CS918: LECTURE 4

Advanced Language Models: Generalisation and Zeros

Arkaitz Zubiaga, 17th October, 2018

LECTURE 4: CONTENTS

WARWICK

- Generalisation of Language Models and Zeros
- Smoothing Approaches for Generalisation
  - Laplace smoothing.
  - Interpolation and backoff.
  - Good Turing Smoothing.
  - Kneser-Ney Smoothing.

ESTIMATING BIGRAM PROBABILITIES

WARWICK

- **Maximum Likelihood Estimate (MLE):**

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

e.g.

$$P(the \mid in) = \frac{count(\text{"in the"})}{count(\text{"in"})}$$

GENERALISATION OF LANGUAGE MODELS

- **Language models:** P(I, am, fine) → high
  P(am, fine, I) → low or 0

- **Remember:** language models will work best on similarly looking data.

- If we train on social media data, that may not work for novels.
  - OMG I'm ROTFL! → may be frequent in SM, unlikely in novels!

GENERALISATION OF LANGUAGE MODELS

WARWICK

- **Limitation:** We're assuming that all n-grams in new, unseen data will have been observed in the training data.
    - Is this the reality though?

- We need to **consider the possibility of new n-grams** in unseen data.

THE SHANNON VISUALISATION METHOD

WARWICK

- Choose a random starting bigram
- (**<s>**, w) based on probability.
- While x != </s>:
  - Choose next random bigram (**w**, x) based on probability.
- Concatenate all bigrams.

```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food  </s>
I want to eat Chinese food
```

SHANNON VIS. METHOD FOR SHAKESPEARE

WARWICK

Unigram

To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
Every enter now severally so, let
Hill he late speaks; or! a more to leg less first you enter
Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like

Bigram

What means, sir. I confess she? then all sorts, he is trim, captain.
Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?

Trigram

Sweet prince, Falstaff shall die. Harry of Monmouth's grave.
This shall forbid it should be branded, if renown made it empty.
Indeed the duke; and had a very good friend.
Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

Quadrigram

King Henry.What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
Will you not tell me who I am?
It cannot be but so.
Indeed the short and the long. Marry, 'tis a noble Lepidus.

SHANNON VIS. METHOD FOR SHAKESPEARE

- Significant **improvement** as we train **longer n-grams**.

- But it’s a tradeoff:
  - Longer n-grams → more accurate.
  - Longer n-grams → more sparse.

- We need to find a balance.

SHAKESPEARE'S BIGRAMS

- Shakespeare used:
  - N = 884,647 tokens.
  - V = 29,066 types → $V^2 \approx$ 845M possible bigrams!

- Shakespeare only produced ~300,000 bigram types (**0.04% of all possible bigrams!**)
  - Other bigrams may be possible, but we haven't observed them.

ZEROS

- **Training set:**
  - … found a penny
  - … found a solution
  - … found a tenner
  - … found a book
- **Test set:**
  - … found a fiver
  - … found a card

P(card | found a) = 0

THE INTUITION OF SMOOTHING

WARWICK

- We have sparse statistics:

  P(w | "found a")

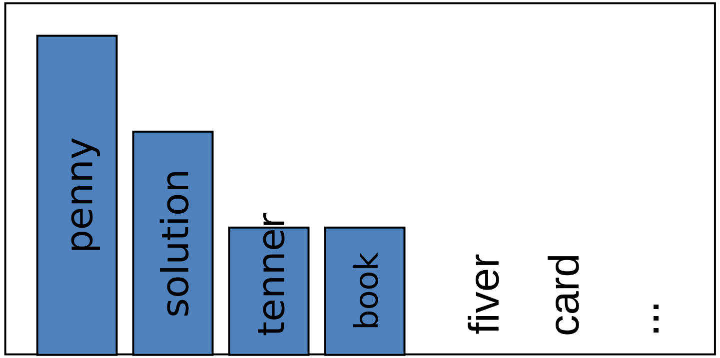
  3 → penny

  2 → solution

  1 → tenner

  1 → book

  7 → total



THE INTUITION OF SMOOTHING

WARWICK

- We'd like to improve the distribution:

  P(w | "found a")

  3 → penny → 2.5

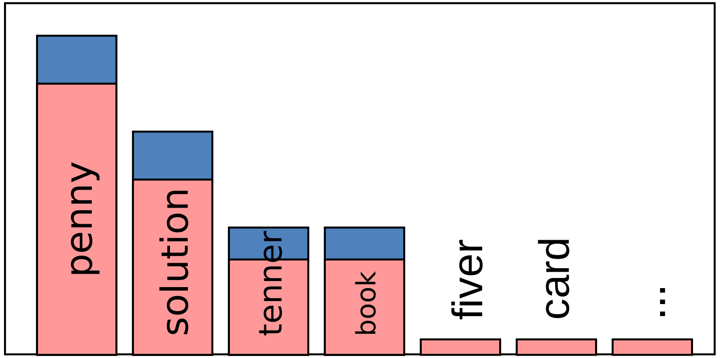
  2 → solution → 1.5

  1 → tenner → 0.5

  1 → book → 0.5

  other → 2

  7 → total



FOUR POSSIBLE SOLUTIONS

- We'll see 4 possible solutions:
  1) Laplace smoothing.
  2) Interpolation and backoff.
  3) Good Turing Smoothing.
  4) Kneser-Ney Smoothing.

LAPLACE SMOOTHING

LAPLACE SMOOTHING: ADD-ONE ESTIMATION

- **Intuition:**

  Pretend we have seen **each word one more time** than we actually did.

LAPLACE SMOOTHING: ADD-ONE ESTIMATION

WARWICK

- Pretend we have seen **each word one more time** than we actually did.

- MLE estimate: $$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- Add-one estimate: $$P_{Add\text{-}1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

SAMPLE OF BIGRAM COUNTS

WARWICK

- Remember our sample of bigrams counted in a corpus of 9,222 sentences.

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

“i” is followed by another “i” on 5 occasions

“to” is followed by “eat” on 686 occasions

LAPLACE SMOOTHED BIGRAM COUNTS

- Add 1 to ALL values in the matrix.

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

LAPLACE SMOOTHED BIGRAM PROBABILITIES

WARWICK

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

RECONSTITUTED COUNTS

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

COMPARING ORIGINAL VS SMOOTHED

WARWICK

- Original:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

- Smoothed:

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

COMPARING ORIGINAL VS SMOOTHED

- Original:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

dividing by 10!

- Smoothed:

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	[illegible]	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

WARWICK

MORE GENERAL LAPLACE SMOOTHING: ADD-k

$$P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

LAPLACE SMOOTHING IS NOT IDEAL FOR LMs

- It’s generally **not the best solution for language models**.
  - With such a sparse matrix, we’re **replacing too many 0’s with 1’s**.

- It’s still **often used for NLP**, particularly when we observe fewer 0’s.

INTERPOLATION AND BACKOFF

WARWICK

- Different words may need different context size captured.

  e.g. I may identify that:

  ”**United States**” needs to be captured as a **bigram**…

  …but…

  ”**look forward to**” is better captured as a **trigram**.

BACKOFF AND INTERPOLATION

- **Backoff:**
  - use trigram if it's very common,
  - otherwise bigram, otherwise unigram

Difficult to implement, define "very common"

- **Interpolation:**
  - mix all unigram, bigram, trigram

LINEAR INTERPOLATION

- Simple interpolation: $$\begin{aligned}\hat{P}(w_n|w_{n-1}w_{n-2}) &= \lambda_1 P(w_n|w_{n-1}w_{n-2}) \\ &+\lambda_2 P(w_n|w_{n-1}) \\ &+\lambda_3 P(w_n)\end{aligned}$$ $$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

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

HOW DO WE SET THOSE LAMBDAS?

- Use a held-out (or development) corpus:

Training Data | Held-Out Data | Test Data

- Fix the N-gram probabilities (on the training data)
- Then search for λs that give largest probability (or lowest perplexity) to held-out set:

$$\log P(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

OPEN VS CLOSED VOCABULARY TASKS

- Two scenarios when processing held out or test data:

  1) We know ALL words in advance.

  - i.e. all words were also in training data.
  - We have a fixed vocabulary V – **closed vocab. Task**

  2) We **may find new, unseen words** (generally the case)

  - **Out Of Vocabulary (OOV)** words – **open vocab. task**

HOW DO WE DEAL WITH OOV WORDS?

WARWICK

- We can use a token for all unknown words: **<OOV>**
- How to train:
  - Create a **fixed lexicon L** of size V.
  - While preprocessing text, change **words not in L to <OOV>**
  - Train the LM, where <OOV> is just another token.

- In test: **new words are assigned the probability of <OOV>**.

GOOD TURING SMOOTHING

INTUITION OF GOOD TURING SMOOTHING

- To estimate counts of **unseen things**

  → use the **count of things we've seen once**

FREQUENCY OF FREQUENCY "C"

- N_c = count of tokens observed C times
- e.g. I am happy and I am fine and I have to go

I → 3
am → 2
and → 2
happy → 1
fine → 1
have → 1
to → 1
go → 1

$$N_1 = 5$$
$$N_2 = 2$$
$$N_3 = 1$$

INTUITION OF GOOD TURING SMOOTHING

- You are fishing, and caught:
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish

- How likely is it that next fish is trout?
  - 1/18
    → easy, we have the probability directly

INTUITION OF GOOD TURING SMOOTHING

- You are fishing, and caught:
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish

- How **likely** is it that next **fish is new**, e.g. haddock?

INTUITION OF GOOD TURING SMOOTHING

- You are fishing, and caught:
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish

- How **likely** is it that next **fish is new**, e.g. haddock?
  - Use our **count of things-we-saw-once**

    3 elements seen once (trout, salmon, eel) → 3/18

INTUITION OF GOOD TURING SMOOTHING

WARWICK

- You are fishing, and caught:
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish

- How **likely** is it that next **fish is new**, e.g. haddock?
  - 3/18
    → OK, but **probabilities won't add up to 1** now!

    we need to revise probabilities.

GOOD TURING CALCULATIONS

- You are fishing, and caught:
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish

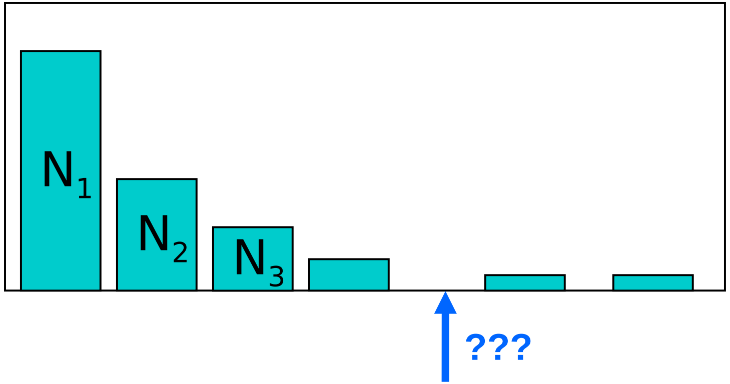
$$\longrightarrow P^*_{GT}(c=0) = \frac{N_1}{N} \longrightarrow 3/18$$

c*

$$\longrightarrow P^*_{GT}(c>0) = \frac{c^*}{N}$$, where: $$c^* = \frac{(c+1)N_{c+1}}{N_c} \longrightarrow P^*_{GT}(c=1) = \frac{2*N2/N1}{N} = \frac{2*1/3}{18} = \frac{1}{27}$$

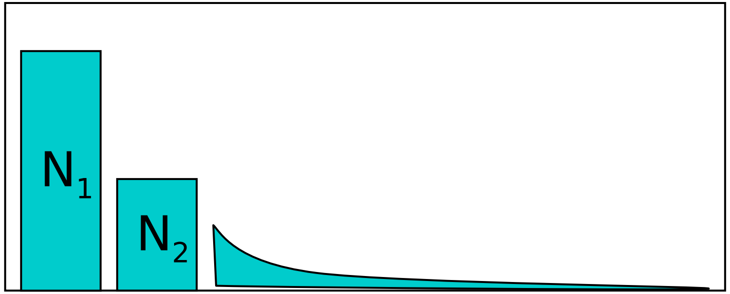
GOOD TURING ISSUES

- **Problem:** what about the word "and"? (say c=12156)
  - For small c, $N_c > N_{c+1}$
  - For large c? N_{12157} will likely be zero.



GOOD TURING ISSUES

- **Solution:** to avoid zeros for large values of c.
  - Simple Good-Turing [Gale and Sampson]: replace $N_c \rightarrow$ best-fit power law for unreliable counts



GOOD TURING NUMBERS

- Corpus with 22 million words of AP Newswire

$$c^* = \frac{(c+1)N_{c+1}}{N_c}$$

Count c	Good Turing c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	5.19
7	6.21
8	7.24
9	8.25

i.e. for every item observed c times, pretend we've seen it c* times

GOOD TURING NUMBERS

WARWICK

- Corpus with 22 million words of AP Newswire

$$c^* = \frac{(c+1)N_{c+1}}{N_c}$$

Count c	Good Turing c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	5.19
7	6.21
8	7.24
9	8.25

- If you look at it, for c >= 2:
  - $c^* \approx (c - .75)$
  - Can we avoid the hassle of doing all Turing calculations?

KNESER-NEY SMOOTHING

ABSOLUTE DISCOUNTING INTERPOLATION

- Save ourselves some time and just subtract 0.75 (or some d)!

discounted bigram

Interpolation weight

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w)$$

unigram

we'll come back to this

KNESER-NEY SMOOTHING

- We need to predict the next word:
    - I can't see without my reading ________?

    - BUT: we have never seen "reading X" in training data.
        - Backoff, rely only on unigrams?

- Now we need to predict the next word:
  - I can't see without my reading ________?
  - If we only look at unigrams, P("Kingdom") > P("glasses").
    - Is it "reading Kingdom" then?

KNESER-NEY SMOOTHING

- Now we need to predict the next word:
  - I can't see without my reading ________?

  - **Intuition of Kneser-Ney smoothing:**
    "Kingdom" ALWAYS comes after "United"
    why would it ever come after "reading"?

KNESER-NEY SMOOTHING

- Instead of P(w): “How likely is w”
  - $P_{continuation}(w)$: “How likely is w to appear as a novel continuation?”

- For each unigram, **count the number of bigram types it completes** → i.e. always paired with a specific word, or likely in many bigrams?

KNESER-NEY SMOOTHING

- How many times does w appear as a novel continuation:

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

- Normalized by the total number of bigram types

$$P_{CONTINUATION}(w) = \frac{\left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|}{\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|}$$

KNESER-NEY SMOOTHING

- In other words, $P_{continuation}$ is:
  - The number of # of word types seen to precede w,
  - normalised by the # of words preceding all words.

- I am
  We are
  You are
  They are
  She is

KNESER-NEY SMOOTHING

WARWICK

- In other words, $P_{continuation}$ is:
  - The number of # of word types seen to precede w,
  - normalised by the # of words preceding all words.

- I am
  We are
  You are ⟶ $P_{continuation}(\text{"are"}) = 3/5$
  They are
  She is

KNESER-NEY SMOOTHING

- In other words, $P_{continuation}$ is:
  - The number of # of word types seen to precede w,
  - normalised by the # of words preceding all words.

- I am
  We are
  You are → $P_{continuation}$("am") = 1/5
  They are
  She is

KNESER-NEY SMOOTHING

- In other words, $P_{continuation}$ is:
  - The number of # of word types seen to precede w,
  - normalised by the # of words preceding all words.

- I am
  We are
  You are → $P_{continuation}$(“is”) = 1/5
  They are
  She is

KNESER-NEY SMOOTHING

WARWICK

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i)$$

λ is a normalising constant; the probability mass we've discounted

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{ w : c(w_{i-1}, w) > 0 \} \right|$$

the normalised discount

The number of word types that can follow w_{i-1}
= # of word types we discounted
= # of times we applied normalised discount

RESOURCES

WARWICK

- KenLM Language Model Toolkit: https://kheafield.com/code/kenlm/
- CMU Statistical Language Modeling Toolkit: http://www.speech.cs.cmu.edu/SLM/toolkit.html
- Google Books N-gram Counts: http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

ASSOCIATED READING

- Jurafsky, Daniel, and James H. Martin. 2009. Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. 3rd edition. **Chapters 3.3-3.6.**