CMPT 413/713: Natural Language Processing

Language Models

Zhiyun (Richard) Peng

2026-09-11

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

What is Language Modeling?

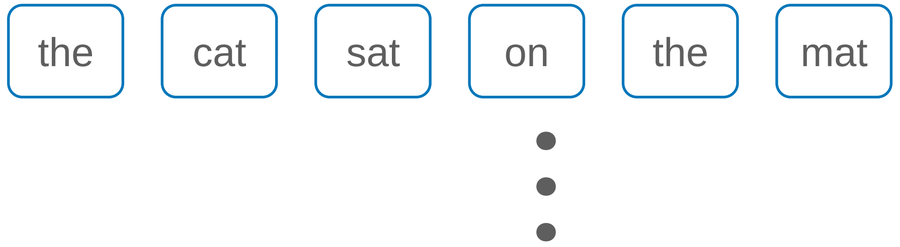
- We want to be able to estimate the probability of a sequence of words
  - How likely is a given phrase / sentence / paragraph / document?
  - We want to be able to compute $P(s) = P(w_1, w_2, \ldots, w_T)$

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

Processing language

the cat sat on the mat
a mouse saw the grey cat
the cat chased the mouse

Tokenized text

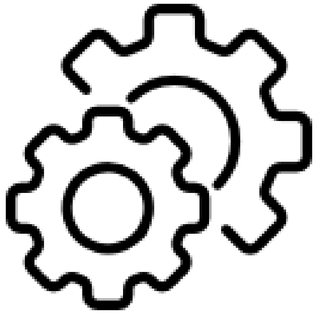


Corpus of text

Vocabulary

a	on
cat	sat
chased	saw
grey	the
mat	
mouse	

Tokenize and build vocabulary



Build model

What is a language model?

Probabilistic model of a sequence of words

Setup: Assume a finite vocabulary of words V

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

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

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

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

What is a language model?

Probabilistic model of a sequence of words

Given a training data set of example sentences $S = \{s_1, s_2, \ldots, s_m\}, s_i \in V^+$

Estimate a probability model

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

Language Model

- p(clown) = 1e-5
- p(killer) = 1e-6
- p(killer clown) = 1e-12
- p(crazy killer clown) = 1e-21
- p(crazy killer clown killer) = 1e-110
- …
- p(crazy slow killer killer) = 1e-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 **tokenized** 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
  - $P(w_1, \ldots, w_n) = \frac{C(w_1,\ldots,w_n)}{N}$

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

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

Example Sentence: “the cat sat on the mat”

$$P(\text{the cat sat on the mat}) = P(\text{the}) \times P(\text{cat}|\text{the}) \times P(\text{sat}|\text{the cat})$$
$$\times \text{P}(\text{on}|\text{the cat sat}) \times P(\text{the}|\text{the cat sat on})$$
$$\times P(\text{mat}|\text{the cat sat on the})$$

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

Estimating n-gram probabilities

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

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

Number of parameters grows as $|V|^n$! (n=1,2, or 3, here)

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

Some combinations will not be in training set!

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 (e.g. words in our corpus)

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

Parameters

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

Corpus of m sentences

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

Likelihood

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

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)

Log-Likelihood

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

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

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 $P_{\mathrm{ML}}(w) = \arg\max_{P(w)} \sum_{j=1}^{N} \sum_{i=1}^{n_j} \log P(w_i^{(j)}) = \arg\max_{P(w)} \sum_{w \in V} C(w) \log P(w)$

$2 \log P(\text{"the"}) + 1 \log P(\text{"cat"}) + 1 \log P(\text{"dog"})$

$\log P(\text{"the"}) + \log P(\text{"cat"}) + \log P(\text{"the"}) + \log P(\text{"dog"})$

- Solve using Lagrange Multipliers $g(\lambda, P(\cdot)) = \sum_{w \in V} C(w) \log P(w) - \lambda \left(\sum_{w \in V} P(w) - 1 \right)$

$$\frac{\partial g}{\partial P(w)} = \frac{C(w)}{P(w)} - \lambda = 0 \implies P(w) = \frac{C(w)}{\lambda} \longrightarrow \underset{\text{MLE}}{P(w)} = \frac{C(w)}{\sum_{w \in V} C(w)} = \frac{C(w)}{N}$$

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

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

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

$$P(w_1) = P(w_1| < s >< s >)$$
$$P(w_2|w_1) = P(w_2| < s > w_1)$$

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

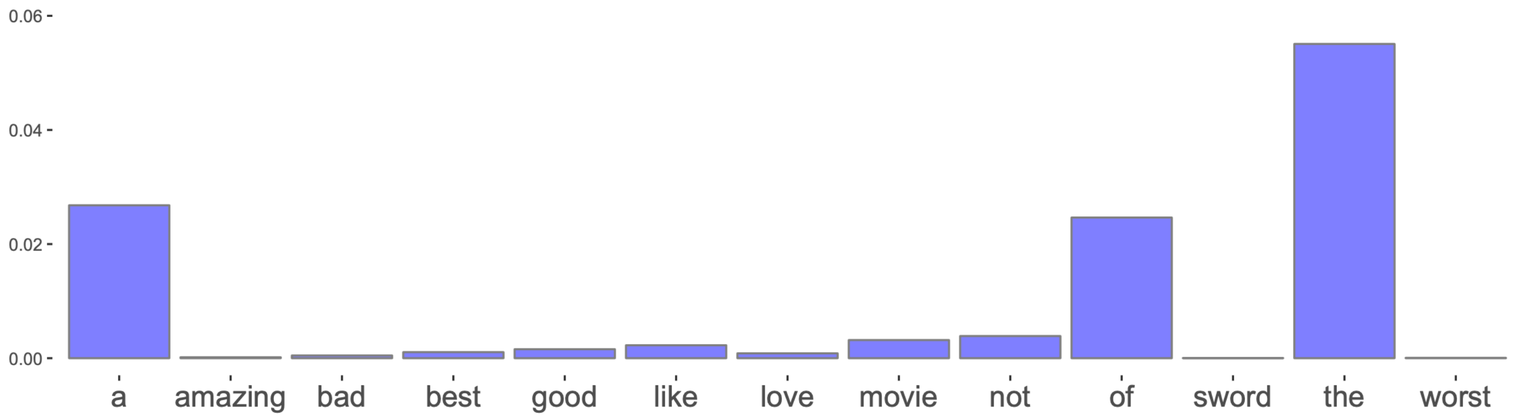
Not proper distribution unless we add a stop symbol (or </s> end of sentence marker)

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

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 (affray | voice doth us) = 0 → P(test corpus) = 0

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: $P_{\text{unk}}(w) = \frac{1}{|V_{\text{all}}|} \quad V_{\text{all}} = V \cup \{< \text{UNK} >\}$
  - Or modify training data so rare words (words that appear < *T* times) are treated as <UNK>

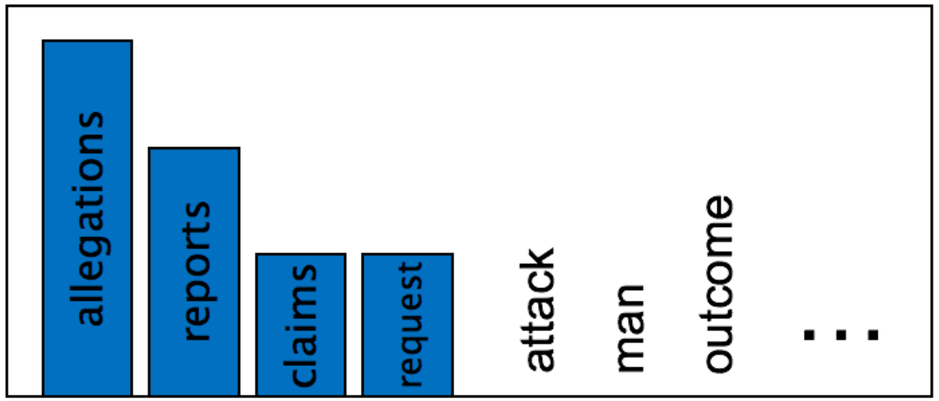
Smoothing n-gram Models

Smoothing intuition

Taking from the rich and giving to the poor

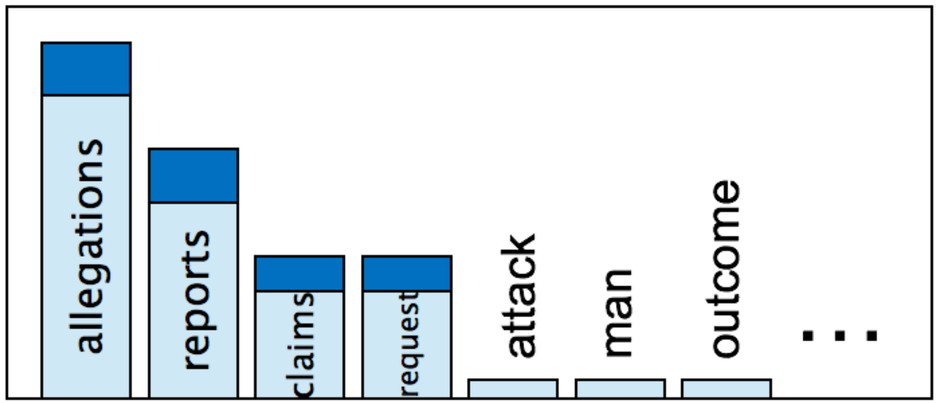
When we have sparse statistics:

P(w | 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



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

(Credits: Dan Jurafsky)

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)

Linear Interpolation (Jelinek-Mercer Smoothing)

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

$$\sum_i \lambda_i = 1$$

- 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

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

λ

- Interpolation parameters () are hyper parameters. Tune them on the "held-out" set.

- Improved JM smoothing, a different λ for each w_i

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

To skip

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

Interpolated absolute discounting

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

α is set so the resulting probability values sums to one

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

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

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

Interpolated absolute discounting

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

α is set so the resulting probability values sums to one

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

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

$$P_{\text{bo}}(w_i|w_{i-n+1}^{i-1})$$

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

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

- α = back-off weight

Backoff Smoothing with Discounting

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

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

$P_{\text{absdis}}(w_i|w_{i-1})$

- Where α is chosen to ensure that $P_{\text{absdis-bo}}$ is a proper probability

Similar to Backoff Knesser-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 wi-1

Backoff Smoothing with Discounting

- Let D = 0.5
- Missing probability mass:

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

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

- Divide this mass between words *w* for which the counts: *C(*the, *w) = 0*

x	$c(x)$	$c(x) - D$	$\frac{c(x)-D}{c(the)}$
the	48		
the,dog	15	14.5	14.5/48
the,woman	11	10.5	10.4/48
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 = Score

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

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

Other challenges

- Efficient storage
- Efficient lookup

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

To skip. end

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 $P(w_t \mid w_{t-1}, z)$
- 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

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

How well do these models perform?

Evaluating Language Models

Evaluation

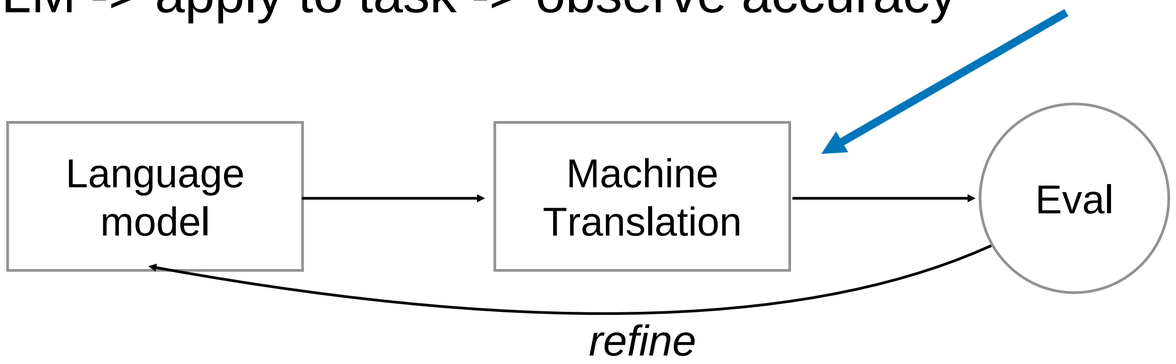
- Extrinsic: measure how useful the language model is at some task - machine translation (MT), automatic speech recognition (ASR), question answer (QA), etc.

- Intrinsic: measure how good we are at modeling language

Extrinsic evaluation--Evaluates the model by embedding it inside a downstream task and measuring end-system performance.

Many different tasks and benchmarks

- 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)

Intrinsic evaluation- Evaluates the model directly on its internal training/objective function (using held-out text)

- 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 ~ 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 $T = s_1, \ldots, s_m$ of independent sentences, the probability of $P(T)$ is:

joint probability

$$P(T) = \prod_{i=1}^{m} P(s_i)$$

higher $P(T)$ = 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.

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

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

log-likelihood

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

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

$$ppl(T) = 2^{-\ell}$$

lower ppl = better LM

$ppl(T)$ 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

p: true prob distribution

q: model distribution

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

$$H(p, q) = -\sum_{x} p(x) \log_2 q(x) = \mathbb{E}_{x \sim p}[-\log_2 q(x)]$$

- For a corpus *T* with sentences $s_1, s_2, \ldots, s_m$

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

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

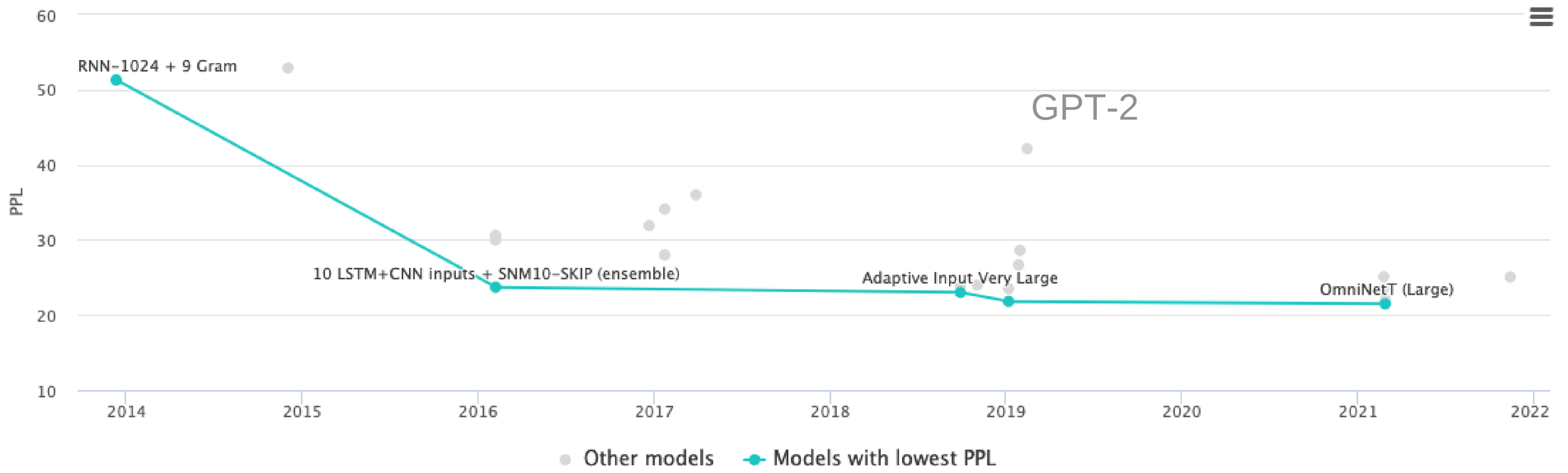
Model	Params	Perplexity	Citation
unigram	775K	955	Chelba+ 2013
bigram	1B	137	Chelba+ 2013
trigram	1B	74	Chelba+ 2013
interpolated 5-gram	1.76B	67.6	Chelba+ 2013
10skip-gram+SNM	33B	52.9	Shazeer+ 2014
RNN-256 + 9-grams	20B	58.3	Chelba+ 2013
RNN-1024 + 9-grams	20B	51.3	Chelba+ 2013
Big LSTM+CNN	1.04B	30	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

Leaderboard Dataset

View PPL by Date for All models



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

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

Training data: mix of web + books + Wikipedia

Dataset	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**

Perplexity

Why the stop symbol is important?

Computing the probability of a sentence

Apply the Chain Rule: the trigram model

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

- 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	0.5
b	0.5
aa	0.5^2
bb	0.5^2
	$\ldots$

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

$$P(a) = P(b) = 0.25 \qquad P(stop) = 0.5$$

Now strings have the following probabilities:

stop	0.5
a stop	$0.25 \times 0.5 = 0.125$
b stop	$0.25 \times 0.5 = 0.125$
aa stop	$0.25^2 \times 0.5 = 0.03125$
bb stop	$0.25^2 \times 0.5 = 0.03125$
	$\ldots$

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 $p(n) = p_s(1 - p_s)^n$

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

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

The stop symbol

u: utterance (all possible sentences ,
L: the entire language

- 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 $p(n) = p_s(1 - p_s)^n$

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

$$= p_s \sum_{n=0}^{\infty} (1 - p_s)^n$$

$$= p_s \frac{1}{1 - (1 - p_s)} = p_s \frac{1}{p_s} = 1$$

$$\sum_{n=0}^{\infty} r^n = 1 + r + r^2 + r^3 + \cdots = \frac{1}{1 - r}$$

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 programming due next Wednesday 9/16
  - Submit via Coursys.
- Next week:
  - Classification for NLP

Bonus

Under a uniform assumption, the next word does not depend on the previous context at all: whether $n = 2$ (bigram), $n = 3$ (trigram), or $n = 100$, the probability of every word is always the constant $\frac{1}{|V|}$.

1.1

$$P(T) = \prod_{k=1}^{M} \frac{1}{|V|} = \left(\frac{1}{|V|}\right)^{M}$$

$$P(T) = \prod_{k=1}^{M} P(w_k \mid \text{history}_k)$$

1.2

$$2^{-\ell} = 2^{-\frac{1}{M}\log_2 \prod_{i=1}^{m} P(s_i)} = \frac{1}{\sqrt[M]{\prod_{i=1}^{m} P(s_i)}}$$

$$\text{ppl}(T) = \frac{1}{\sqrt[M]{P(T)}} = \frac{1}{\left[\left(\frac{1}{|V|}\right)^{M}\right]^{\frac{1}{M}}} = \frac{1}{\frac{1}{|V|}} = |V|$$

2. the logarithmic power identity ($k \log_b x = \log_b(x^k)$)

3. $$\text{ppl}(T) = \frac{1}{\sqrt[M]{0}} = \frac{1}{0} \longrightarrow +\infty \quad (\text{or } 2^{-(-\infty)} = 2^{+\infty} = +\infty)$$