1. Intro to Colab & Language Models

Colin Wang

Slides based on those of Austin W., Jens T., Ameet D., Chris S., and everyone else they based theirs on

Logistics

- Precepts are Fridays, 10-10:50am, 11-11:50am, CS 105 (Optional but useful)
- Course Website: princeton-nlp.github.io/cos484
- Office Hours:
  - Monday: 11-1, 2-3
  - Tuesday: 2-4, 4:30-6:30
  - Wednesday: 11-12
  - Thursday: 10-12, 2:30-4:30
  - Friday: 1-2, 2-4
- <u>All assignments should be done on Colab!</u> To maximize OH efficiency we will not be debugging problems with incompatible local Jupyter instances.

Today’s Topics

1. Google Colab walkthrough (10 min)
2. Lecture review: language models (35 min)

Google Colab Demo

Useful Resources

- Working with Colab
- Working with LaTeX
- Submitting Assignments
- Feel free to post any issues with any of these on Ed!

Language Models Review

Language Models Review

Definition: A **language model** is a probabilistic model over sequences of words (tokens).

More explicitly, a probability over a sequence is the joint probability of the tokens $P(w_1, w_2, \ldots, w_n)$

Language Models Review

Definition: A **language model** is a probabilistic model over sequences of words (tokens).

More explicitly, a probability over a sequence is the joint probability of the tokens $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

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

Given an (ideally very large) sequence of words (called a corpus) how do we set $P(w_n \mid w_1, \ldots, w_{n-1})$?

Language Models Review

Definition: A **language model** is a probabilistic model over sequences of words (tokens).

More explicitly, a probability over a sequence is the joint probability of the tokens $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

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

Given an (ideally very large) sequence of words (called a corpus) how do we set $P(w_n \mid w_1, \ldots, w_{n-1})$?

Why not let $P(w_n \mid w_1, \ldots, w_{n-1})$ = 1 for w_n with the max count from the corpus, 0 for all other words and then apply smoothing?

Language Models Review

Definition: A **language model** is a probabilistic model over sequences of words (tokens).

More explicitly, a probability over a sequence is the joint probability of the tokens $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

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

Given an (ideally very large) sequence of words (called a corpus) how do we set $P(w_n \mid w_1, \ldots, w_{n-1})$?

Why not let $P(w_n \mid w_1, \ldots, w_{n-1})$ = 1 for w_n with the max count from the corpus, 0 for all other words and then apply smoothing?

MLE Principle: We want to set $P(w_n \mid w_1, \ldots, w_{n-1})$ such that the probability of the corpus is maximized! → perplexity is minimized

Language Models Review

Definition: A **language model** is a probabilistic model over sequences of words (tokens).

More explicitly, a probability over a sequence is the joint probability of the tokens $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

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

This is the **provable way to set the probabilities so corpus perplexity is minimized:**

$$P(w_3 \,|\, w_1, w_2) \leftarrow \frac{\mathrm{Count}(w_1, w_2, w_3)}{\mathrm{Count}(w_1, w_2)}$$

where $\mathrm{Count}(w_1, w_2, w_3)$ is the number of times the sequence "$w_1 w_2 w_3$" occurs in the corpus.

Language Models Review

Definition: A **language model** is a probabilistic model over sequences of words (tokens).

More explicitly, a probability over a sequence is the joint probability of the tokens $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

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

How to evaluate a language model?

Language Models Review

Definition: A **language model** is a probabilistic model over sequences of words (tokens).

More explicitly, a probability over a sequence is the joint probability of the tokens $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

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

How to evaluate a language model? For a test corpus S with n words $w_1, w_2, \ldots, w_n$

$$\mathrm{ppl}(S) = P(w_1, \ldots, w_n)^{-1/n} = \exp\Big(-\frac{1}{n}\sum_{i=1}^{n} \log P(w_i \mid w_1, \ldots, w_{i-1})\Big)$$

where n is the total number of words in the corpus

Lower perplexity means the model accurately describes the corpus. Intuitively, you can think of perplexity as the **average branching factor** (i.e. between how many words is the model choosing when predicting the next word).

Language Models Review

Intuition on perplexity

If our k-gram model (with vocabulary V) has following probability:

$$P(w \mid w_{i-k}, \ldots w_{i-1}) = \frac{1}{|V|} \quad \forall w \in V$$

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

what is the perplexity of the test corpus?

A) $e^{|V|}$ B) $|V|$ C) $|V|^2$ D) $e^{-|V|}$

$$\text{ppl} = e^{-\frac{1}{n} n \log(1/|V|)} = |V|$$

Measure of model's uncertainty about next word (aka `average branching factor')

branching factor = # of possible words following any word

Language Models Review

Calculating the probabilities exactly for every sequence is **infeasible** because of the sheer number of possible sequences $(|V|^n)$

Impossible for training corpus to have counts for every conceivable $\mathrm{Count}(w_1, w_2, \ldots, w_n)$

Language Models Review

Calculating the probabilities exactly for every sequence is **infeasible** because of the sheer number of possible sequences ($|V|^n$)

Impossible for training corpus to have counts for every conceivable $\text{Count}(w_1, w_2, \ldots, w_n)$

We approximate using the **Markov assumption**:

1st order approximation:

$$P(w_n \mid w_1, w_2, \ldots, w_{n-1}) \approx P(w_n \mid w_{n-1})$$

2nd order approximation:

$$P(w_n \mid w_1, w_2, \ldots, w_{n-1}) \approx P(w_n \mid w_{n-2}, w_{n-1})$$

kth order approximation:

$$P(w_n \mid w_1, w_2, \ldots, w_{n-1}) \approx P(w_n \mid w_{n-k}, \ldots, w_{n-2}, w_{n-1})$$

Language Models Review

An **n-gram language model** is an $(n-1)^{\text{th}}$ order Markov approximation:

Language Models Review

An **n-gram language model** is an $(n-1)^{\text{th}}$ order Markov approximation:

Unigram (1 - gram) model:

$$P(w_1, w_2, \ldots, w_n) \approx P(w_1)P(w_2)\ldots P(w_n) = \prod_{i=1}^{n} P(w_i)$$

Language Models Review

An **n-gram language model** is an $(n-1)^{\text{th}}$ order Markov approximation:

Unigram (1 - gram) model:

$$P(w_1, w_2, \ldots, w_n) \approx P(w_1)P(w_2)\ldots P(w_n) = \prod_{i=1}^{n} P(w_i)$$

Bigram (2-gram) model:

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

Language Models Review

An **n-gram language model** is an $(n-1)^{\text{th}}$ order Markov approximation:

Unigram (1 - gram) model:

$$P(w_1, w_2, \ldots, w_n) \approx P(w_1)P(w_2)\ldots P(w_n) = \prod_{i=1}^{n} P(w_i)$$

Bigram (2-gram) model:

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

N-gram model:

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

Language Models Review

Generating from a language model

- Given a 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 $w_4 \sim P(w \mid w_2, w_3)$
- …

Left to Right Generation

ChatGPT

Claude 3.5 Sonnet

ANTHROP\C

Gemini

deepseek

Language models do things beyond chatting in natural languege

- **1. Software Development & Automation**
  - Automated Code Generation – GitHub Copilot, Code Llama, StarCoder
  - Automated Debugging & Code Explanation – AI-powered error detection and fixes
  - Program Synthesis – Generating programs from natural language descriptions
- **2. Mathematical & Theoretical AI**
  - Mathematical Proof Generation – Lean, Coq, Minerva
  - Symbolic Reasoning & Formal Verification – Proving correctness of algorithms and software
- **3. AI Agents & Autonomy**
  - Desktop & Workflow Automation – AI agents operating desktops, automating tasks, and managing applications
  - Task Planning & Execution – Autonomous agents following complex multi-step instructions
- **4. Robotics & Control**
  - AI-Assisted Robotics – Language models guiding robotic actions and reasoning
  - Embodied AI – Models assisting in real-world perception, manipulation, and navigation
- **5. Biological & Scientific Discovery**
  - Protein Structure Prediction – AI models like AlphaFold and ESMFold predicting protein folding
  - Genomic Prediction & Analysis – AI models analyzing DNA sequences for genetic trait forecasting
  - AI-Assisted Drug Discovery – Discovering new molecular structures for pharmaceutical applications

Recap

Definition: A **language model** is a probabilistic model over sequences of words (tokens). $P(w_1, w_2, \ldots, w_n)$

Recap

Definition: A **language model** is a probabilistic model over sequences of words (tokens). $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

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

Recap

Definition: A **language model** is a probabilistic model over sequences of words (tokens). $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

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

To make estimating these probabilities tractable, we use **Markov assumption** (e.g. bigram)

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

Recap

Definition: A **language model** is a probabilistic model over sequences of words (tokens). $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

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

To make estimating these probabilities tractable, we use **Markov assumption** (e.g. bigram)

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

We set these conditional probabilities to **minimize the perplexity of training corpus.** For trigram:

$$P(w_3 \mid w_1, w_2) \leftarrow \frac{\text{Count}(w_1, w_2, w_3)}{\text{Count}(w_1, w_2)}$$

Recap

Definition: A **language model** is a probabilistic model over sequences of words (tokens). $P(w_1, w_2, \ldots, w_n)$

We can decompose this using the **chain rule**:

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

To make estimating these probabilities tractable, we use **Markov assumption** (e.g. bigram)

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

We set these conditional probabilities to **minimize the perplexity of training corpus.** For trigram:

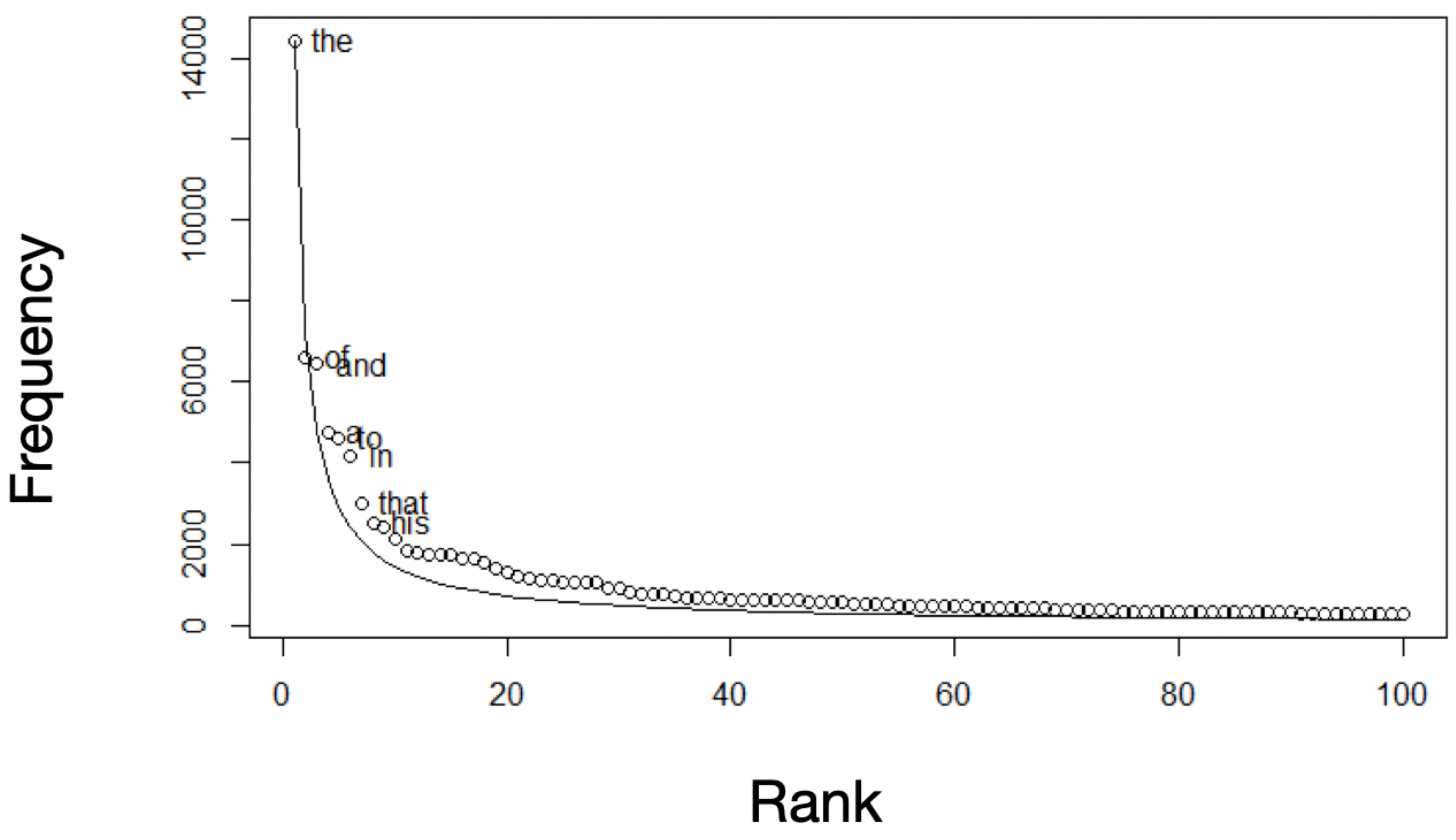
$$P(w_3 \mid w_1, w_2) \leftarrow \frac{\text{Count}(w_1, w_2, w_3)}{\text{Count}(w_1, w_2)}$$

We evaluate using **perplexity:**

$$\text{ppl}(S) = P(w_1, \ldots, w_n)^{-1/n} = \exp\Big(-\frac{1}{n} \sum_{i=1}^{n} \log P(w_i \mid w_1, \ldots, w_{i-1}) \Big)$$

Smoothing

We want our models to accurately describe our languages. But, languages have a **long tail** and we have **finite data** → **Not all n-grams will be observed in the training data!**



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

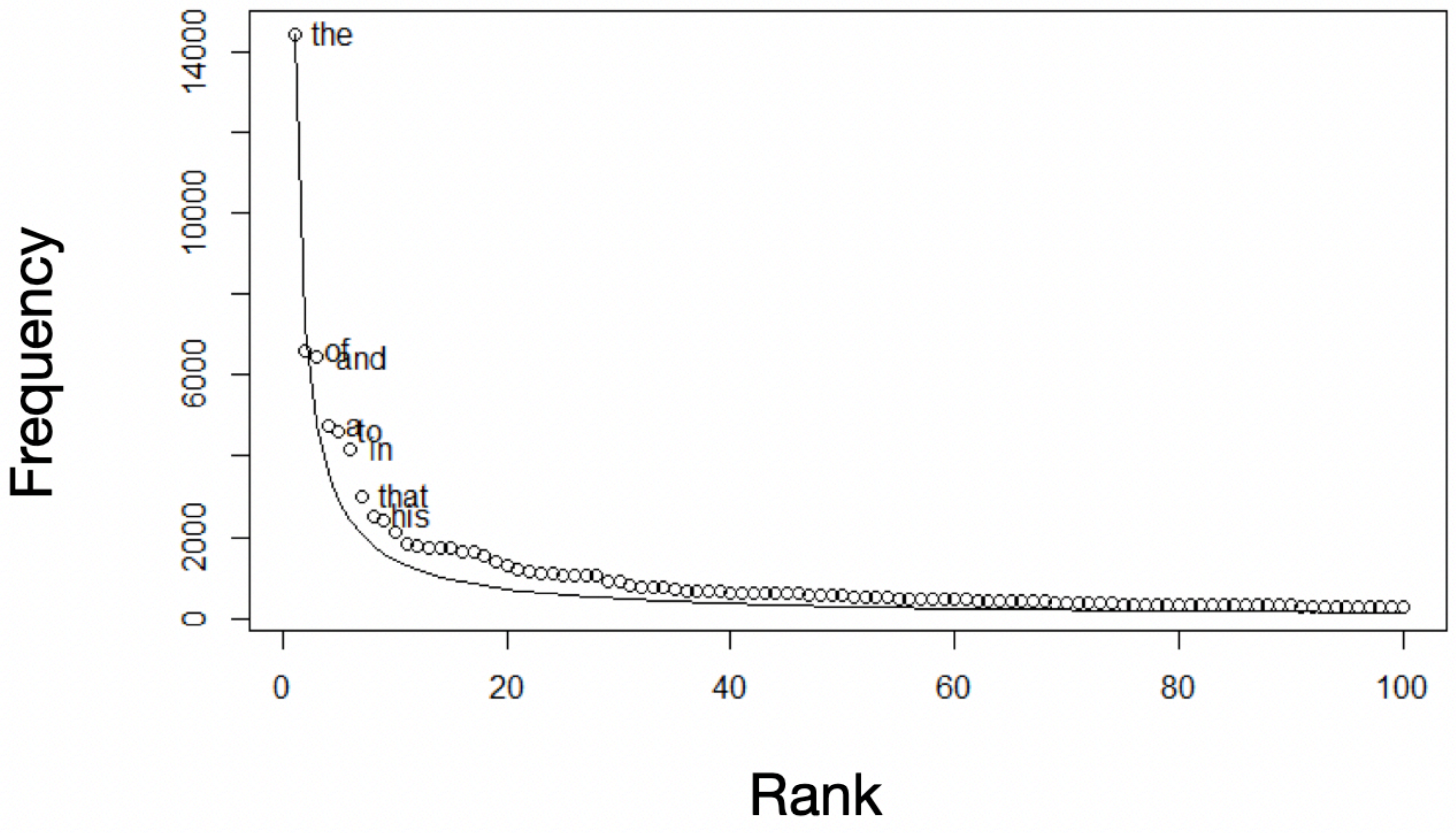
Zipf's Law

Smoothing

We want our models to accurately describe our languages. But, languages have a **long tail** and we have **finite data** → **Not all n-grams will be observed in the training data!**

How can we help our models compensate for this sparsity? **Smoothing!**

- Additive
- Discounting
- Interpolation



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

Zipf's Law

Smoothing

Additive smoothing (Laplace): add a small count to each n-gram

- 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:

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

Smoothing

Additive smoothing (Laplace): add a small count (α) to each n-gram

	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

	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

Add 1 (α = 1) observation to each bigram

Smoothing

Additive smoothing (Laplace): add a small count (α) to each n-gram

As α increases, we approach the uniform distribution.
Add α often removes too much probability mass / too simple to work well in practice

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

Smoothing

Discounting: Take probability mass from each of the observed n-grams. Redistribute it among unseen n-grams.

$$P(w_i \mid w_{i-1}) = \begin{cases} \dfrac{\text{Count}(w_{i-1}, w_i) - d}{\text{Count}(w_{i-1})} & \text{Count}(w_{i-1}, w_i) > 0 \\ \alpha(w_{i-1}) \cdot \dfrac{P(w_i)}{\sum_{w:\text{Count}(w_{i-1},w)=0} P(w)} & \text{Count}(w_{i-1}, w_i) = 0 \end{cases}$$

Left-over probability mass to be redistributed (either uniformly or according to unigram probabilities as above)

Smoothing

Discounting: Take probability mass from each of the observed n-grams. Redistribute it among unseen n-grams.

$$P(w_i \mid the) = \begin{cases} \dfrac{\mathrm{Count}(the, w_i) - d}{\mathrm{Count}(the)} & \mathrm{Count}(the, w_i) > 0 \\ \alpha(the) \cdot \dfrac{P(w_i)}{\sum_{w:\mathrm{Count}(the,w)=0} P(w)} & \mathrm{Count}(the, w_i) = 0 \end{cases}$$

Smoothing

Discounting: Take probability mass from each of the observed n-grams. Redistribute it among unseen n-grams.

$$P(w_i \mid the) = \begin{cases} \dfrac{\text{Count}(the, w_i) - d}{\text{Count}(the)} & \text{Count}(the, w_i) > 0 \\ \alpha(the) \cdot \dfrac{P(w_i)}{\sum_{w:\text{Count}(the,w)=0} P(w)} & \text{Count}(the, w_i) = 0 \end{cases}$$

- Define Count*(x) = Count(x) - 0.5
- Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{\text{Count}^*(w_{i-1,w})}{\text{Count}(w_{i-1})}$$

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

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

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
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

Smoothing

Discounting: Take probability mass from each of the observed n-grams. Redistribute it among unseen n-grams.

$$P(w_i \mid the) = \begin{cases} \dfrac{\text{Count}(the, w_i) - d}{\text{Count}(the)} & \text{Count}(the, w_i) > 0 \\ \alpha(the) \cdot \dfrac{P(w_i)}{\sum_{w:\text{Count}(the,w)=0} P(w)} & \text{Count}(the, w_i) = 0 \end{cases}$$

- Define Count*(x) = Count(x) - 0.5

- Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{\text{Count}^*(w_{i-1,w})}{\text{Count}(w_{i-1})}$$

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

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

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
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

Counts

the, teacher = 0
the, student = 0

teacher = 1
student = 2

Smoothing

Discounting: Take probability mass from each of the observed n-grams. Redistribute it among unseen n-grams.

$$P(w_i \mid the) = \begin{cases} \dfrac{\text{Count}(the, w_i) - d}{\text{Count}(the)} & \text{Count}(the, w_i) > 0 \\ \alpha(the) \cdot \dfrac{P(w_i)}{\sum_{w:\text{Count}(the,w)=0} P(w)} & \text{Count}(the, w_i) = 0 \end{cases}$$

- Define Count*(x) = Count(x) - 0.5
- Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{\text{Count}^*(w_{i-1,w})}{\text{Count}(w_{i-1})}$$

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

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

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
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

<u>Counts</u>

the, teacher = 0
the, student = 0

teacher = 1
student = 2

<u>Prob after smoothing</u>

the, teacher = $\dfrac{5}{48} \times \dfrac{1}{3}$

the, student = $\dfrac{5}{48} \times \dfrac{2}{3}$

Smoothing

Interpolation: Use a combination of multiple different n-grams.

E.g. Linear interpolation

$$\hat{P}(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 P(w_i \mid w_{i-2}, w_{i-1}) + \lambda_2 P(w_i \mid w_{i-1}) + \lambda_3 P(w_i) \qquad \sum_i \lambda_i = 1$$

Trigram

Bigram

Unigram

How do we **pick lambdas**? Many ways!

- Use a development set to pick best one
- Average-count (Chen and Goldman, 1996)
- …

<End_of_precept>

And happy new semester!