Lecture 3:
N-gram Language Modelling

First-Year Project 4:
Natural Language Processing

Christian Hardmeier

26 April 2022

Language Modelling

- *Language models* learn a probability distribution over sequences of words.

$$p(w_1, w_2, \ldots, w_N)$$

- Encode properties of a certain type of language.
- Distinguish plausible from less plausible word sequences.
- Generate plausible-sounding word sequences.
- Learn generic relations between words.
- Often used as components in other models:
  - ASR: Acoustic model + Language model
  - Statistical MT: Translation model + Language model

Chain rule of probability

Star	light	,	star	bright	,	first	star	I	see	tonight	!

$$\begin{aligned} p(w_1, w_2, \ldots, w_N) &= p(w_1) \cdot p(w_2|w_1) \cdot p(w_3|w_2, w_1) \cdot \\ &\quad p(w_4|w_3, w_2, w_1) \cdots p(w_N|w_{N-1}, \ldots, w_2, w_1) \\ &= p(w_1) \prod_{i=1}^{N} p(w_i|w_1, \ldots, w_{i-1}) \end{aligned}$$

- We need to estimate values for all these probabilities, for all possible instantiations of $w_1, \ldots, w_N$.
- That's a lot of parameters!

Independence assumptions

- Recall that natural language has strong *local* dependencies.
  - *the* strongly favours a following noun (or adjective)
  - After a full stop, we're likely to see a word starting with a capital letter.
- *Long-range dependencies* do exist and are important, but not as strong (and more difficult to model).
- We can make *independence assumptions* to simplify the model.

Markov assumption

- **Markov assumption:**
  Each element of the sequence depends only on the immediately preceding element and is *independent* of the previous history.

$$p(w_i|w_1, \ldots, w_{i-1}) \approx p(w_i|w_{i-1})$$

- **k-th order Markov assumption:**
  Each element of the sequence depends only on the k immediately preceding elements.

$$p(w_i|w_1, \ldots, w_{i-1}) \approx p(w_i|w_{i-k}, \ldots, w_{i-1})$$

- Note: These are *approximations*!

2nd order Markov assumption

Star light , star bright , first star I see tonight !

$$\begin{aligned} p(w_1, w_2, \ldots, w_N) \approx\ & p(w_1) \cdot p(w_2|w_1) \cdot p(w_3|w_2, w_1) \cdot \\ & p(w_4|w_3, w_2) \cdots p(w_N|w_{N-2}, w_{N-1}) \\ =\ & p(w_1) \prod_{i=1}^{N} p(w_i|w_{i-2}, w_{i-1}) \end{aligned}$$

Model size

- Let V be the vocabulary size and
  N be the maximum sentence length.
- Each w_i can be any vocabulary item $\rightarrow V$ choices.
- For a model *without* independence assumptions,
  - we need to estimate $p(w_N | w_1, w_2, \ldots, w_{N-1})$.
  - up to V^N model parameters
- For a k-th order Markov model,
  - we need to estimate $p(w_{k+1} | w_1, \ldots, w_k)$.
  - up to V^{k+1} model parameters
- In a realistic language model,
  - $V \approx 10^4$ to 10^5
  - $N \approx 30$ to 80
  - $k \approx 2$ to 5

Sequence padding

- Is this a good complete sentence???

  Star light , star bright , first star I
- Is this a good start of a sentence???

  , star bright ,
- Add special symbols to mark the start and end of each sentence!
  - ⟨s⟩ or BOS for *beginning of sentence*
  - ⟨/s⟩ or EOS for *end of sentence*

⟨s⟩ Star light , star bright , first star I see tonight ! ⟨/s⟩

N-gram language model

- N-gram models are Markov models for language modelling.
- N-gram: sequence of n tokens in a text.
- 1-gram = unigram; 2-gram = bigram; 3-gram = trigram

⟨s⟩ Star light , star bright , first star I see tonight ! ⟨/s⟩
⟨s⟩ Star light
Star light ,
light , star
, star bright
star bright ,
bright , first
, first star
first star I
star I see
. . .

Scoring sentences

- N-gram models estimate *probabilities*.
- One small factor per token in the sequence:
  - Numbers become *really* small very quickly.
  - Numerical precision suffers the closer you get to zero.
  - At some point, your score will get rounded to zero.
- Use *log-probabilities* instead!
  - Much better numerical stability.
  - Multiplication becomes addition.
  - Do this *whenever* you use probabilities!

Perplexity

- To compare language models, it's also common to use *perplexity*:

$$PPL = p(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}}$$

- Perplexity is an indication of how "confused" the language model is: How many continuations does it consider plausible on average per step?
- *Lower* perplexity is better!
- *Important note:* Perplexity values are only comparable if they refer to the **same vocabulary**!

Parameter Estimation/ Model Training

Maximum-likelihood estimation

Simplest method to estimate a conditional probability:
Count how often the target event occurs
in the context conditioned on.

$$p(w_3|w_1, w_2) = \frac{\#\mathrm{tokens}(w_1 w_2 w_3)}{\#\mathrm{tokens}(w_1 w_2 \bullet)}$$

Example:

⟨s⟩ Star light , star bright , first star I see tonight ! ⟨/s⟩

$$p(\text{bright}|\text{star}) = \frac{\#\mathrm{tokens}(\text{star bright})}{\#\mathrm{tokens}(\text{star } \bullet)} = \frac{1}{2} = 0.5$$

Problems with maximum likelihood estimation

Any token that has not been seen in a particular context will have a count of 0, and therefore a probability of zero.

⟨s⟩ I wish I might have the wish I wish tonight ! ⟨/s⟩

$$p(\text{I}|\text{wish}) = \frac{\#\mathrm{tokens}(\text{wish I})}{\#\mathrm{tokens}(\text{wish } \bullet)} = \frac{2}{3} = 0.667$$

Now score this (assuming a bigram model):

I wish *you* might have the wish *you* wish tonight !

$$p(w_1, \ldots, w_N) = p(w_1) \prod_{i=1}^{N} p(w_i|w_{i-1})$$

Problems with maximum likelihood estimation

Any token that has not been seen in a particular context will have a count of 0, and therefore a probability of zero.

⟨s⟩ I wish I might have the wish I wish tonight ! ⟨/s⟩

$$p(\text{I}|\text{wish}) = \frac{\#\mathrm{tokens}(\text{wish I})}{\#\mathrm{tokens}(\text{wish } \bullet)} = \frac{2}{3} = 0.667$$

Now score this (assuming a bigram model):

I wish *you* might have the wish *you* wish tonight !

$$p(\text{you}|\text{wish}) = \frac{\#\mathrm{tokens}(\text{wish you})}{\#\mathrm{tokens}(\text{wish } \bullet)} = \frac{0}{3} = 0$$

Problems with maximum likelihood estimation

- MLE *underestimates* the probability of n-grams not seen in the training data.
- MLE *overestimates* the probability of n-grams seen only a few times.

$$p(\text{the}|\text{have}) = \frac{\#\text{tokens}(\text{have the})}{\#\text{tokens}(\text{have } \bullet)} = \frac{1}{1} = 1$$

- We get good estimates of very frequent tokens.
- But Zipf's law says *most tokens are **not** frequent*!

Add-one estimate (Laplace smoothing)

- Add one to each count to avoid zero counts.
- If we add one to the count of each word in the nominator, we need to add $|V|$ (the vocabulary size) to the denominator:

$$\begin{aligned} p(w_3|w_1,w_2) &= \frac{\#\text{tokens}(w_1w_2w_3)+1}{\sum_{w\in V}(\#\text{tokens}(w_1w_2w)+1)} \\ &= \frac{\#\text{tokens}(w_1w_2w_3)+1}{\#\text{tokens}(w_1w_2\bullet)+|V|} \end{aligned}$$

- The probability of seeing *something new* after the context w_1w_2 is

$$p(\text{new}|w_1,w_2) = \frac{|V| - \#\text{types}(w_1w_2\bullet)}{\#\text{tokens}(w_1w_2\bullet)+|V|}$$

Too much probability mass is moved

Estimated bigram frequencies from AP data

- 22M tokens training
- 22M tokens test

Add-one smoothing significantly overestimates unseen events.

$r = f_{MLE}$	f_{emp}	$f_{add\text{-}1}$
0	0.000027	0.000137
1	0.448	0.000274
2	1.25	0.000411
3	2.24	0.000548
4	3.23	0.000685
5	4.21	0.000822
6	5.23	0.000959
7	6.21	0.00109
8	7.21	0.00123
9	8.26	0.00137

http://www.cs.cornell.edu/courses/cs6740/2008fa/lectures/smoothing2+backoff.pdf
Data from Church and Gale, *Computer Speech and Language* 5 (1991) 19–54

Smoothing

Various *smoothing* methods produce better estimates of language model probabilities.

General principle:

- Take away probability mass from the n-grams we *have* seen by *discounting* their estimates.
- Assign this probability mass to n-grams we *have not* seen.
- The total probability still sums to 1 *for each context*!

$$\sum_{\bullet} p(\bullet|w_1, \ldots, w_k) = 1$$

Backoff models

What to do when an actual unseen event occurs?

- Discounting sets aside probability mass for unseen event, but this is for the *totality* of such events, not for an *individual* one.
- We may not have enough data for a good trigram model but perhaps it's enough for a bigram model?
- A **backoff model** uses a lower-order n-gram whenever the higher order isn't available.

  I wish I may $\rightarrow$ wish I may $\rightarrow$ I may $\rightarrow$ may

- If we haven't even seen the unigram, we assume a uniform distribution.

Interpolated models

- An **interpolated model** always uses lower-order n-grams and combines them with the higher-order estimates.

  $$p'(w_3|w_1, w_2) = \lambda_1 p(w_3|w_1, w_2) + \lambda_2 p(w_3|w_2) + \lambda_3 p(w_3)$$

  The λs must sum to one.
- Recursive formulation:

  $$p'(w_{k+1}|w_1, \ldots, w_k) = \lambda\, p_{\text{ML}}(w_{k+1}|w_1, \ldots, w_k) + (1-\lambda)\, p'(w_{k+1}|w_2, \ldots, w_k)$$

  Note: λ will be different for each $w_{k+1}|w_1, \ldots, w_k$.
- λ can be seen as the probability of choosing between the higher-order model and the backoff distribution.

Witten-Bell smoothing

- Witten-Bell smoothing treats "seeing a new word" as an event in its own right, so we can model its probability explicitly.
- In training, the "new word" event occurs as many times as we have different words.

$$p(\text{new}|w_1, w_2) = \frac{\#\text{types}(w_1 w_2 \bullet)}{\#\text{tokens}(w_1 w_2 \bullet) + \#\text{types}(w_1 w_2 \bullet)}$$

- In the interpolated model, this probability corresponds to the weight $(1 - \lambda)$:

$$\begin{aligned} p'(w_{k+1}|w_1, \dots, w_k) &= \lambda\, p_{\text{ML}}(w_{k+1}|w_1, \dots, w_k) \\ &\quad + (1 - \lambda)\, p'(w_{k+1}|w_2, \dots, w_k) \end{aligned}$$

Absolute discounting

- Subtract a constant discount $(0 < d < 1)$ from the nominator of the counts:

$$p(w_3|w_1, w_2) = \frac{\#\text{tokens}(w_1 w_2 w_3) - d}{\#\text{tokens}(w_1 w_2 \bullet)}$$

- This will have a large effect on small counts, but a small effect on large counts.
- d can be estimated, e.g. from held-out data.

Absolute discounting

$r = f_{\text{MLE}}$	f_{emp}	$f_{\text{add-1}}$
0	0.000027	0.000137
1	0.448	0.000274
2	1.25	0.000411
3	2.24	0.000548
4	3.23	0.000685
5	4.21	0.000822
6	5.23	0.000959
7	6.21	0.00109
8	7.21	0.00123
9	8.26	0.00137

`http://www.cs.cornell.edu/courses/cs6740/2008fa/lectures/smoothing2+backoff.pdf`
Data from Church and Gale, *Computer Speech and Language* 5 (1991) 19–54

Kneser-Ney smoothing

I can't see without my reading ________

- The continuation *glasses* is far more likely than *Kong*.
- But in an English new corpus, *Kong* is more frequent than *glasses*.
- *Kong* only occurs in specific contexts (mostly *Hong Kong*).
  - We only expect to see *Kong* in a bigram we know.
  - We *don't* expect *Kong* to occur in a context we don't know.
  - Contexts we don't know correspond to *backoff situations*.

(Improved) Kneser-Ney smoothing

- Kneser-Ney smoothing uses different distributions for the *higher-order* and the *backoff* distributions.
- For the higher-order distribution, it uses absolute discounting.
  - Discounts estimated separately for counts 1 and 2.
- Backoff distributions are estimated based on the *number of contexts* a word occurs in:

$$p_{\text{cont}}(w) = \frac{\#\text{types}(\bullet w)}{\#\text{types}(\bullet \bullet)}$$

Smoothing methods

- **Laplace smoothing**
  - Avoids 0 probabilities, very easy to implement.
  - Performs worse than other methods.
- **(Improved) Kneser-Ney smoothing**
  - One of the best-performing smoothing methods for natural language.
  - Based on absolute discounting, with clever handling of backoff distribution.
  - Makes specific assumptions about the distribution of infrequent tokens that work well for natural language.
  - For sequences with few infrequent tokens, estimation may fail!
- **Witten-Bell smoothing**
  - Good method for sequences that don't meet the Kneser-Ney assumptions.
  - Uses the number of different continuations of an n-gram to estimate how likely yet another new continuation will be.

N-gram modelling tools

- NLTK
  - The n-gram library in `nltk` is a teaching tool.
  - You would *not* use it for real projects.
- KenLM – `https://kheafield.com/code/kenlm/`
  - Very fast and scalable implementation.
  - Only supports one smoothing method (Kneser-Ney).
  - Free software.
- SRILM – `http://www.speech.sri.com/projects/srilm/`
  - Very complete and well-documented package.
  - Supports many different methods and options.
  - Non-free, free of charge for many non-profit use cases.
  - Commercial use costs money.

Exercises

- Experimentation with n-gram models.
- Train models for
  - different domains (Tweets and News),
  - different n-gram orders, and
  - different smoothing techniques.

IT UNIVERSITY OF COPENHAGEN