729G86/TDP030 Language Technology (VT2025)

Language Modelling

Marcel Bollmann

Department of Computer and Information Science (IDA)

LINKÖPING UNIVERSITY

Adapted from slides by Marco Kuhlmann.

Today's lecture

1. Introduction

2. N-gram Models
   - N-grams
   - Markov Assumption
   - Simple Uses

3. Training
   - MLE
   - Smoothing
   - Unknown Words

4. Evaluation
   - Entropy
   - Perplexity

5. Outlook: LLMs

Introduction to Language Modelling

1

Language modelling

- What is the **probability** of a sequence of words?

$$p(\text{“}I\ like\ books\text{”}) > p(\text{“}books\ like\ I\text{”})$$
$$p(\text{“}my\ favorite\ food\ is\ pizza\text{”}) > p(\text{“}my\ favorite\ food\ is\ airplanes\text{”})$$

- What is the **conditional probability** of a word given context?

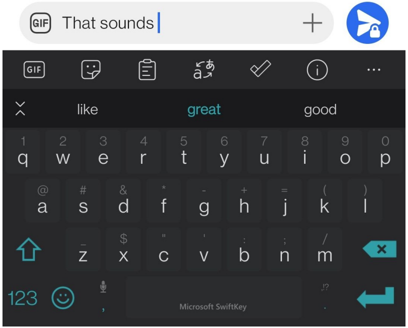
$$p(\text{“}pizza\text{”}|\text{“}my\ favorite\ food\ is\text{”}) = \ ?$$

(These two formulations are equivalent.)

Example: Predictive text input

- Given the words already typed, **predict the most likely next word**.

$$\arg\max_{w \in V} P(w \mid \text{“}\textit{that sounds}\text{”})$$

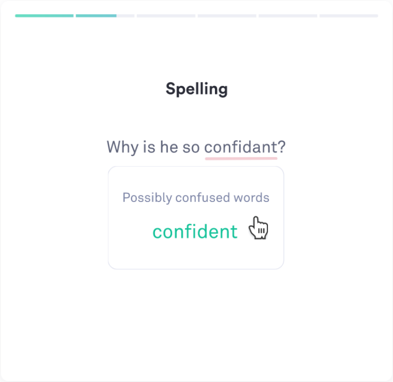


SwiftKey

Example: Spelling correction

- **Spelling mistakes** should result in a low probability.
- If there exists a similar word with much higher probability, we can **suggest spelling corrections**.

$$P(\text{“confident”} \mid \text{“why is he so”})$$
$$> \quad P(\text{“confidant”} \mid \text{“why is he so”})$$



Grammarly

Example: Machine translation

- **Rank candidate translations** for this Chinese sentence:

他	向	记者	介绍了	主要	内容
He	*to*	*reporters*	*introduced*	*main*	*content*

⊗ he introduced reporters to the main contents

⊗ he briefed to reporters the main contents

☛ **he briefed reporters on the main contents**

Example from Jurafsky and Martin (2021)

N-gram Language Models

2

N-grams

Definition

An ***n*-gram** is a sequence of n consecutive words or characters.

- **Unigram**: *that*
- **Bigram**: *that sounds*
- **Trigram**: *that sounds great*
- After that, we would typically speak of 4-grams, 5-grams, etc.

N-gram models

- An ***n*-gram language model** defines a probability distribution over sequences of n words.

$$P(w_1 \cdots w_n)$$

- We often look at the **conditional probability** of seeing the **last word** in an n-gram, given the previous words.

$$P(\boxed{w_n} | w_1 \cdots w_{n-1})$$

- n is also called the **"order"** of the language model.

Unigram models

- A unigram language model is just a **bag-of-words model**.

$$P(w_1 \cdots w_N) = \prod_{i=1}^{N} P(w_i)$$

- Here, the probabilities of each word in a text are **mutually independent**.

Markov models

- **Markov assumption**: word w_n only depends on the $n-1$ previous words

~~I think that~~ sounds ***great***

For $n=2$, the probability of *great* depends only on *sounds*

- To estimate the probability of a longer sentence, we **multiply the conditional probabilities** of subsequent n-grams.

$$P(think|I) \cdot P(that|think) \cdot P(sounds|that) \cdot P(great|sounds)$$

Marking sentence boundaries

BOS I think that sounds great EOS

- For a well-defined model, we also need to **mark the sentence boundaries**.

$$P(I|\underbrace{\boxed{\text{BOS}}}_{\text{beginning-of-sentence}}) \cdot P(think|I) \cdot P(that|think) \cdot P(sounds|that) \cdot P(great|sounds) \cdot P(\underbrace{\boxed{\text{EOS}}}_{\text{end-of-sentence}}|great)$$

Bigram models

- A bigram language model is a **Markov model** on sequences of words.

$$P(w_1 \cdots w_N) = P(w_1 | \textsc{bos}) \cdot \left(\prod_{i=2}^{N} P(w_i | w_{i-1}) \right) \cdot P(\textsc{eos} | w_N)$$

- The probabilities of each word depend only on the **immediately preceding word**.

N-gram models

BOS BOS BOS I think that sounds great EOS

Padded input with $n = 4$

- If we assume that the **input sequence is padded** with $n-1$ BOS tokens, we can define language models for arbitrary n.

$$P(w_1 \cdots w_N) = \prod_{i=n}^{N} P(w_i | \boxed{w_{i-(n-1)} \cdots w_{i-1}})$$

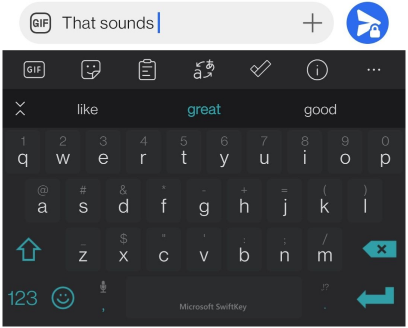
- The probabilities of each word only depend on the $n-1$ **preceding words**.

Using n-gram models for prediction

- To predict the next word, we can **choose the word with the highest probability** among all possible words w:

$$\underset{w \in \boxed{V}}{\arg\max}\; P(w \mid \textit{"that sounds"})$$

vocabulary =
set of all possible words



SwiftKey

Using n-gram models for generation

- We can **generate new text** by **sampling** from the vocabulary.
  - sampling: picking words with probability $P(w|preceding\ words)$

- Without further conditioning, this does not produce meaningful text.

to him at the chamber at his best , and my words brought a newspaper i expect , will reallocate resources , and which analysis alone can bring one of those categories .

Text sampled from a small trigram model

Important concepts

- language model
- unigrams, bigrams, trigrams, *n*-grams
- Markov assumption
- special tokens for marking sentence boundaries (BOS, EOS)

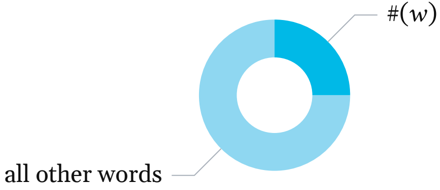
Training n-gram Models

3

MLE for unigram models

- We can use **maximum likelihood estimation (MLE)** to obtain probabilities for n-gram models.

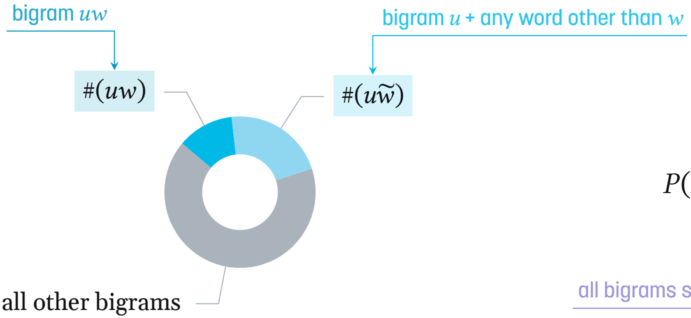
- For unigram models, we can simply **count the words**:



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

MLE for bigram models

- For bigram models, we have to take the **context** into account.



$$P(w|u) = \frac{\#(uw)}{\#(u\bullet)}$$

all bigrams starting with u

MLE for n-gram models

- We can simplify the equation a bit:

$$P(w|u) = \frac{\#(uw)}{\boxed{\#(u\bullet)}} = \frac{\#(uw)}{\boxed{\#(u)}}$$

why?

- We can perform MLE for **arbitrary** n – e.g. for trigrams:

$$P(great|that\ sounds) = \frac{\#(that\ sounds\ great)}{\#(that\ sounds)}$$

A problem with maximum likelihood estimation

- If the bigram "*det sägs*" never occurred in the training data, MLE estimates a **probability of zero**.

$$P(\textit{sägs}|\textit{det}) = \frac{\boxed{\#(\textit{det sägs})}}{\#(\textit{det})}$$

- Under a Markov model, **each sentence containing this bigram** will receive probability zero.

$$P(\textit{det sägs att en blick säger mer än tusen ord}) = 0$$

Sparsity issues in practice

- Shakespeare's collected works contain ca. 31,000 word types.

👉 There are **961 million possible bigrams** with these words.

  – $31000^2 = 961000000$

- Shakespeare's collected works contain ca. 300,000 unique bigrams.

👉 This means that **99.97% of all possible bigrams** have count zero.

Additive smoothing

- We can do **add-k smoothing** as for the Naive Bayes classifier:
  - Unigram with smoothing

$$P(w) = \frac{\#(w) + k}{N + kV}$$

  - Bigram with smoothing

$$P(w|u) = \frac{\#(uw) + k}{\#(u) + kV}$$

A problem with additive smoothing

- We only have a constant amount of probability mass to distribute.
  - Probabilities must always sum up to 1.
- Additive smoothing…
  - **subtracts** probability from **observed *n*-grams**, and
  - **redistributes** it equally among **all possible *n*-grams**.
- Better smoothing techniques for language modelling have been proposed *(but we won't look at them here)*.
  - Witten–Bell smoothing, Kneser–Ney smoothing

Redistributing the probability mass

- Assume the vocabulary:

  {awesome, great, sounds, that}

- Let's estimate a unigram model from a single training sentence:

  that sounds great

- **After smoothing,** each observation **loses ca. 14%** of its original probability mass.

awesome	great	sounds	that
0/3	1/3	1/3	1/3
0.00	0.33	0.33	0.33

awesome	great	sounds	that
1/7	2/7	2/7	2/7
0.14	0.29	0.29	0.29

Unknown words

- Text may contain **out-of-vocabulary** words.

> *One ecological change dams bring to rivers is caused by something called* ***hydropeaking***

Source: New York Times, 02.08.2022

- Smoothing **will not help** with that!
  - It only considers in-vocabulary words.

Handling unknown words

- We can **introduce a new symbol** for "unknown words":

> *One ecological change dams bring to rivers is caused by something called* ***UNK***

- During training, we replace very rare words with UNK.
  - *e.g.* all words occurring only a single time
  - This makes the model learn a distribution for UNK.

- During testing, we replace all out-of-vocabulary words with UNK.

Important concepts

- maximum likelihood estimation *(for n-gram language models)*
- additive smoothing *(for n-gram language models)*
- special token for out-of-vocabulary words (UNK)

Evaluating Language Models

4

Intrinsic and extrinsic evaluation

1. **Intrinsic evaluation:** How does the model score on some **evaluation measure**?
   - In classification, we used accuracy, precision, recall, F1-score.
2. **Extrinsic evaluation:** How much does it help on a **downstream task**?
   - *Such as:* next word prediction, spelling correction, machine translation, ...

- For language modelling, accuracy/precision/recall is not very meaningful.
  - We predict over a vocabulary with thousands of "classes."
  - We wouldn't expect the next word in a sentence to be perfectly predictable.

Intrinsic evaluation of language models

1. **Train a language model** on some training data.
2. Compute the **probability of sentences** in some held-out **test data**.
   - *e.g.* train on Washington Post, test on New York Times

💡 Intuition

If the language model is good, the probability of the test sentences should be high.

From probabilities to surprisal

- Instead of probabilities, we compute **negative log probabilities**.

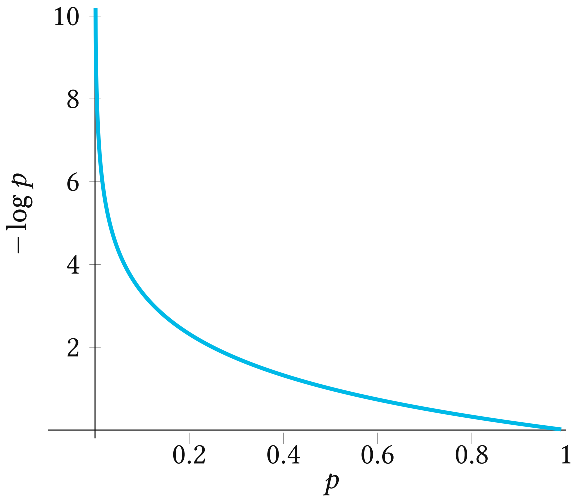
$$-\log P(w_1 \cdots w_N)$$

- Intuitively, this measures how **"surprised"** we are about seeing the test sentences.
  - high probability = low negative log probability = low surprisal

The **entropy** of a language model is its **average surprisal** per word.

$$-\frac{1}{N} \log P(w_1 \cdots w_N)$$

Negative log probabilities



Entropy and perplexity

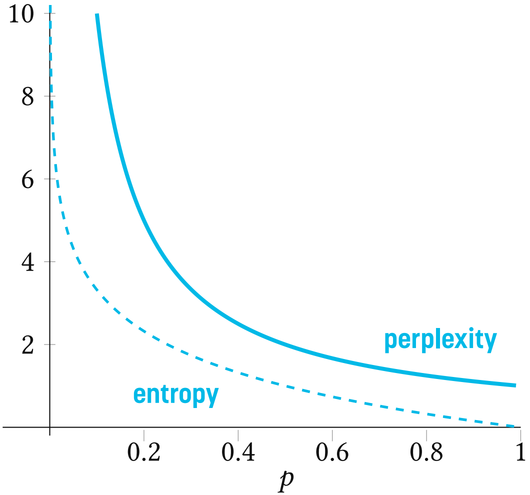
- Language models are more commonly evaluated via **perplexity**:

$$PPL(x) = 2^{\text{entropy}} = 2^{-\frac{1}{N}\log_2 P(w_1 \cdots w_N)}$$

💡 Intuition

The model is as "perplexed" from seeing the test sentences as if it had to **randomly pick between $PPL(x)$ tokens** at every step.

Entropy vs. perplexity



Perplexity values

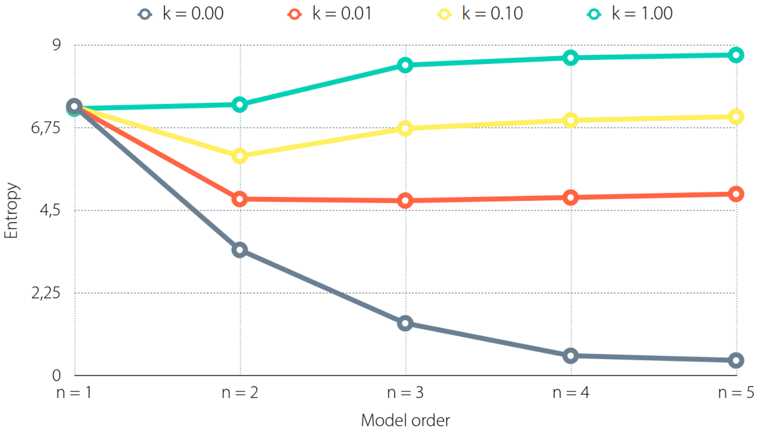
- Typically, perplexity is a **value between 1 and $|V|$.**
  - $PPL(x) = 1 \quad \longrightarrow$ text can be **predicted perfectly**.
  - $PPL(x) = |V| \quad \longrightarrow$ words were **guessed randomly** from the entire vocabulary.
- In practice, the absolute perplexity value **depends on the vocabulary**.
  - larger vocabulary = more words to distribute the probability mass between
- Perplexity values of two language models are *only* directly comparable if they use **identical vocabularies**!

Effects of order and smoothing

Assuming that the test data is similar to the training data:

- A model with **higher order** will have **lower entropy/perplexity**.
  - higher order = longer sequences of words
- A model with **smoothing** will have **higher entropy/perplexity**.
  - Smoothing "takes away" probability mass from the actual observations.

Entropy vs. order and smoothing



Plot by Marco Kuhlmann

Important concepts

- negative log probabilities, entropy
- perplexity *(& how to interpret it)*

Outlook: Large Language Models

5

Large language models

- Applications like ChatGPT or Copilot are built on **large language models**.
- Most current large language models are **next word predictors**.
  - Just like the models we saw in this lecture!
- What makes them so powerful is that they can use a **much larger context window**.
  - n-gram models can look at the previous $(n-1)$ words.
  - GPT-4 Turbo can look at the previous **128,000** words!

Source: OpenAI

Why is language modelling so effective?

1. *Berlin is the capital of **Germany*** — WORLD KNOWLEDGE
2. *Kenya's long-distance runner broke the world **record*** — LEXICAL KNOWLEDGE
3. *I almost fell asleep because this movie was so **boring*** — SENTIMENT
4. *If Alice is Bob's daughter, Bob is Alice's **father*** — SEMANTIC RELATION

- We can model a **variety of knowledge** just from next word prediction!
  - provided the context window is large enough

ChatGPT vs. statistical models

- On a **conceptual level**, ChatGPT uses language modelling in a very similar sense as *n*-gram models.
- On a **technical level**, it uses much more advanced machine learning techniques.

You
I know about simple, statistical n-gram language modeling. In just a few words, what concepts do I need to learn to understand how ChatGPT works?

ChatGPT
To understand how ChatGPT works, you should familiarize yourself with the following concepts:

1. **Transformer Architecture:** ChatGPT is built on the Transformer architecture, which is a type of neural network architecture that excels in processing sequential data.
2. **Attention Mechanism:** Transformers use attention mechanisms to weigh different parts of the input sequence differently when making predictions, allowing them to focus on relevant information.
3. **Self-Attention:** This is a specific type of attention mechanism where the model weighs different positions in the input sequence to compute a representation for each position.
4. **Pre-training and Fine-tuning:** ChatGPT is pre-trained on a massive dataset to learn general language patterns and then fine-tuned on specific tasks or domains to make it more useful for certain applications.

Source: ChatGPT

LINKÖPING
UNIVERSITY