Introduction to (n-gram) language models

Statistical Methods in NLP 2
ISCL-BA-08

Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Summer Semester 2025

version: dbfcbc8+ @2025-06-27

What is a language model?

What is a language model?

> *A statistical language model estimates the prior probability values* $P(W)$ *for strings of words* W *in a vocabulary* V *...*
>
> — *Chelba (2010)*

What is a language model?

> *A statistical language model estimates the prior probability values* P(W) *for strings of words* W *in a vocabulary* V ...
>
> — *Chelba* (2010)

> *A language model is a machine learning model that predicts upcoming words. More formally, a language model assigns a probability to each possible next word, or equivalently gives a probability distribution over possible next words.*
>
> — *Jurafsky and Martin* (2025)

Where do we use language models?

- Historically, language models used to have important, but rather limited applicability in NLP, in particular
  - machine translation
  - Speech recognition
  - Spelling correction / predictive text
- With recent success of *neural* language models, the application areas of language models also grew
  - Practically all NLP tasks can be solved with the help of language models

The general formulation

We want to solve two related problems:

- Given a sequence of words $\boldsymbol{w} = (w_1 w_2 \dots w_m)$,

  what is the probability of the sequence $P(\boldsymbol{w})$?

  (machine translation, automatic speech recognition)

- Given a sequence of words $w_1 w_2 \dots w_{m-1}$,

  what is the probability of the next word $P(w_m \mid w_1 \dots w_{m-1})$?

  (predictive text)

Language models in practice: spelling correction

- How would a spell checker know that there is a spelling error in the following sentence?

  I like pizza wit spinach

- Or this one?

  Zoo animals on the lose

Language models in practice: spelling correction

- How would a spell checker know that there is a spelling error in the following sentence?

  I like pizza wit spinach

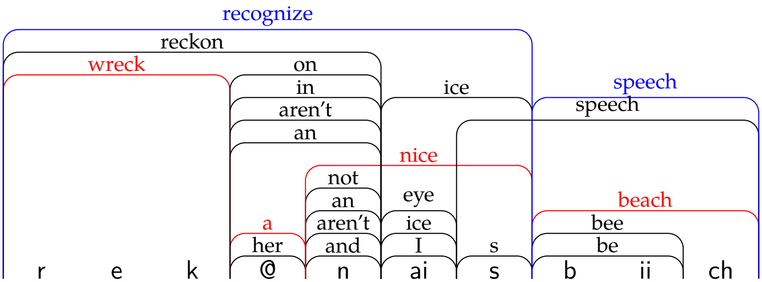
- Or this one?

  Zoo animals on the lose

We want:

$$
\begin{aligned}
P(\text{I like pizza with spinach}) &> P(\text{I like pizza wit spinach}) \\
P(\text{Zoo animals on the loose}) &> P(\text{Zoo animals on the lose})
\end{aligned}
$$

Language models in practice: speech recognition



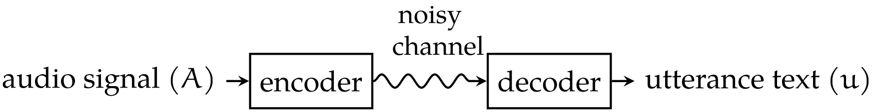
We want:

$$P(\text{recognize speech}) > P(\text{wreck a nice beach})$$

* Reproduced from Shillcock (1995)

Language models in practice: speech recognition

Noisy channel model



- We want $P(u \mid A)$, probability of the utterance given the acoustic signal
- The model of the noisy channel gives us $P(A \mid u)$
- We can use Bayes' formula

$$P(u \mid A) = \frac{P(A \mid u)P(u)}{P(A)}$$

- $P(u)$, probabilities of utterances, come from a language model

Language models in practice: representations & generation

- Language models can be trained without labeled data
- (Most) language models are *generative* models, we can generate text from them
- (Neural) language models build representations for text during language-model training
- Most recent applications of language models are based on representations build during (pre)training

N-gram language models

- An n-gram language model is a model that assign probabilities to sequences based on their parts
- N-gram language models are symbolic, they treat words (tokens) as discrete units
- Until recently, 'language model' meant '*n-gram* language model'
- They are replaced by neural models for almost any application
- Still, most of the concepts about language models is easier to introduce through n-gram models

Assigning probabilities to sentences

count and divide?

How do we calculate the probability of a sentence like
$P(\text{I like pizza with spinach})$

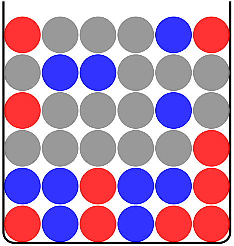
Assigning probabilities to sentences

count and divide?

How do we calculate the probability of a sentence like
P(I like pizza with spinach)

- Can we count the occurrences of the sentence, and divide it by the total number of sentences (in a large corpus)?

P(●) = ?



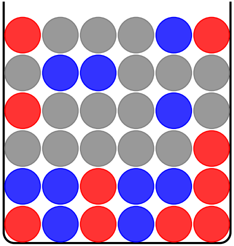
Assigning probabilities to sentences

count and divide?

How do we calculate the probability of a sentence like
P(I like pizza with spinach)

- Can we count the occurrences of the sentence, and divide it by the total number of sentences (in a large corpus)?
- Short answer: No.

P(●) = ?



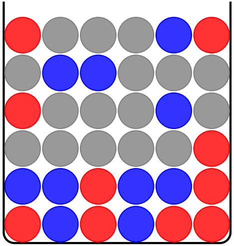
Assigning probabilities to sentences

count and divide?

How do we calculate the probability of a sentence like
P(I like pizza with spinach)

- Can we count the occurrences of the sentence, and divide it by the total number of sentences (in a large corpus)?
- Short answer: No.
  - Many sentences are not observed even in very large corpora

P(●) = ?



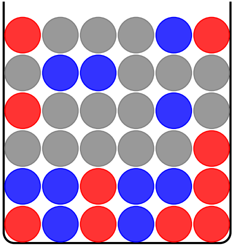
Assigning probabilities to sentences

count and divide?

How do we calculate the probability of a sentence like
P(I like pizza with spinach)

- Can we count the occurrences of the sentence, and divide it by the total number of sentences (in a large corpus)?
- Short answer: No.
  - Many sentences are not observed even in very large corpora
  - For the ones observed in a corpus, probabilities will not reflect our intuitions, or will not be useful in most applications

P(●) = ?



Assigning probabilities to sentences

applying the chain rule

- The solution is to *decompose*

  We use probabilities of parts of the sentence (words) to calculate the probability of the whole sentence

- Using the chain rule of probability (without loss of generality), we can write

$$\begin{aligned} P(w_1, w_2, \ldots, w_m) = \; & P(w_2 \mid w_1) \\ & \times P(w_3 \mid w_1, w_2) \\ & \times \ldots \\ & \times P(w_m \mid w_1, w_2, \ldots w_{m-1}) \end{aligned}$$

Example: applying the chain rule

$$
\begin{aligned}
P(\text{I like pizza with spinach}) = \quad & P(\text{like} \mid \text{I}) \\
\times\ & P(\text{pizza} \mid \text{I like}) \\
\times\ & P(\text{with} \mid \text{I like pizza}) \\
\times\ & P(\text{spinach} \mid \text{I like pizza with})
\end{aligned}
$$

- Did we solve the problem?

Example: applying the chain rule

$$
\begin{aligned}
P(\text{I like pizza with spinach}) = \; & P(\text{like} \mid \text{I}) \\
& \times P(\text{pizza} \mid \text{I like}) \\
& \times P(\text{with} \mid \text{I like pizza}) \\
& \times P(\text{spinach} \mid \text{I like pizza with})
\end{aligned}
$$

- Did we solve the problem?
- Not really, the last term is equally difficult to estimate

Example: bigram probabilities of a sentence

$$
\begin{aligned}
P(\text{I like pizza with spinach}) = \;& P(\text{like} \mid \text{I}) \\
&\times P(\text{pizza} \mid \text{I like}) \\
&\times P(\text{with} \mid \text{I like pizza}) \\
&\times P(\text{spinach} \mid \text{I like pizza with})
\end{aligned}
$$

Example: bigram probabilities of a sentence

with first-order Markov assumption

$$
\begin{aligned}
P(\text{I like pizza with spinach}) = \;& P(\text{like} \mid \text{I}) \\
& \times P(\text{pizza} \mid \text{like}) \\
& \times P(\text{with} \mid \text{pizza}) \\
& \times P(\text{spinach} \mid \text{with})
\end{aligned}
$$

- Now, hopefully, we can count them in a corpus

Maximum-likelihood estimation (MLE)

- The MLE of n-gram probabilities is based on their frequencies in a corpus
- We are interested in conditional probabilities of the form: $P(w_i \mid w_1, \ldots, w_{i-1})$, which we estimate using

$$P(w_i \mid w_{i-n+1}, \ldots, w_{i-1}) = \frac{C(w_{i-n+1} \ldots w_i)}{C(w_{i-n+1} \ldots w_{i-1})}$$

where, $C()$ is the frequency (count) of the sequence in the corpus.

- For example, the probability $P(\text{like} \mid \text{I})$ would be

$$\begin{aligned} P(\text{like} \mid \text{I}) &= \frac{C(\text{I like})}{C(\text{I})} \\ &= \frac{\text{number of times I like occurs in the corpus}}{\text{number of times I occurs in the corpus}} \end{aligned}$$

MLE estimation of an n-gram language model

An n-gram model conditioned on $n - 1$ previous words.

$$\text{unigram} \qquad P(w_i) = \frac{C(w_i)}{N}$$

$$\text{bigram} \qquad P(w_i) = P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$$

$$\text{trigram} \qquad P(w_i) = P(w_i \mid w_{i-2}w_{i-1}) = \frac{C(w_{i-2}w_{i-1}w_i)}{C(w_{i-2}w_{i-1})}$$

Parameters of an n-gram model are these conditional probabilities.

N-gram models define probability distributions

- An n-gram model defines a probability distribution over words

$$\sum_{w \in V} P(w) = 1$$

- They also define probability distributions over word sequences of equal size. For example (length 2),

$$\sum_{w \in V} \sum_{v \in V} P(w)P(v) = 1$$

N-gram models define probability distributions

- An n-gram model defines a probability distribution over words

$$\sum_{w \in V} P(w) = 1$$

- They also define probability distributions over word sequences of equal size. For example (length 2),

$$\sum_{w \in V} \sum_{v \in V} P(w)P(v) = 1$$

- Example: probabilities in sentence *I'm sorry, Dave, I'm afraid I can't do that.*

word	prob
I	0.200
'm	0.133
.	0.133
't	0.067
,	0.067
Dave	0.067
afraid	0.067
can	0.067
do	0.067
sorry	0.067
that	0.067
	1.000

N-gram models define probability distributions

- An n-gram model defines a probability distribution over words

$$\sum_{w \in V} P(w) = 1$$

- They also define probability distributions over word sequences of equal size. For example (length 2),

$$\sum_{w \in V} \sum_{v \in V} P(w)P(v) = 1$$

- Example: probabilities in sentence
I'm sorry, Dave, I'm afraid I can't do that.
- What about sentences?

word	prob
I	0.200
'm	0.133
.	0.133
't	0.067
,	0.067
Dave	0.067
afraid	0.067
can	0.067
do	0.067
sorry	0.067
that	0.067
	1.000

Bigrams

Bigrams are overlapping sequences of two tokens.

I 'm sorry , Dave .

I 'm afraid I can 't do that .

Bigram counts

ngram	freq	ngram	freq	ngram	freq	ngram	freq
I 'm	2	, Dave	1	afraid I	1	n't do	1
'm sorry	1	Dave .	1	I can	1	do that	1
sorry ,	1	'm afraid	1	can 't	1	that .	1

Bigrams

Bigrams are overlapping sequences of two tokens.

I 'm sorry , Dave .

I 'm afraid I can 't do that .

Bigram counts

ngram	freq	ngram	freq	ngram	freq	ngram	freq
I 'm	2	, Dave	1	afraid I	1	n't do	1
'm sorry	1	Dave .	1	I can	1	do that	1
sorry ,	1	'm afraid	1	can 't	1	that .	1

- What about the bigram ' `. I` '?

Sentence boundary markers

If we want sentence probabilities, we need to mark them.

$\langle s \rangle$ I 'm sorry , Dave . $\langle /s \rangle$
$\langle s \rangle$ I 'm afraid I can 't do that . $\langle /s \rangle$

- The bigram ' $\langle$s$\rangle$ I ' is not the same as the unigram ' I '
  Including $\langle s \rangle$ allows us to predict likely words at the beginning of a sentence
- Including $\langle /s \rangle$ allows us to assign a proper probability distribution to sentences

A note on n-gram order

- Larger values for 'n' allows modeling long-range dependencies
- It also requires large amounts of data, otherwise results in overfitting
- It used to be common to use up to 5-gram language models (with additional tricks)
- Increasing n-gram order also increases the number of parameters of the model

How to evaluate (n-gram) language models?

Intrinsic: the higher the probability assigned to a test set better the model. A few measures:

- Likelihood
- (cross) entropy
- perplexity

Extrinsic: improvement of the target application due to the language model:

- Speech recognition accuracy
- BLEU score for machine translation
- Keystroke savings in predictive text applications
- More recently: large benchmark datasets on various tasks (QA, NLI, paraphrasing, summarization, translation, ...)

Likelihood

- Likelihood of a model M is the probability of the (test) set $\boldsymbol{w}$ given the model

$$\mathcal{L}(M \mid \boldsymbol{w}) = P(\boldsymbol{w} \mid M) = \prod_{s \in \boldsymbol{w}} P(s)$$

Likelihood

- Likelihood of a model M is the probability of the (test) set $\boldsymbol{w}$ given the model

$$\mathcal{L}(M \mid \boldsymbol{w}) = P(\boldsymbol{w} \mid M) = \prod_{s \in \boldsymbol{w}} P(s)$$

- The higher the likelihood (for a given test set), the better the model

Likelihood

- Likelihood of a model M is the probability of the (test) set $\boldsymbol{w}$ given the model

$$\mathcal{L}(M \mid \boldsymbol{w}) = P(\boldsymbol{w} \mid M) = \prod_{s \in \boldsymbol{w}} P(s)$$

- The higher the likelihood (for a given test set), the better the model
- Likelihood is sensitive to the test set size

Likelihood

- Likelihood of a model M is the probability of the (test) set $\boldsymbol{w}$ given the model

$$\mathcal{L}(M \mid \boldsymbol{w}) = P(\boldsymbol{w} \mid M) = \prod_{s \in \boldsymbol{w}} P(s)$$

- The higher the likelihood (for a given test set), the better the model
- Likelihood is sensitive to the test set size
- Practical note: (negative) log likelihood is used more commonly, because of ease of numerical manipulation

Cross entropy

- Cross entropy of a language model on a test set $\boldsymbol{w}$ is

$$H(\boldsymbol{w}) = -\frac{1}{N} \sum_{w_i} \log_2 \widehat{P}(w_i)$$

Cross entropy

- Cross entropy of a language model on a test set $\boldsymbol{w}$ is

$$H(\boldsymbol{w}) = -\frac{1}{N}\sum_{w_i} \log_2 \widehat{P}(w_i)$$

- The lower the cross entropy, the better the model

Cross entropy

- Cross entropy of a language model on a test set $\boldsymbol{w}$ is

$$H(\boldsymbol{w}) = -\frac{1}{N} \sum_{w_i} \log_2 \widehat{P}(w_i)$$

- The lower the cross entropy, the better the model
- Cross entropy is not sensitive to the test-set size

Cross entropy

- Cross entropy of a language model on a test set $\boldsymbol{w}$ is

$$H(\boldsymbol{w}) = -\frac{1}{N} \sum_{w_i} \log_2 \widehat{P}(w_i)$$

- The lower the cross entropy, the better the model
- Cross entropy is not sensitive to the test-set size

Cross entropy

- Cross entropy of a language model on a test set $\boldsymbol{w}$ is

$$H(\boldsymbol{w}) = -\frac{1}{N} \sum_{w_i} \log_2 \widehat{P}(w_i)$$

- The lower the cross entropy, the better the model
- Cross entropy is not sensitive to the test-set size

Reminder: Cross entropy is the bits required to encode the data coming from P using another (approximate) distribution $\widehat{P}$.

$$H(P, Q) = -\sum_{x} P(x) \log \widehat{P}(x)$$

Perplexity

- Perplexity is a more common measure for evaluating language models

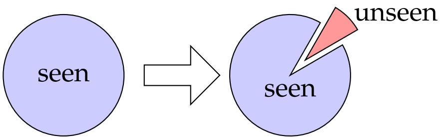
$$\mathrm{PP}(\boldsymbol{w}) = 2^{\mathrm{H}(\boldsymbol{w})} = \mathrm{P}(\boldsymbol{w})^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{\mathrm{P}(\boldsymbol{w})}}$$

- Perplexity is the average branching factor
- Similar to cross entropy
  - lower better
  - not sensitive to test set size

What do we do with unseen n-grams?

...and other issues with MLE estimates

- Words (and word sequences) are distributed according to the Zipf's law: *many words are rare*.
- MLE will assign 0 probabilities to unseen words, and sequences containing unseen words
- Even with non-zero probabilities, MLE *overfits* the training data
- One solution is smoothing: take some probability mass from known words, and assign it to unknown words



Laplace smoothing

(Add-one smoothing)

- The idea (from 1790): add one to all counts
- The probability of a word is estimated by

$$P_{+1}(w) = \frac{C(w)+1}{N+V}$$

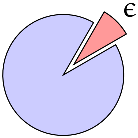
N number of word tokens

V number of word types - the size of the vocabulary

- Then, probability of an unknown word is:

$$\frac{0+1}{N+V}$$

Absolute discounting



- An alternative to the additive smoothing is to reserve an explicit amount of probability mass, ϵ, for the unseen events
- The probabilities of known events has to be re-normalized
- How do we decide what ϵ value to use?

Good-Turing smoothing

- Estimate the probability mass to be reserved for the novel n-grams using the observed n-grams
- Novel events in our training set is the ones that occur once

$$p_0 = \frac{n_1}{n}$$

where n_1 is the number of distinct n-grams with frequency 1 in the training data

- Now we need to discount this mass from the higher counts
- The probability of an n-gram that occurred r times in the corpus is

$$(r+1)\frac{n_{r+1}}{n_r n}$$

Back-off

Back-off uses the estimate if it is available, 'backs off' to the lower order n-gram(s) otherwise:

$$P(w_i \mid w_{i-1}) = \begin{cases} P^*(w_i \mid w_{i-1}) & \text{if } C(w_{i-1}wi) > 0 \\ \alpha P(w_i) & \text{otherwise} \end{cases}$$

where,

- $P^*(\cdot)$ is the discounted probability
- α makes sure that $\sum P(w)$ is the discounted amount
- $P(w_i)$, typically, smoothed unigram probability

Interpolation

Interpolation uses a linear combination:

$$P_{int}(w_i \mid w_{i-1}) = \lambda P(w_i \mid w_{i-1}) + (1 - \lambda) P(w_i)$$

In general (recursive definition),

$$P_{int}(w_i \mid w_{i-n+1}^{i-1}) = \lambda P(w_i \mid w_{i-n+1}^{i-1}) + (1 - \lambda) P_{int}(w_i \mid w_{i-n+2}^{i-1})$$

- $\sum \lambda_i = 1$
- Recursion terminates with
  - either smoothed unigram counts
  - or uniform distribution $\frac{1}{V}$

Some shortcomings of the n-gram language models

The n-gram language models are simple and successful, but ...

- They cannot handle long-distance dependencies:
  `In the last race, the horse he bought last year finally ______.`
- The success often drops in morphologically complex languages
- The smoothing methods are often 'a bag of tricks'
- They are highly sensitive to the training data: you do not want to use an n-gram model trained on business news for medical texts

Some shortcomings of the n-gram language models

The n-gram language models are simple and successful, but …

- They cannot handle long-distance dependencies:
  `In the last race, the horse he bought last year finally _____.`
- The success often drops in morphologically complex languages
- The smoothing methods are often 'a bag of tricks'
- They are highly sensitive to the training data: you do not want to use an n-gram model trained on business news for medical texts

Cluster-based n-grams

- The idea is to cluster the words, and fall-back (back-off or interpolate) to the cluster
- For example,
  - a clustering algorithm is likely to form a cluster containing words for food, e.g., {`apple`, `pear`, `broccoli`, `spinach`}
  - if you have never seen `eat your broccoli`, estimate

$$P(\texttt{broccoli} \mid \texttt{eat your}) = P(\texttt{FOOD} \mid \texttt{eat your}) \times P(\texttt{broccoli} \mid \texttt{FOOD})$$

- Clustering can be

hard a word belongs to only one cluster (simplifies the model)

soft words can be assigned to clusters probabilistically (more flexible)

Skipping

- The contexts
  - `boring | the lecture was`
  - `boring | (the) lecture yesterday was`

  are completely different for an n-gram model
- A potential solution is to consider contexts with gaps, 'skipping' one or more words
- We would, for example model $P(e \mid abcd)$ with a combination (e.g., interpolation) of
  - $P(e \mid abc\_)$
  - $P(e \mid ab\_d)$
  - $P(e \mid a\_cd)$
  - ...

Modeling sentence types

- Another way to improve a language model is to condition on the sentence types
- The idea is different types of sentences (e.g., ones related to different topics) have different behavior
- Sentence types are typically based on clustering
- We create multiple language models, one for each sentence type
- Often a 'general' language model is used, as a fall-back

Caching

- If a word is used in a document, its probability of being used again is high
- Caching models condition the probability of a word, to a larger context (besides the immediate history), such as
  - the words in the document (if document boundaries are marked)
  - a fixed window around the word

Structured language models

- Another possibility is using a generative parser
- Parsers try to explicitly model (good) sentences
- Parsers naturally capture long-distance dependencies
- Parsers require much more computational resources than the n-gram models
- The improvements are often small (if any)

Maximum entropy models

- We can fit a logistic regression 'max-ent' model predicting $P(w \mid \text{context})$
- Main advantage is to be able to condition on arbitrary features

Neural language models

- Similar to maxent models, we can train a feed-forward network that predicts a word from its context
- (gated) Recurrent networks are more suitable to the task:
  - Train a recurrent network to predict the next word in the sequence
  - The hidden representations reflect what is useful in the history
- Neural/RNN language models are generally more successful than n-gram models
- In recent years, language models based on Transformers architecture dominated the field

Summary

- We want to assign probabilities to sentences
- N-gram language models do this by
  - estimating probabilities of parts of the sentence (n-grams)
  - use the n-gram probability and a conditional independence assumption to estimate the probability of the sentence
- MLE estimate for n-gram overfit
- Smoothing is a way to fight overfitting
- Back-off and interpolation yields better 'smoothing'
- There are better ways of building language models
- Reading: Jurafsky and Martin, 2025, Chapter 3

Summary

- We want to assign probabilities to sentences
- N-gram language models do this by
  - estimating probabilities of parts of the sentence (n-grams)
  - use the n-gram probability and a conditional independence assumption to estimate the probability of the sentence
- MLE estimate for n-gram overfit
- Smoothing is a way to fight overfitting
- Back-off and interpolation yields better ‘smoothing’
- There are better ways of building language models
- Reading: Jurafsky and Martin, 2025, Chapter 3

Next:

- Encoder–decoder networks and neural language models

Additional reading, references, credits

Chelba, Ciprian (2010). "Statistical language modeling". In: *The Handbook of Computational Linguistics and Natural Language Processing*. Ed. by Alexander Clark, Chris Fox, and Shalom Lappin. Wiley Online Library, pp. 74–104.

Jurafsky, Daniel and James H. Martin (2025). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 12, 2025. URL: `https://web.stanford.edu/~jurafsky/slp3/`.

Shillcock, Richard (1995). "Lexical Hypotheses in Continuous Speech". In: *Cognitive Models of Speech Processing*. Ed. by Gerry T. M. Altmann. MIT Press.