Einführung in die Computerlinguistik

N-grams and language models

Laura Kallmeyer

Heinrich-Heine-Universität Düsseldorf

Summer 2022

Table of contents

Motivation

Goals:

- Estimate the probability that a given sequence of words occurs in a specific language.
- Model the most probable next word for a given sequence of words.

Jurafsky & Martin (2020), chapter 3 and 7, and Chen & Goodman (1999)

Motivation

Examples from Jurafsky & Martin (2020)

- (1) Please turn your homework …
  What is a probable continuation? Rather in or over and not refrigerator.

- (2) a. all of a sudden I notice three guys standing on the sidewalk
  b. on guys all I of notice sidewalk three a sudden standing the

  Which of the two word orders is better?

Language model (LM): Probabilistic model that gives $\mathrm{P}(\mathrm{w}_1 \ldots \mathrm{w}_\mathrm{n})$ and $\mathrm{P}(\mathrm{w}_\mathrm{n} | \mathrm{w}_1 \ldots \mathrm{w}_{\mathrm{n}-1})$

Motivation

Applications:

- Tasks in which we have to identify words in noisy, ambiguous input: speech recognition, handwriting recognition, ...
- spelling correction

Example

(3) a. their is only one written exam in this class
    b. there is only one written exam in this class

- machine translation: among a series of different word orders in the target language, one has to choose the best one.

Example

(4) a. Das Fahrrad wird er heute reparieren.
    b. The bike will he today repair
    c. The bike he will today repair.
    d. The bike he will repair today.

N-grams

Notation: $\mathrm{w}_1^{\mathrm{m}} = \mathrm{w}_1 \ldots \mathrm{w}_{\mathrm{m}}$.
Question: How can we compute $\mathrm{P}(\mathrm{w}_1^{\mathrm{m}})$?

$$\begin{aligned} \mathrm{P}(\mathrm{w}_1^{\mathrm{m}}) &= \mathrm{P}(\mathrm{w}_1)\mathrm{P}(\mathrm{w}_2|\mathrm{w}_1)\mathrm{P}(\mathrm{w}_3|\mathrm{w}_1^2)\ldots\mathrm{P}(\mathrm{w}_{\mathrm{m}}|\mathrm{w}_1^{\mathrm{m}-1}) \\ &= \prod_{\mathrm{k}=1}^{\mathrm{m}} \mathrm{P}(\mathrm{w}_{\mathrm{k}}|\mathrm{w}_1^{\mathrm{k}-1}) \end{aligned}$$

But: computing $\mathrm{P}(\mathrm{w}_{\mathrm{k}}|\mathrm{w}_1^{\mathrm{k}-1})$ for a large k is computationally expensive.

Approximation of $\mathrm{P}(\mathrm{w}_{\mathrm{k}}|\mathrm{w}_1^{\mathrm{k}-1})$: N-grams, i.e., look at just the $\mathrm{n}-1$ last words, $\mathrm{P}(\mathrm{w}_{\mathrm{k}}|\mathrm{w}_{\mathrm{k}-\mathrm{n}+1}^{\mathrm{k}-1})$ for some fixed n.

Special cases:

- unigrams: $\mathrm{n} = 1$, $\mathrm{P}(\mathrm{w}_{\mathrm{k}})$
- bigrams: $\mathrm{n} = 2$, $\mathrm{P}(\mathrm{w}_{\mathrm{k}}|\mathrm{w}_{\mathrm{k}-1})$
- trigrams: $\mathrm{n} = 3$, $\mathrm{P}(\mathrm{w}_{\mathrm{k}}|\mathrm{w}_{\mathrm{k}-2}\mathrm{w}_{\mathrm{k}-1})$

N-grams

With n-grams, we get

1. Probability of a sequence of words:

$$P(w_1^l) \approx \prod_{k=1}^{l} P(w_k | w_{k-n+1}^{k-1})$$

2. Probability of a next word:

$$P(w_l | w_1^{l-1}) \approx P(w_l | w_{l-n+1}^{l-1})$$

These are strong independence assumptions called Markov assumptions. E.g. with bigrams

Example

$P(\text{einfach}|\text{die Klausur war nicht}) \approx P(\text{einfach}|\text{nicht})$

Maximum likelihood estimation (MLE)

Question: How do we estimate the n-gram probabilities?

Maximum likelihood estimation (MLE): Get n-gram counts from a (large) corpus and normalize so that the values lie between 0 and 1.

$$\mathrm{P}(\mathrm{w_k}|\mathrm{w_{k-n+1}^{k-1}}) = \frac{\mathrm{C}(\mathrm{w_{k-n+1}^{k-1} w_k})}{\mathrm{C}(\mathrm{w_{k-n+1}^{k-1}})}$$

In the bigram case, this amounts to

$$\mathrm{P}(\mathrm{w_k}|\mathrm{w_{k-1}}) = \frac{\mathrm{C}(\mathrm{w_{k-1} w_k})}{\mathrm{C}(\mathrm{w_{k-1}})}$$

We augment sentences with an initial $\langle \mathrm{s} \rangle$ and a final $\langle /\mathrm{s} \rangle$

Maximum likelihood estimation (MLE)

Example from Jurafsky & Martin (2020)

Training data:

$< \text{s} >$ I am Sam $< /\text{s} >$
$< \text{s} >$ Sam I am $< /\text{s} >$
$< \text{s} >$ I do not like green eggs and ham $< /\text{s} >$

Some bigram probabilities:

$\text{P}(\text{I}\|< \text{s} >) = \frac{2}{3}$	$\text{P}(\text{Sam}\|< \text{s} >) = \frac{1}{3}$	$\text{P}(\text{am}\|\text{I}) = \frac{2}{3}$
$\text{P}(< /\text{s} >\|\text{Sam}) = \frac{1}{2}$	$\text{P}(\text{Sam}\|\text{am}) = \frac{1}{2}$	$\text{P}(\text{do}\|\text{I}) = \frac{1}{3}$

Maximum likelihood estimation (MLE)

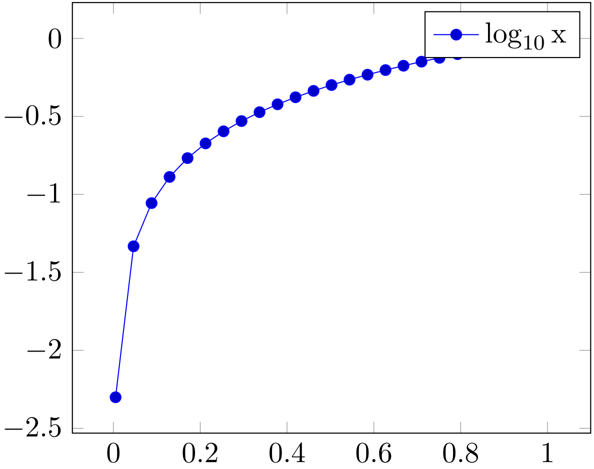
Practical issues:

- In practice, n is mostly between 3 and 5, i.e., we use tri-grams, 4-grams or 5-grams.
- LM probabilities are always represented as log probabilities. Advantage: Adding replaces multiplying and numerical underflow is avoided.

$$\mathrm{p}_1 \cdot \mathrm{p}_2 \cdot \ldots \mathrm{p}_\mathrm{l} = \exp(\log \mathrm{p}_1 + \log \mathrm{p}_2 + \cdots + \log \mathrm{p}_\mathrm{l})$$

Maximum likelihood estimation (MLE)

Reminder: $\log 1 = 0, \log 0 = -\infty$



Evaluating language models

The data is usually separated into

- a training set (80% of the data),
- a test set (10% of the data),
- and sometimes a development set (10% of the data).

The model is estimated from the training set. Intuitively, the higher the probability of the test set, the better the model.

But: Using probabilities prefers shorter sentences to longer ones, regardless of the sentence lengths in the training data.

Instead of measuring the probability of the test set, LMs are usually evaluated with respect to the perplexity of the test set. The lower the perplexity, the better the sentence.

Evaluating language models

The perplexity of a test set $W = w_1 w_2 \dots w_N$ is defined as

$$\begin{aligned} PP(W) &= P(W)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(W)}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \\ &= \sqrt[N]{\frac{1}{\prod_{k=1}^{N} P(w_k \mid w_1^{k-1})}} \end{aligned}$$

With our n-gram model, we get then for the perplexity:

Perplexity

$$PP(W) = \sqrt[N]{\frac{1}{\prod_{k=1}^{N} P(w_k \mid w_{k-n+1}^{k-1})}}$$

Evaluating language models

Example

Training data (toy example):

<s> a b c a b c </s> <s> a a b b a a b b a a </s>
<s> b b a a b b a a </s> <s> a b c a b a </s>

Some of the resulting bigram probabilities:

$P(a|<s>) = \frac{3}{4}$ $P(a|a) = \frac{5}{15} = \frac{1}{3}$ $P(b|a) = \frac{7}{15}$
$P(b|b) = \frac{4}{12} = \frac{1}{3}$ $P(c|b) = \frac{3}{12} = \frac{1}{4}$ $P(a|c) = \frac{2}{3}$

Probabilities and perplexities of the following sequencies:

(1) <s> a b c (2) <s> a b c a a b b

(1): probability $\frac{3}{4} \cdot \frac{7}{15} \cdot \frac{1}{4} = \frac{21}{240} = \frac{7}{80} = 0.0875$

perplexity $\sqrt[3]{\frac{80}{7}} \approx 2.252$

(2): probability $\frac{3}{4} \cdot \frac{7}{15} \cdot \frac{1}{4} \cdot \frac{2}{3} \cdot \frac{1}{3} \cdot \frac{7}{15} \cdot \frac{1}{3} = \frac{3 \cdot 7 \cdot 2 \cdot 7}{4 \cdot 15 \cdot 4 \cdot 3 \cdot 3 \cdot 15 \cdot 3} = \frac{49}{16200} \approx 0.003$

perplexity $\sqrt[7]{\frac{16200}{49}} \approx 2.29$

Evaluating language models

A different way to think about perplexity: it measures the weighted average branching factor of a language.

Example

$L = \{a, b, c, d\}^*$. Frequencies are such that $P(a) = P(b) = P(c) = P(d) = \frac{1}{4}$ (independent from the context).
For any $w \in L$, given this model, we obtain

$$PP(w) = \sqrt[|w|]{\frac{1}{\prod_{k=1}^{|w|} \frac{1}{4}}} = \sqrt[|w|]{\frac{1}{\frac{1}{4}^{|w|}}} = \sqrt[|w|]{4^{|w|}} = 4$$

The perplexity of any $w \in L$ under this model is 4.

Evaluating language models

Example

$L = \{a, b, c, d\}^*$. Words in the language contain three times as many a's as they contain b's, c's or d's. $P(a) = \frac{1}{2}$ and $P(b) = P(c) = P(d) = \frac{1}{6}$. For any $w \in L$ with these frequencies and with $|w| = 6n$:

$$PP(w) = \sqrt[6n]{\frac{1}{\prod_{k=1}^{n} \frac{1}{2 \cdot 2 \cdot 2 \cdot 6 \cdot 6 \cdot 6}}} = \sqrt[6n]{2^{6n} \cdot \sqrt{3}^{6n}} = 2\sqrt{3} = 3.46$$

Assume that we use the same model but test it on a w with equal numbers of as, bs, cs and ds, $|w| = 4n$. Then we get

$$PP(w) = \sqrt[4n]{\frac{1}{\prod_{k=1}^{n} \frac{1}{2 \cdot 6 \cdot 6 \cdot 6}}} = \sqrt[4n]{2^{4n} \cdot 3^{3n}} = 2\,\sqrt[4n]{3^{\frac{3}{4}4n}} = 2\sqrt[4]{27} = 4.56$$

Unknown words

Problem: New text can contain

- unknown words; or
- unseen n-grams.

In these cases, with the algorithm seen so far, we would assign probability 0 to the entire text. (And we would not be able to compute perplexity at all.)

Example from (Jurafsky & Martin, 2020)

Words following the bigram denied the in WSJ Treebank 3 with counts:

denied the allegations	5
denied the speculation	2
denied the rumors	1
denied the report	1

If the test set contains denied the offer or denied the loan, the model would estimate its probability as 0.

Unknown words

Unknown or out of vocabulary words:

- Add a pseudo-word $\langle$UNK$\rangle$ to your vocabulary.
- Two ways to train the probabilities concerning $\langle$UNK$\rangle$:
    1. Choose a vocabulary V fixed in advance. Any word $w \notin V$ in the training set is converted to $\langle$UNK$\rangle$. Then estimate probabilities for $\langle$UNK$\rangle$ as for all other words.
    2. Replace the first occurrence of every word w in the training set with $\langle$UNK$\rangle$. Then estimate probabilities for $\langle$UNK$\rangle$ as for all other words.

Smoothing

Unseen n-grams: To avoid probabilities 0, we do smoothing: Take off some probability mass from the events seen in training and assign it to unseen events.

Laplace Smoothing (or add-one smoothing):

- Add 1 to the count of all n-grams in the training set before normalizing into probabilities.
- Not so much used for n-grams but for other tasks, for instance text classification.

- For unigrams, if N is the size of the training set and $|\mathrm{V}|$ the size of the vocabulary, we replace $\mathrm{P(w)} = \frac{\mathrm{C(w)}}{\mathrm{N}}$ with $\mathrm{P_{Laplace}(w)} = \frac{\mathrm{C(w)+1}}{\mathrm{N+|V|}}$.
- For bigrams, we replace $\mathrm{P(w_n|w_{n-1})} = \frac{\mathrm{C(w_{n-1}w_n)}}{\mathrm{C(w_{n-1})}}$ with $\mathrm{P_{Laplace}(w_n|w_{n-1})} = \frac{\mathrm{C(w_{n-1}w_n)+1}}{\mathrm{C(w_{n-1})+|V|}}$.

Smoothing

Smoothing methods for n-grams that use the $(n-1)$-grams, $(n-2)$-grams etc.:

- Backoff: use the trigram if it has been seen, otherwise fall back to the bigram and, if this has not been seen either, to the unigram.
- Interpolation: Use always a weighted combination of the trigram, bigram and unigram probabilities.

Linear interpolation:

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$$

with $\sum_i \lambda_i = 1$.

Smoothing

More sophisticated: each λ is computed conditioned on the context.

$$\begin{aligned}\hat{\mathrm{P}}(\mathrm{w_n}|\mathrm{w_{n-2}w_{n-1}}) &= \lambda_1(\mathrm{w_{n-2}w_{n-1}})\mathrm{P}(\mathrm{w_n}|\mathrm{w_{n-2}w_{n-1}}) \\ &\quad +\lambda_2(\mathrm{w_{n-2}w_{n-1}})\mathrm{P}(\mathrm{w_n}|\mathrm{w_{n-1}}) \\ &\quad +\lambda_3(\mathrm{w_{n-2}w_{n-1}})\mathrm{P}(\mathrm{w_n})\end{aligned}$$

In both cases,

- the probabilities are first estimated from the training corpus,
- and the λ parameters are then estimated from separate held-out data.
- They are estimated such that they maximize the likelihood of the held-out data.

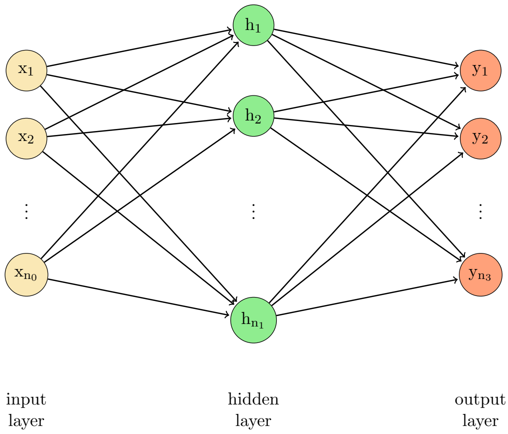
Neural Language Models

State-of-the-art language models are oftentimes based on neural network architectures.

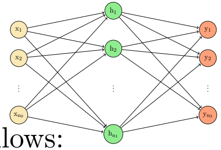
In the following, we briefly sketch how a simple feedforward neural language model might look like (Jurafsky & Martin, 2020, chapter 7).

Neural Language Models

A feedforward neural network with one hidden layer works as follows:



Neural Language Models



- The neural network takes an input vector with components $x_1, \ldots x_{n_0}$
- for the n_1 nodes of the hidden layer, activations $h_1, \ldots, h_{n_1}$ are computed as follows:

  $$h_j = \sigma\left(\sum_{i=1}^{n_0} w_{ji} x_i + b_j\right)$$

  σ is a non-linear activation function.
- for the n_2 nodes of the output layer, activations $z_1, \ldots, z_{n_2}$ are computed as follows:

  $$z_j = \sum_{i=1}^{n_1} u_{ji} h_i$$

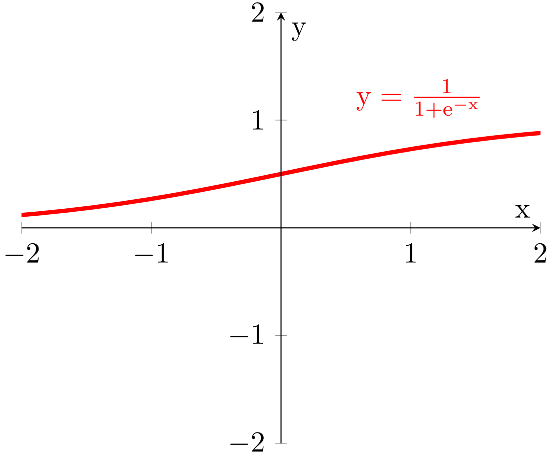
  A subsequent softmax yields $y_1, \ldots, y_{n_2}$ with $\sum_{i=1}^{n_2} y_i = 1$.

The w_{ji}, b_j and u_{ji} are parameters that are learned during training.

Neural Language Models

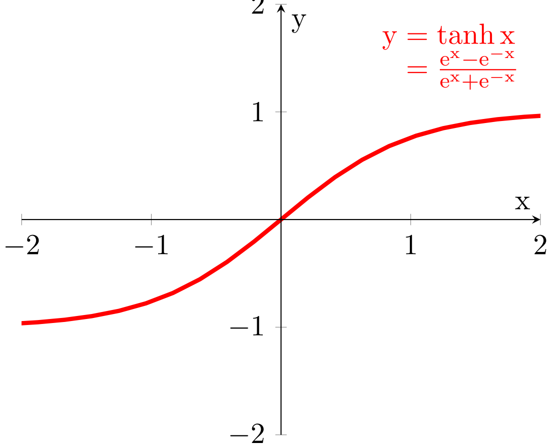
Possible activation functions σ:

- sigmoid



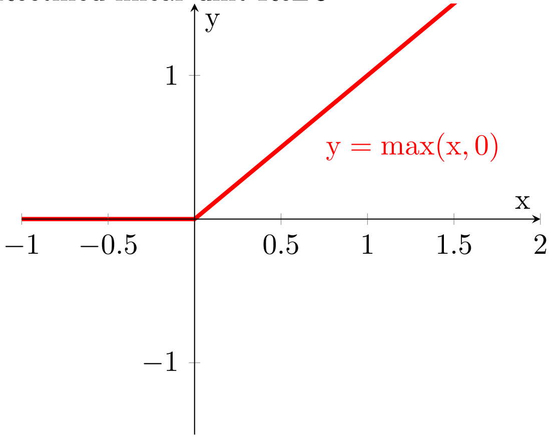
Neural Language Models

- tanh



Neural Language Models

- Rectified linear unit ReLU



Neural Language Models

The softmax function turns a sequence of output activations $z_1, \ldots, z_{n_2}$ into probabilities by

1. mapping every z_i to e^{z_i}, and
2. normalizing, i.e., dividing by the sum of all e^{z_j} for $1 \leq j \leq n_2$

It yields output values $y_1, \ldots, y_{n_2}$ with

$$y_i = \frac{e^{z_i}}{\sum_{j=1}^{n_2} e^{z_j}}$$

for all $1 \leq i \leq n_2$.

Neural Language Models

For a language model, we assume that we have some precompiled vectors for each possible input word, these vectors are all of dimension d. And we assume that $|V|$ is the size of our vocabulary.

Then a neural network that predicts for each sequence of $n-1$ preceding words w_{l-n+1}^{l-1} and for each word $w_l \in V$, the probability $P(w_l | w_{l-n+1}^{l-1})$, can be as follows:

- The input layer takes a concatenation of the vectors of w_{l-n+1}^{l-1}, i.e., it has $d \cdot (n-1)$ components.
- The output layer has $|V|$ components, the ith component is $P(w_l | w_{l-n+1}^{l-1})$ for the case where w_l is the ith element of V.

Neural Language Models

The parameters of the network are learned by training on large amounts of text (all sentences concatenated), starting with some random initialization and then going through several rounds of adjusting the parameters in order to minimize the loss that arises from incorrect predictions of the next word.

References

Chen, Stanley F. & Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. Computer Speech and Language 13. 359–394.

Jurafsky, Daniel & James H. Martin. 2020. Speech and language processing. an introduction to natural language processing, computational linguistics, and speech recognition. Draft of the 3rd edition. Available here: https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf.