COMS 4705.H: Language Models

Karl Stratos

January 27, 2017
Motivation

How likely are the following sentences?

- the dog barked
- the cat barked
- dog the barked
- oqc shgwqw#w 1g0
Motivation

How likely are the following sentences?

- the dog barked
  
  “probability 0.1”

- the cat barked
  
  “probability 0.03”

- dog the barked
  
  “probability 0.00005”

- oqc shgwqw#w 1g0
  
  “probability $10^{-13}$”
A **language model** is a function that defines a probability distribution \( p(x_1 \ldots x_m) \) over all sentences \( x_1 \ldots x_m \).

**Goal:** Design a *good* language model, in particular

\[
p(\text{the dog barked}) > p(\text{the cat barked}) > p(\text{dog the barked}) > p(\text{oqc shgwqw#w 1g0})
\]
Overview

The Probability of a Sentence

The $n$-Gram Language Models
  Unigram, Bigram, Trigram Models
  Estimation from Data
  Evaluation

Smoothing, Discount Methods
Problem Statement

▶ We’ll assume a finite vocabulary $V$ (i.e., the set of all possible word types).

▶ Sample space: $\Omega = \{x_1 \ldots x_m \in V^m : m \geq 1\}$

▶ Task: Design a function $p$ over $\Omega$ such that

$$p(x_1 \ldots x_m) \geq 0 \quad \forall x_1 \ldots x_m \in \Omega$$

$$\sum_{x_1 \ldots x_m \in \Omega} p(x_1 \ldots x_m) = 1$$

▶ What are some challenges?
Challenge 1: Infinitely Many Sentences

- Can we “break up” the probability of a sentence into probabilities of individual words?

- **Yes**: Assume a *generative process*.

- We may assume that each sentence $x_1 \ldots x_m$ is generated as
  
  (1) $x_1$ is drawn from $p(\cdot)$,
  
  (2) $x_2$ is drawn from $p(\cdot|x_1)$,
  
  (3) $x_3$ is drawn from $p(\cdot|x_1, x_2)$,

  $\ldots$

  (m) $x_m$ is drawn from $p(\cdot|x_1, \ldots, x_{m-1})$,

  (m + 1) $x_{m+1}$ is drawn from $p(\cdot|x_1, \ldots, x_m)$.

  where $x_{m+1} = \text{STOP}$ is a special token at the end of every sentence.
Justification of the Generative Assumption

By the **chain rule**, 

\[
p(x_1 \ldots x_m \text{ STOP}) = p(x_1) \times p(x_2|x_1) \times p(x_3|x_1, x_2) \times \cdots \\
\quad \cdots \times p(x_m|x_1, \ldots, x_{m-1}) \times p(\text{STOP}|x_1, \ldots, x_m)
\]

Thus we have solved the first challenge.

- Sample space = *finite* \( V \)
- The model still defines a proper distribution over all sentences.

(Does the generative process need to be left-to-right?)
Challenge 2: Infinitely Many Distributions

Under the generative process, we need infinitely many conditional word distributions:

\[
\begin{align*}
p(x_1) & \quad \forall x_1 \in V \\
p(x_2|x_1) & \quad \forall x_1, x_2 \in V \\
p(x_3|x_1, x_2) & \quad \forall x_1, x_2, x_3 \in V \\
p(x_4|x_1, x_2, x_3) & \quad \forall x_1, x_2, x_3, x_4 \in V \\
\vdots & \quad \vdots
\end{align*}
\]

Now our goal is to redesign the model to have only a finite, compact set of associated values.
Overview

The Probability of a Sentence

The $n$-Gram Language Models

Unigram, Bigram, Trigram Models
Estimation from Data
Evaluation

Smoothing, Discount Methods
Independence Assumptions

$X$ is independent of $Y$ if

$$ P(X = x | Y = y) = P(X = x) $$

$X$ is conditionally independent of $Y$ given $Z$ if

$$ P(X = x | Y = y, Z = z) = P(X = x | Z = z) $$

Can you think of such $X, Y, Z$?
Assumption. A word is independent of all previous words:

\[ p(x_i | x_1 \ldots x_{i-1}) = p(x_i) \]

That is,

\[ p(x_1 \ldots x_m) = \prod_{i=1}^{m} p(x_i) \]

Number of parameters: \( O(|V|) \)

Not a very good language model:

\[ p(\text{the dog barked}) = p(\text{dog the barked}) \]
Assumption. A word is independent of all previous words condition on the preceding word:

\[ p(x_i|x_1 \ldots x_{i-1}) = p(x_i|x_{i-1}) \]

That is,

\[ p(x_1 \ldots x_m) = \prod_{i=1}^{m} p(x_i|x_{i-1}) \]

where \( x_0 = * \) is a special token at the start of every sentence.

Number of parameters: \( O(|V|^2) \)
Trigram Language Model

**Assumption.** A word is independent of all previous words conditioning on the two preceding words:

\[ p(x_i|x_1 \ldots x_{i-1}) = p(x_i|x_{i-2}, x_{i-1}) \]

That is,

\[ p(x_1 \ldots x_m) = \prod_{i=1}^{m} p(x_i|x_{i-2}, x_{i-1}) \]

where \( x_{-1}, x_0 = * \) are special tokens at the start of every sentence.

Number of parameters: \( O(|V|^3) \)
The $n$-Gram Language Model

Assumption. A word is independent of all previous words conditioning on the $n - 1$ preceding words:

$$p(x_i|x_1 \ldots x_{i-1}) = p(x_i|x_{i-n+1}, \ldots, x_{i-1})$$

Number of parameters: $O(|V|^n)$

This kind of conditional independence assumption ("depends only on the last $n - 1$ states...") is called a **Markov assumption**.

- Is this a reasonable assumption for language modeling?
Overview

The Probability of a Sentence

The $n$-Gram Language Models
-Unigram, Bigram, Trigram Models
- Estimation from Data
- Evaluation

Smoothing, Discount Methods
A Practical Question

▶ Summary so far: We have designed probabilistic language models parametrized by finitely many values.

▶ Bigram model: Stores a table of $O(|V|^2)$ values

\[ q(x'|x) \quad \forall x, x' \in V \]

(plus $q(x|\ast)$ and $q(\text{STOP}|x)$) representing transition probabilities and computes

\[ p(\text{the cat barked}) = q(\text{the}|\ast) \times q(\text{cat}|\text{the}) \times q(\text{barked}|\text{cat}) \times q(\text{STOP}|\text{barked}) \]

▶ Q. But where do we get these values?
Estimation from Data

- Our data is a **corpus** of \( N \) sentences \( x^{(1)} \ldots x^{(N)} \).

- Define \( \text{count}(x, x') \) to be the number of times \( x, x' \) appear together (called “bigram counts”):

\[
\text{count}(x, x') = \sum_{i=1}^{N} \sum_{j=1: \begin{cases} x_j = x' \\ x_{j-1} = x \end{cases}}^{l_i + 1} 1
\]

\( (l_i = \text{length of } x^{(i)} \text{ and } x_{l_i + 1} = \text{STOP}) \)

- Define \( \text{count}(x) := \sum_{x'} \text{count}(x, x') \) (called “unigram counts”).
Example Counts

Corpus:
- the dog chased the cat
- the cat chased the mouse
- the mouse chased the dog

Example bigram/unigram counts:

- \( \text{count}(x_0, \text{the}) = 3 \)
- \( \text{count}(\text{the}) = 6 \)
- \( \text{count}(\text{chased}, \text{the}) = 3 \)
- \( \text{count}(\text{chased}) = 3 \)
- \( \text{count}(\text{the}, \text{dog}) = 2 \)
- \( \text{count}(x_0) = 3 \)
- \( \text{count}(\text{cat}, \text{STOP}) = 1 \)
- \( \text{count}(\text{cat}) = 2 \)
Parameter Estimates

- For all $x, x'$ with $\text{count}(x, x') > 0$, set
  \[ q(x'|x) = \frac{\text{count}(x, x')}{\text{count}(x)} \]
  Otherwise $q(x'|x) = 0$.

- In the previous example:
  \[ q(\text{the}|x_0) = \frac{3}{3} = 1 \]
  \[ q(\text{chased}|\text{dog}) = \frac{1}{3} = 0.\bar{3} \]
  \[ q(\text{dog}|\text{the}) = \frac{2}{6} = 0.\bar{3} \]
  \[ q(\text{STOP}|\text{cat}) = \frac{1}{2} = 0.5 \]
  \[ q(\text{dog}|\text{cat}) = 0 \]

- Called maximum likelihood estimation (MLE).
Justification of MLE

Claim. The solution of the constrained optimization problem

\[
q^* = \arg \max_{q} \sum_{i=1}^{N} \sum_{j=1}^{l_i+1} \log q(x_j|x_{j-1})
\]

is given by

\[
q^*(x'|x) = \frac{\text{count}(x, x')}{\text{count}(x)}
\]

(Proof?)
MLE: Other $n$-Gram Models

Unigram:

\[ q(x) = \frac{\text{count}(x)}{N} \]

Bigram:

\[ q(x' | x) = \frac{\text{count}(x, x')}{\text{count}(x)} \]

Trigram:

\[ q(x'' | x, x') = \frac{\text{count}(x, x', x'')}{\text{count}(x, x')} \]
Overview

The Probability of a Sentence

The $n$-Gram Language Models
  Unigram, Bigram, Trigram Models
  Estimation from Data
  Evaluation

Smoothing, Discount Methods
Evaluation of a Language Model

“How good is the model at predicting unseen sentences?”

**Held-out corpus**: Used for evaluation purposes only

- One metric: log likelihood of unseen sentences $y^{(1)} \ldots y^{(T)}$

$$\text{LL} = \sum_{i=1}^{T} \log p(y^{(i)})$$

- More popular metric: **perplexity** of the model:

$$\text{PP} = 2^{-\frac{1}{M} \sum_{i=1}^{T} \log p(y^{(i)})}$$

where $M$ is the number of words + STOP symbols
Motivation of Perplexity: The Branching Factor

- How many times do we expect to flip a coin until we get a head, if it comes up head with probability $\epsilon$?

- $1/\epsilon$ times
  - Mean of the geometric distribution with parameter $\epsilon$

- Examples
  - $\epsilon = 0.5$: expect to flip two times
  - $\epsilon = 0.1$: expect to flip ten times
  - $\epsilon = 0.001$: expect to flip a thousand times

- The higher the “branching factor” $1/\epsilon$ is, the more “surprised” the model.
The Branching Factor of Language Models

- For simplicity, assume a single sentence $y = y_1 \ldots y_{M-1}$ STOP.

- The branching factor of the model at word $y_i$:

  $\frac{1}{p(y_i|y_1 \ldots y_{i-1})}$

- Geometric average of the branching factors:

  $\prod_{i=1}^{M} \left( \frac{1}{p(y_i|y_1 \ldots y_{i-1})} \right)^{\frac{1}{M}}$
Perplexity = Average Branching Factor

\[
\prod_{i=1}^{M} \left( \frac{1}{p(y_i|y_1 \cdots y_{i-1})} \right)^{\frac{1}{M}} = \prod_{i=1}^{M} 2^{\log \left( \frac{1}{p(y_i|y_1 \cdots y_{i-1})} \right)^{\frac{1}{M}}}
\]
Perplexity = Average Branching Factor

\[
\prod_{i=1}^{M} \left( \frac{1}{p(y_i|y_1 \cdots y_{i-1})} \right)^{\frac{1}{M}} = \prod_{i=1}^{M} 2^{\log \left( \frac{1}{p(y_i|y_1 \cdots y_{i-1})} \right)^{\frac{1}{M}}}
= \prod_{i=1}^{M} 2^{-\frac{1}{M} \log p(y_i|y_1 \cdots y_{i-1})}
\]
Perplexity = Average Branching Factor

\[ \prod_{i=1}^{M} \left( \frac{1}{p(y_i|y_1 \cdots y_{i-1})} \right) \frac{1}{M} = \prod_{i=1}^{M} 2^{\log \left( \frac{1}{p(y_i|y_1 \cdots y_{i-1})} \right)} \frac{1}{M} \]

\[ = \prod_{i=1}^{M} 2^{-\frac{1}{M} \log p(y_i|y_1 \cdots y_{i-1})} \]

\[ = 2^{-\frac{1}{M} \sum_{i=1}^{M} \log p(y_i|y_1 \cdots y_{i-1})} \]
Perplexity = Average Branching Factor

\[ \prod_{i=1}^{M} \left( \frac{1}{p(y_i|y_1 \cdots y_{i-1})} \right)^{\frac{1}{M}} = \prod_{i=1}^{M} 2^{\log \left( \frac{1}{p(y_i|y_1 \cdots y_{i-1})} \right)}^{\frac{1}{M}} \]

\[ = \prod_{i=1}^{M} 2^{-\frac{1}{M} \log p(y_i|y_1 \cdots y_{i-1})} \]

\[ = 2^{-\frac{1}{M} \sum_{i=1}^{M} \log p(y_i|y_1 \cdots y_{i-1})} \]

\[ = 2^{-\frac{1}{M} \log \prod_{i=1}^{M} p(y_i|y_1 \cdots y_{i-1})} \]
Perplexity = Average Branching Factor

\[
\prod_{i=1}^{M} \left( \frac{1}{p(y_i | y_1 \cdots y_{i-1})} \right)^{\frac{1}{M}} = \prod_{i=1}^{M} 2^{\log \left( \frac{1}{p(y_i | y_1 \cdots y_{i-1})} \right)^{\frac{1}{M}}}
\]

\[
= \prod_{i=1}^{M} 2^{-\frac{1}{M} \log p(y_i | y_1 \cdots y_{i-1})}
\]

\[
= 2^{-\frac{1}{M} \sum_{i=1}^{M} \log p(y_i | y_1 \cdots y_{i-1})}
\]

\[
= 2^{-\frac{1}{M} \log \prod_{i=1}^{M} p(y_i | y_1 \cdots y_{i-1})}
\]

\[
= 2^{-\frac{1}{M} \log p(y)}
\]
Perplexity = Average Branching Factor

$$\prod_{i=1}^{M} \left( \frac{1}{p(y_i|y_1 \ldots y_{i-1})} \right)^{\frac{1}{M}} = \prod_{i=1}^{M} 2^{\log \left( \frac{1}{p(y_i|y_1 \ldots y_{i-1})} \right)^{\frac{1}{M}}}$$

$$= \prod_{i=1}^{M} 2^{- \frac{1}{M} \log p(y_i|y_1 \ldots y_{i-1})}$$

$$= 2^{- \frac{1}{M} \sum_{i=1}^{M} \log p(y_i|y_1 \ldots y_{i-1})}$$

$$= 2^{- \frac{1}{M} \log \prod_{i=1}^{M} p(y_i|y_1 \ldots y_{i-1})}$$

$$= 2^{- \frac{1}{M} \log p(y)}$$

$$= \text{PP}$$
Example Perplexity Values

- If the model perfectly predicts test sentence,
  \[
  \text{PP} = 2^{-\frac{1}{M} \log \prod_{i=1}^{M} p(y_i|y_1...y_{i-1})} = 2^{-\frac{1}{M} \log 1} = 1
  \]

- If the model predicts words uniformly at random,
  \[
  \text{PP} = 2^{-\frac{1}{M} \sum_{i=1}^{M} \log p(y_i|y_1...y_{i-1})} = 2^{-\frac{1}{M} \sum_{i=1}^{M} \log 1/|V|} = |V|
  \]

- Empirical values for $|V| = 50,000$ (Goodman, 2001)
  - Unigram: 955, Bigram: 137, Trigram: 74
The Probability of a Sentence

The \textit{n}-Gram Language Models
\hspace{1em} Unigram, Bigram, Trigram Models
\hspace{1em} Estimation from Data
\hspace{1em} Evaluation

Smoothing, Discount Methods
Smoothing

In practice, it’s important to **smooth** estimation of higher-order models:

\[
q^{\text{smoothed}}(x'' | x, x') = \lambda_1 q(x'' | x, x') + \\
\lambda_2 q(x'' | x') + \\
\lambda_3 q(x'')
\]

where \( \lambda_1 + \lambda_2 + \lambda_3 = 1 \) and \( \lambda_i \geq 0 \). Called **linear interpolation**.
Discount Methods

At test time, how do we handle words that were unobserved in the training corpus?

- Naively, we assign probability 0 to the entire held-out data!

A solution: “steal” some probability mass from observed words and allocate it for unobserved words.

Called *discount methods*. Will cover more details in video lectures / textbook.