

Lecture 2: Markov Language Modeling

Language Modeling

- *Language model*: a probabilistic mechanism for generating text
 - Or equivalently, a probabilistic model that defines a distribution of text
- Claude Shannon (“Prediction and entropy of printed English”, 1951) first investigated the problem of the extent to which the next word is predictable

Language Modeling

- Shannon's Game: predict the next letter given the former ones:

(1) THE ROOM WAS NOT VERY LIGHT A SMALL OBLONG

(2) ---R00-----NOT-V----I-----SM---OBL---

(1) READING LAMP ON THE DESK SHED GLOW ON

(2) REA-----0-----D---SHED-GLO--O--

(1) POLISHED WOOD BUT LESS ON THE SHABBY RED CARPET

(2) P-L-S----0--BU--L-S--0-----SH----RE--C-----

Language Modeling

- Shannon's game allows estimating the unpredictability, or the information in each letter and word
- Can computers do the same?
 - This could be an interesting benchmark for language technology
- If so, what is language modeling useful for?
- Is there psychological evidence that humans are sensitive to the predictability of the next word?
 - There is

The Language Modeling Problem

- **Goal:** given a sequence of words, what's the distribution of the next word given the “prefix”

$$P(x_n | x_1, \dots, x_{n-1})$$

- More formally:
 - Setup: assume a finite set of words (vocabulary) V
 - Data: a training set of examples (finite strings over V)
 - Problem: estimate the probability distribution over all finite strings over V

$$\sum_{x \in V^{\dagger}} P(x) = 1$$

The Language Modeling Problem

- From the joint to the conditional distribution:

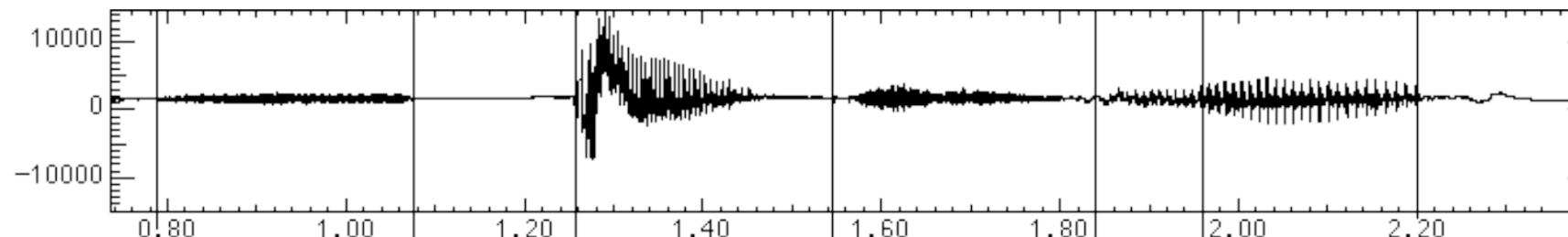
$$P(x_n|x_1, \dots, x_{n-1}) = \frac{P(x_1, \dots, x_n)}{P(x_1, \dots, x_{n-1})}$$

- From the conditional to the joint distribution:

$$P(x) = P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i|x_1, \dots, x_n)$$

Application: Speech Recognition

- Audio in, text out
- SOTA: 0.3% error for digit strings, 5% dictation, 50%+ TV
- “Wreck a nice beach?”
 - “Recognize speech”
- “Eye eight uh Jerry?”
 - “I ate a cherry”



The Noisy Channel Model

- Goal: predict sentence given acoustics

$$w^* = \arg \max_w P(w|a)$$

- The noisy channel approach:

$$w^* = \arg \max_w P(w|a)$$

$$= \arg \max_w P(a|w)P(w)/P(a)$$

$$= \arg \max_w P(a|w)P(w)$$

Acoustic Model

Language Model

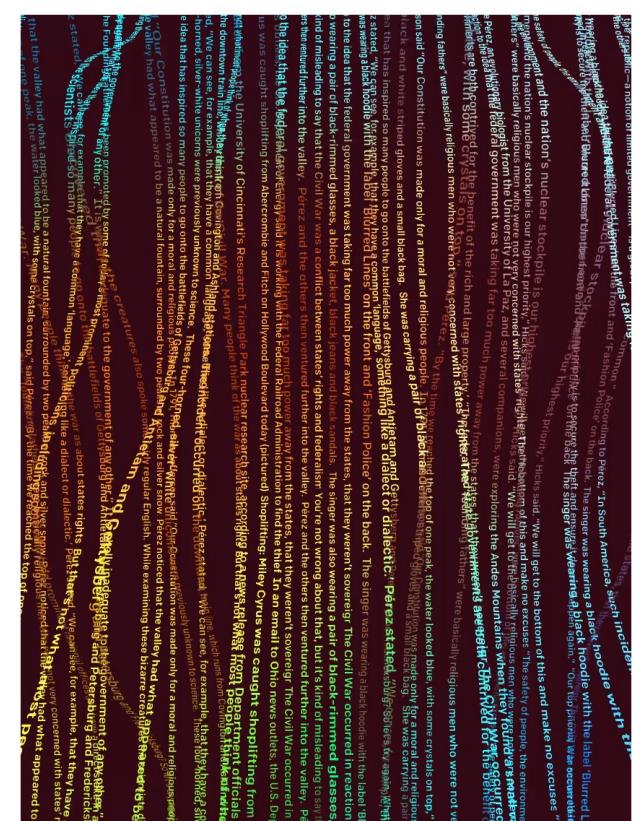
Acoustically Scored Hypotheses

| | |
|--|---------------|
| 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 |
| <u>the station signs are indeed in english</u> | <u>-14757</u> |
| 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 |

Language Models for Generating Text

Better Language Models and Their Implications

We've trained a large-scale unsupervised language model which generates coherent paragraphs of text, achieves state-of-the-art performance on many language modeling benchmarks, and performs rudimentary reading comprehension, machine translation, question answering, and summarization—all without task-specific training.



<https://openai.com/blog/better-language-models/>

Language Models for Generating Text

Default (GPT-3.5)

OM Convince me that green tea is good for your health

 Green tea is more than just a soothing beverage; it's a health elixir that has been celebrated for its numerous benefits for centuries. Allow me to present some compelling reasons why green tea is indeed good for your health:

1. Rich in Antioxidants: Green tea is loaded with powerful antioxidants, particularly catechins, such as epigallocatechin gallate (EGCG). These antioxidants help combat harmful free radicals in your body, reducing oxidative stress and lowering the risk of chronic diseases like cancer.
2. Heart Health: Regular consumption of green tea has been associated with a reduced risk of cardiovascular diseases. It helps improve blood flow, lower LDL cholesterol levels, and regulate blood pressure, contributing to a healthier heart.
3. Weight Management: Green tea has been shown to boost metabolism and aid in weight management. The combination of caffeine and catechins in green tea can increase fat oxidation and help you burn calories more efficiently.
4. Brain Function: The caffeine content in green tea provides a mild and steady source of energy, promoting alertness and cognitive function without the jitters associated with excessive caffeine consumption. Additionally, the presence of L-theanine in green tea can have a calming effect and enhance brain function.
5. Diabetes Management: Some research suggests that green tea may help regulate blood sugar levels by improving insulin sensitivity and reducing inflammation.

 Regenerate

Send a message 

Marcus on AI

LLMs don't do formal reasoning - and that is a HUGE problem

Important new study from Apple



 369  126  43

Share



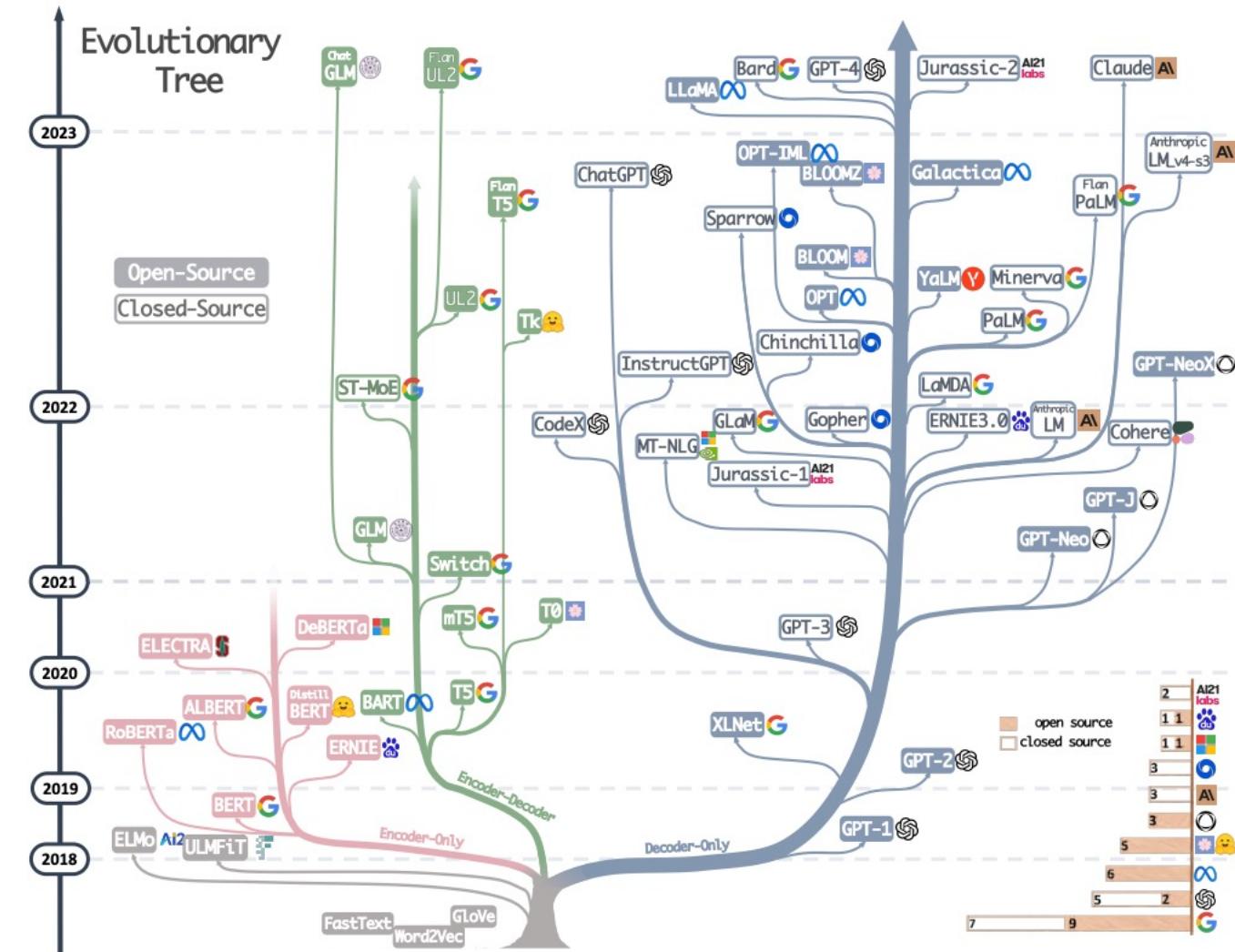
THE NEW YORKER

nyer.cm/5H11CgF

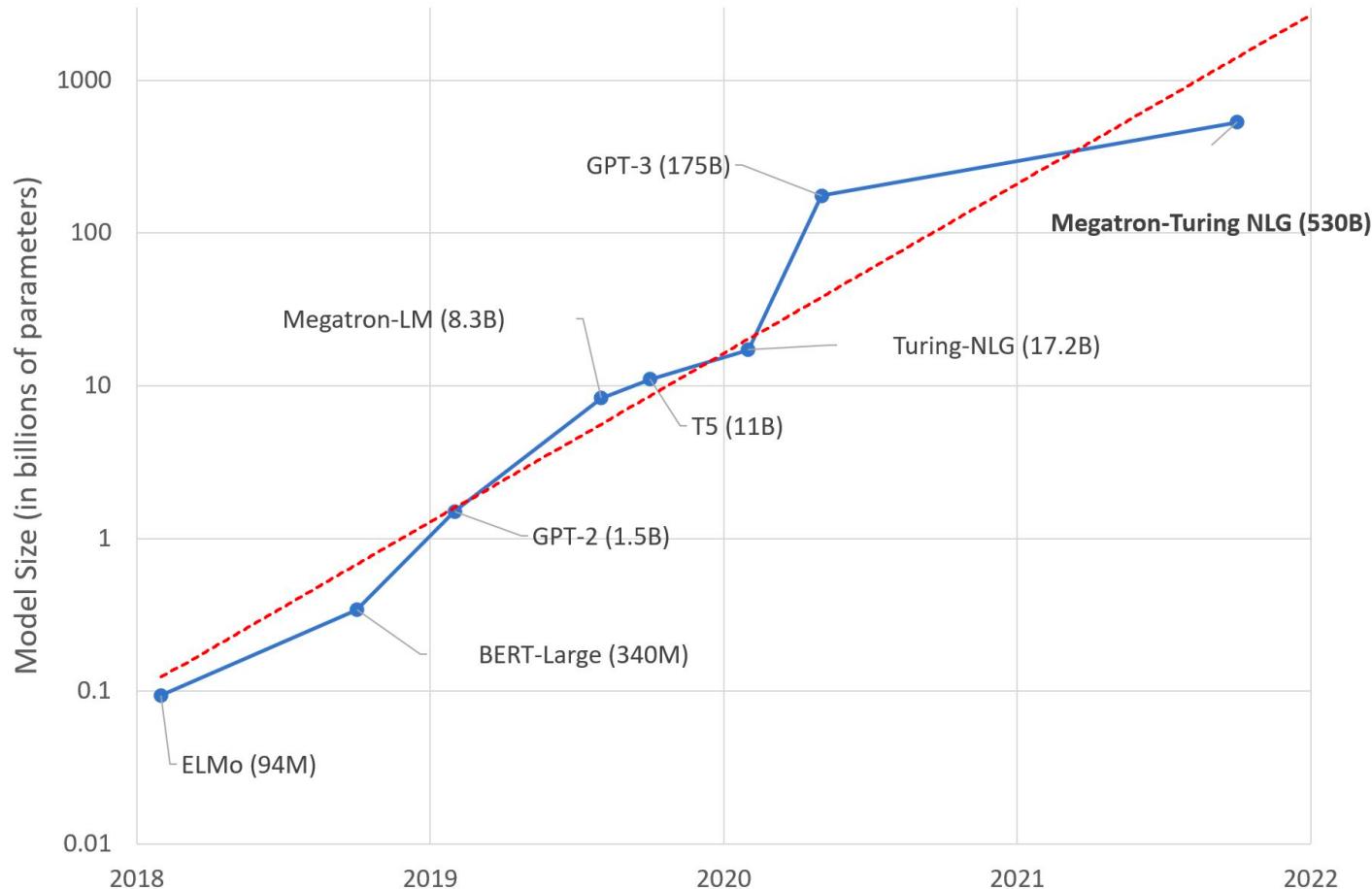
Language Modeling

- Language modeling started off as a means to introduce a “prior” into text generation systems such as machine translation or speech2text.
- With the advent of large language models (mostly starting from OpenAI’s GPT-2 in 2019), the use of language models has become much more ambitious
 - Sometimes viewed as a one-size-fits all solution to NLP problems

Large Language Models



Large (Neural) Langauge Models



Different Uses of Language Models

- We can use language models to answer questions:

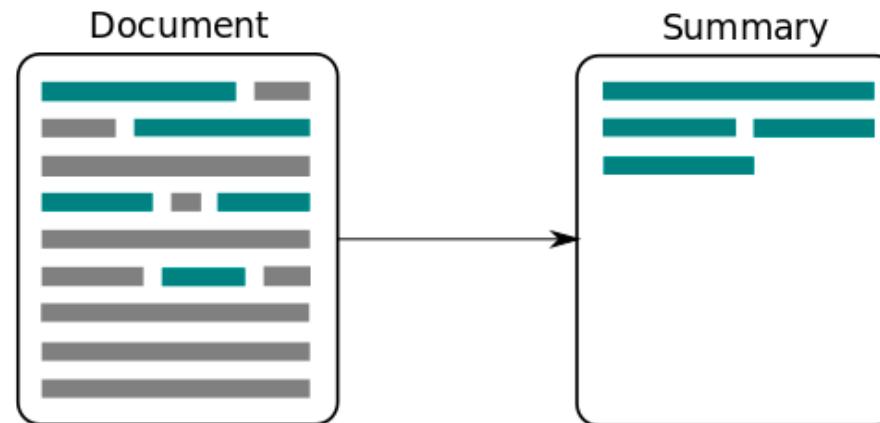
Q: how big is France?



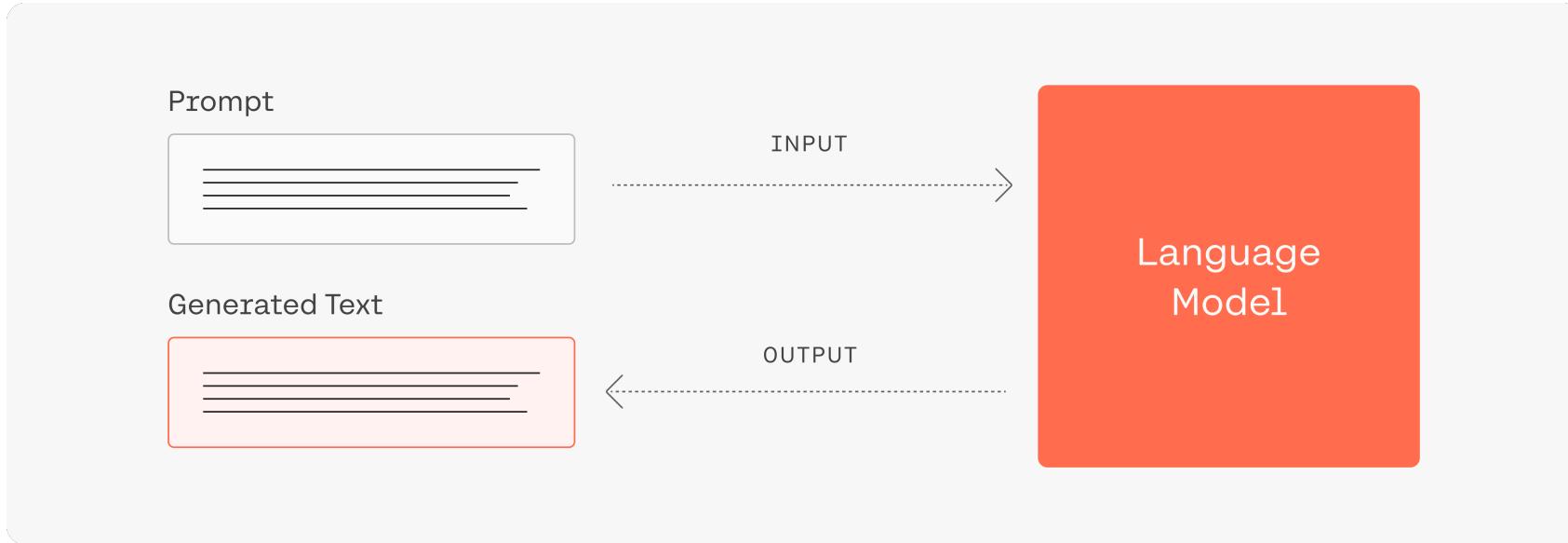
543,940 km²

A:

- Or for summarization:



Prompting



- Prompt: a prefix for the text such that the most likely continuation will be the desired answer
- Bypasses the need for additional training of the LM. The LM stays the same, and only its input changes
- Prompt Engineering

Prompting

Context → Title: The Blitz

Background: From the German point of view, March 1941 saw an improvement. The Luftwaffe flew 4,000 sorties that month, including 12 major and three heavy attacks. The electronic war intensified but the Luftwaffe flew major inland missions only on moonlit nights. Ports were easier to find and made better targets. To confuse the British, radio silence was observed until the bombs fell. X- and Y-Gerät beams were placed over false targets and switched only at the last minute. Rapid frequency changes were introduced for X-Gerät, whose wider band of frequencies and greater tactical flexibility ensured it remained effective at a time when British selective jamming was degrading the effectiveness of Y-Gerät.

Q: How many sorties were flown in March 1941?

A: 4,000

Q: When did the Luftwaffe fly inland missions?

A:

Target Completion → only on moonlit nights

Figure G.28: Formatted dataset example for SQuADv2

Language Modeling

- But what is this distribution?
 - Is there a “ground truth” distribution?
 - If not, what are useful distributions? what should we aim to capture?
- It is not clear whether there is a sense in which there is a “ground truth” distribution. Surely, it needs to depend on the domain, context and speaker/writer

Language Modeling

- At a minimum, probabilities should broadly indicate plausibility of sentences:
 - $P(I \text{ saw a van}) >> P(\text{eyes awe of an})$
 - Not only grammaticality: $P(\text{artichokes intimidate zippers}) \approx 0$
 - In principle, plausibility depends on the domain, context, speaker
- The minimal requirements reflect the classic approach to LMs

Language Modeling

- Using LMs to generate text makes much stronger assumptions, and essentially aims to reduce all dimensions of language into a single number
- This is its strength - we will see it greatly simplifies the task of NLP
- But also its weakness - language is fundamentally multi-dimensional in its core and much is lost by placing different strings on a scalar scale

Learning Language Models

- Goal: Assign useful probabilities $P(x)$ to sentences x
- Input: many observations of training sentences x
- Output: system capable of computing $P(x)$
- The straightforward approach is viewing it as density estimation over the space of finite strings over a given alphabet

Trivial Language Model

- Simplest option: empirical distribution over training sentences...

$$p(x_1 \dots x_n) = \frac{c(x_1 \dots x_n)}{N} \text{ for sentence } x = x_1 \dots x_n$$

- Problem: does not generalize (at all)
- We need to assign non-zero probability to previously unseen sentences!

Unigram Model

- Generative process: pick a word, pick a word, pick a word ... until you pick STOP

$$p(x_1 \dots x_n) = \prod_{i=1}^n p(x_i)$$

- Problem: these models disregard context completely

Bigram Models

- Condition on previous single word
- Generative process:
 - pick START, pick a word conditioned on previous one, repeat until STOP
- The model can be factored thusly:

$$p(x_1 \dots x_n) = \prod_{i=1}^n p(x_i | x_{i-1})$$

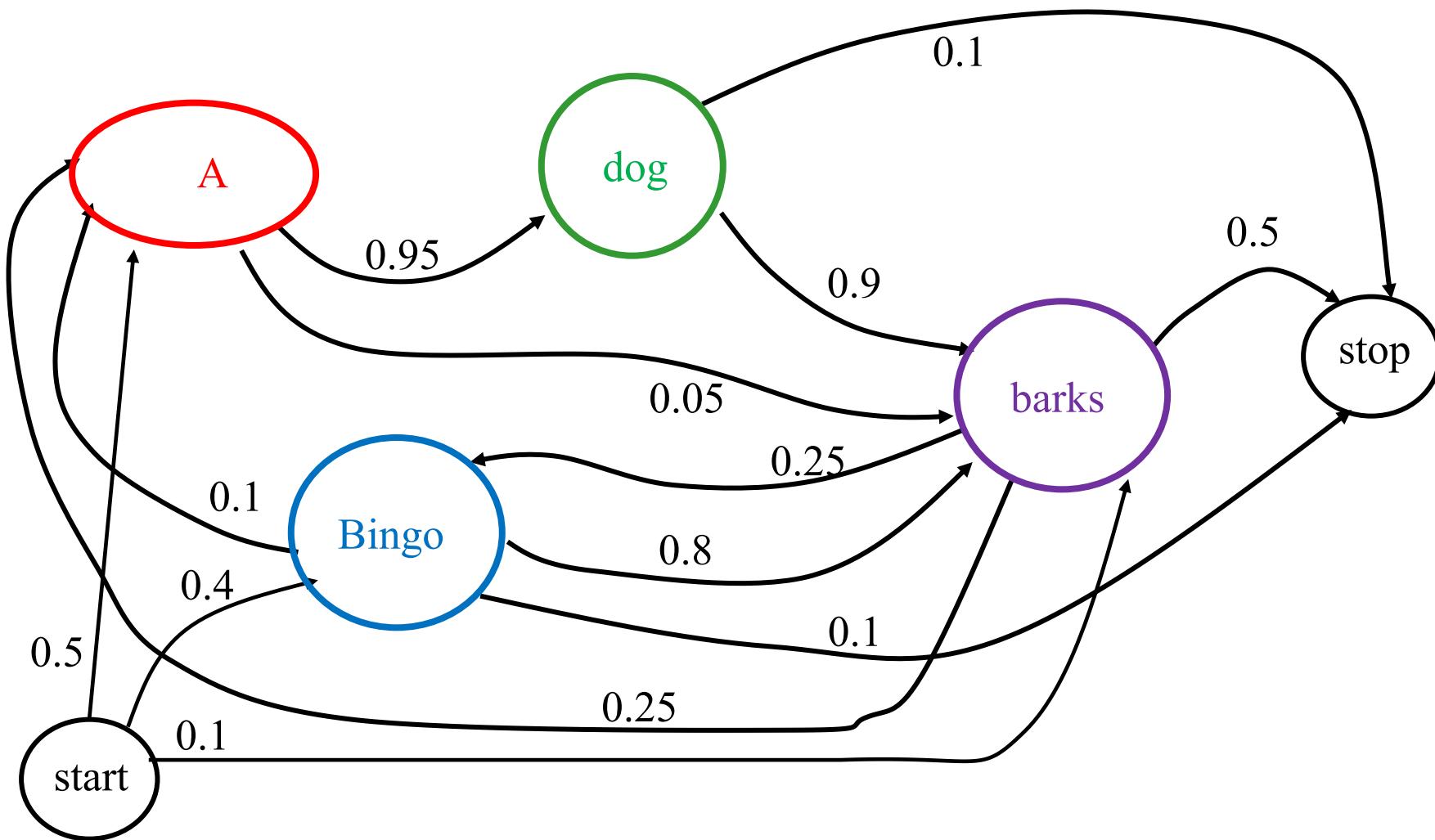
- This is equivalent to assuming the words of the sentence are (homogeneous, 1st order) Markov chains, namely that:

$$p(x_i | x_{i-1} \dots x_0) = p(x_i | x_{i-1})$$

Markov Model / Markov Chain

- A finite state automaton with probabilistic state transitions
- Makes Markov assumption that next state only depends on the current state and independent of previous history **given the current state**

Sample Markov Model for LM



Formal Representation of Markov LMs

- The parameters of a Markov LM can be represented as a stochastic, positive-valued matrix (*transition matrix*)
 - Sometimes the distribution of the first symbol is represented separately, but we can think about the sequences as always starting with n instances of a special START symbol

$$A_{ij} = P(x_m = w_j | x_{m-1} = w_i)$$

$$w_i, w_j \in WORDS \cup \{START, STOP\}$$

- Example:

| | START | FUNNY | LITTLE | THINGS | STOP |
|--------|-------|-------|--------|--------|------|
| START | 0 | 0.5 | 0.5 | 0 | 0 |
| FUNNY | 0 | 0.8 | 0.1 | 0.1 | 0 |
| LITTLE | 0 | 0.1 | 0.1 | 0.8 | 0 |
| THINGS | 0 | 0 | 0 | 0.1 | 0.9 |
| STOP | 0 | 0 | 0 | 0 | 1 |

Higher-order Markov Models can Help

- A bigram model will probably assign a similar probability to the sentences:

The birds I saw last night **sing** beautifully

The birds I saw last night **sings** beautifully

Or:

He saw himself in the mirror

He saw herself in the mirror

Higher-order Markov Models can Help

| | | |
|-------------------------|-------------------------|------------------------------|
| 198015222 the first | 197302 close the window | 3380 please close the door |
| 194623024 the same | 191125 close the door | 1601 please close the window |
| 168504105 the following | 152500 close the gap | 1164 please close the new |
| 158562063 the world | 116451 close the thread | 1159 please close the gate |
| ... | 87298 close the deal | ... |
| 14112454 the door | ----- | 0 please close the first |
| ----- | 3785230 close the * | ----- |
| 23135851162 the * | | 13951 please close the * |

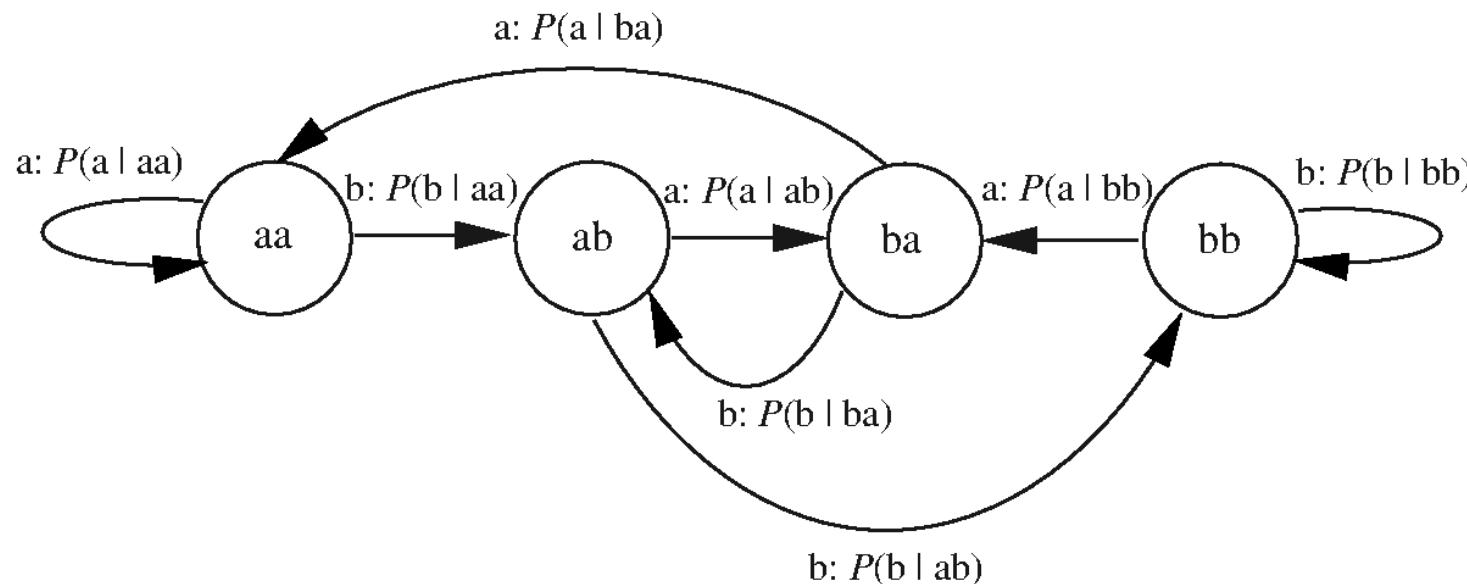
Higher-order Models

- k-th order Markov model

$$p(x_1, \dots, x_n) = \prod_{i=1}^n p(x_i | x_{i-1}, \dots, x_{i-k})$$

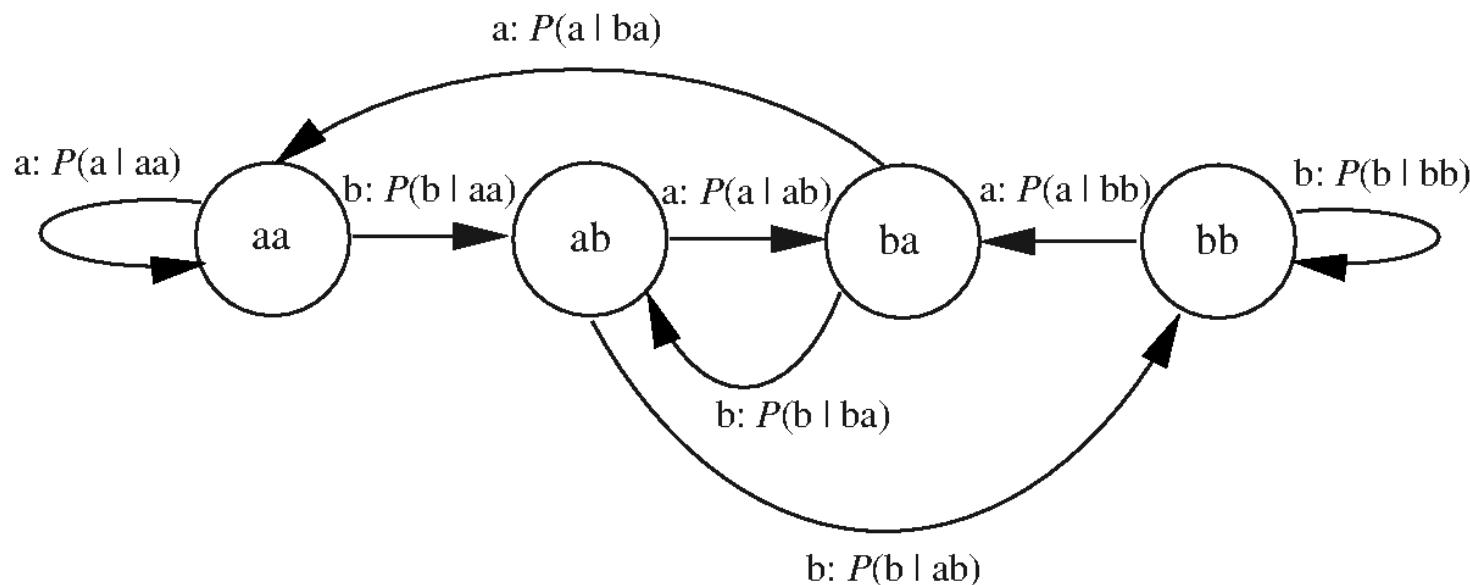
Higher-order Markov Language Models

- k-gram models are equivalent to bi-gram models where the state space (the words) are k-tuples of the bi-gram state space
- So we think of these models as modeling the transition between a k-tuple of states to a k-tuple of states, where the first $k-1$ coordinates of x_n must be equal to last $k-1$ coordinates of x_{n-1}



Higher-order Markov Language Models

- The difficulty: the number of parameters increases exponentially with k .
- So a k -gram language model would have about $|\text{Words}|^k$ entries in its transition matrix



Learning Bigram Language Models

- The parameters of a Markov LM can be represented as a stochastic, positive-valued matrix (*transition matrix*)
 - Sometimes the distribution of the first symbol is represented separately, but we can think about the sequences as always starting with n instances of a special START symbol

$$A_{ij} = P(x_m = w_j | x_{m-1} = w_i)$$

Learning Bigram Language Models

- A maximum likelihood estimator for the transition matrix is obtained by counting:

$$q_{ML}(x_m = w_j | x_{m-1} = w_i) = \frac{\text{count}(w_i, w_j)}{\sum_{j'} \text{count}(w_i, w_{j'})}$$

Learning Higher-order Language Models

- A maximum likelihood estimator for a k^{th} order language model is:

$$q_{ML}(x_m = w_j | x_{m-1} = w_{i1}, x_{m-2} = w_{i2}, \dots, x_{m-k} = w_{ik}) = \frac{\text{count}(w_{ik}, \dots, w_{i1}, w_j)}{\sum_{j'} \text{count}(w_{ik}, \dots, w_{i1}, w_{j'})}$$

Zero Counts

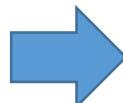
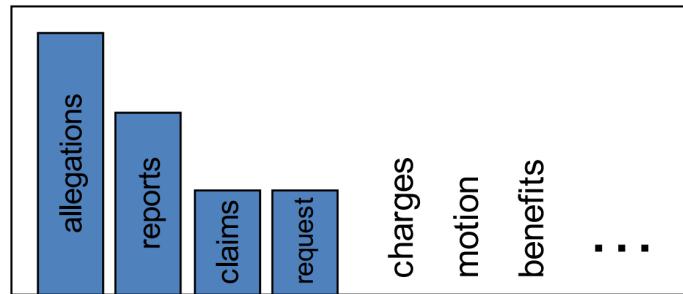
- Training set:
 - ... denied the allegations
 - ... denied the reports
 - ... denied the claims
 - ... denied the request
- Test set:
 - ... denied the offer
 - ... denied the loan

$$q_{ML}(offer|the, denied)$$

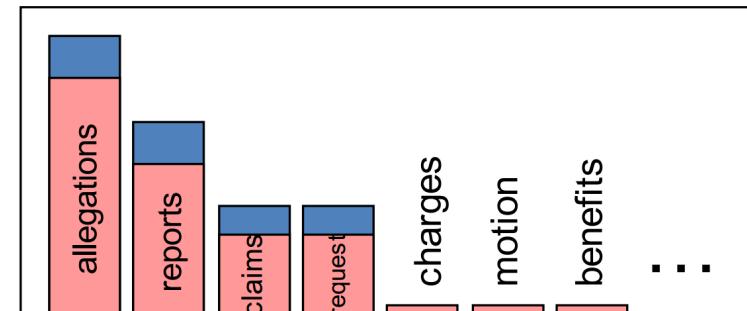
Zero Counts

- But, zero transition probabilities mean that the whole sentence receives a 0 probability!
- Therefore, zero probability estimates are generally avoided
- Methods for assigning probability mass to rare events are often called *smoothing*
 - Some have statistical motivation, others don't

$P(\text{offer}|\text{the}, \text{denied})$
3 allegations
2 reports
1 claims
1 request
7 total



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



Add- δ (Laplace) Smoothing

- Classic solution: add counts
 - If $q(w)$ is some distribution over words w then the smoothed $q_{add-\delta}(w)$ is

$$q_{add-\delta}(w) = \frac{c(w) + \delta}{\sum_{w'}(c(w') + \delta)} = \frac{c(w) + \delta}{c() + \delta|\mathcal{V}|}$$

- Problem: doesn't work well in practice for LMs

n-gram Smoothing

- The problem is not only zero counts
 - In general, where the event conditioned on has a low probability, the maximum likelihood estimator is unreliable
- Estimating the probability of an n-gram is a classical task in NLP
 - Not only in language models
 - For instance, knowing the common n-grams a word participates in can be telling in terms of its meaning
- The problem becomes worse as n increases
 - The denominator grows exponentially with n

Back-off Models

- **The idea:** assume we want to estimate a trigram distribution for a word x_m given its two previous words x_{m-1} and x_{m-2} :

$$q_{ML}(x_m = w_j | x_{m-1} = w_i, x_{m-2} = w_l) = \frac{\text{count}(w_l, w_i, w_j)}{\sum_{j'} \text{count}(w_l, w_i, w_{j'})}$$

- If the bigram (x_{m-1}, x_{m-2}) didn't appear enough times, we back-off to conditioning only on x_{m-1} :

$$q_{ML}(x_m = w_j | x_{m-1} = w_i) = \frac{\text{count}(w_i, w_j)}{\sum_{j'} \text{count}(w_i, w_{j'})}$$

- If w_{i-1} is not frequent enough, we back-off to w_i 's frequency:

$$q_{ML}(x_m = w_j) = \frac{\text{count}(w_j)}{\sum_{j'} \text{count}(w_{j'})}$$

Notation

- For brevity, we will assume that the transitions are not dependent on m (“homogeneous Markov models”) and use the notation

$$P(w_j|w_i, w_l)$$

instead of

$$P(x_m = w_j | x_{m-1} = w_i, w_{m-2} = w_l)$$

Back-off: Linear Interpolation

- Interpolate the trigram, bigram and unigram distributions by a convex combination:

$$\begin{aligned} q(w_i \mid w_{i-2}, w_{i-1}) = & \lambda_1 \times q_{\text{ML}}(w_i \mid w_{i-2}, w_{i-1}) \\ & + \lambda_2 \times q_{\text{ML}}(w_i \mid w_{i-1}) \\ & + \lambda_3 \times q_{\text{ML}}(w_i) \end{aligned}$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and $\lambda_i \geq 0$ for all i .

- To estimate λ_i , we split the training data into two: compute the n-gram probabilities q_{ML} on one part, and search for the λ_i that yield the highest probability on the other

The Probability of a Corpus

- The log-likelihood of a corpus is the product of the probabilities

$$P(\text{corpus}) = \prod_{i=1}^N P(s_i)$$

- If the model is a Markov LM (say a tri-gram model), it can be written as: (sub-script is token index, super-script is sentence index)

$$\prod_{i=1}^N P(s_i) = \prod_{i=1}^N \prod_{j=1}^{\text{length}(s_i)} P(w_j^{(i)} | w_{j-1}^{(i)}, w_{j-2}^{(i)})$$

- And if we denote the number of times the trigram (w_1, w_2, w_3) appears in the corpus with c' , we get:

$$\prod_{i=1}^N \prod_{j=1}^{\text{length}(s_i)} P(w_j^{(i)} | w_{j-1}^{(i)}, w_{j-2}^{(i)}) = \prod_{(w_1, w_2, w_3) \in V^3} P(w_3 | w_1, w_2)^{c'(w_1, w_2, w_3)}$$

Back-off: Linear Interpolation

- That is, maximize L , the log-likelihood, where $c'(w_1, w_2, w_3)$ is the number of times the trigram (w_1, w_2, w_3) appeared in the validation set

$$L(\lambda_1, \lambda_2, \lambda_3) = \log \left(\prod_{(w_1, w_2, w_3) \in V^3} q(w_3 | w_1, w_2)^{c'(w_1, w_2, w_3)} \right)$$

- Which yields this (concave) optimization problem:

$$L(\lambda_1, \lambda_2, \lambda_3) = \sum_{w_1, w_2, w_3} c'(w_1, w_2, w_3) \log q(w_3 | w_1, w_2)$$

such that $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and $\lambda_i \geq 0$ for all i , and where

$$\begin{aligned} q(w_i | w_{i-2}, w_{i-1}) = & \lambda_1 \times q_{\text{ML}}(w_i | w_{i-2}, w_{i-1}) \\ & + \lambda_2 \times q_{\text{ML}}(w_i | w_{i-1}) \\ & + \lambda_3 \times q_{\text{ML}}(w_i) \end{aligned}$$

Back-off: Linear Interpolation

- It is common to take more ls to get a better fit
- Here is one example of how to do that. Define:

$$\Pi(w_{i-2}, w_{i-1}) = \begin{cases} 1 & \text{If } \text{Count}(w_{i-1}, w_{i-2}) = 0 \\ 2 & \text{If } 1 \leq \text{Count}(w_{i-1}, w_{i-2}) \leq 2 \\ 3 & \text{If } 3 \leq \text{Count}(w_{i-1}, w_{i-2}) \leq 5 \\ 4 & \text{Otherwise} \end{cases}$$

And maximize:

$$q(w_i | w_{i-2}, w_{i-1}) = \lambda_1^{\Pi(w_{i-2}, w_{i-1})} \times q_{\text{ML}}(w_i | w_{i-2}, w_{i-1}) + \lambda_2^{\Pi(w_{i-2}, w_{i-1})} \times q_{\text{ML}}(w_i | w_{i-1}) + \lambda_3^{\Pi(w_{i-2}, w_{i-1})} \times q_{\text{ML}}(w_i)$$

where $\lambda_1^{\Pi(w_{i-2}, w_{i-1})} + \lambda_2^{\Pi(w_{i-2}, w_{i-1})} + \lambda_3^{\Pi(w_{i-2}, w_{i-1})} = 1$,
and $\lambda_i^{\Pi(w_{i-2}, w_{i-1})} \geq 0$ for all i .

Discounting Methods

- Maximum likelihood estimates tend to be too high for low count items:

| x | Count(x) | $q_{\text{ML}}(w_i \mid w_{i-1})$ |
|--------------|--------------|-----------------------------------|
| X | 48 | |
| X, dog | 15 | 15/48 |
| X, woman | 11 | 11/48 |
| X, man | 10 | 10/48 |
| X, park | 5 | 5/48 |
| X, job | 2 | 2/48 |
| X, telescope | 1 | 1/48 |
| X, manual | 1 | 1/48 |
| X, afternoon | 1 | 1/48 |
| X, country | 1 | 1/48 |
| X, street | 1 | 1/48 |

Discounting Methods

- One simple way to handle this is to discount all counts by a constant term: (say, 0.5)
- But what do we do with the missing probability mass?

| x | Count(x) | Count * (x) | $\frac{\text{Count}^*(x)}{\text{Count}(\text{the})}$ |
|--------------|--------------|--------------------|--|
| X | 48 | | |
| X, dog | 15 | 14.5 | 14.5/48 |
| X, woman | 11 | 10.5 | 10.5/48 |
| X, man | 10 | 9.5 | 9.5/48 |
| X, park | 5 | 4.5 | 4.5/48 |
| X, job | 2 | 1.5 | 1.5/48 |
| X, telescope | 1 | 0.5 | 0.5/48 |
| X, manual | 1 | 0.5 | 0.5/48 |
| X, afternoon | 1 | 0.5 | 0.5/48 |
| X, country | 1 | 0.5 | 0.5/48 |
| X, street | 1 | 0.5 | 0.5/48 |

Discounting Methods

- In a bigram model, the missing probability mass for a preceding word w_{i-1} is:

$$\lambda(w_{i-1}) = 1 - \sum_w \frac{c^*(w_{i-1}, w)}{c(w_{i-1})}$$

- In our example: $\lambda(w_{i-1}) = 10 \cdot 0.5 / 48$
- We will distribute *this* mass (also) to bigrams with probability 0
 - In fact, $P(w_i)$ will be non-zero for every word in the training data
- But how?

Discounting + Unigram Interpolation

- First guess: smooth by reducing some fixed quantity from the bi-gram count, and interpolate with the unigram estimate:

discounted bigram Interpolation weight

$$P_{KN}(w_i | w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i)$$

unigram

Kneser-Ney Smoothing

- Better estimate for the probabilities of unigrams:
 - What's the next word: *I can't see without my reading_____?*
 - “Francisco” is more common than “glasses”
 - ... but “Francisco” always follows “San”
- So we want to interpolate with a measure of “How likely is w to appear as a novel continuation?”
 - For each word, count the number of bigram types it completes
 - Every bigram type was a novel continuation the first time it was seen

$$P_{CONTINUATION}(w) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

Kneser-Ney Smoothing

- Properly normalized:

$$P_{CONTINUATION}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|}$$

- A frequent word (“Francisco”) occurring in only one context (“San”) will have a low continuation probability

Kneser-Ney Smoothing

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i)$$

- λ is a normalizing constant; assume $d \leq 1$.
- The probability mass we've discounted:

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{w : c(w_{i-1}, w) > 0\} \right|$$

Generalization in LMs

- Much of the criticism against Markov LMs (and about statistical methods to Computational Linguistics) has focused on the difficulty of such models in generalizing beyond the observed data
- Smoothing is a simple example for such a generalization
- But what about?

“Colorless green ideas sleep furiously”

vs.

“Furiously sleep ideas green colorless”

Chomsky (1957)

Pereira's Solution

- Now assume that when a word appears, it is accompanied by a category, but that this category is never observed
- The identity of the next word is determined based on this category
 - So words share a category if they share a distribution of the next word
- Examples:
 - The words “the” and “a” may share a category, because they are both followed by nouns and verbs

the/a cat
the/a dog
the/a university

Pereira's Solution

- In probabilistic terms we will say that each word w_i is accompanied by a latent category c_i such that

$$\begin{aligned} Pr(w_i|w_{i-1}) &= \sum_{c \in CATS} Pr(w_i, c|w_{i-1}) = \sum_{c \in CATS} Pr(c|w_{i-1}) \cdot Pr(w_i|c, w_{i-1}) = \\ &\quad \sum_{c \in CATS} Pr(c|w_{i-1}) \cdot Pr(w_i|c) \end{aligned}$$

- The last transition assumes that w_{i-1} and w_i are conditionally independent given c
- The model is now free to assign words to categories, such that knowing the category of a word is the best predictor of the following word

Pereira's Solution

- Using 16 categories and newspaper corpora to train the model (using EM), Pereira found that:

$$\frac{\Pr(\textit{Colorless green ideas sleep furiously})}{\Pr(\textit{Furiously sleep ideas green colorless})} \approx 200000$$

- Today's LMs use similar technology to generalize to novel word sequences, by creating a multi-dimensional representation of each word

LM Evaluation

- Assume we have m sentences s_1, s_2, \dots, s_m
- The common way to evaluate language models is by the probability they assign to held-out data
- More exactly, we use *perplexity*:

$$\text{Perplexity} = e^{-l} \quad l = \frac{1}{M} \sum_{i=1}^m \log P(s_i) = \frac{1}{M} \sum_{i=1}^m \sum_j \log P(x_j^{(i)} | x_1^{(i)}, \dots, x_{j-1}^{(i)})$$

M is the total number of tokens (words) in the test data

m is the number of sentences in the test data

Some Perplexity Values

- Perplexity measures the effective number of possibilities for the next word (on average)
- If our model predicts the probability of a word to be $1/N$, regardless of the context (a uniform model), the perplexity will be N
- Perplexity values would change considerably between corpora, but Markov models often vary between 50 and 500. Newer models can be an order of magnitude or more lower
- **Extrinsic evaluation** (by applying it to a downstream application) is usually conducted in parallel to perplexity evaluation

Predictability and Psycholinguistics

- Language modeling as a statistical process has been much studied in the context of psycholinguistics
- Examples:
 - Reading time of a word has been shown to be logarithmic in its predictability ([Smith and Levy, 2013](#))
 - The length of words has been shown to be related to their predictability ([Piantadosi et al., 2010](#))
 - Related to the “Uniform Information Density” hypothesis, which states that information is evenly spread out through a sentence
 - Event related potentials (ERPs) have been found where prediction is violated (see [here](#))

Questions?

We'll also talk about Neural Network-based LM later on