AdvNLP/E Lecture 3

Language Modelling 1

Dr Julie Weeds, Spring 2024

Language models

PREVIOUSLY

- Lexical and distributional semantics
  - semantic relationships
  - WordNet
  - distributional hypothesis
  - vector spaces and word representations
  - sparsity and Zipf's Law
  - dimensionality reduction
  - word embeddings

THIS TIME

- Probabilistic language models
  - n-gram modelling
  - evaluation and perplexity
  - generation
  - generalization and smoothing

N-gram models

Lecture 2, Part 1

Why do we want to be able to assign a probability to a sentence?

- Machine translation

  P(high winds tonight) > P(large winds tonight)

- Spelling correction

  P(The office is about 15 **minutes** from my house)
  > P(The office is about 15 **minuets** from my house)

- Speech recognition

  P(I saw a van) > P(eyes awe of an)

Probabilistic language modelling

- Goal: compute the probability of a sentence of sequence of words

$$P(W) = P(w_1, w_2, \dots , w_n)$$

- Related task: probability of an upcoming word

$$P(w_5 | w_1, w_2, w_3, w_4)$$

- A model that computes either of these is called a **language model (LM)**

The Chain Rule for Probabilities

- The definition of conditional probabilities give us:

$$P(B|A) = \frac{P(A,B)}{P(A)} \Rightarrow P(A,B) = P(A)P(B|A)$$

- More variables

$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$$

- The general case (chain rule):

$$P(x_1, x_2, x_3, \ldots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \ldots P(x_n|x_1, \ldots, x_{n-1})$$

Applying the chain rule to words

$$P(w_1, w_2, w_3, \ldots, w_k) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \ldots P(w_k|w_1, \ldots, w_{k-1})$$

$$\begin{aligned}
&P(\text{“Then he hovered over the hill”}) \\
&= P(\text{“Then”}) \times P(\text{“he”} \,|\, \text{“Then”}) \times P(\text{“hovered”} \,|\, \text{“Then he”}) \\
&\times P(\text{“over”} \,|\, \text{“Then he hovered”}) \\
&\times P(\text{“the”} \,|\, \text{“Then he hovered over”}) \\
&\times P(\text{“hill”} \,|\, \text{“Then he hovered over the”})
\end{aligned}$$

Estimating probabilities

- Can we just count and divide?

$$P(\text{“}hill\text{”} \mid \text{“}Then\ he\ hovered\ over\ the\text{”}) = \frac{freq(\text{“}Then\ he\ hovered\ over\ the\ hill\text{”})}{freq(\text{“}Then\ he\ hovered\ over\ the\text{”})}$$

Markov Assumptions

- First order

$$\boxed{P(\text{“}hill\text{”} \mid \text{“}Then\ he\ hovered\ over\ the\text{”}) \approx P(\text{“}hill\text{”} \mid \text{“}the\text{”})}$$

- Second order

$$\boxed{P(\text{“}hill\text{”} \mid \text{“}Then\ he\ hovered\ over\ the\text{”}) \approx P(\text{“}hill\text{”} \mid \text{“}over\ the\text{”})}$$

N-gram language model

$$P(w_1, w_2, w_3, \dots, w_k) = \prod_{i=1}^{k} P\left(w_i \middle| w_{i-(n-1)}, \dots, w_{i-1}\right)$$

Considers only *n* words at a time, the current word and the previous *n*-1 words

- approximates each component in the product
- these approximations can be estimated using maximum likelihood estimation (MLE) on a training corpus

$$P\left(w_i \middle| w_{i-(n-1)}, \dots, w_{i-1}\right) = \frac{freq(w_{i-(n-1)},\dots,w_{i-1},w_i)}{freq(w_{i-(n-1)},\dots,w_{i-1})}$$

Unigram model

- n = 1

$$P(w_1, w_2, w_3, \ldots, w_k) = \prod_{i=1}^{k} P(w_i)$$

$$P(\text{“}Then\ he\ hovered\ over\ the\ hill\text{”})$$
$$= P(\text{“}Then\text{”}) \times P(\text{“}he\text{”}) \times P(\text{“}hovered\text{”}) \times P(\text{“}over\text{”}) \times P(\text{“}the\text{”}) \times P(\text{“}hill\text{”})$$

Bigram model

- n=2

$$P(w_1, w_2, w_3, \ldots, w_k) = \prod_{i=1}^{k} P(w_i | w_{i-1})$$

$$P(\text{“}Then\ he\ hovered\ over\ the\ hill\text{”})$$
$$= P(\text{“}Then\text{”}) \times P(\text{“}he\text{”} | \text{“}Then\text{”}) \times P(\text{“}hovered\text{”} | \text{“}he\text{”})$$
$$\times P(\text{“}over\text{”} | \text{“}hovered\text{”}) \times P(\text{“}the\text{”} | \text{“}over\text{”}) \times P(\text{“}hill\text{”} | \text{“}the\text{”})$$

Trigrams and beyond

- We can extend to:
  - trigrams (n=3)
  - quadrigrams (n=4)
  - 5-grams(n=5) …..
- The higher ***n*** is, the more long range dependencies can be captured … but the models will also become more sparse and unreliable

6-gram

`Presently he emerged, looking even more ______ than before.`

Products of probabilities

- Use logs
- Avoid underflow
- Computationally more efficient (adding is easier than multiplying)
- Convert back into probability at the end (if necessary!)

$$\boxed{\log(p_1 \times p_2 \times \cdots \times p_n) = \log(p_1) + \log(p_2) + \cdots + \log(p_n)}$$

Evaluation

- How good is a language model?
- Does it prefer “good” sentences to “bad” ones?
- Does it assign higher probabilities to “real” sentences rather than “ungrammatical” or “implausible” sentences?

Extrinsic evaluation

- Put each model in a task which requires a language model
  - spelling correction
  - machine translation
  - speech recognition
- Run the task and get an accuracy for each model
  - how many misspelt words corrected properly?
  - how many words translated correctly?
- Problems:
  - time-consuming
  - other factors affecting performance

Intrinsic evaluation

- Does the model assign higher probabilities to seen sentences than to unseen sentences?
- We trained the model's parameters on a training set
- We must test it on data that was **not used to train the model**
  - a test set
  - if we test on the training set, sentences will have artificially high probabilities
  - and it would be cheating!

Perplexity

- The best language model is one that best predicts an unseen test set
  - returns the highest P(sentences)
- Perplexity is the inverse probability of the test set, normalised by the number of words

$$PP(W) = P(w_1, w_2, w_3, \dots, w_N)^{-1/N}$$

$$PP(W) = e^{-1/N \log P(w_1, w_2, w_3, \dots, w_N)}$$

- this assumes that we have calculated probability as a sum of logs
- multiplying by -1/N first and then raising e to this power, makes the computation possible with floating point numbers

Minimising perplexity

- Example:
  - training 38 million words, testing 1.5 million words (WSJ text)

	unigram	bigram	trigram
Perplexity	962	170	109

Maximising probability is the same as minimising perplexity

- Perplexity should only really be compared for the same training and testing corpora

Generalisation in N-gram Language Models

Lecture 2, Part 2

A Toy Bigram Model

- I like to cook Chinese food.
- I want to eat dinner.
- They want to eat Indian food.

w_2

$P(w_2\|w_1)$	_ST.	I	They	like	want	to	cook	eat	Chinese	dinner	Indian	food	_END
_ST.	0	2/3	1/3	0	0	0	0	0	0	0	0	0	0
I	0	0	0	1/2	1/2	0	0	0	0	0	0	0	0
They	0	0	0	0	1	0	0	0	0	0	0	0	0
like	0	0	0	0	0	1	0	0	0	0	0	0	0
want	0	0	0	0	0	1	0	0	0	0	0	0	0
to	0	0	0	0	0	0	1/3	2/3	0	0	0	0	0
cook	0	0	0	0	0	0	0	0	1	0	0	0	0
eat	0	0	0	0	0	0	0	0	0	1/2	1/2	0	0
Chinese	0	0	0	0	0	0	0	0	0	0	0	1	0
dinner	0	0	0	0	0	0	0	0	0	0	0	0	1
Indian	0	0	0	0	0	0	0	0	0	0	0	1	0
food	0	0	0	0	0	0	0	0	0	0	0	0	1

w_1

A Toy Bigram Model

- I like to cook Chinese food.
- I want to eat dinner.
- They want to eat Indian food.
- They eat Chinese dinner.

w_2

w_1

$P(w_2\|w_1)$	_ST	I	They	like	want	to	cook	eat	Chinese	dinner	Indian	food	_END
_ST	0	**1/2**	**1/2**	0	0	0	0	0	0	0	0	0	0
I	0	0	0	1/2	1/2	0	0	0	0	0	0	0	0
They	0	0	0	0	**1/2**	0	0	**1/2**	0	0	0	0	0
like	0	0	0	0	0	1	0	0	0	0	0	0	0
want	0	0	0	0	0	1	0	0	0	0	0	0	0
to	0	0	0	0	0	0	1/3	2/3	0	0	0	0	0
cook	0	0	0	0	0	0	0	0	1	0	0	0	0
eat	0	0	0	0	0	0	0	0	**1/3**	**1/3**	**1/3**	0	0
Chinese	0	0	0	0	0	0	0	0	0	**1/2**	0	**1/2**	0
dinner	0	0	0	0	0	0	0	0	0	0	0	0	1
Indian	0	0	0	0	0	0	0	0	0	0	0	1	0
food	0	0	0	0	0	0	0	0	0	0	0	0	1

Generation

- The Shannon-Visualisation Method
  - Choose a random bigram (_ST.,w) according to its probability
  - Now choose another random bigram (w,x) according to its probability
  - And so on until we choose _END
  - Then string the words together

_ST.	I					
	I	want				
		want	to			
			to	eat		
				eat	Chinese	
					Chinese	food

I want to eat Chinese food

Approximating Shakespeare

1 gram	-To him swallowed confess hear both. Which. Of save on train for are ay device and rote life have
2 gram	- What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	- This shall forbid it should be branded, if renown made it empty.
4 gram	- King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; - It cannot be but so.

- N = 884, 647 tokens. V = 29, 066
- Shakespeare produced 300,000 bigram types out of V^2 possible bigrams (840 million)
  - so 99.96% of the possible bigrams will have 0 probabilities
- Quadrigrams even worse
- What's coming out looks like Shakespeare because it is Shakespeare

Overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
- It often doesn't
- If Shakespeare had written one more play … new possibilities for quadrigrams, trigrams, bigrams and even unigrams
- Models need to be robust – they need to generalise to unseen data

Zeros

TRAINING SET

- … denied the allegations
- … denied the reports
- … denied the claims
- … denied the request

TEST SET

- … denied the loan
- … denied the offer

$$\boxed{P(\text{“offer”} \mid \text{“denied the”}) = 0}$$

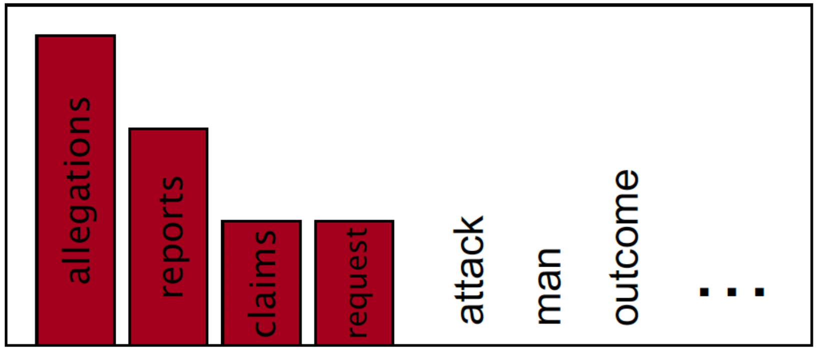
Trigrams (or even bigrams or unigrams) with zero probability in the training set mean that we

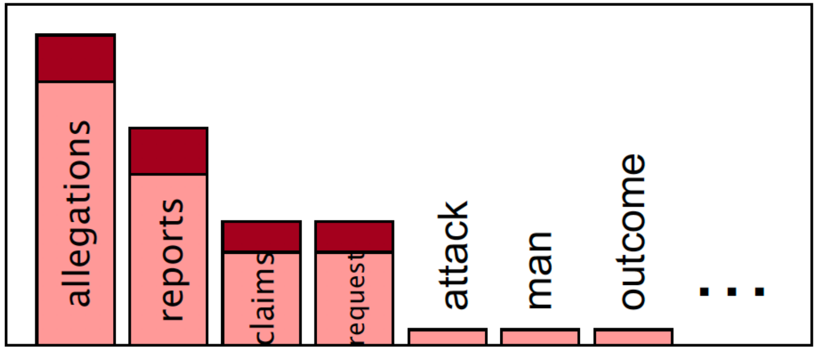
- assign zero probability to test set
- cannot calculate perplexity

Smoothing intuition

- When we have sparse statistics, steal probability mass from observed events to generalise to unobserved events

count (w| "denied the")	smoothed count
allegations 3	allegations 2.5
reports 2	reports 1.5
claims 1	claims 0.5
request 1	request 0.5
	OTHER 2
total 7	total 7





Add-one estimation

- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add on one to each count!
- Can be very effective for some problems
  - where number of zeros isn't so huge
  - e.g., text classification
- But for n-grams, rarely used
  - assigns too much mass to unseen co-occurrences (even add-k)
  - and leads to massive, unwieldy models

Unknown words

- Test corpus contains words that the training corpus doesn't
- Training corpus also contains words that the test corpus doesn't
- Which training corpus words are least likely to be in the test corpus?
- Fix the vocabulary (top *N* words in training corpus or all words which occur *f* or more times)
- Create a **<UNK>** token which captures probabilities for ***Out-Of-Vocabulary (OOV)*** words.

Unseen bigrams

- The **<UNK>** token allows us to estimate the probability of seeing an ***OOV*** word
- It even lets us estimate the probability of seeing two ***OOV*** words together or an ***OOV*** word with an in-vocabulary word
- But it does not allow us to estimate the probability of two in-vocabulary words which have not been seen together before
- What to do?

Absolute discounting

- **Subtract** a little from each bigram count in order to **save probability mass** for unseen events.
- How much?
- Church and Gale (1991)
  - Divided 22 million words of newswire text into training and testing sets
  - for each bigram count in the training set, what is its average bigram count in the test set?

Training count	Testing count
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Absolute discounting interpolation

- If we subtract *d* from each bigram count, how much probability mass do we save for unobserved bigrams?
- We need to keep track of the discounts made for each word
  - each time we discount a bigram $\boldsymbol{c(w_2|w_1)}$, we add that discount to a dummy token ***lambda*** for that word $\boldsymbol{c(\lambda|w_1)}$
  - normalise counts as probability distributions as before
  - For a smoothed probability estimate of any bigram, **interpolate**
    - sum the observed (discounted) probability and a proportion of reserved probability mass (according to the unigram probability of w_2)

$$\boxed{P_e(w_2|w_1) = P_d(w_2|w_1) + P_d(\lambda|w_1) \times P(w_2)}$$

The San Francisco problem

- The *absolute discounting interpolation method* divides up the reserved probability mass according to the **unigram** probability of the target word
- Assumption: Higher probability words are more likely to be seen in novel word combinations
- Not always true
- ”Francisco” is a high frequency word but only in the context of “San”
- In fact, high frequency means we have more evidence that a novel combination is unlikely

Kneser-Ney smoothing

- Don’t assign the reserved probability mass according to the unigram probability
- Calculate a separate probability for each word which is its likelihood of being seen in novel word combinations

$$\boxed{P_{KN}(w) = \frac{|\{w_j | c(w_j, w) > 0\}|}{\sum_i |\{w_j | c(w_j, w_i) > 0\}|}}$$

$$\boxed{P_e(w_2|w_1) = P_d(w_2|w_1) + P_d(\lambda|w_1) \times P_{KN}(w_2)}$$

Web-scale language models

- Google n-gram corpus
  - 1 billion five-word sequences over 13 million unique word types

4-gram	Count
serve as the incoming	92
serve as the incubator	99
serve as the independent	794
serve as the index	223
serve as the indication	72
serve as the indicator	120
serve as the indicators	45

Efficiency considerations:

- words stored as 64-bit hash number
- probabilities quantized using 4-8 bits (rather than 8-byte floats)
- n-grams stored in reverse tries
- n-grams shrunk by pruning

Stupid backoff

- Can apply full Kneser-Ney smoothing to web-scale language models
- But Brants et al. (2007) showed that a much simpler algorithm might be sufficient at this scale
- Stupid backoff gives up on the idea of making it a true probability distribution
- No discounting of higher order probabilities
- If a higher-order n-gram has a zero count, simply "*backoff*" to a lower order n-gram, with a fixed weight ($\lambda = 0.4$)

$$S(w_i|w_{i-k+1}^{i-1}) = \begin{cases} \frac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if count}(w_{i-k+1}^{i}) > 0 \\ \lambda S(w_i|w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

Coming up

- Neural language models (week 4)
  - feed-forward
  - RNNs and LSTMs
  - character-based

References

- Brants, T. et al. 2007. Large language models in machine translation. *In EMNLP/CONLL 2007*
- Church, K.W. and Gale, W.A. 1991. A Comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, 5, 19-54
- Mikolov, Yih and Zweig 2013 – Linguistic Regularities in Continuous Space Word Representations, (NAACL-HCT 2013)
- Zweig, G. and Burges, A. 2011. The Microsoft Research Sentence Completion Challenge. Microsoft Technical Report