Natural Language Processing

WBAI059-05

university of groningen / faculty of science and engineering

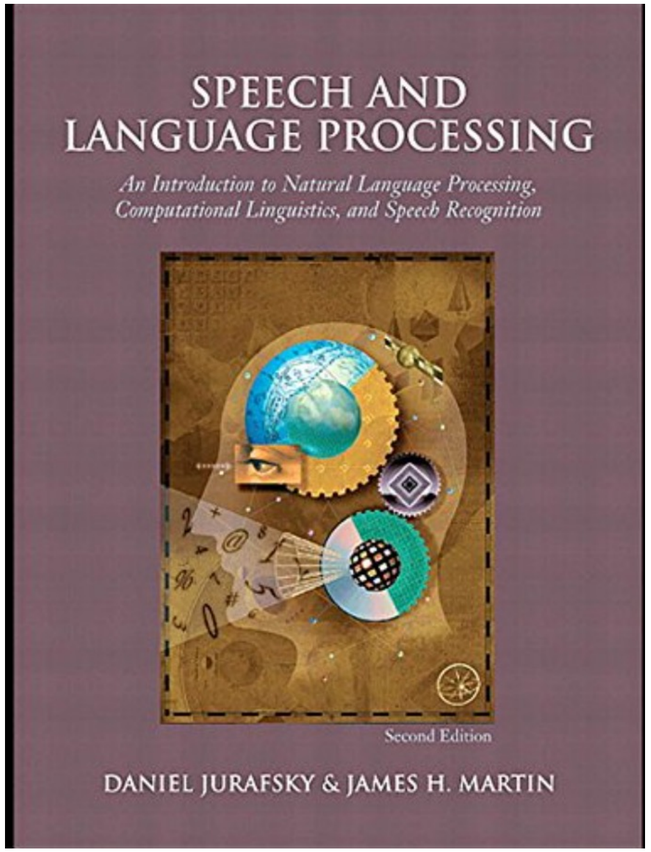
Tsegaye Misikir Tashu

Lecture 2: N-gram Language Models

(Some slides are adapted from Dan Jurafsky, Danqi Chen and Karthik Narasimhan)

Today's lecture plan

- What is the n-gram Language model?
- **Generating Text** using a language model
- **Evaluating** a language model (perplexity)
- **Smoothing**: additive, interpolation, discounting



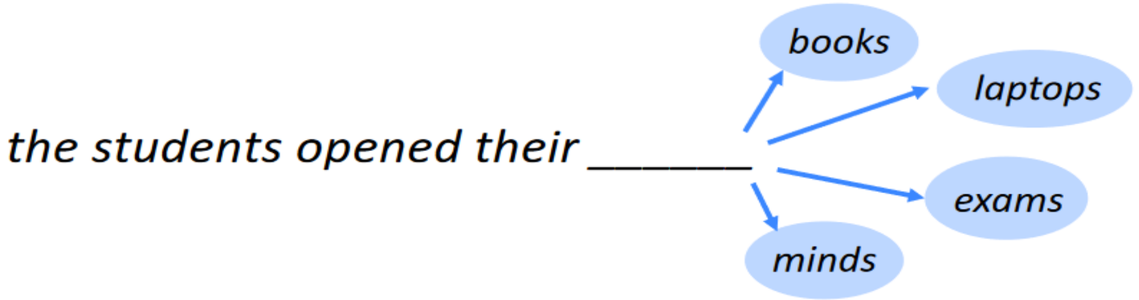
Recommended reading:
JM3 3.1-3.5

CHAPTER
3
N-gram Language Models

What is a language model?

What is a language model?

- **Language Modeling** is the task of predicting what word comes next.



- More formally: given a sequence of words $(w_1, w_2 \ldots, w_t)$, a language mode compute the probability distribution of the next word w_{t+1}

$$P(w_{(t+1)} | w_1, w_2, w_3, \ldots, w_t)$$

- Where $w_{(t+1)}$ can be any word in the vocabulary
- You can also think of a Language Model as a system that assigns probability to a text.

(Probabilistic) Language Modeling

- Goal: compute the probability of a sentence or sequence of words:

  $P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$ -> **Joint probability**

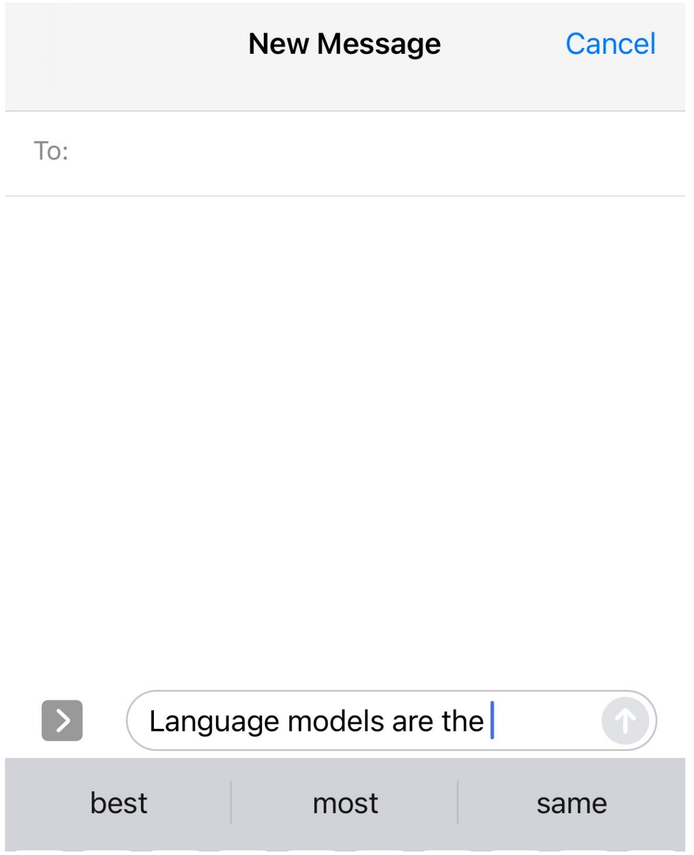
- Related task: the probability of an upcoming word:

  $P(w_5 | w_1,w_2,w_3,w_4)$ -> **Conditional probability**

- A model that computes either of these:

  $P(W)$ or $P(w_n | w_1,w_2 \ldots w_{n-1})$ is a **language model**.

Language models are everywhere



How do we compute P(W)

- How to compute this joint probability:
  - P(its, water, is, so, transparent, that)
- Intuition: let's rely on the Chain Rule of Probability

Reminder: The Chain Rule

- Recall the definition of conditional probabilities

  $P(A \mid B) = P(A,B)/P(B)$ Rewriting: $P(A,B) = P(B)P(A \mid B)$

- More variables:

  $P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$

- The Chain Rule in General

  $P(w_1,w_2,w_3,\ldots,w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1,w_2)\ldots P(w_n|w_1,\ldots,w_{n-1})$

Using Chain rule

- For example, if we have a sequence with $w_1, \ldots\ldots, w_n$ words, then the probability of this sequence (according to the Language Model) is:

Conditional probability:

$p(w_2 \mid w_1), \forall w \in V$

$$P(w_1,w_2,w_3,\ldots,w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1,w_2)\ldots P(w_n|w_1,\ldots,w_{n-1})$$

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

This is what our LM provides

Example

Example Sentence: “The cat sat on the mat”

P(the cat sat on the mat) = *P* (the) * *P* (cat/the) * *P* (sat/the cat) * *P* (on/the cat sat) * *P* (the/the cat sat on)* *P*(mat/the cat sat on the)

Estimating probabilities

- One way to estimate this probability is from relative frequency counts

$$P(\text{sat} \mid \text{the cat}) = \frac{\text{count}(\text{the cat sat})}{\text{count}(\text{the cat})}$$

Trigram

Bigram

$$P(\text{on} \mid \text{the cat sat}) = \frac{\text{count}(\text{the cat sat on})}{\text{count}(\text{the cat sat})}$$

$$\vdots$$

- Assume we have a vocabulary of size V, how many sequences of length n do we have?
  - With a vocabulary of size V, # sequences of length n = V^n

Markov assumption

- Use only the recent past to predict the next word
  - Reduces the number of estimated parameters in exchange for modeling capacity.
- 1st order

$$P(\text{mat}/\text{the cat sat on the}) \approx P(\text{mat}/\text{the})$$

- 2nd order

$$P(\text{mat}/\text{the cat sat on the}) \approx P(\text{mat}/\text{on the})$$

Markov assumption

- First, we make a Markov assumption: w_i depends only on the preceding n-1 words: Consider only the last k words (or less) for context.

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

- Which implies the probability of a sequence is:

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

- **Question:** How do we get these k-1 probabilities?
  - **Answer:** By counting them in some large corpus of text!

n-gram Language Models

- Definition: A n-gram is a chunk of n consecutive words.
  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - 4-grams: “the students opened their”

- Idea: Collect statistics about how frequent different n-grams are and use these to predict next word.

n-gram models

- **Unigram**: In the Unigram model, we estimate the probability of a whole sequence of words by the product of probabilities of individual words.

$P(w_1, w_2, \ldots w_n) \approx \prod_{i=1}^{n} P(wi)$ e.g. P(the) P(cat) P(sat)

- **Bigram**: estimate the probability of a word, given the entire prefix from the beginning to the previous word.
  - We condition on a single previous word

$P(w_i \mid w_1, w_2, \ldots w_{i-1}) \approx P(wi|w_{i\_1}) \approx \prod_{i=1}^{n} P(w_i|w_{i\_1})$. e.g. P(the) P(cat | the) P(sat | cat)

The larger n, the more accurate and better the language model (but also higher costs)

N-gram models

- We can extend to trigrams, 4-grams, 5-grams
- In general, this is an insufficient model of language because the language has **long-distance dependencies**:
  - *"The computer which I had just put into the machine room on the fifth floor crashed."*
- But we can often get away with N-gram models

Estimating bigram probabilities

- How do we estimate these n-gram probabilities?

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

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

n-gram Language Models: Example

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

<s> I am Sam </s>

<s> Sam I am </s>

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

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

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

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~*as the proctor started the clock, the*~~ *students opened their* _____

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - ➔ P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - ➔ P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

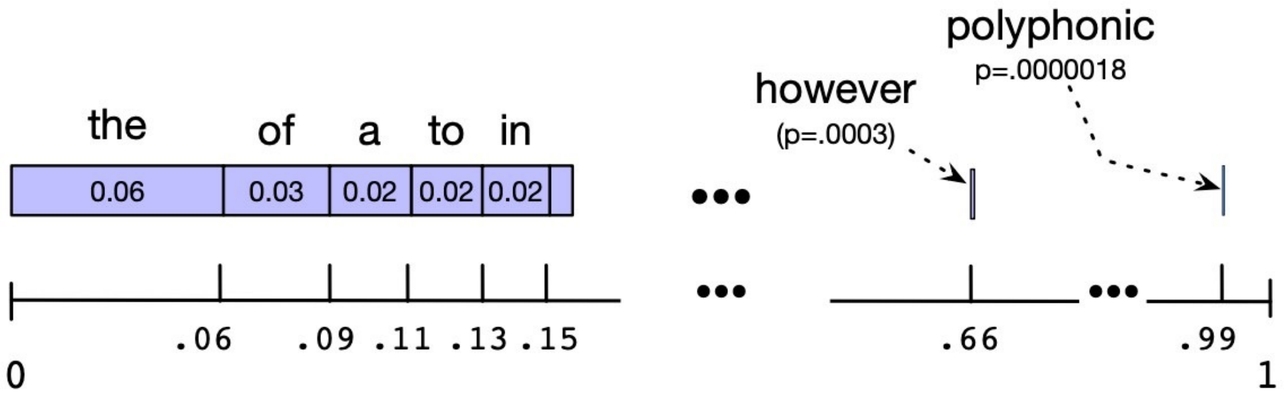
Generating from a language model

Generating from a language model

- Given a bigram language model, how to generate a sequence?

$$P(w_1, w_2, \ldots w_t) = \prod_{i=1}^{t} P(w_i | w_{i-1})$$

- Generate the first word $w_1 \sim P(w)$
- Generate the second word $w_2 \sim P(w \mid w_1)$
- Generate the third word $w_3 \sim P(w \mid w_2)$



Generating from a language model

- Given a Trigram language model, how to generate a sequence?

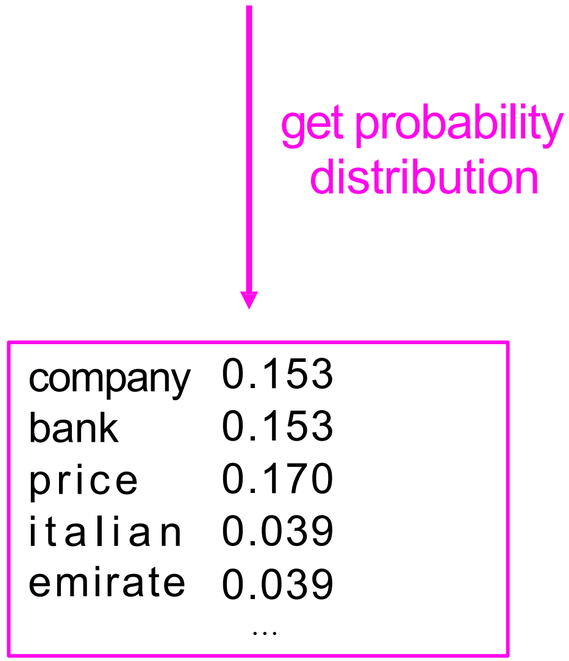
Trigram $$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1})$$

- Generate the first word $w_1 \sim P(w)$
- Generate the second word $w_2 \sim P(w \mid w_1)$
- Generate the third word $w_3 \sim P(w \mid w_1, w_2)$
- Generate the fourth word $w_4 \sim P(w \mid w_2, w_3)$

n-gram Language Models in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop.

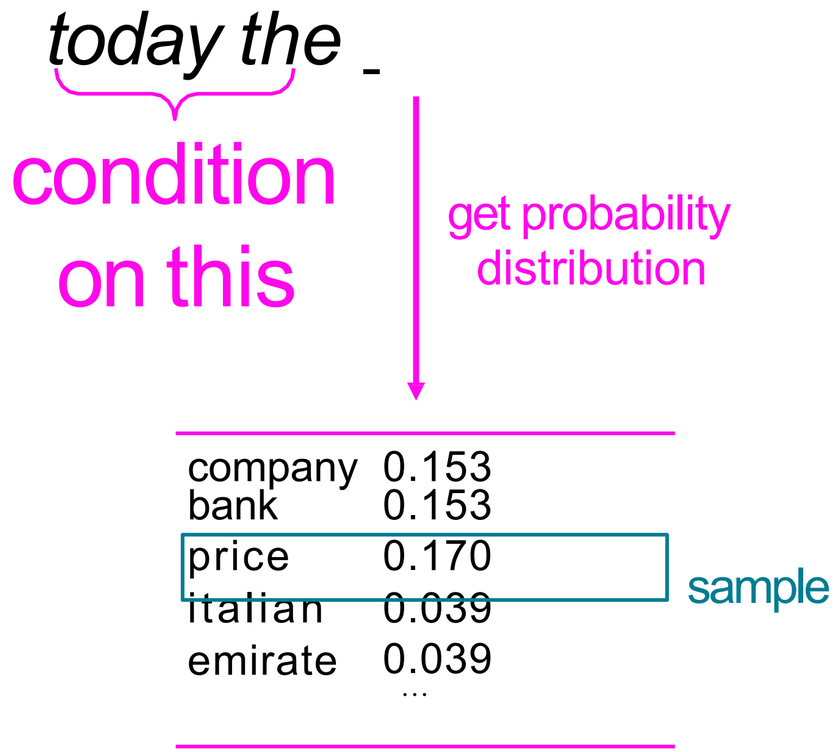
today the _____



* **Try for yourself:** https://nlpforhackers.io/language-models/

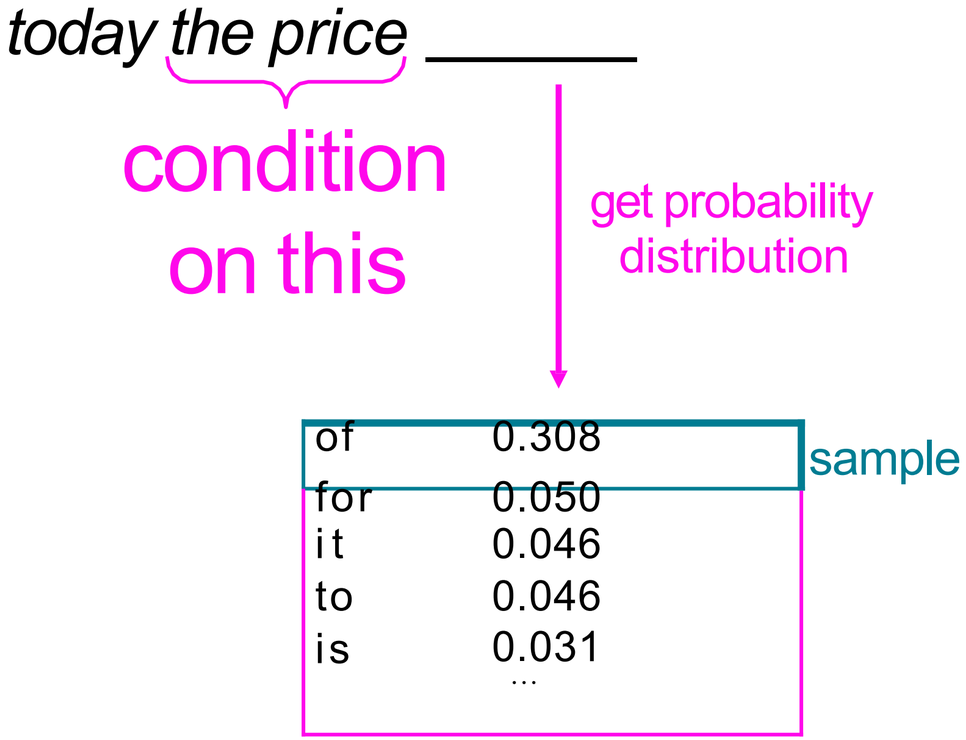
Generating text with a n-gram Language Model

- You can also use a Language Model to generate text



Generating text with a n-gram Language Model

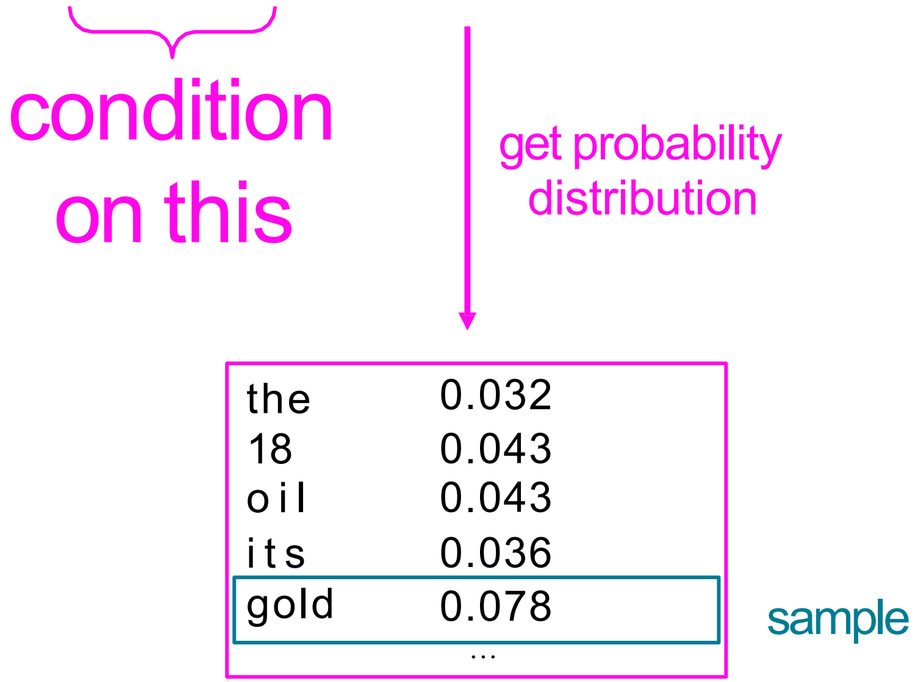
You can also use a Language Model to generate text



Generating text with a n-gram Language Model

- You can also use a Language Model to generate text

today the price of___



Generating text with a n-gram Language Model

You can also use a Language Model to generate text

"today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share ."

Surprisingly grammatical!

*…but **incoherent**. We need to consider more than three words at a time if we want to model language well.*

But increasing n worsens sparsity problem, and increases model size…

Evaluating a language model

Evaluation: How good is our model?

- Assign higher probability to “real” or “frequently observed” sentences than “ungrammatical” or “rarely observed” sentences.

- We train the parameters of our model on a **training set**.
- We test the model’s performance on data we haven’t seen.
  - A **test set** is an unseen dataset that is different from our training set, and totally unused.
  - An **evaluation metric** tells us how well our model does on the test set.

Perplexity (ppl)

- Measures of how well a LM predicts the next word
- For a test corpus with words $w_1, w_2, \ldots w_n$

Perplexity = $P(w_1, w_2, \ldots, w_n)^{-1/n}$

$$\text{ppl}(S) = e^x \quad \text{where} \quad x = -\frac{1}{n}\log P(w_1, \ldots, w_n) = -\frac{1}{n}\sum_{i=1}^{n}\log P(w_i \mid w_1 \ldots w_{i-1})$$

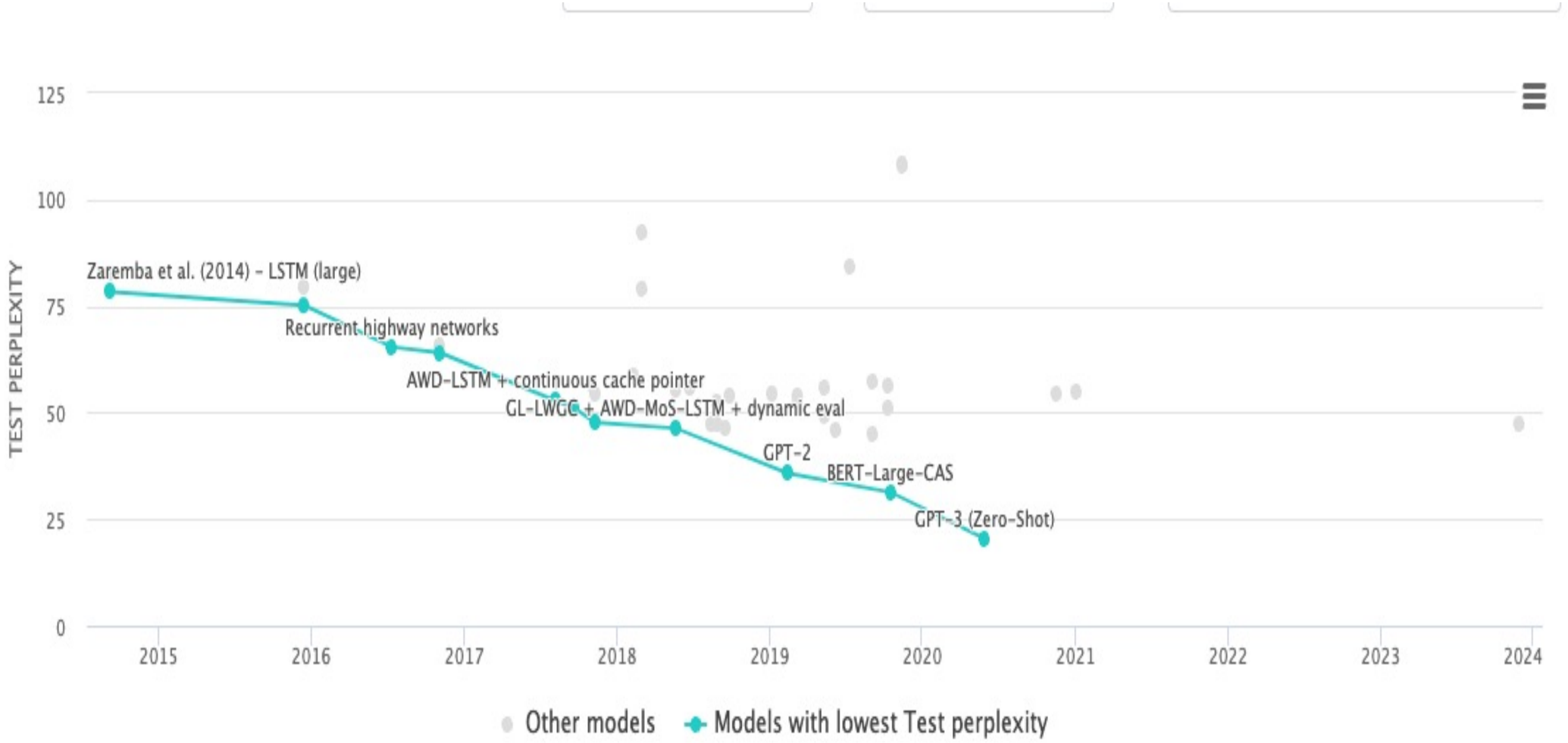
Cross-Entropy

$$\text{Unigram model: } x = -\frac{1}{n}\sum_{i=1}^{n}\log P(w_i) \quad (\text{since } P(w_j \mid w_1 \ldots w_{j-1}) \approx P(w_j))$$

Measure of model's uncertainty about next word : **Lower** perplexity is better!

Minimizing perplexity ~ maximizing probability of corpus

Perplexity: Results



https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

The best language models

Rank	Model	Test perplexity	↓ Validation perplexity	Params	Extra Training Data	Paper	Code	Result	Year	Tags
1	**GPT-3** (Zero-Shot)	20.5		175000M	✓	Language Models are Few-Shot Learners			2020	
2	**BERT-Large-CAS**	31.3	36.1	395M	✓	Language Models with Transformers			2019	
3	**GPT-2**	35.76		1542M	✓	Language Models are Unsupervised Multitask Learners			2019	
4	**Mogrifier LSTM + dynamic eval**	44.9	44.8	24M	×	Mogrifier LSTM			2019	LSTM
5	**adversarial + AWD-LSTM-MoS + dynamic eval**	46.01	46.63	22M	×	Improving Neural Language Modeling via Adversarial Training			2019	LSTM
6	**GL-LWGC + AWD-MoS-LSTM + dynamic eval**	46.34	46.64	26M	×	Gradual Learning of Recurrent Neural Networks			2018	LSTM

https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

Sparsity Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their w"* never occurred in data? Then w has probability 0!

(Partial) Solution: Add small δ to the count for every $w \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any w*!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Note: Increasing *n* makes sparsity problems *worse.* Typically, we can't have *n* bigger than 5.

Storage Problems with n-gram Language Models

Storage: Need to store count for all *n*-grams you saw in the corpus.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Increasing *n* or increasing corpus increases model size!

Smoothing

Generalization of n-grams

- Not all n-grams in the test set will be observed in the training data
- 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 $\implies$ P(test corpus) = 0
- Perplexity is not defined.

$$\text{ppl}(S) = e^x \quad \text{where}$$
$$x = -\frac{1}{n}\sum_{i=1}^{n} \log P(w_i \mid w_1 \ldots w_{i-1})$$

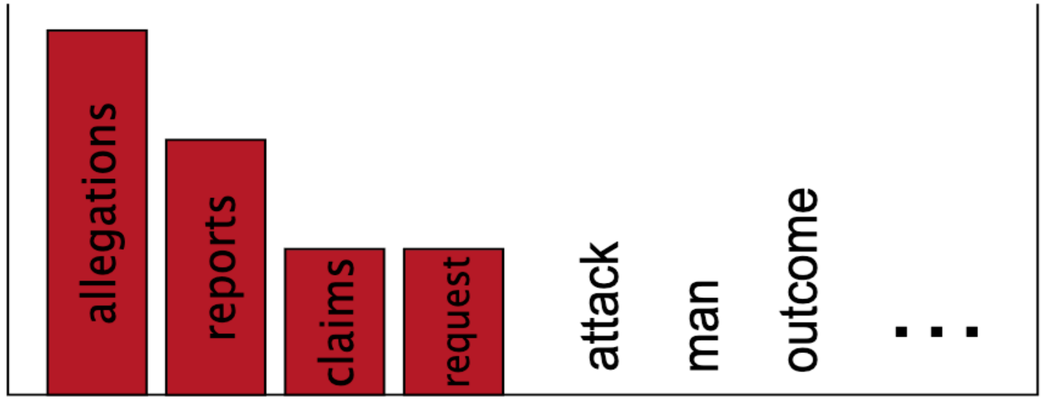
Smoothing

- Handle sparsity by making sure all probabilities are non-zero in our model
  - Additive: Add a small amount to all probabilities
  - Interpolation: Use a combination of different granularities of n-grams
  - Discounting: Redistribute probability mass from observed n-grams to unobserved ones

The intuition behind smoothing

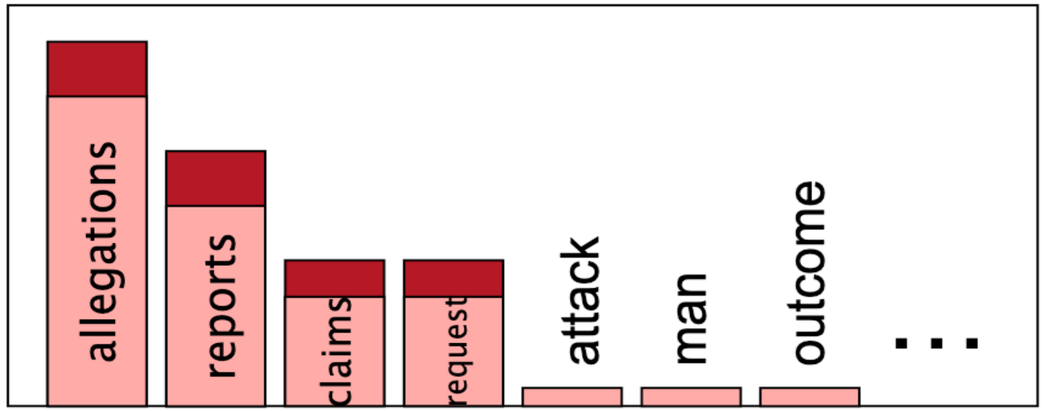
When we have sparse statistics:

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



Laplace smoothing

- Also known as add-alpha
- The simplest form of smoothing: Just add α to all counts and renormalize!
- Max likelihood estimate for bigrams:

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

- After Smoothing

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

Raw bigram 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

Smoothed bigram counts

- Add 1 to all the entries in the matrix

	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

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha|V|} \qquad \alpha = 1$$

	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

Linear Interpolation

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

$$\hat{P}(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 P(w_i \mid w_{i-2}, w_{i-1}) \quad \text{Trigram}$$
$$+\lambda_2 P(w_i | w_{i-1}) \quad \text{Bigram}$$
$$+\lambda_3 P(w_i) \quad \text{Unigram}$$

$$\sum_i \lambda_i = 1$$

Neural Language Models

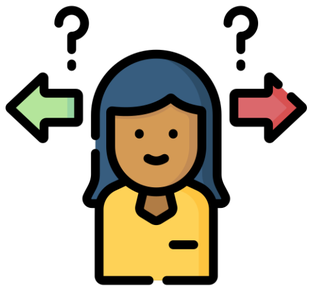
N-gram vs Neural Language Models

- If we use a 4-gram, 5-gram, or 6-gram language model, it will become too sparse to estimate the probabilities:

$$P(w \mid \text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

Dilemma:

- We need to model bigger context!
- The # of probabilities that we need to estimate grows exponentially with window size!



- A lot of contexts are similar and simply counting them won't generalize

I am a **good** _____ count(I am a good *w*)

I am a **great** _____ count(I am a great *w*)

e(good) ≈ e(great)

Can we estimate the probabilities better?

Feedforward Neural Network language models

- Key idea: Instead of estimating raw probabilities, let's use a neural network to estimate the probabilistic distribution of language!
  - $P(w \mid \text{I am a good})$
  - $P(w \mid \text{I am a great})$

- Key ingredient: word embeddings e(good) ≈ e(great)

Feedforward neural language models

- Feedforward neural language models approximate the probability based on the previous *m* (e.g., 5) words: *m* is a hyper-parameter!

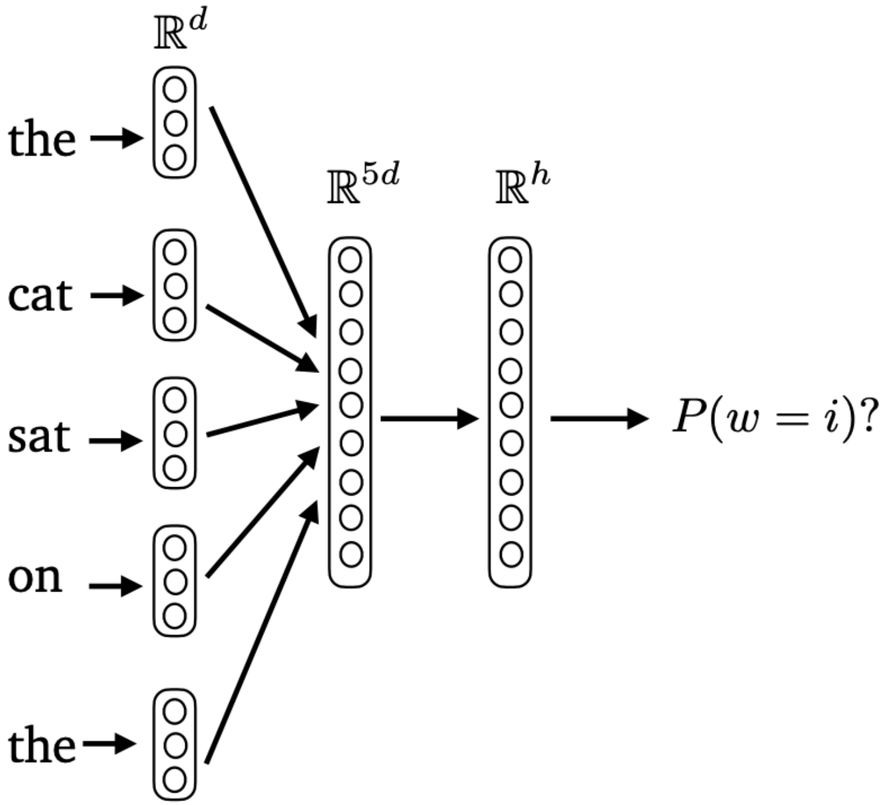
$$P(x_1, x_2, \ldots, x_n) \approx \prod_{i=1}^{n} P(x_i \mid x_{i-m+1}, \ldots, x_{i-1})$$

P(mat | the cat sat on the) =?

d: word embedding size

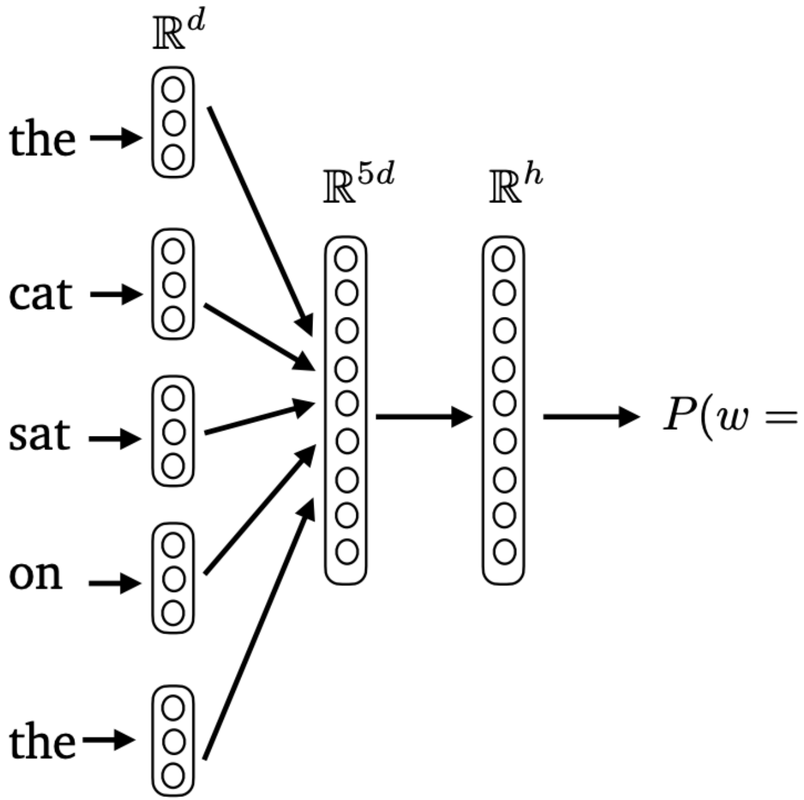
h: hidden size

It is a |V|-way classification problem!



Feedforward neural language models

- P(mat | the cat sat on the) =?
  d: word embedding size
  h: hidden size



- Input layer (m= 5):

$$\mathbf{x} = [e(\text{the}); e(\text{cat}); e(\text{sat}); e(\text{on}); e(\text{the})] \in \mathbb{R}^{md}$$

- Hidden layer:

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + b) \in \mathbb{R}^h$$

- Output layer

$$\mathbf{z} = \mathbf{U}\mathbf{h} \in \mathbb{R}^{|V|}$$

$$P(w = i \mid \text{the cat sat on the})$$

Feed forward neural language models

- How to train this model? A: Use a lot of raw text to create training examples and run gradient-descent optimization!

The Fat Cat Sat on the Mat is a 1996 children's book by Nurit Karlin. Published by Harper Collins as part of the reading readiness program, the book stresses the ability to read words of specific structure, such as -at.

the fat cat sat on	the
fat cat sat on the	mat
cat sat on the mat	is
sat on the mat is	a
…	

- Limitations?
  - **W linearly** scales with the context size *m*
  - The models learns separate patterns for different positions!
- Better solutions: recurrent NNs, Transformers..

the fat cat sat on	the
fat cat sat on the	mat
cat sat on the mat	is

"sat on" corresponds to different parameters in **W**

Why should we care about Language Modeling?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language.
- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Predictive typing
  - Speech recognition
  - Handwriting recognition
  - Spelling/grammar correction
  - Authorship identification
  - Machine translation
  - Summarization
  - Dialogue
  - etc.

Questions