Introduction to Natural Language Processing

Part VIII: NLP using Language Models

Henning Wachsmuth

https://ai.uni-hannover.de

Learning Objectives

Concepts

- n-gram probability distributions
- Perplexity of language models
- The notion of prompting

Methods

- Text generation with n-gram language models
- Dealing with unknown words in language models
- Different types of smoothing to alleviate model sparsity
- Beam search for improved text generation

Covered tasks

- Free text generation

Outline of the Course

Introduction

Language Models

Example: Next words

- Given the following sequence of words:

  ChatGPT is based on a neural language __________

- Which of the following is the most likely next word?

  that | model | learning | language | ...

Example: Probabilities of word sequences

- Given the following two sequences of words:

  language models have become a key technique in NLP

  NLP models language in key have become a technique

- Which of them seems more likely?

Language Models

n-Gram Language Model

Language model (LM)

- A language model represents a probability distribution over sequences of tokens, $s = (w_1, \ldots, w_k)$, with $k \geq 1$.
- It thus defines the probability $P(s)$ of any token sequence s.
- Also, it assigns a probability $P(w_{k+1}|s)$ to any next token w_{k+1} after s.

Where do the probabilities come from?

- $P(s)$ can be approximated by the relative frequency of s in a corpus.
- For longer s, $P(s)$ may be unreliable (or even 0) due to data sparsity.

n-gram language model

- An n-gram LM derives the probability of s from the probability of all token sequences of length n contained in s.
- $n \geq 1$ is a predefined hyperparameter of the LM.
- The larger n, the more data is needed to get reliable estimations $P(s)$.

Language Models

Challenges in Language Modeling

Vanishing probabilities

- In real-world data, the probability of most token sequences s is near 0, which may lead to vanishing probabilities.
- A way to deal with this problem is to use *log probabilites*.

Unknown words and sequences

- Some tokens may never appear in a training corpus.
- Even without unknown tokens, there may always be sequences s that do not appear in training corpus, but appear in other data.
- A technique used to deal with these problems is called *smoothing*.

Exactness vs. generalization

- The higher n, the more exact the estimated probabilities.
- Sometimes, less context (i.e., a lower n) may aid generalization.
- Two techniques to deal with this problem are *backoff* and *interpolation*.

Language Models
Applications

When to use LMs?

- Probabilities of token sequences are essential in any task where tokens have to be inferred from ambiguous input.
- Ambiguity may be due to linguistic variations or due to noise.
- LMs are a key technique in generation, but are also used for analysis.

Selected applications

- Speech recognition. Disambiguate unclear words based on likelihood.

  wreck a nice beach   recognize speech

- Spelling/Grammar correction. Find likely errors and suggest alternatives.

  I booked one and Tim booked too   I booked one and Tim booked two

- Machine translation. Find likely interpretation/order in target language.

  爱国人 → love country human → country loving human

Language Models

Applications: Free Text Generation

Free text generation

- Nowadays, the key application of LMs is free text generation.
- Input. An n-gram representing the beginning of a text, called the *prompt*
- Output. The most likely sequence of text following the prompt

Input. Introduction to Natural	$\rightarrow$	Output. Language Processing is just madness.
Input. What is INLP?	$\rightarrow$	Output. Just madness.

How to generate text?

- Stepwise predict the most likely next token (diversity can be enforced).

$$w_k := \text{argmax}_w \, P(w \mid w_{k-(n-1)}, \ldots, w_{k-1})$$

How to *stop* generating text?

- The maximum length of the output sequence may be prespecified.
- Also, LMs may learn to generate a special end tag, </s>.

Outlook: Beyond N-Gram Language Models

Neural language models

- LMs that rely on neural networks to get the probabilities of next tokens
- Main difference: Tokens modeled as real-valued vectors (*embeddings*)
- This enables generalizing learned dependencies to unseen sequences.

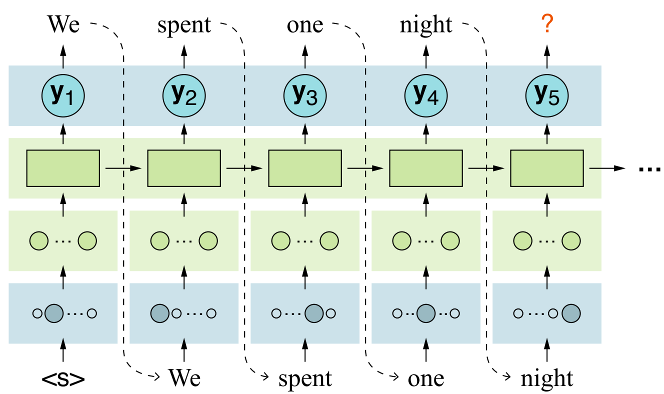
Training: $\text{argmax}_w\ P(w \mid \text{the people were}) = \text{lovely}$
Application: $P(\text{lovely} \mid \text{the peepz were}) = ?$

How are probabilities computed?

- As for an n-gram LM, probabilities are derived from a corpus.
- Neural LMs are *trained* (unsupervised) to predict probabilities.

Autoregressive text generation

- Stepwise append the most likely next token to the prompt and its previously appended tokens.



Outlook: Beyond N-Gram Language Models

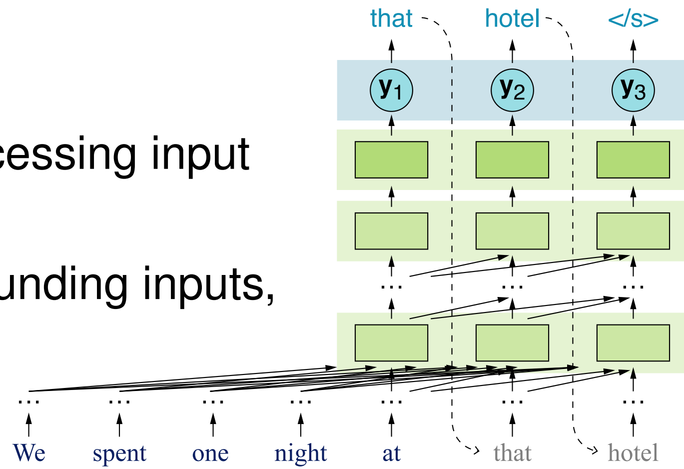
Large Language Models

Large language model (LLM)

- A neural language model trained on huge amounts of textual data
- Usually based on the *transformer* architecture

Transformer

- A neural network architecture for processing input sequences in parallel
- Models each input based on its surrounding inputs, called *self-attention*
- Examples. GPT-x, BART, T5, ...



Example: ChatGPT https://chat.openai.com

- A dialogue system based on GPT-3.5/GPT-4 that answers reasonably (and often impressively) to nearly any human-written input
- Notice that ChatGPT still has clear limitations, e.g., in terms of factuality.

n-Gram Language Models

N-Grams

n-gram

- An n-gram s is a sequence of n tokens for a fixed $n \geq 1$
- A text with $m \geq n$ tokens consists of $m - n + 1$ (overlapping) n-grams.
- Example. “The quick brown fox jumps over the lazy dog.”

1-grams (unigrams).	“The”, “quick”, “brown”, “fox”, ..., “dog”, “.”
2-grams (bigrams).	“The quick”, “quick brown”, ..., “lazy dog”, “dog.”
3-grams (trigrams).	“The quick brown”, “quick brown fox”, ..., “lazy dog.”

Notation

- $P(w)$. The probability that a variable X_i has the value “w”, $P(X_i = \text{“}w\text{”})$
- $P(w_1, \ldots, w_k)$. The joint probability $P(X_1 = \text{“}w_1\text{”}, \ldots, X_k = \text{“}w_k\text{”})$

Chain rule of probabilities (CRP)

- The joint probability of a sequence of values “w_1”, . . . , “w_k” is defined as:

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

N-Grams

Estimating Probabilities

Problem

- How to determine the probability of “model” in the initial example?

$$P(\text{model} \mid \text{ChatGPT is based on a neural language})$$

Solution?

- Given a corpus, it can be estimated from frequency counts:

$$\frac{\#\ \text{ChatGPT is based on a neural language model}}{\#\ \text{ChatGPT is based on a neural language}}$$

Problem

- Even a huge corpus does not allow for good estimates in many cases.
- This is due to language diversity: too many sequences are possible.

Solution

- Simplify the estimation of probabilities. $\rightarrow$ n-gram language model

N-Grams

Intuition of the n-gram Language Model

Simplification

- Instead of modeling the full history of a token (i.e., *all* previous tokens), approximate the history by the previous $n-1$ tokens only.
- So, the probability of a token w_k given its previous tokens $w_1, \ldots, w_{k-1}$ is approximated as follows:

$$P(w_k|\, w_1, \ldots, w_{k-1}) \;\approx\; P(w_k|\, w_{k-(n-1)}, \ldots, w_{k-1})$$

Example: Bigrams

- Approximate the probability of token w_k given $w_1, \ldots, w_{k-1}$ only based on its previous token w_{k-1}:

$$P(w_k|\, w_1, \ldots, w_{k-1}) \;\approx\; P(w_k|\, w_{k-1})$$

- The conditional probability sought for above is thus simplified to:

$P($model | ChatGPT is based on a neural language$)$ $\approx (P($model | language$)$

N-Grams

Maximum Likelihood Estimation (MLE)

Maximum likelihood estimation (MLE)

- In general, the conditional probability of a token w_k in a sequence of tokens $s = (w_1, \ldots, w_k)$ can be estimated as:

$$P(w_k | \, w_1, \ldots, w_{k-1}) \;\approx\; \frac{\#(w_1, \ldots, w_k)}{\#(w_1, \ldots, w_{k-1})},$$

  where $\#$ refers to the count of the sequences in a corpus.

- With the n-gram simplification, only $n-1$ previous tokens are modeled:

$$P(w_k | \, w_{k-(n-1)}, \ldots, w_{k-1}) \;\approx\; \frac{\#(w_{k-(n-1)}, \ldots, w_k)}{\#(w_{k-(n-1)}, \ldots, w_{k-1})}$$

- Later, we see how to further adjust the MLE to get better estimates.

Example: Bigrams

- Only the previous token is modeled:

$$P(w_k | \, w_{k-1}) \;\approx\; \frac{\#(w_{k-1}, w_k)}{\#(w_{k-1})}$$

N-Gram Language Model

Language model (LM)

- A probability distribution over a sequence of tokens
- An LM assigns a probability $P(w_1, \ldots, w_k)$ to each sequence of tokens $s = (w_1, \ldots, w_k)$ for any length $k \geq 1$.

n-gram language model

- An LM that approximates the probability of a sequence $s = (w_1, \ldots, w_k)$ of $k \geq 1$ tokens for some $n \geq 1$ as:

$$P(w_1, \ldots, w_k) = \prod_{i=1}^{k} P(w_i | w_1, \ldots, w_{i-1}) \quad \approx \quad \prod_{i=1}^{k} P(w_i | w_{i-(n-1)}, \ldots, w_{i-1})$$

Start and end tags

- Start tags. To have a history for the first tokens in s (where $n > i$), start tags $<\mathrm{s}>$ are prepended to s.

  $n - 1$ start tags must be prepended, in general.

- End tag. $</\mathrm{s}>$ is appended to s to obtain a true probability distribution.

N-Gram Language Model

Example: Estimation of Conditional Probabilities

A mini training set with three sentences

<s> <s> language models model language </s>
<s> <s> model language as a language model </s>
<s> <s> language models as a model </s>

Selected bigram probabilities (only green tags considered)

$P(\text{language} \mid \text{<s>}) = \frac{2}{3} \approx 0.67$	$P(\text{model} \mid \text{<s>}) = \frac{1}{3} \approx 0.33$	$P(\text{a} \mid \text{<s>}) = \frac{0}{3} = 0.00$
$P(\text{models} \mid \text{language}) = \frac{2}{5} = 0.40$	$P(\text{</s>} \mid \text{language}) = \frac{1}{4} = 0.25$	$P(\text{a} \mid \text{as}) = \frac{2}{2} = 1.00$

Selected trigram probabilities (both blue and green tags considered)

$P(\text{language} \mid \text{<s><s>}) = \frac{2}{3} \approx 0.67$	$P(\text{model} \mid \text{<s><s>}) = \frac{1}{3} \approx 0.33$
$P(\text{models} \mid \text{<s> language}) = \frac{2}{2} = 1.00$	$P(\text{as} \mid \text{model language}) = \frac{1}{2} = 0.5$

N-Gram Language Model

Example: Computation of Sequence Probabilities

A test sentence

$s = $ <s><s> model language as a model </s>

Probability computation under bigram LM (only green tags considered)

$$\begin{aligned} P_{n=2}(s) &= P(\text{model} \mid \text{<s>}) \cdot P(\text{language} \mid \text{model}) \cdot P(\text{as} \mid \text{language}) \\ &\quad \cdot P(\text{a} \mid \text{as}) \cdot P(\text{model} \mid \text{a}) \cdot P(\text{</s>} \mid \text{model}) \\ &\approx 0.33 \cdot 0.5 \cdot 0.2 \cdot 1.0 \cdot 0.5 \cdot 0.67 \quad \approx 0.0111 \end{aligned}$$

Probability computation under trigram LM (both blue and green tags considered)

$$\begin{aligned} P_{n=3}(s) &= P(\text{model} \mid \text{<s><s>}) \cdot P(\text{language} \mid \text{<s> model}) \\ &\quad \cdot P(\text{as} \mid \text{model language}) \cdot P(\text{a} \mid \text{language as}) \\ &\quad \cdot P(\text{model} \mid \text{as a}) \cdot P(\text{</s>} \mid \text{a model}) \\ &\approx 0.33 \cdot 1.0 \cdot 0.5 \cdot 1.0 \cdot 0.5 \cdot 1.0 \quad = 0.0825 \end{aligned}$$

N-Gram Language Model

Practical Issues

What n to use?

- Bigrams are used in the examples above mainly for simplicity.
- In practice, mostly trigrams, 4-grams, or 5-grams are used.
- The higher n, the more training data is needed for reliable probabilities.

  Besides, notice that LMs may also consider capitalization and non-word tokens.

Log probabilites

- Computations are done in log space to avoid vanishing probabilities.
- Addition in log space is equivalent to multiplication in linear space.
- The actual probabilities can be recovered when needed:

$$p_1 \cdot \ldots \cdot p_k \quad = \quad e^{\log p_1 + \ldots + \log p_k}$$

n-gram vs. neural LMs

- The n-gram LM is the simplest way to map sequences to probabilities.
- Neural LMs extend them but build on the same language modeling idea.

Evaluation and Application of Language Models

Evaluation of LMs

- Extrinsic. Measure/Compare impact of LMs within an application.
- Intrinsic. Measure the quality of LMs independent of an application.

Example: Extrinsic evaluation of spelling/grammar correction

$P(\text{too} \mid \text{booked})$ vs. $P(\text{two} \mid \text{booked})$

$P(\text{too} \mid \text{Tim booked})$ vs. $P(\text{two} \mid \text{Tim booked})$

Intrinsic evaluation

- Compute all probabilities of an LM on the training set of a corpus.
- Measure the quality the LM on the test set.

  As usual, a validation set may also be needed during development.

How to measure the quality of an LM intrinsically?

- An LM is better, the higher the probability that it assigns to the test set.
- Rationale: The LM then predicts the test set more accurately.
- The measure used to reflect the probability is called *perplexity*.

Evaluation and Application of Language Models

Perplexity

Perplexity

- The perplexity PPL of an LM on a test set is the inverse probability of the test set, normalized by the number of tokens.
- If the test set is given as one long sequence, $s = (w_1, \ldots, w_m)$, then:

$$PPL(s) = P(w_1, \ldots, w_m)^{-\frac{1}{m}} = \sqrt[m]{\frac{1}{P(w_1, \ldots, w_m)}} \overset{\text{CRP}}{=} \sqrt[m]{\prod_{i=1}^{m} \frac{1}{P(w_i | w_1 \ldots, w_{i-1})}}$$

Perplexity of bigram LMs

- Under a bigram LM, the perplexity is accordingly computed as follows:

$$PPL(s) = \sqrt[m]{\prod_{i=1}^{m} \frac{1}{P(w_i | w_{i-1})}}$$

Notice

- Each sentence is included in s with start and end tag `<s>` and `</s>`.
- The end tags are counted as part of the length m (the start tags not).

Branching factor (BF)

- The number of next tokens in a language that can follow any token
- Perplexity can be understood as the *weighted average* branching factor.
- Example. The language of digits, $\Sigma = \{0, 1, \ldots, 9\}$

If $P(w) = 0.1$ for each $w \in \Sigma$ in a test set s, then $BF = 10$ and $PPL(s) = 10$.
If $P(w) = 0.95$ for any $w \in \Sigma$ in a test set s, then $BF = 10$ but $PPL(s) < 10$.

Example: Perplexity of n-gram models

- Training set. 38 million tokens from Wall Street Journal articles
- Test set. 1.5 milion tokens from other Wall Street Journal articles

Unigram LM: $PPL \approx 962$ | Bigram LM: $PPL \approx 170$ | Trigram LM: $PPL \approx 109$

Notice

- Perplexity values are comparable only for LMs with same vocabulary.
- Better (so, lower) perplexity does not imply more extrinsic effectiveness.

Evaluation and Application of Language Models

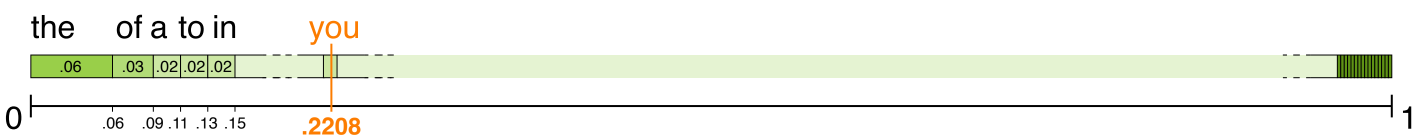
Sequence Sampling

Sampling of sequences

- The probabilities of an LM encode knowledge from the training set.
- To see this, sequences s can be sampled based on their likelihood $P(s)$.

Unigram sampling

- Decompose the probability space $[0, 1]$ into intervals, each reflecting the probability of one unigram from the LM vocabulary.



- Choose a random point in the space, and write the associated unigram.
- Repeat this process until $</s>$ is written.

Bigram sampling

- Same technique, starting by sampling random $w_1 = w$ from $P(w_1|<s>)$.
- Repeat process for $P(w_2| w)$ and so forth, until $</s>$ is written.

Evaluation and Application of Language Models

Text Generation using Sequence Sampling

Example: Sampling from Shakespeare's works (900k words, 29k unique words)

1-**grams:**	To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have	Hill he late speaks; or! a more to leg less first you enter
2-**grams:**	Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.	What means, sir. I confess she? then all sorts, he is trim, captain.
3-**grams:**	Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.	This shall forbid it should be branded, if renown made it empty.
4-**grams:**	King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;	It cannot be but so.

Observations

- As n is increased, n-gram LMs improve in generating coherent text.
- Under a 4-gram LM, some sequences are just copies of Shakespeare.

  The reason is data sparsity: $7 \cdot 10^{17}$ possible 4-grams, but less than $900k$ examples.

Advanced Language Modeling

Advanced Language Modeling

Sparsity

- n-grams frequent in a training set may get reliable probability estimates.
- But even huge training sets will not contain *all* possible n-grams.

Example: Wall Street Journal Treebank

- Counts of trigrams starting with “denied the”:

denied the allegation = 5 ... rumors = 1 ... speculation = 2 ... report = 1

- Probabilities of other trigrams starting with “denied the”:

$P(\text{denied the offer}) = 0$ $P(\text{denied the loan}) = 0$

Why are zero probabilities problematic?

- The probability of any unknown token (sequence) is underestimated.
- If any test set probability is 0, the probability of the entire test set is 0.
  What is the perplexity in this case?
- No next token can be predicted for any unknown token or sequence.

Advanced Language Modeling

Unknown Tokens

Out-of-vocabulary (OOV) tokens

- OOV tokens are those that appear in a test set but not in a training set.
- They are *unknown* to an LM built on the training set.
- Common examples. Slang words, misspellings, URLs, rare words, ...

Solution

- Replace all unknown tokens in a test set by a special tag, `<UNK>`.
- As for any token, estimate the probability of `<UNK>` on the training set.
- Two common ways to obtain `<UNK>` training instances exist.

Alternative 1: Closed vocabulary

1. Choose a fixed vocabulary of known tokens in advance.
2. Convert any other (OOV) token to `<UNK>`.

Alternative 2: Frequency pruning

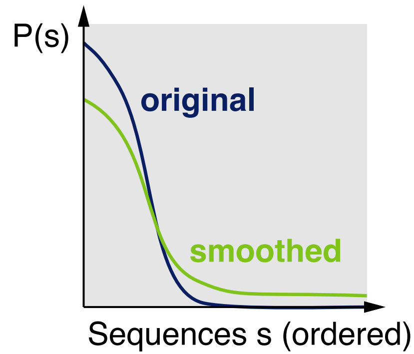
1. Choose a minimum absolute or relative frequency threshold, τ.
2. Convert any token with training frequency $< \tau$ to `<UNK>`.

Smoothing

Unknown sequences

- Even if all tokens in a sequence s are known, s as a whole might have never appeared in a training set.
- Techniques to avoid that $P(s) = 0$ in such cases are called *smoothing*.

General idea of smoothing (aka discounting)



- Reduce the probability mass of known sequences.
- Distribute gained mass over unknown sequences.

Main types of smoothing

- Laplace smoothing and Add-k smoothing
- Backoff, simple interpolation, and conditional interpolation
- Absolute discounting and Kneser-Ney smoothing
- Stupid backoff

Smoothing

Laplace Smoothing

Laplace smoothing (aka add-1 smoothing)

- Add 1 to the count of *all* n-gram counts before estimating probabilites.

  So, an unseen n-gram gets a count of 1, one with count 100 has 101, ...

Unigram MLE under Laplace smoothing

- Given a training set with m tokens, the unsmoothed unigram probability estimate of a token w is:

$$P(w) = \frac{\#w}{m}$$

- If the vobulary size is v, then the MLE of w is modified to:

$$P_{\text{Laplace}}(w) := \frac{\#w + 1}{m + v}$$

Notice

- Laplace smoothing is not used in practice, due to issues shown below.
- Rather, it shows the key smoothing idea and may serve as a baseline.

Smoothing

Laplace Smoothing: Example

Modified bigram counts and unigram counts for the mini training set

	language	model	models	as	a	</s>	# Unigram
<s>	2+1 = 3	... 2	1	1	1	1	3
language	0+1 = 1	... 2	3	2	1	2	5
model	... 3	1	1	1	1	3	4
models	1	2	1	2	1	1	2
as	1	1	1	1	3	1	2
a	2	2	1	1	1	1	2

Bigram probability estimation

- Under Laplace smoothing, the bigram probabilities are estimated as:

$$P_{\text{Laplace}}(w_i | w_{i-1}) \;:=\; \frac{\#(w_{i-1}, w_i) + 1}{\#w_{i-1} + v}$$

- Selected probabilities, given the vocabulary of size $v = 6$:

$P_{\text{Laplace}}(\text{language} \mid \text{<s>}) = \frac{2+1}{3+6} \approx 0.33$

$P_{\text{Laplace}}(\text{models} \mid \text{model}) = \frac{0+1}{4+6} = 0.10$

- Some probabilities are strongly reduced, as $P(\text{language} \mid \text{<s>})$ here.

  Before, $P(\text{language} \mid \text{<s>}) = 0.67$, as seen above.

Smoothing

Add-k Smoothing

Problem with Laplace smoothing

- Adding one to all counts may strongly change the probabilities.
- Too much probability mass is moved to all the (former) zero counts.
- A relaxation is to do *add-k smoothing* instead.

  This does not *solve* the problem, though. Further refinements follow below.

Add-k smoothing

- Add only a fractional count k to the count of all n-grams, $0 < k < 1$.
- k is a hyperparameter that can be optimized on a validation set.

  Typical values might be $k = 0.5$, $k = 0.05$, or $k = 0.01$.

Bigram probability estimation

- Under add-k smoothing, the bigram probabilities are estimated as:

$$P_{\text{Add}-k}(w_i|w_{i-1}) \;\; := \;\; \frac{\#(w_{i-1}, w_i) + k}{\#w_{i-1} + k \cdot v}$$

Backoff and Interpolation

Less context for better generalization

- If an n-gram probability cannot be computed, it can be approximated by probabilities of subsequences.
- Example. $P(w_i|w_{i-1})$ and/or $P(w_i)$ may be used for $P(w_i|w_{i-2}, w_{i-1})$.
- Two techniques to limit context this way are *backoff* and *interpolation*.

Backoff

- Reduce n by 1, if an n-gram probability $P(w_i|\ w_{i-(n-1)}, \ldots, w_{i-1}) = 0$.
- Repeat until $P(w_i|\ w_{i-(n-1)}, \ldots, w_{i-1}) > 0$ (latest at $n{=}1$, if `<UNK>` used).
- To maintain a probability distribution, discount higher-order n-grams.

Katz backoff

- Discount probabilities P^* for known n-grams.
- Use function λ to assign probabilities to lower-order n-grams of others.

  P^* and λ are estimated using *Good Turing smoothing* (beyond the scope here).

$$P_{\text{KB}}(w_i|w_{i-(n-1)}, \ldots, w_{i-1}) := \begin{cases} P^*(w_i|w_{i-(n-1)}, \ldots, w_{i-1}) & \text{if } \#(w_{i-(n-1)}, \ldots, w_i) > 0 \\ \lambda(w_{i-(n-1)}, \ldots, w_{i-1}) \cdot P_{\text{KB}}(w_i|w_{i-(n-2)}, \ldots, w_{i-1}) & \text{else} \end{cases}$$

Backoff and Interpolation

Interpolation

Interpolation

- Always mix weighted probability estimates from all n-gram estimators.
- Weights are usually chosen such that they maximize the likelihood of a validation set (i.e., minimizing perplexity).

Simple interpolation

- Combine different order n-gram probabilities via linear interpolation, using weights λ_j with $\sum_j \lambda_j = 1$
- Example. Unigrams, bigrams, and trigrams:

$$P_{\text{SI}}(w_i|w_{i-2}, w_{i-1}) \;:=\; \lambda_1 \cdot P(w_i) + \lambda_2 \cdot P(w_i|w_{i-1}) + \lambda_3 \cdot P(w_i|w_{i-2}, w_{i-1})$$

Conditional interpolation

- Condition each weight λ_j on the given context:

$$\begin{aligned} P_{\text{CI}}(w_i|w_{i-2}, w_{i-1}) \;:=\; & \lambda_1(w_{i-2}, w_{i-1}) \cdot P(w_i) + \lambda_2(w_{i-2}, w_{i-1}) \cdot P(w_i|w_{i-1}) \\ & + \lambda_3(w_{i-2}, w_{i-1}) \cdot P(w_i|w_{i-2}, w_{i-1}) \end{aligned}$$

Absolute Discounting

Discounting

- Smoothing discounts frequent sequences, to save probability for unknown sequences.
- Question: How much discounting is best?

Idea of absolute discounting

- Compare training set count to mean count on some validation set.

# Bigrams		
Train.	Valid.	$\triangle$
1	0.45	0.55
2	1.25	0.75
3	2.24	0.76
4	3.23	0.77
5	4.21	0.79
6	5.23	0.77
7	6.21	0.79
8	7.21	0.79
9	8.26	0.74

Table: Bigram counts in a 22M words news corpus.

- Choose fixed discount value d on this basis.

(Interpolated) Absolute discounting

- Subtract a fixed absolute discount d from each count.
- Distribute gained probability mass weighted over lower-order n-grams:

$$P_{\text{AD}}(w_i|w_{i-(n-1)}, \ldots, w_{i-1}) := \frac{\#(w_{i-(n-1)}, \ldots, w_i) - d}{\#(w_{i-(n-1)}, \ldots, w_{i-1})} + \lambda(w_{i-(n-1)}, \ldots, w_{i-1}) \cdot P(w_i|w_{i-(n-2)}, \ldots, w_{i-1})$$

Smoothing

Kneser-Ney Smoothing: Intuition

Kneser-Ney Smoothing in a nutshell

- Absolute discounting with a refined handling of lower-order distributions
- One of the best n-gram smoothing methods proposed so far

Unigram intuition of refinement

> ChatGPT is based on a neural language ___________

- A default unigram model will assign "york" a higher probability than "model", since "york" is more frequent in general.
- But "model" appears in many contexts, "york" mostly in "new york" only.

Refined lower-order n-gram handling

- Define probability of a sequence $s = (w_1, \ldots, w_k)$ from its likelihood to appear in novel contexts.
- Derive estimate from number of unigrams continued by s in a corpus:

$$\#\{w_0 : \#(w_0, \ldots, w_k) > 0\}$$

Smoothing

Kneser-Ney Smoothing

(Interpolated) Kneser-Ney Smoothing

- Use count $\#$ for highest order and continuation count for lower orders.
- Discount counts by d as in absolute discounting.
- Recursively distribute probability mass, normalized by a constant λ.

  λ is the normalized discount, multiplied by how often it is applied (see below).

$$P_{\mathsf{KN}}(w_i|w_{i-(n-1)}, \ldots, w_{i-1}) \;:=\; \frac{\max(c_{\mathsf{KN}}(w_{i-(n-1)}, \ldots, w_i) - d, 0)}{c_{\mathsf{KN}}(w_{i-(n-1)}, \ldots, w_{i-1})} + \lambda(w_{i-(n-1)}, \ldots, w_{i-1}) \cdot P_{\mathsf{KN}}(w_i|w_{i-(n-2)}, \ldots, w_{i-1})$$

where

$$c_{KN}(w_1, \ldots, w_k) \;:=\; \begin{cases} \#(w_1, \ldots, w_k) & \text{for highest-order } n\text{-grams} \\ \#\{w_0 : \#(w_0, w_1, \ldots, w_k) > 0\} & \text{for lower-order } n\text{-grams} \end{cases}$$

and

$$\lambda(w_{i-(n-1)}, \ldots, w_{i-1}) \;:=\; \frac{d}{\#(w_{i-(n-1)}, \ldots, w_i)} \cdot \#\{w_i : \#(w_{i-(n-1)}, \ldots, w_i) > 0\}$$

Large N-Gram Language Models

Large n-gram language models

- The larger the training set, the more reliable the estimated probabilities
- By employing web-scale text corpora, extremely large LMs can be built.

Example n-gram corpora

- Google Web NGrams. 1 trillion English word n-grams, $1 \leq n \leq 5$, all with 40+ occurrences
- Google Books NGrams. 800 billion token n-grams in eight languages

4-grams	Count
serving as the independent	794
serving as the index	223
serving as the indicator	120
serving as the incubator	99
serving as the incoming	92
...	...

Efficiency challenges

- The number of sequences and resulting n-gram probabilities explodes.
- Technical space optimizations may be necessary, such as hashing.
- To reduce time and space needs, less frequent n-grams can be pruned.

Large N-Gram Language Models

Stupid Backoff

Smoothing under large LMs

- It is possible to realize Kneser-Ney smoothing at web scale.
- Alternatively, however, the scale enables the resort to a much simpler method called *stupid backoff*.

Stupid backoff

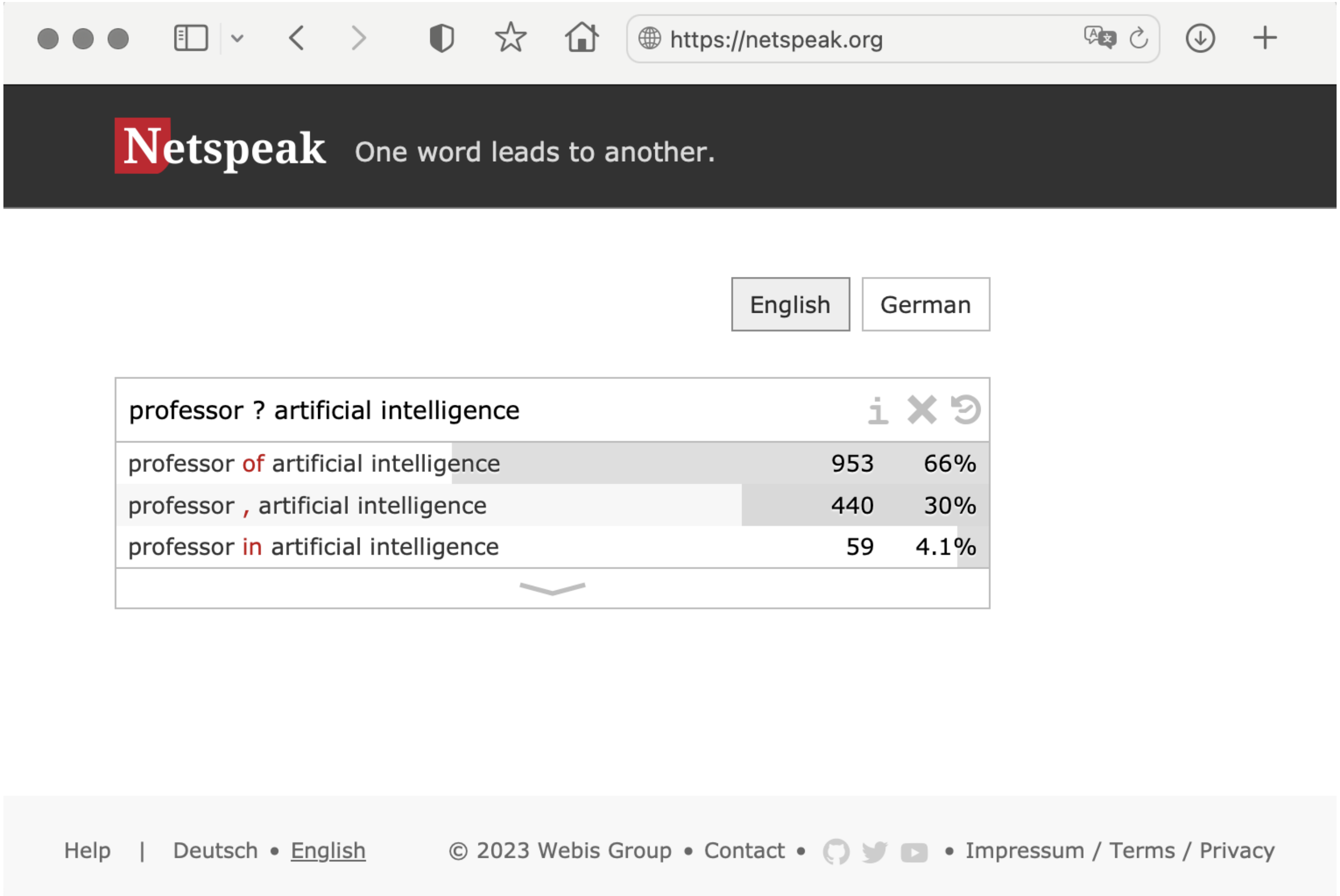
- Do not discount higher-order probabilities, i.e., drop the requirement to have a true probability distribution.
- If a higher-order n-gram is unknown, approximate its probability from a lower-order n-gram, weighted by a constant weight λ.

  $\lambda = 0.4$ has been found to work well in experiments.

$$S(w_i | w_{i-(n-1)}, \ldots, w_{i-1}) := \begin{cases} \frac{\#(w_{i-(n-1)}, \ldots, w_i)}{\#(w_{i-(n-1)}, \ldots, w_{i-1})} & \text{if } \#(w_{i-(n-1)}, \ldots, w_{i-1}) > 0 \\ \lambda \cdot S(w_i | w_{i-(n-2)}, \ldots, w_{i-1}) & \text{otherwise} \end{cases}$$

Evaluation and Application of Language Models

Netspeak: Writing Support based on n-Grams



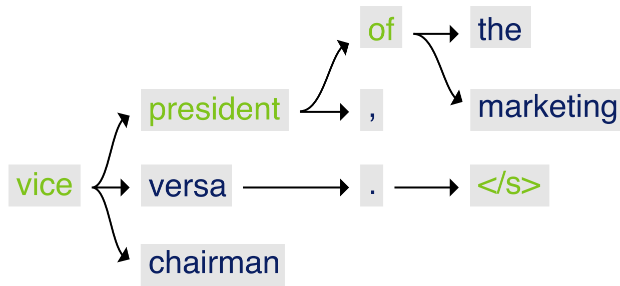
https://netspeak.org

Evaluation and Application of Language Models

Improving Results in Text Generation

Beam search

- A simple heuristic search method often used to find optimal sequences
- Instead of extending only the most likely sequence s^*, extend all $\beta > 1$ most likely sequences to finally generate best.
- Rationale: Another sequence $s \neq s^*$ may turn out better later.



Diversification using randomization

- A simple way to generate more diverse text is to randomize each step.
- Instead of writing the most likely token w_{k+1}, write any of the top $l \geq 1$.

Prompting

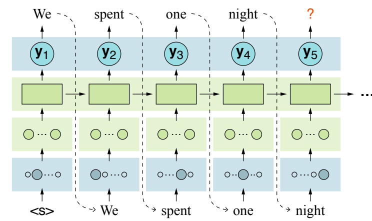
- The quality of the output of an LM always depends on the prompt.
- This is why *prompt engineering* is currently a hot topic in academia and industry (but beyond the scope of this course).

Conclusion

Conclusion

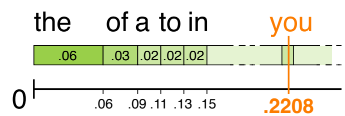
NLP using language models (LMs)

- LMs are probability distributions over token sequences
- Nowadays, one of the most central NLP techniques
- Used particularly for free text generation



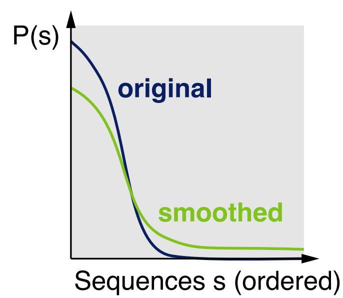
n-gram language models

- Estimation of probabilities from n tokens only
- The higher n, the more training data is needed
- The quality of an LM can be quantified as perplexity



Advanced language modeling

- Smoothing enables dealing with unknown sequences
- Backoff/Interpolation reduce context for generalization
- Different techniques to improve outputs exist



References

Much content and multiple examples taken from

- **Jurafsky and Martin (2021).** Daniel Jurafsky and James H. Martin. Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. Draft or 3rd edition, December 29, 2021. https://web.stanford.edu/ jurafsky/slp3/