Lecture 05:
Language modelling

OVERVIEW

- What is language modeling
- Why do we need language modeling
- Probabilistic Language Models
- Language modeling with n-grams

LANGUAGE MODEL

Formal definition

- Given a finite vocabulary *V* of words (tokens):
- Formal language: let *Ω* be a set of sequences of words from *V,*

  $\forall\, x \in \Omega$, *x* is called a sentence

- Language Model: A probability distribution over the formal language *Ω*,

$$P : \Omega \rightarrow [0, 1] \qquad \sum_{x \in \Omega} P(x) = 1$$

- A **Language Modeling** is a probability distribution over a sequence of words (tokens).

LANGUAGE MODEL

A language model aims to answer the question which sequences are more likely?

I would like to eat.
I would like eat to.
I like would eat to.
I to like would eat.

We expect that regular and grammatically correct sentences will occur more often in text and speech than other weird sequences

WHY DO WE NEED LANGUAGE MODELS?

Many NLP tasks require natural language output estimated from a sentence probability

- Machine translation: *"vents violents ce soir"*

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

- Speech recognition: return a transcript of what was spoken

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

- Spell-correction: *"the office is about fifteen minuets from my house"*

  $$P(about\ fifteen\ minutes\ from) > P(about\ fifteen\ minuets\ from)$$

- Summarization, question answering, etc

PROBABILISTIC LANGUAGE MODEL

- Probabilistic language models are based on the grouping of words into chunks called n-grams in order to estimate

$$P(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

- The main idea:
  - collect statistics about the frequency of different *n-grams*
    - n-gram probabilities are estimated by counting the frequency of occurrence in the vocabulary
  - use this to estimate the next word given a history of observed words

LANGUAGE MODELING

How do we compute the probability of a sequence of tokens $x^1, x^2, \dots, x^{t-1}, x^t$ from our vocabulary V?

The probability of a sentence $x^1, x^2, \dots, x^t$ is the joint probability $P(x^1, x^2, \dots, x^t)$ and estimate by the chain rule of probability

$$\begin{aligned} P(x^1, x^2, \dots, x^{t-1}, x^t) &= P(x^1 | x^2, \dots, x^{t-1}, x^t) P(x^2, \dots, x^{t-1}, x^t) \\ &= P(x^1) P(x^2 | x^1) P(x^3 | x^1, x^2) \dots . P(x^t \, | x^1, \dots, x^{t-1}) \\ &= P(x^1) \prod_{k=1}^{t} P(x^k | x^1, \dots, x^{k-1}) \end{aligned}$$

LANGUAGE MODEL

Given the sentence *"the student open their books"*
How do we compute the probability of *P(the student open their books)*

$$
\begin{aligned}
P(the\ student\ open\ their\ books) = \; & P(books|the\ student\ open\ their) * \\
& P(their|the\ student\ open) * \\
& P(open|the\ student) * \\
& P(student|the) * \\
& P(the)
\end{aligned}
$$

We estimate the probabilities by counting

$$P(books|the\ student\ open\ their) = \frac{count(the\ student\ open\ their\ books)}{count(the\ student\ open\ their)}$$

LANGUAGE MODEL – ESTIMATING PROBABILITIES

A language model is a distribution over all word-sequences $x_1x_2, \dots, x_n$ in a vocabulary V

$$\sum_{\langle x_1x_2,\dots,x_n\rangle} P(x_1x_2, \dots, x_n) = 1$$

by derivation is given by $P(x^1, x^2, \dots, x^{t-1}, x^t) = P(x^1) \prod_{k=1}^{t} P(x^k | x^1, \dots, x^{k-1})$

To estimate the probability of each sequence we need to

- first estimate $P(x^1)$
- Estimate probabilities $P(x^k | x^1, \dots, x^{k-1})$ for all $x^1, \dots, x^k$

LANGUAGE MODEL – ESTIMATING PROBABILITIES

- Relative frequency from a corpus

$$P(x_i|x_1, \ldots, x_{i-1}) = \frac{count(x_1, \ldots, x_{i-1}, x_i)}{count(x_1, \ldots, x_{i-1})}$$

$$= \frac{count(x_1, \ldots, x_{i-1}, x_i)}{\sum_{x \in V} count(x_1, \ldots, x_{i-1}, \ x)}$$

NB: this is the estimate for all sequences of length *l*

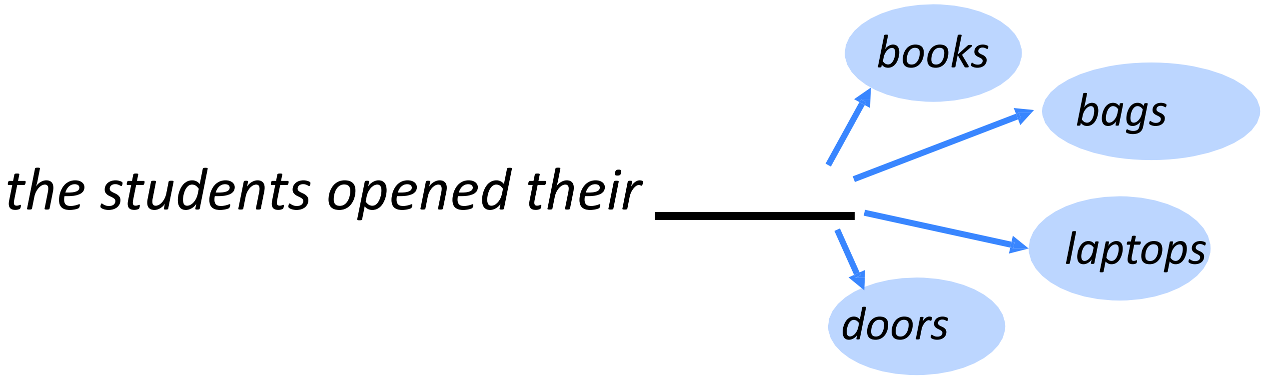
- Suppose /V/ = 1000, all sentences are approximately 10 word long, then we need to estimate 1000^{10} probabilities.

- no corpus is large enough to obtain an unbiased estimate of the probabilities

LANGUAGE MODELING

Language Modeling can be reduced to the task of predicting the next

$$P(x^1, x^2, \ldots, x^{t-1}, x^t) = P(x^1) \prod_{k=1}^{t} \boldsymbol{P(x^k | x^1, \ldots, x^{k-1})}$$



Given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$ compute the probability distribution of the next word $x^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

LANGUAGE MODELING – MARKOV ASSUMPTION

Markov assumption

- Independent and identical trials
- There is a fixed and finite *k* such that all word depends only on the preceding *k-1* words

$$P(x_{i+1}|x_1, \dots, x_i) \approx P(x_{i+1}|x_{i-k}, \dots, w_i) \ \forall k \geq 0$$

- Model: an k^{th}- order Markov model
- n-gram: statistics of an k-order Markov model is $\mathrm{k} + 1 - gram$ model

$$P(x_i|\boxed{x_{i-k}}, \dots, x_{i-1}) = \frac{count(\boxed{x_{i-k}} \dots, x_{i-1}, x_i)}{\sum_{x \in V} count(\boxed{x_{i-k}} \dots, x_{i-1}, \ x)}$$

LANGUAGE MODEL – MARKOV ASSUMPTION AND N-GRAMS

- N-gram language models are based on probabilities of chunks of word.

The student open their

- <u>Definition:</u> An n-gram is a chunk of n consecutive words.
  - *unigram: unit of single word – "the", "student", "opened", "their"*
  - *bigrams: unit of double words – "the student", "student opened", "opened their"*
  - *trigram: unit of triple words – "the student opened", "student opened their"*
  - *4-gram: unit of 4 words – "the student opened their"*

- The main idea behind *n-gram* models is to collect statistics about the frequency of different *n-grams* and use this to predict the next word.

LANGUAGE MODELING – MARKOV ASSUMPTION

The order of a Markov model is defined by the length of its history or *n-gram* *(n = k+1)*

$0^{th} - order$: $P(x_1, \dots, x_n) \approx P(x_1) \prod_{i=1}^{n-1} P(x_{i+1}) \approx P(x_1)P(x_2) \dots P(x_n)$
|history| = 0

$1^{st} - order$: $P(x_1, \dots, x_n) \approx P(x_1) \prod_{i=1}^{n-1} P(x_{i+1} | x_i)$
|history| = 1

$2^{nd} - order$: $P(x_1, \dots, x_n) \approx P(x_1) \prod_{i=1}^{n-1} P(x_{i+1} | x_i, x_{i-1})$
|history| = 2

$k^{th} - order$: $P(x_1, \dots, x_n) \approx P(x_1) \prod_{i=1}^{n-1} P\left(x_{i+1} \middle| \, x_{i-k}^{i}\right)$
|history| = k

FROM N-GRAM PROBABILITIES TO LANGUAGE MODEL

In a trigram model

$$P(x^1x^2x^3) = P(x^1)P(x^2|x^1)P(x^3|x^1x^2)$$

- The only trigram is $P(x^3|x^1x^2)$
  - $P(x^1)$ and $P(x^2|x^1)$ are not trigrams thus are from a different probability distribution

- Solution:
  - add *n-1* beginning of sentence (***<s>***) symbols

    <s><s>$x^1x^2x^3$......
  - similarly add *n-1* end of sentence symbols

    .....$x^1x^2x^3$ </s></s>

ESTIMATING BIGRAM PROBABILITIES

Bigram estimate of the probability of the sentence *“I want English food”*

P(<s> I want english food </s>) =

P(I|<s>)

× P(want|I)

× P(english|want)

× P(food|english)

× P(</s>|food)

= .000031

How is it estimated

- P(english|want) = .0011
- P(chinese|want) = .0065
- P(to|want) = .66
- P(eat | to) = .28
- P(food | to) = 0
- P(want | spend) = 0
- P (i | <s>) = .25

FROM N-GRAM PROBABILITIES TO LANGUAGE MODEL

- With the start or end of sentence token(s) we define a new vocabulary

$$V^* = V \, U \, \{</s>\}$$
$$or$$
$$V^* = V \, U \, \{< s >\}$$

- With the new vocabulary we can get a single distribution over strings of any length

- **Why?**
  - because *P(</s>|…)* will be high enough that we are always guaranteed to stop after generating a finite number of words.

LANGUAGE MODELING WITH N-GRAM

Maximum likelihood estimate

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

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

Example

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

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$$P(\texttt{I} \mid \texttt{<s>}) = \tfrac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \tfrac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \tfrac{2}{3} = .67$$

ESTIMATING PROBABILITIES - PRACTICAL ISSUES

- We do everything in log space
  - Avoid underflow
  - Computationally adding is faster than multiplying

$$\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

NUMBER OF POSSIBLE PARAMETERS

- Estimating the number of parameter per n-gram language model.
- Given a vocabulary V of $|V|$ unique tokens, where $|V| = 10^4$

  - **Unigram** model: $|V|$ parameters ⇔ 10^4 parameters
  - **Bigram** model: $|V|^2$ parameters ⇔ 10^8 parameters
  - **Trigram** model: $|V|$ parameters ó 10^{12} parameters

SHAKESPEARE AS CORPUS

- Number of words (symbols) = 884,647
- Tokens, V=29,066
- Shakespeare produced 300,000 bigrams
- bigram types out of V^2= 844 million possible bigrams
- So, 99.96% of the possible bigrams were never seen (have zero entries in the table)

- Quadrigrams worse:
  - What's coming out looks like Shakespeare because it ***is*** Shakespeare

EVALUATION: HOW GOOD IS OUR MODEL?

- Does our language model prefer good sentences to bad ones?
- Assign higher probability to "real" or "frequently observed" sentences
  - than "ungrammatical" or "rarely observed" sentences?
- We train parameters of our model on a **training set**.
- We test the model's performance on data we haven't seen.
  - A **test set** is an unseen dataset that is different from our training set, totally unused.
  - An **evaluation metric** tells us how well our model does on the test set.

EXTRINSIC EVALUATION OF N-GRAM MODELS

- Best evaluation for comparing models A and B
- Embed each model in a task
  - spelling corrector,
  - speech recognizer,
  - Machine Translation system
- Run the task, get an accuracy for A and for B
  - How many misspelled words corrected properly
  - How many words translated correctly
- Compare accuracy for A and B

Time-consuming; can take days, weeks or even months

INTRINSIC EVALUATION OF N-GRAM MODELS

- Sometimes use **intrinsic** evaluation: **perplexity**
- Bad approximation
  - unless the test data looks **just** like the training data
  - So **generally, only useful in pilot experiments**
- But is helpful to think about.

PERPLEXITY

- Perplexity is the inverse probability of the test set, normalized by the number of words *N*:

$$perplexity(W) = P(x_1 x_2, \dots, x_N)^{-1/N} = \sqrt[N]{\frac{1}{P(x_1 x_2, \dots, x_N)}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(x_i | x_1, \dots, x_{i-1})}}$$

- For unigram $perplexity(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(x_i)}}$

- For bigram $perplexity(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(x_i)}}$

NB: Minimizing perplexity is the same as maximizing probability

INTUITION OF PERPLEXITY

- The **Shannon Game**:
  - How well can we predict the next word?

I always order pizza with cheese and ____

The 33rd President of the US was ____

I saw a ____

laptop	0.09
mushrooms	0.02
pepperoni	0.1
anchovies	0.0001
fried rice	0.005
and	1e-10
cook	2e-15

  - Unigrams are terrible at this game. (Why?)
- A better language model is the one that assigns a higher probability to the most appropriate word

LIMITATIONS – STORAGE PROBLEMS

Storage Problem:
The need to store count for all n-grams you saw in the corpus.

$$P(\boldsymbol{w} | \ I\, want\, to\, eat) = \frac{\boxed{count(I\, want\, to\, eat\, \boldsymbol{w})}}{count(I\, want\, to\, eat)}$$

Increasing *n* or increasing corpus size <=> increases model size

LIMITATIONS - SPARSITY PROBLEM

Sparsity Problem 1:
- What if "*I want to w*" never occurred in the corpus?
- Then *w* has probability 0

Solution - smoothing: Add small δ to the count for every $w \in V$.

$$P(\boldsymbol{w}|waiting\ for\ the) = \frac{\boxed{count(waiting\ for\ the\ \boldsymbol{w})}}{\boxed{count(waiting\ for\ the)}}$$

Sparsity Problem 2:
- What if "*I want to*" never occurred in the corpus?
- Then we cannot calculate probabiliity of any *w.*

Solution - backoff: condition on the (n-1)-gram, i.e. "*I want*"

Larger *n* makes sparsity problem worse. Typically *n* should be less than or equal to 5

THE PERILS OF OVERFITTING - SPARSITY

N-grams only work well for word prediction if the test corpus looks like the training corpus.

- In real life, it often doesn’t
  - We need to train robust models that generalize!
- One kind of generalization: Zeros!
  - Things that don’t ever occur in the training set but occur in the test set

SMOOTHING METHODS

Smoothing methods

- Additive smoothing
- Good-Turing estimate
- Jelinek-Mercer smoothing (interpolation)
- Katz smoothing (backoff)
- Witten-Bell smoothing
- Absolute discounting
- Kneser-Ney smoothing

ADDITIVE SMOOTHING

- Idea: pretend we've seen each n-gram δ times more than we have.
  - Typically, $0 < \delta \leq 1$.
  - Lidstone and Jeffreys advocate $\delta = 1$.
  - Gale & Church (1994) argue that this method performs poorly.

$$p_{add}(w_i|w_{i-n+1}^{i-1}) = \frac{\delta + c(w_{i-n+1}^{i})}{\delta|V| + \sum_{w_i} c(w_{i-n+1}^{i})}$$

ADD-ONE ESTIMATION

- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add one to all the counts!
- MLE estimate:

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

- Add-1 estimate:

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

BACKOFF AND INTERPOLATION

- Sometimes it helps to use **less** context
  - Condition on less context for contexts you haven't learned much about
- **Backoff:**
  - use trigram if you have good evidence,
  - otherwise, bigram, otherwise unigram
- **Interpolation:**
  - mix unigram, bigram, trigram
- NB: Interpolation works better

REFERENCES