acoustic signal to candidate transcriptions

| Hypothesis | Score |
|--|--------|
| the station signs are in deep in english | -14732 |
| the stations signs are in deep in english | -14735 |
| the station signs are in deep into english | -14739 |
| the station 's signs are in deep in english | -14740 |
| the station signs are in deep in the english | -14741 |
| the station signs are indeed in english | -14757 |
| the station 's signs are indeed in english | -14760 |
| the station signs are indians in english | -14790 |
| the station signs are indian in english | -14799 |
| the stations signs are indians in english | -14807 |
| the stations signs are indians and english | -14815 |

Hypothesis scores for machine translation

From source language to target language candidates

| Hypothesis | Score |
|-------------------------------------|--------|
| we must also discuss a vision . | -29.63 |
| we must also discuss on a vision . | -31.58 |
| it is also discuss a vision . | -31.96 |
| we must discuss on greater vision . | -36.09 |
| ⋮ | ⋮ |

Why is language model important?

- Much of the current successes in NLP comes from large pre-trained language models (BERT, GPT, T5, ...)
- By training neural language models, we can obtain useful representations for words and sentences.
- Can take the pre-trained language fine tune for specific tasks or use in zero-shot setting



matte painting of a bonsai tree;
trending on artstation.



Language Modeling

Predict probability of sequence of words

$$P(s) = P(w_1, \dots, w_T) = \prod_{t=1}^T P(w_t | w_{<t})$$

with n-grams

$$P(w_t | w_{<t}) \approx P(w_t | w_{t-n+1, t-1})$$

with HMMs

$$P(w_t | w_{<t}) \approx P(w_t | h_t) P(h_t | h_{t-n+1, t-1})$$

with neural networks

$$p(w_t | w_{<t}) \approx p(w_t | \phi(w_1, \dots, w_{t-1}))$$

with fixed window

$$P(w_t | w_{<t}) \approx P(w_t | \phi(w_{t-n+1, t-1}))$$

with RNNs

$$P(w_t | w_{<t}) \approx P(w_t | \mathbf{h}_t), \mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

What to know about LMs?

- What is a language model?
- Statistical language model using **ngrams**
- How to build a language model? **Training the model from data**
Learning/estimating model parameters
- MLE and smoothing
- How to use the language model? **Generation**
- How to tell if our language model is working well? **Evaluation**

What is a language model?

Probabilistic model of a sequence of words

Setup: Assume a finite vocabulary of words V

$$V = \{\text{cat, clown, crazy, killer, mat, on, sat, the}\}$$

V can be used to construct an infinite set of sentences (sequences of words)

$$V^+ = \{\text{clown, cat sat, killer clown, crazy clown, crazy cat, crazy killer clown, killer crazy clown, ...}\}$$

where a sentence is defined as $s \in V^+$ where $s = \{w_1, \dots, w_n\}$



What is a language model?

Probabilistic model of a sequence of words

Given a **training data** set of example sentences

$$S = \{s_1, s_2, \dots, s_m\}, s_i \in V^+$$

Estimate a probability model

$$\sum_{s_j \in V^+} P(s_j) = \sum_j P(w_1, \dots, w_{n_j}) = 1.0$$

Language Model

- $p(\text{clown}) = 1\text{e-}5$
- $p(\text{killer}) = 1\text{e-}6$
- $p(\text{killer clown}) = 1\text{e-}12$
- $p(\text{crazy killer clown}) = 1\text{e-}21$
- $p(\text{crazy killer clown killer}) = 1\text{e-}110$
- $p(\text{crazy slow killer killer}) = 1\text{e-}127$
- ...

Where do we get the vocabulary?

Common Setup: Assume a finite vocabulary of words V

- Get from a list of words (say a dictionary)
- Build from training data
- Decide on vocabulary size (say $|V| = 50K$) and then pick most frequent words
- Take words that occur more than T times.



Learning language models

How to estimate the probability of a sentence?

- We can **directly count** using a training data set of sentences

$$P(w_1, \dots, w_n) = \frac{C(w_1, \dots, w_n)}{N}$$

Problem: does not generalize to new sentences **unseen** in the training data

- C is a function that counts how many times each sentence occurs
- N is the sum over all possible $C(\cdot)$ values

Estimating joint probabilities with the chain rule

$$\begin{aligned} P(w_1, w_2, \dots, w_n) &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \times \dots \times P(w_n|w_1, w_2, \dots, w_{n-1}) \\ &= \prod_{i=1}^n P(w_i|w_1, \dots, w_{i-1}) \end{aligned}$$

Example

Sentence: “the cat sat on the mat”

$$\begin{aligned} P(\text{the cat sat on the mat}) &= P(\text{the}) \times P(\text{cat}|\text{the}) \times P(\text{sat}|\text{the cat}) \\ &\quad \times P(\text{on}|\text{the cat sat}) \times P(\text{the}|\text{the cat sat on}) \\ &\quad \times P(\text{mat}|\text{the cat sat on the}) \end{aligned}$$

Markov assumption

- Use only the recent past to predict the next word
- Reduces the number of estimated parameters in exchange for modeling capacity

Unigram

- 0th order $P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat})$

Bigram

- 1st order $P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{the})$

Trigram

- 2nd order $P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{on the})$

- kth order $P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-k} \dots w_{i-1})$

- Probability of sequence: $P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i | w_{i-k} \dots w_{i-1})$

Estimating n-gram probabilities

- **Maximum likelihood estimate (MLE):** Use counts from text corpus

Unigram
$$P(w_i) = \frac{C(w_i)}{\sum_{w_i \in V} C(w_i)} = \frac{C(w_i)}{N}$$

Bigram
$$P(w_i | w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

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

Can reuse counts for multiple estimations

Maximum Likelihood Estimation

We want to find the set of parameters $\hat{\theta}$ that maximize the probability of the training data

$$\hat{\theta} = \arg \max_{\theta \in \Theta} \hat{L}(\theta; \mathcal{D})$$

Parameters

$$\theta : \{p(w_i | w_1, \dots, w_{i-1})\}$$

Corpus of N sentences

Using our model, we can estimate the probabilities of these sentences

Likelihood

$$\hat{L}_m(\theta; \mathcal{D}) = \prod_{j=1}^m P(w_1^{(j)}, \dots, w_{n_j}^{(j)}) = \prod_{j=1}^m \prod_{i=1}^{n_j} P(w_i^{(j)} | w_1^{(j)}, \dots, w_{n_j}^{(j)})$$

Log-Likelihood

$$\hat{\ell}_m(\theta; \mathcal{D}) = \sum_{j=1}^m \log P(w_1^{(j)}, \dots, w_{n_j}^{(j)}) = \sum_{j=1}^m \sum_{i=1}^{n_j} \log P(w_i^{(j)} | w_1^{(j)}, \dots, w_{n_j}^{(j)})$$

Easier to work with (products to sums)
Numeric underflow less of a issue

Likelihood function

- How likely it is to see the examples in the training data
- Function of the parameters you are using to model the probability
- Probability density over data samples (sentences)

Typical assumptions

- Samples are iid (independently and identically distributed)

Maximum Likelihood Estimation (for categorical distributions)

- Unigram: $P(w)$
- Probability: $P(w) \geq 0, \sum_{w \in V} P(w) = 1$
- MLE is the **sample mean**
- Optimize $P_{\text{ML}}(w) = \arg \max_{P(w)} \sum_{j=1}^N \sum_{i=1}^{n_j} \log P(w_i^{(j)}) = \arg \max_{P(w)} \sum_{w \in V} C(w) \log P(w)$
- Solve using Lagrange Multipliers $g(\lambda, P(\cdot)) = \sum_{w \in V} C(w) \log P(w) - \lambda (\sum_{w \in V} P(w) - 1)$

$$P(w) = \frac{C(w)}{\lambda} \quad \longrightarrow \quad P(w) = \frac{C(w)}{\sum_{w \in V} C(w)} = \frac{C(w)}{N}$$

Using Language Models

How to use n-gram LMs?

- Computing probability

$$P(\text{high school } \mathbf{principal}) > P(\text{high school } \mathbf{principle})$$

- Completion

$$\arg \max_{w \in V} P(w | \text{where, is, SFU})$$

- Generating text

Computing the probability of a sentence

Apply the Chain Rule: the trigram model

$$\begin{aligned} P(w_1, \dots, w_n) \\ &\approx P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2) \dots P(w_n \mid w_{n-2}, w_{n-1}) \\ &\approx P(w_1)P(w_2 \mid w_1) \prod_{i=3}^n P(w_i \mid w_{i-2}, w_{i-1}) \end{aligned}$$

Not trigrams!
Pad our sentence
with <s>
(start of sentence
markers)

$$P(w_1) = P(w_1 \mid \langle s \rangle \langle s \rangle)$$

$$P(w_2 \mid w_1) = P(w_2 \mid \langle s \rangle w_1)$$

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

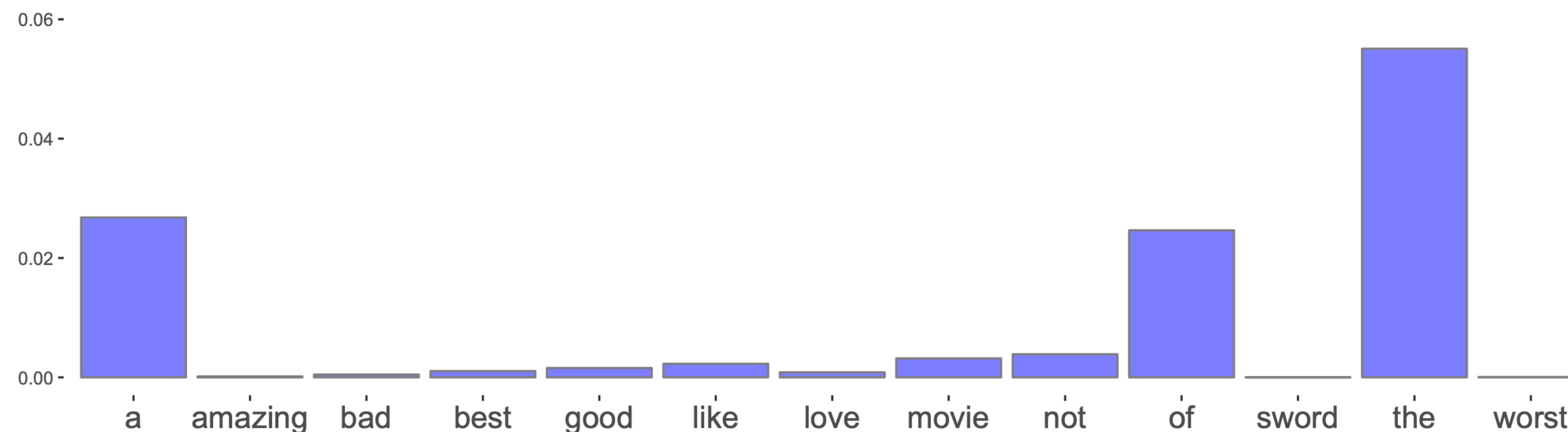
Not proper
distribution unless we
add a **stop** symbol
(or </s> end of
sentence marker)

Generating text from n-grams

generative model

- How do you generate text from an n-gram model?
- Sample from distribution, generating tokens from left to right
- Use the last $n - 1$ words for context
- Select from multinomial over the vocabulary that include a **STOP** token. Repeat until **STOP** is generated.

| context1 | context2 | generated word |
|----------|----------|----------------|
| START | START | The |
| START | The | dog |
| The | dog | walked |
| dog | walked | in |



Generalization

Number of Parameters

How many probabilities in each n -gram model

$$\mathcal{V} = \{\text{cat}, \text{crazy}, \text{mat}, \text{sat}\}$$

Question

How many unigram probabilities: $P(x)$ for $x \in \mathcal{V}$?

Number of Parameters

How many probabilities in each n -gram model

$$\mathcal{V} = \{\text{cat}, \text{crazy}, \text{mat}, \text{sat}\}$$

Question

How many bigram probabilities: $P(y|x)$ for $x, y \in \mathcal{V}$?

Number of Parameters

How many probabilities in each n -gram model

$$\mathcal{V} = \{\text{cat}, \text{crazy}, \text{mat}, \text{sat}\}$$

Question

How many trigram probabilities: $P(z|x, y)$ for $x, y, z \in \mathcal{V}$?

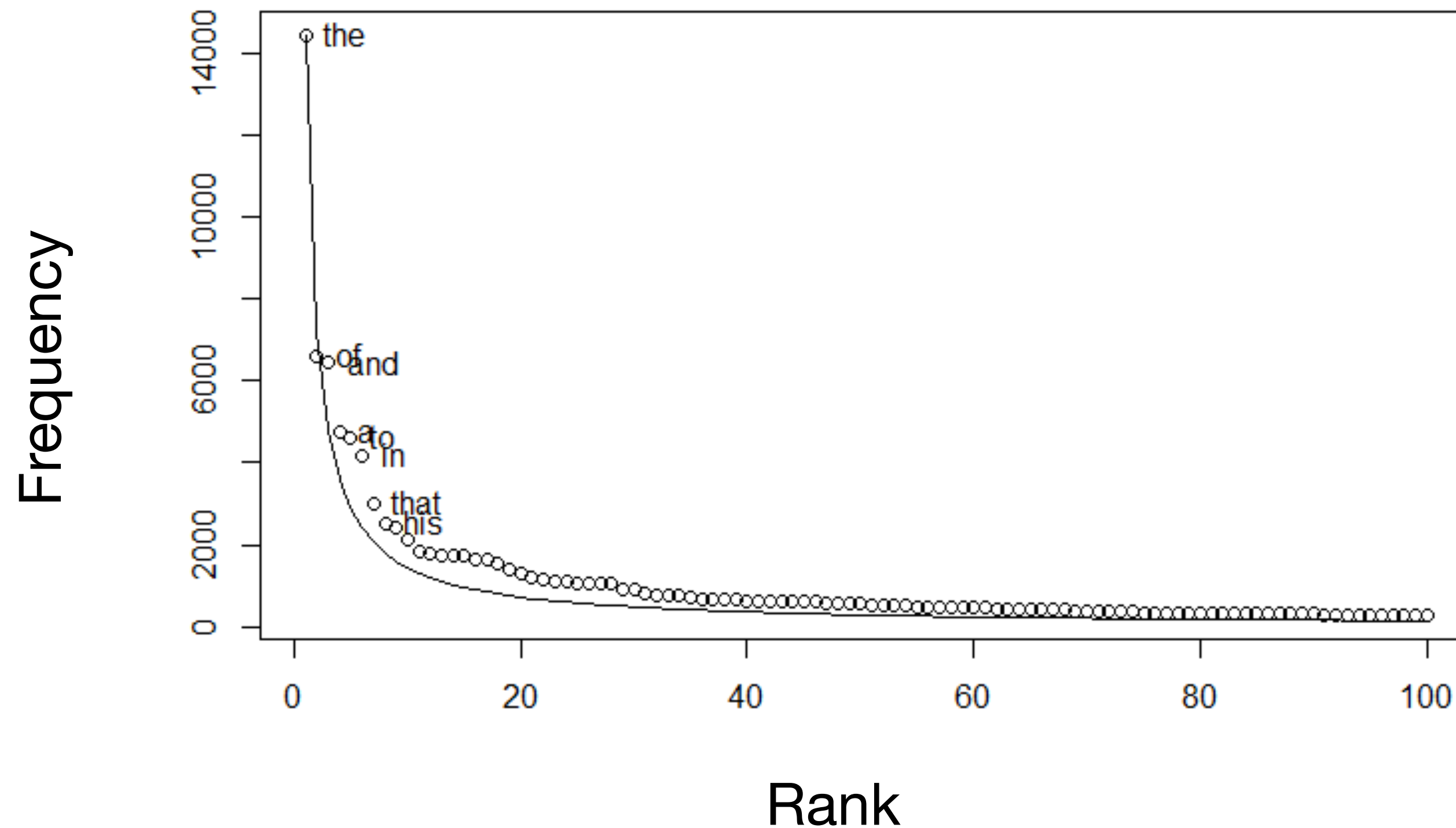
Number of parameters

- ▶ Assume $|\mathcal{V}| = 50,000$ (a realistic vocabulary size for English)
- ▶ What is the minimum size of training data in tokens?
 - ▶ If you wanted to observe all unigrams at least once.
 - ▶ If you wanted to observe all trigrams at least once.

Generalization of n-grams

- Not all n-grams will be observed in training data!
- There can be unknown words in the test set!
- Test corpus might have some that have zero probability under our model
- Training set: *Google news*
- Test set: *Shakespeare*
- $P(\text{affray} \mid \text{voice doth us}) = 0 \quad \Rightarrow \quad P(\text{test corpus}) = 0$

Sparsity in language



$$freq \propto \frac{1}{rank}$$

Zipf's Law

- Long tail of infrequent words
- Most finite-size corpora will have this problem.

Unknown words

- Typically assume closed vocabulary
- What about words not in the vocabulary?
 - Known as OOV (out-of-vocabulary) words.
 - Introduce **<UNK> token** to represent the unknown words
- If we never see these words in the training data, any sentence with these words will get a probability of 0!
- Can handle these unknown words by:
 - Estimate the probability of unknown word as: $P_{\text{unk}}(w) = \frac{1}{|V_{\text{all}}|}$ $V_{\text{all}} = V \cup \{< \text{UNK} >\}$
 - Or modify training data so rare words (words that appear $< T$ times) are treated as **<UNK>**

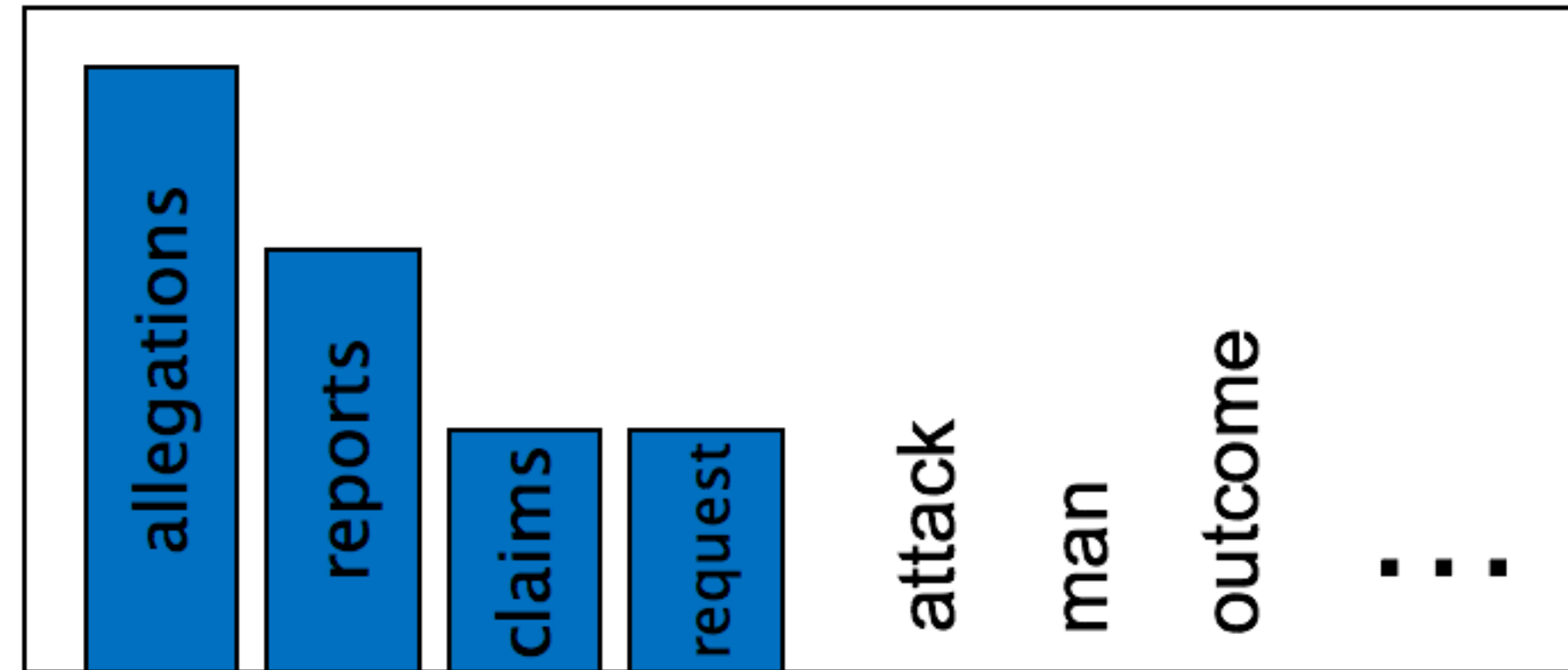
Smoothing n-gram Models

Smoothing intuition

Taking from the rich and giving to the poor

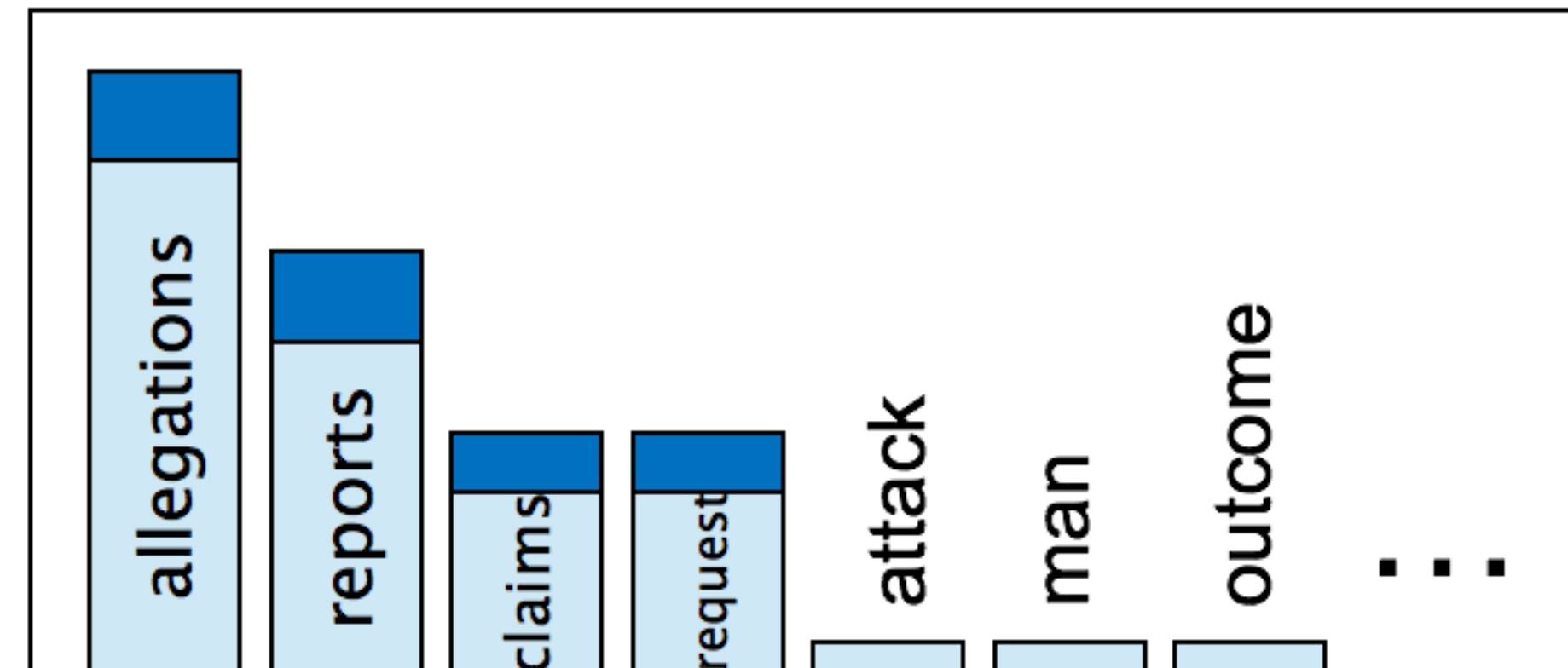
When we have sparse statistics:

$P(w \mid \text{denied the})$
3 allegations
2 reports
1 claims
1 request
7 total



Steal probability mass to generalize better

$P(w \mid \text{denied the})$
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



(Credits: Dan Klein)

Smoothing

- **Smoothing** deals with events that have been observed zero or very few times
- Handle sparsity by making sure all **probabilities are non-zero** in our model
- **Additive**: Add a small amount to all probabilities
- **Discounting**: Redistribute probability mass from observed n-grams to unobserved ones
- **Interpolation**: Use a combination of different n-grams
- **Back-off**: Use lower order n-grams if higher ones are too sparse

Ensure proper probability distribution

Add-one (Laplace) smoothing

- Why add 1? 1 is an overestimate for unobserved events

- Max likelihood estimate for bigrams: $P_{\text{ML}}(w_i | w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$

- Let $|V|$ be the number of words in our vocabulary. Assign count of 1 to unseen bigrams

- After smoothing:

$$P_{\text{Add1}}(w_i | w_{i-1}) = \frac{1 + C(w_{i-1}, w_i)}{|V| + C(w_{i-1})}$$

Additive smoothing

(Lidstone 1920, Jeffreys 1948)

- Why add 1? 1 is an overestimate for unobserved events

$$P_{\text{Add1}}(w_i | w_{i-1}) = \frac{1 + C(w_{i-1}, w_i)}{|V| + C(w_{i-1})}$$

- Additive smoothing ($0 < \delta \leq 1$):

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

- Also known as add-alpha (the symbol α is used instead of δ)

Raw bigram counts (Berkeley restaurant corpus)

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

- 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 |

(Credits: Dan Jurafsky)

Smoothed bigram counts

| | 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 |

Smoothed bigram probabilities

(Laplace Add-1 smoothing)

$$P_{\text{Add1}}(w_i | w_{i-1}) = \frac{1 + C(w_{i-1}, w_i)}{|V| + C(w_{i-1})}$$

| | 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 |

(Credits: Dan Jurafsky)

The problem with Laplace smoothing

Too much discounted
from popular words!

Raw
counts

| | 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 |

Reconstituted
counts

| | 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 |

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

(Credits: Dan Jurafsky)

Linear Interpolation (Jelinek-Mercer Smoothing)

$$\begin{aligned} P_{\text{interp}}(w_i | w_{i-1}, w_{i-2}) = & \lambda_1 P(w_i | w_{i-1}, w_{i-2}) \\ & + \lambda_2 P(w_i | w_{i-1}) \\ & + \lambda_3 P(w_i) \\ & + \lambda_4 \frac{1}{|V|} \end{aligned}$$
$$\sum_i \lambda_i = 1$$

- Use a combination of models to estimate probability
- Strong empirical performance

Jelinek and Mercer (1980)

Linear Interpolation (Jelinek-Mercer Smoothing)

- It's also possible to formulate the interpolation in a recursive manner:

$$P_{\text{JM}}(n\text{gram}) = \lambda_n P_{\text{ML}}(n\text{gram}) + (1 - \lambda_n) P_{\text{JM}}(n - 1\text{gram})$$

$$P_{\text{JM}}(w_i | w_{i-n+1}^{i-1}) = \lambda_n P_{\text{ML}}(w_i | w_{i-n+1}^{i-1}) + (1 - \lambda_n) P_{\text{JM}}(w_i | w_{i-n+2}^{i-1})$$

$$P_{\text{JM}}(w_i) = \lambda_1 P_{\text{ML}}(w_i) + (1 - \lambda_1) \frac{\delta}{|V|}$$

Brown et al (1992)

Linear Interpolation: Finding lambda

$$P_{\text{JM}}(\text{ngram}) = \lambda P_{\text{ML}}(\text{ngram}) + (1 - \lambda) P_{\text{JM}}(n - 1\text{gram})$$

- Interpolation parameters (λ) are hyper parameters. Tune them on the “held-out” set.



- Improved JM smoothing, a different λ for each w_i

$$P_{\text{JM}}(w_i | w_{i-1}) = \lambda(w_{i-1}) P_{\text{ML}}(w_i | w_{i-1}) + (1 - \lambda(w_{i-1})) P_{\text{JM}}(w_i)$$

Discounting

| 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 |

- Determine some “mass” to remove from probability estimates
- Redistribute mass among unseen n-grams
- Just choose an absolute value (D) to discount

more properly

$$\max(c(w_{i-1}, w_i) - D, 0)$$

α is set so the resulting probability values sums to one

Interpolated
absolute
discounting

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

Very similar to
Interpolated
Knesser-Ney


With interpolation, can also be with “backoff” as we will see

Interpolated Knesser-Ney

Popular state of the art n-gram smoothing

Cleverer count (based
on number of contexts)

Modified Knesser-Ney:
different discounting values

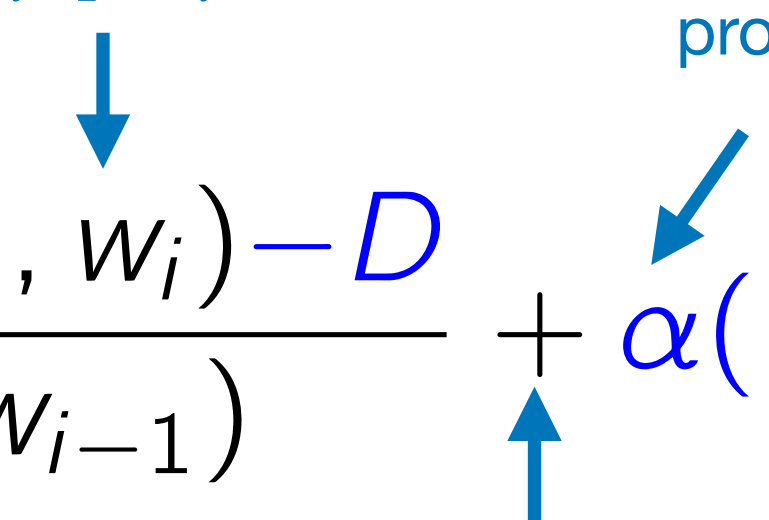

$$P_{\text{KN-i}}(w_i | w_{i-1}) = \frac{\max(C_{\text{KN}}(w_{i-1}, w_i) - D)}{\sum_{w'} C_{\text{KN}}(w_{i-1}, w')} + \alpha(w_{i-1}) P_{\text{KN-i}}(w_i)$$

more properly

$\max(c(w_{i-1}, w_i) - D, 0)$

α is set so the resulting
probability values sums to one

Interpolated
absolute
discounting


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

Very similar to
Interpolated
Knesser-Ney

With interpolation, can also be with “backoff” as we will see

Back-off

- Use **n-gram** if enough evidence, else back off to **(n-1)-gram**

$$P_{\text{bo}}(w_i | w_{i-n+1}^{i-1}) \\ = \begin{cases} d_{w_{i-n+1}^i} \frac{C(w_{i-n+1}^i)}{C(w_{i-n+1}^{i-1})} & \text{if } C(w_{i-n+1}^i) > 0 \\ \alpha_{w_{i-n+1}^{i-1}} P_{\text{bo}}(w_i | w_{i-n+2}^{i-1}) & \text{otherwise} \end{cases}$$

- d = amount of discounting (Katz back-off, 1987)
- α = back-off weight

Backoff Smoothing with Discounting

- Absolute Discounting with backoff (Ney, Essen, Knessler)

$$P_{\text{absdis-bo}}(w_i | w_{i-1}) = \begin{cases} \frac{C(w_{i-1} w_i) - D}{C(w_{i-1})} & \text{if } C(w_{i-1} w_i) > 0 \\ \alpha(w_{i-1}) P_{\text{absdis-bo}}(w_i) & \text{otherwise} \end{cases}$$

- Where $\alpha(w_{i-1})$ is chosen to ensure that $P_{\text{abs}}(w_i | w_{i-1})$ is a proper probability

Similar to
Backoff
Knessler-Ney,
1994

$$\alpha(w_{i-1}) = 1 - \sum_{w_i} \frac{C(w_{i-1} w_i) - D}{C(w_i)}$$

Different value of α for
each context word w_{i-1}

Backoff Smoothing with Discounting

- Let $D = 0.5$

- Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w_i} \frac{C(w_{i-1}w_i) - D}{C(w_i)}$$

$$\alpha(\text{the}) = 10 \times 0.5/48 = 5/48$$

- Divide this mass between words w for which the counts: $C(\text{the}, w) = 0$

| x | $c(x)$ | $c(x) - D$ | $\frac{c(x) - D}{c(\text{the})}$ |
|---------------|--------|------------|----------------------------------|
| the | 48 | | |
| the,dog | 15 | 14.5 | 14.5/48 |
| the,woman | 11 | 10.5 | 10.5/48 |
| the,man | 10 | 9.5 | 9.5/48 |
| the,park | 5 | 4.5 | 4.5/48 |
| the,job | 2 | 1.5 | 1.5/48 |
| the,telescope | 1 | 0.5 | 0.5/48 |
| the>manual | 1 | 0.5 | 0.5/48 |
| the,afternoon | 1 | 0.5 | 0.5/48 |
| the,country | 1 | 0.5 | 0.5/48 |
| the,street | 1 | 0.5 | 0.5/48 |
| TOTAL | | | 0.8958 |
| the,UNK | 0 | | 0.1042 |

Web-scale N-grams Smoothing

Keeping track of everything gets complicated

Not even a proper distribution!

- “Stupid backoff” (Brants et al, 2007)

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

S = Score

$$S(w_i) = \frac{C(w_i)}{|V|}$$

Other challenges

- Efficient storage
- Efficient lookup

<https://www.aclweb.org/anthology/D07-1090.pdf>

Beyond n-grams

Other types of language models

- Discriminative models:
 - train n-gram probabilities to directly maximize performance on end task (e.g. as feature weights)
- Parsing-based models
 - handle syntactic/grammatical dependencies
- Topic models
- Neural Language Models

Summary: Estimating language models

- ▶ Predict probability of sequence of words
- ▶ Need to handle **data sparsity**
use Markov assumption and smoothing
Independence assumptions
Reallocate probability mass
Ensure proper probability
- ▶ Later: Neural language models

Use Chain rule and approximate using a neural network

$$p(w_1, \dots, w_n) \approx \prod_t p(w_{t+1} \mid \underbrace{\phi(w_1, \dots, w_t)}_{\text{capture history with vector } s(t)})$$

How well do these models perform?

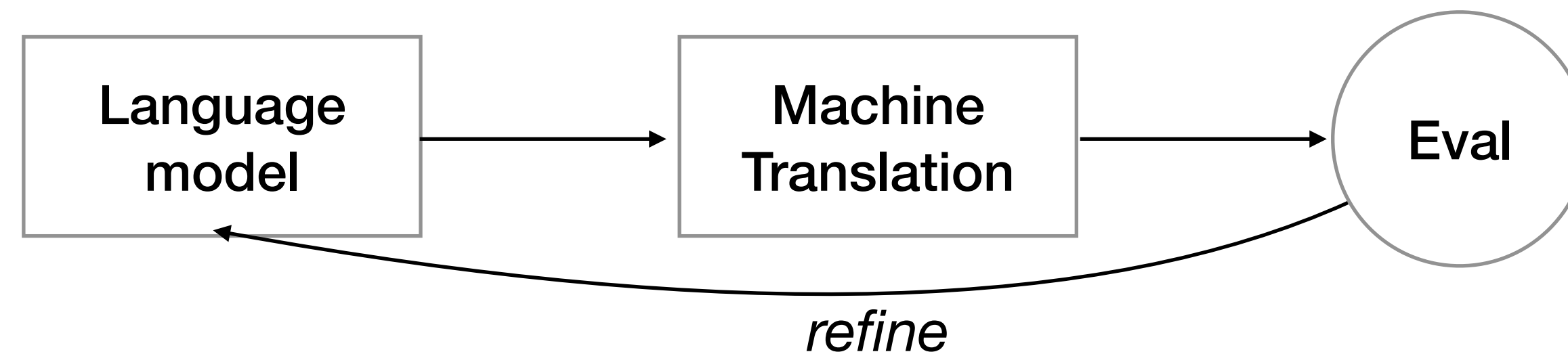
Evaluating Language Models

Evaluation

- **Extrinsic**: measure how useful the language model is at some task (MT, ASR, etc).
- **Intrinsic**: measure how good we are at modeling language

Extrinsic evaluation

- Train LM -> apply to task -> observe accuracy



- Directly optimized for downstream tasks
 - higher task accuracy -> better model
- Expensive, time consuming
- Hard to optimize downstream objective (indirect feedback)

Evaluating language models

- A good language model should assign higher probability to typical, grammatically correct sentences
- Research process:
 - **Train** parameters on a suitable training corpus
 - Assumption: observed sentences \sim good sentences
 - **Test** on *different, unseen* corpus
 - Training on any part of test set not acceptable!
 - **Evaluation metric**



Evaluation of language models

Computing the **average** probability of the test corpus

- Given a test corpus $T = s_1, \dots, s_m$ of independent sentences, the probability of $P(T)$ is:

$$P(T) = \prod_{i=1}^m P(s_i) \quad \text{higher } P(T) = \text{better LM}$$

- But T can be any size and $P(T)$ will be lower if T is larger.
- So let's compute the **average** probability. Let M be the total number of tokens in the test corpus T .

$$M = \sum_{i=1}^m \text{length}(s_i)$$

- The average **log** probability of the test corpus T is:

$$\ell = \frac{1}{M} \log_2 \prod_{i=1}^m P(s_i) = \frac{1}{M} \sum_{i=1}^m \log_2 P(s_i)$$

Evaluation of language models

Perplexity

M = total # of words

m = # of sentences

- The average log probability of the test corpus T is:

$$\ell = \frac{1}{M} \sum_{i=1}^m \log_2 P(s_i)$$

higher ℓ = better LM

Note that ℓ is
a negative number

- Language models are evaluated using **perplexity**

$$\text{ppl}(T) = 2^{-\ell} \quad \text{lower ppl} = \text{better LM}$$

$\text{ppl}(T)$ is
a positive number

Note that the exponent ($-\ell$) can be regarded as the **cross entropy** between the empirical distribution of test corpus and the language model

Intuition: **Measure of model's uncertainty about next word**

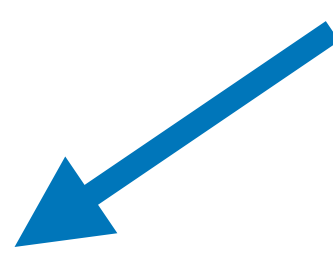
Perplexity summary

- Measure of how well a probability distribution (or model) predicts a sample

- For a corpus T with sentences s_1, s_2, \dots, s_m

$$\text{ppl}(T) = 2^{-\ell} \text{ where } -\ell = -\frac{1}{M} \sum_{i=1}^m \log_2 P(s_i)$$

cross entropy
between the
empirical
distribution of test
corpus and the
language model



where M is the total number of words in test corpus

- Unigram model: $-\ell = -\frac{1}{M} \sum_{i=1}^m \sum_{j=1}^{n_i} \log_2 P(w_j^{(i)}) = -\sum_{w \in V} \frac{C(w)}{M} \log_2 P(w)$
- Minimizing perplexity ~ maximizing probability

Intuition: Measure of model's uncertainty about next word
branching factor

Pros and cons of perplexity

| Pros | Cons |
|--|--|
| Easy to compute | Domain match between train and test |
| standardized | Limited to sequence models |
| directly useful, easy to use to correct sentences | might not correspond to end task optimization |
| nice theoretical interpretation - matching distributions | $\log 0$ undefined |
| | can be cheated by predicting common tokens |
| | size of test set matters |
| | can be sensitive to low prob tokens/ sentences |

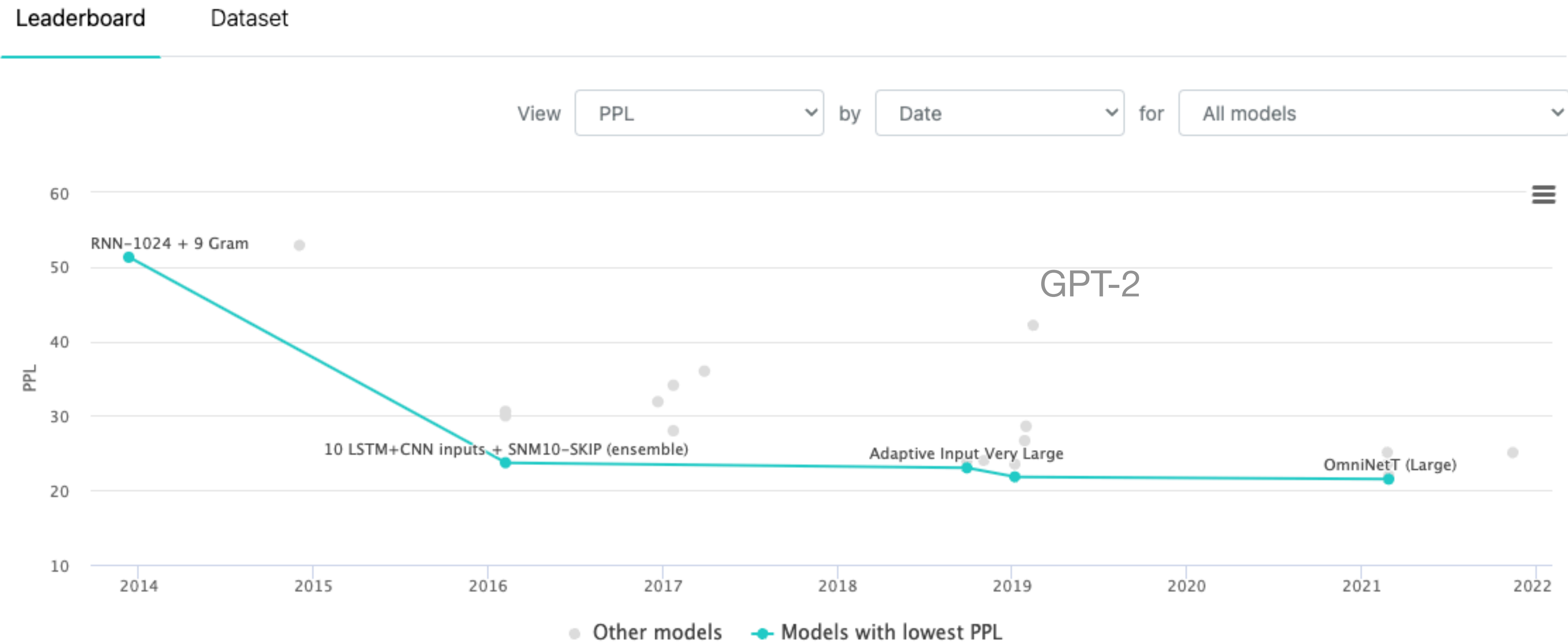
Perplexity values for different language models

Progress on the 1B Word Benchmark

| Model | Params | Perplexity | Citation |
|---------------------|--------|------------|----------------------------------|
| unigram | 775K | 955 | Chelba+ 2013 |
| bigram | 1B | 137 | Chelba+ 2013 |
| trigram | 1B | 74 | Chelba+ 2013 |
| interpolated 5-gram | 1.76B | 67.6 | Chelba+ 2013 |
| 10skip-gram+SNM | 33B | 52.9 | Shazeer+ 2014 |
| RNN-256 + 9-grams | 20B | 58.3 | Chelba+ 2013 |
| RNN-1024 + 9-grams | 20B | 51.3 | Chelba+ 2013 |
| Big LSTM+CNN | 1.04B | 30 | Jozefowicz+ 2016 |
| 10 LSTMs+10skip-SNM | 43B | 23.7 | Jozefowicz+ 2016 |
| GPT2 | 1.54B | 42.16 | Radford+ 2019 |
| Transformer XL | 1.04B | 21.8 | Dai+ 2019 |
| OmniNet | 100M | 21.5 | Tay+ 2021 |

Perplexity of current LMs

Language Modelling on One Billion Word



<https://paperswithcode.com/sota/language-modelling-on-one-billion-word>

For other datasets: see <https://paperswithcode.com/task/language-modelling>

Where are we now?

Neural models with lots of data!

300 Billion tokens

Training data: mix of web + books + Wikipedia

| Model Name | n_{params} |
|-----------------------|---------------------|
| GPT-3 Small | 125M |
| GPT-3 Medium | 350M |
| GPT-3 Large | 760M |
| GPT-3 XL | 1.3B |
| GPT-3 2.7B | 2.7B |
| GPT-3 6.7B | 6.7B |
| GPT-3 13B | 13.0B |
| GPT-3 175B or “GPT-3” | 175.0B |

| Dataset | <small>1B Word Benchmark test set</small> | Quantity (tokens) | Weight in training mix | Epochs elapsed when training for 300B tokens |
|-------------------------|---|----------------------|---------------------------|---|
| Common Crawl (filtered) | | 410 billion | 60% | 0.44 |
| WebText2 | | 19 billion | 22% | 2.9 |
| Books1 | | 12 billion | 8% | 1.9 |
| Books2 | | 55 billion | 8% | 0.43 |
| Wikipedia | | 3 billion | 3% | 3.4 |

Open AI's GPT 3

Language Models are Few-Shot Learners
(Brown et al, 2020)

<https://arxiv.org/pdf/2005.14165.pdf>

| Setting | PTB |
|------------------|-------------------|
| SOTA (Zero-Shot) | 35.8 ^a |
| GPT-3 Zero-Shot | 20.5 |

Why the stop symbol is important?

Computing the probability of a sentence

Apply the Chain Rule: the trigram model

$$\begin{aligned} P(w_1, \dots, w_n) \\ &\approx P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2) \dots P(w_n \mid w_{n-2}, w_{n-1}) \\ &\approx P(w_1)P(w_2 \mid w_1) \prod_{i=3}^n P(w_i \mid w_{i-2}, w_{i-1}) \end{aligned}$$

- Notice that the length of the sentence n is variable
- What is size of the event space (e.g. the total number of possible events/sentences)?

Variable length sequences

Let $V = \{a, b\}$ and the language L be V^*

Consider a unigram model: $P(a) = P(b) = 0.5$

So strings in this language L are:

$a \quad 0.5$

$b \quad 0.5$

$aa \quad 0.5^2$

$bb \quad 0.5^2$

\dots

The sum over all strings in L should be equal to 1

But $P(a) + P(b) + P(aa) + P(bb) = 1.5!!!$

The **stop** symbol

What went wrong? We need to model variable length sentences

Add an explicit probability for the stop symbol

$$P(a) = P(b) = 0.25 \quad P(\text{stop}) = 0.5$$

Now strings have the following probabilities:


$$\begin{array}{ll} \text{stop} & 0.5 \\ a \text{ stop} & 0.25 \times 0.5 = 0.125 \\ b \text{ stop} & 0.25 \times 0.5 = 0.125 \\ aa \text{ stop} & 0.25^2 \times 0.5 = 0.03125 \\ bb \text{ stop} & 0.25^2 \times 0.5 = 0.03125 \\ & \dots \end{array}$$

The sum is no longer greater than one!

The stop symbol

- With this new stop symbol, we can show that $\sum_{u \in L} P(u) = 1$
- Let $p_s = P(\text{stop})$, the probability of the stop symbol
- Then, we can show that the probability of all sequences of length n is $p(n) = p_s(1 - p_s)^n$

$$\begin{aligned}
 p(n) &= \sum_{w_1, \dots, w_n} p(w_1, \dots, w_n) \times p_s \text{ where } w_j \neq \text{stop} \\
 &= p_s \sum_{w_1} \dots \sum_{w_n} p(w_1, \dots, w_n) \\
 &= p_s \sum_{w_1} \dots \sum_{w_n} p(w_1) \dots p(w_n) \\
 &= p_s \sum_{w_1} p(w_1) \dots \sum_{w_n} p(w_n) \\
 &= p_s \prod_{j=1}^n \sum_{w_j} p(w_j) \\
 &= p_s (1 - p_s)^n
 \end{aligned}$$

$\sum_{w \neq \text{stop}} P(w) = 1 - p_s$


The stop symbol

- With this new stop symbol, we can show that $\sum_{u \in L} P(u) = 1$
- Let $p_s = P(\text{stop})$, the probability of the stop symbol
- Using that the probability of all sequences of length n is $p(n) = p_s(1 - p_s)^n$

$$\begin{aligned}\sum_{u \in L} P(u) &= \sum_{n=0}^{\infty} p(n) = \sum_{n=0}^{\infty} p_s(1 - p_s)^n \\ &= p_s \sum_{n=0}^{\infty} (1 - p_s)^n \\ &= p_s \frac{1}{1 - (1 - p_s)} = p_s \frac{1}{p_s} = 1\end{aligned}$$

Summary

- Language models estimates the probability of a sentence
- Statistical LMs: N-grams: $P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$
- Smoothing to handle data sparsity
- Perplexity for evaluating language models
- Modern NLP powered by neural-based language models

Reminders

- HW-0 due next Wednesday 11/17
 - Submit via gradescope and coursys
- Next week:
 - Classification for NLP