Language Modeling 2

CS287
Let $\mathcal{V}$ be the vocabulary of English and let $s$ score any window of size $d_{\text{win}} = 5$, if we see the phrase

$[\text{the dog walks to the}]$

It should score higher by $s$ than

$[\text{the dog house to the}]$

$[\text{the dog cats to the}]$

$[\text{the dog skips to the}]$

$...$
Review: Continuous Bag-of-Words (CBOW)

\[ \hat{y} = \text{softmax}\left(\frac{\sum_i x_i^0 W^0}{d_{\text{win}} - 1}\right) W^1 \]

- Attempt to predict the middle word
  
  \[ \text{[ the dog walks to the ]} \]

Example: CBOW

\[ x = \frac{v(w_3) + v(w_4) + v(w_6) + v(w_7)}{d_{\text{win}} - 1} \]

\[ y = \delta(w_5) \]

\( W^1 \) is no longer partitioned by row (order is lost)
Review: Softmax Issues

Use a softmax to force a distribution,

$$\text{softmax}(z) = \frac{\exp(z)}{\sum_{c \in C} \exp(z_c)}$$

$$\log \text{softmax}(z) = z - \log \sum_{c \in C} \exp(z_c)$$

- **Issue:** class $C$ is huge.
- For C&W, 100,000, for word2vec 1,000,000 types
- Note largest dataset is 6 billion words
We have trained a depth-two balanced soft-max tree. Give an algorithm and time-complexity for:

- Computing the optimal class decision?
- Greedily approximating the optimal class decision.
- Appropriately sampling a word.
- Computing the full distribution over classes?
Answers I

- Computing the optimal class decision i.e. $\arg \max_y p(y = y|x)$?
  - Requires full enumeration $O(|\mathcal{V}|)$
    $$\arg \max_y p(y = y|x) = \arg \max_y p(y|C, x)p(C|x)$$
  - Greedily approximating the optimal class decision.
    - Walk tree $O(\sqrt{|\mathcal{V}|})$
      $$\arg \max_y p(y = y|x)$$
      $$c^* = \arg \max_y p(C|x)$$
      $$y^* = \arg \max_y p(y|C = c^*, x)$$
Answers II

- Appropriately sampling a word $y \sim p(y = \delta(c)|x)$?
  - Walk tree $O(\sqrt{|\mathcal{V}|})$

\[
y \sim p(y = \delta(c)|x) = \mathcal{C} \\
\hat{c} \sim p(C|x) \\
\hat{y} \sim p(y|C = \hat{c}, x)
\]

- Computing the full distribution over classes?
  - Enumerate $O(|\mathcal{V}|)$

\[
p(y = \delta(c)|x)p(y|C = \hat{c}, x)
\]
Contents

Language Modeling

NGram Models

Smoothing

Language Models in Practice
Language Modeling Task

Given a sequence of text give a probability distribution over the next word.

The Shannon game. Estimate the probability of the next letter/word given the previous.

THE ROOM WAS NOT VERY LIGHT A SMALL OBLONG READING LAMP ON THE DESK SHED GLOW ON POLISHED ___
Shannon (1948) *Mathematical Model of Communication*

*We may consider a discrete source, therefore, to be represented by a stochastic process. Conversely, any stochastic process which produces a discrete sequence of symbols chosen from a finite set may be considered a discrete source. This will include such cases as: 1. Natural written languages such as English, German, Chinese.* ...
4. Third-order approximation (trigram structure as in English).

5. First-Order Word Approximation. Rather than continue with tetragram, ... , ll-gram structure it is easier and better to jump at this point to word units. Here words are chosen independently but with their appropriate frequencies.

Shannon’s Babblers I
6. Second-Order Word Approximation. The word transition probabilities are correct but no further structure is included. THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH ’RITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED

The resemblance to ordinary English text increases quite noticeably at each of the above steps.
It is a capital mistake to theorize before one has data.

Insensibly one begins to twist facts to suit theories, instead of theories to suit facts. -Sherlock Holmes, A Scandal in Bohemia
It is a capital mistake to theorize before one has _____ . . .
108 938 285 28 184 29 593 219 58 772 _____ . . .
Language Modeling

Crucially important for:

- Speech Recognition
- Machine Translation
- Many deep learning applications
  - Captioning
  - Dialogue
  - Summarization
  - ...

How permanent are their records?
Statistical Model of Speech
Transcription: Help peppermint on their records
Transcription: How permanent are their records
Language Modeling Formally

**Goal:** Compute the probability of a sentence,

- **Factorization:**

  \[ p(w_1, \ldots, w_n) = \prod_{t=1}^{n} p(w_t \mid w_1, \ldots, w_{t-1}) \]

  k Estimate the probability of the next word, conditioned on prefix,

  \[ p(w_t \mid w_1, \ldots, w_{t-1}; \theta) \]
Machine Learning Setup

Multi-class prediction problem,

\[(x_1, y_1), \ldots, (x_n, y_n)\]

- \(y_i\); the one-hot next word
- \(x_i\); representation of the prefix \((w_1, \ldots, w_{t-1})\)

Challenges:
- How do you represent input?
- Smoothing is crucially important.
- Output space is very large (next class)
Machine Learning Setup

Multi-class prediction problem,

\[(x_1, y_1), \ldots, (x_n, y_n)\]

- \(y_i\); the one-hot next word
- \(x_i\); representation of the prefix \((w_1, \ldots, w_{t-1})\)

Challenges:

- How do you represent input?
- Smoothing is crucially important.
- Output space is very large (next class)
Problem Metric

Previously, used accuracy as a metric.

Language modeling uses of version average negative log-likelihood

- For test data $\bar{w}_1,\ldots,\bar{w}_n$

$$NLL = -\frac{1}{n} \sum_{i=1}^{n} \log p(w_i|w_1,\ldots,w_{i-1})$$

Actually report perplexity,

$$perp = \exp\left(-\frac{1}{n} \sum_{i=1}^{n} \log p(w_i|w_1,\ldots,w_{i-1})\right)$$

Requires modeling full distribution as opposed to argmax (hinge-loss)
Perplexity: Intuition

- Effective uniform distribution size.
- If words were uniform: \( \text{perp} = \left| \mathcal{V} \right| = 10000 \)
- Using unigram: \( \text{perp} \approx 400 \)
- \( \log_2 \text{perp} \) gives average number of bits needed per word.
In practice representation doesn't use all \((w_1, \ldots, w_{t-1})\)

\[ p(w_i | w_1, \ldots, w_{i-1}) \approx p(w_i | w_{i-n+1}, \ldots, w_{i-1}; \theta) \]

- We call this an n-gram model.
- \(w_{i-n+1}, \ldots, w_{i-1};\) the context
Bigram Models

Back to count-based multinomial estimation,

- Feature set $\mathcal{F}$: previous words
- Input vector $\mathbf{x}$ is sparse
- Count matrix $\mathbf{F} \in \mathbb{R}^{V \times V}$
- Counts come from training data:

$$F_{c,w} = \sum_i 1(w_{i-1} = c, w_i = w)$$

$$p_{ML}(w_i | w_{i-1}; \theta) = \frac{F_{c,w}}{F_c}.$$
Trigram Models

- Feature set $\mathcal{F}$: previous two words (conjunction)
- Input vector $\mathbf{x}$ is sparse
- Count matrix $\mathbf{F} \in \mathbb{R}^{V \times V}$

$$F_{c,w} = \sum_i 1(w_{i-2:i-1} = c, w_i = w)$$

$$p_{ML}(w_i | w_{i-2:i-1} = c; \theta) = \frac{F_{c,w}}{F_{c,.}}$$
Notation

- \( c = w_{i-n+1:i-1}; \) context
- \( c' = w_{i-n+2:i-1}; \) context without first word
- \( c'' = w_{i-n+3:i-1}; \) context without first two words
- \([x]_+ = \max\{0, x\}; \) positive part (ReLU)
- \( F_{c,w} = \sum_c F_{c,w}; \) sum-over-dimension
- Maximum-likelihood,
  \[
p_{\text{ML}}(w|c) = \frac{F_{c,w}}{F_{c}}.
  \]
- Non-zero count words, for all \( c, w \)
  \[
  N_{c,w} = 1(F_{c,w} > 0)
  \]
NGram Models

It is common to go up to 5-grams,

\[ F_{c,w} = \sum_i 1(w_{i-5+1} \ldots w_{i-1} = c, w_i = w) \]

Matrix becomes very sparse at 5-grams.
### Google 1T

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tokens</td>
<td>1,024,908,267,229</td>
</tr>
<tr>
<td>Number of sentences</td>
<td>95,119,665,584</td>
</tr>
<tr>
<td>Size compressed (counts only)</td>
<td>24 GB</td>
</tr>
<tr>
<td>Number of unigrams</td>
<td>13,588,391</td>
</tr>
<tr>
<td>Number of bigrams</td>
<td>314,843,401</td>
</tr>
<tr>
<td>Number of trigrams</td>
<td>977,069,902</td>
</tr>
<tr>
<td>Number of fourgrams</td>
<td>1,313,818,354</td>
</tr>
<tr>
<td>Number of fivegrams</td>
<td>1,176,470,663</td>
</tr>
</tbody>
</table>
Contents

Language Modeling

NGram Models

Smoothing

Language Models in Practice
NGram Models

- Maximum likelihood models word terribly.
- NGram models are sparse, need to handle unseen cases.
- Presentation follows work of Chen and Goodman (1999)
Count Modifications

- Laplace Smoothing

\[ \tilde{F}_{c,w} = F_{c,w} + \alpha \text{ for all } c, w \]

- Good-Turing Smoothing (Good, 1953)

\[ \tilde{F}_{c,w} = (F_{c,w} + 1) \frac{\text{hist}(F_{c,w} + 1)}{\text{hist}(F_{c,w})} \text{ for all } c, w \]
Real Issues N-Gram Sparsity

- Histogram of trigrams
Idea 1: Interpolation

For trigrams:

\[ p_{\text{interp}}(w|c) = \lambda_1 p_{\text{ML}}(w|c) + \lambda_2 p_{\text{ML}}(w|c') + \lambda_3 p_{\text{ML}}(w|c'') \]

Ensure that \( \lambda \)s form convex combination

\[ \sum_i \lambda_i = 1 \]

\[ \lambda_i \geq 0 \text{ for all } i \]
Idea 1: Interpolation (Jelinek-Mercer Smoothing)

Can write recursively,

\[ p_{\text{interp}}(w|c) = \lambda p_{\text{ML}}(w|c) + (1 - \lambda) p_{\text{interp}}(w|c') \]

Ensure that \( \lambda \) form convex combination

\[ 0 \leq \lambda \leq 1 \]
How to set parameters: Validation Tuning

- Treat $\lambda$ as hyper-parameters
- Can estimate $\lambda$ using EM (or gradients directly)
- Alternatively: Grid-search to held-out data.
- Can add more parameters $\lambda(c, w)$
- Language models are often stored with prob and backoff value ($\lambda$)
How to set parameters: Witten-Bell

Define $\lambda(c, w)$ as a function of $c, w$

$$(1 - \lambda) = \frac{N_{c, \cdot}}{N_{c, \cdot} + F_{c, \cdot}}$$

$\triangleright \ N_{c, \cdot};$ Number of unique word types seen with context

$$p_{wb}(w|c) = \frac{F_{c, w} + N_{c, \cdot} \times p_{wb}(w|c')}{F_{c, \cdot} + N_{c, \cdot}}$$

Interpolation counts are a new events estimated as proportional to seeing a new word.
Idea 2: Absolute Discounting

Similar form to interpolation

\[ p_{abs}(w|c) = \lambda(w, c)p_{ML}(w|c) + \eta(c)p_{abs}(w|c') \]

Delete counts and redistribute:

\[ p_{abs} = \frac{[F_{c,w} - D]_+}{F_{c,\cdot}} + \eta(c)p_{abs}(w|c') \]

Need for a given \( c \)

\[ \sum_w \lambda(w, c)p_1(w) + \eta(c)p_2(w) = \sum_w \lambda(w)p_1(w) + \eta(c) = 1 \]

**In-class:** To ensure a valid distribution, what are \( \lambda \) and \( \eta \) with \( D = 1? \)
Answer

\[ \lambda(w) = \frac{[F_{c,w} - D]^+}{F_{c,w}} \]

\[ \eta(w) = 1 - \sum_w \frac{[F_{c,w} - D]^+}{F_{c,\cdot}} = \sum_w \frac{F_{c,\cdot} - [F_{c,w} - D]^+}{F_{c,\cdot}} \]

Total counts minus counts discounted.

\[ \eta(w) = \frac{D \times N_{c,\cdot}}{F_{c,\cdot}} \]
Lower-Order Smoothing Issue

Consider the example: San Francisco and Los Franciso

Would like:

- $p(w_{i-1} = \text{San}, w_i = \text{Francisco})$ to be high
- $p(w_{i-1} = \text{Los}, w_i = \text{Francisco})$ to be very low

However, interpolation alone doesn’t ensure this. Why?

$$(1 - \lambda)p(w_i = \text{Francisco})$$

Could be quite high, from seeing San Francisco many times.
Lower-Order Smoothing Issue

Consider the example: San Francisco and Los Francisco
Would like:

- \( p(w_{i-1} = \text{San}, w_i = \text{Francisco}) \) to be high
- \( p(w_{i-1} = \text{Los}, w_i = \text{Francisco}) \) to be very low

However, interpolation alone doesn’t ensure this. Why?

\[
(1 - \lambda)p(w_i = \text{Francisco})
\]

Could be quite high, from seeing San Francisco many times.
Kneser-Ney Smoothing

- Uses Absolute Discounting instead of ML
- However want distribution to match ML on lower-order terms.
- Ensure this by marginalizing out and matching lower-order distribution.
- Uses a different function for the lower-order term KN'.
- Most commonly used smoothing technique (details follow).
Review: Chain-Rule and Marginalization

Chain rule:

\[ p(X, Y) = P(X|Y)P(Y) \]

Marginalization:

\[ p(X, Y) = \sum_{z \in Z} P(X, Y, Z = z) \]

Marginal matching constraint.

\[ p_A(X, Y) \neq p_B(X, Y) \]

\[ \sum_y p_A(X, Y = y) = p_B(X) \]
Kneser-Ney Smoothing

Main Idea: match ML marginals for all $c'$

$$\sum_{c_1} p_{KN}(c_1, c', w) = \sum_{c_1} p_{KN}(w|c)p_{ML}(c) = p_{ML}(c', w)$$

$$\sum_{c_1} p_{KN}(w|c) \frac{F_{c,:}}{F_{,:}} = \frac{F_{c',w,:}}{F_{,:}}$$

$$[ c_1 \quad c' \quad w ]$$
Kneser-Ney Smoothing

\[
p_{KN} = \frac{[F_{c,w} - D]^+}{F_{c,\cdot}} + \eta(c) p_{KN'}(w | c')
\]

\[
F_{c',w,\cdot} = \sum_{c_1} F_{c,\cdot} [p_{KN}(w | c)]
\]

\[
= \sum_{c_1} F_{c,\cdot} \left[ \frac{[F_{c,w} - D]^+}{F_{c,\cdot}} + \frac{D}{F_{c,\cdot}} N_{c,\cdot} \times p_{KN'}(w | c') \right]
\]

\[
= \sum_{c_1 : N_{c,w} > 0} F_{c,\cdot} \frac{F_{c,w} - D}{F_{c,\cdot}} + \sum_{c_1} DN_{c,\cdot} \times p_{KN'}(w | c')
\]

\[
= F_{c',w} - N_{c',w} D + DN_{c',\cdot} \times p_{KN'}(w | c')
\]
Kneser-Ney Smoothing

Final equation for unigram

\[ p_{KN'}(w) = \frac{N_{.,w}}{N_{.,.}}. \]

- Intuition: Prob of a unique bigrams ending in \( w \).

\[ p_{KN'}(w|c') = \frac{N_{c',.,w}}{N_{c',.}}. \]

- Intuition: Prob of a unique ngram (with middle \( c' \)) ending in \( w \).
Modified Kneser-Ney

Set $D$ based on $F_{c,w}$

$$D = \begin{cases} 
0 & F_{c,w} = 0 \\
D_1 & F_{c,w} = 1 \\
D_2 & F_{c,w} = 2 \\
D_{3+} & F_{c,w} \geq 3 
\end{cases}$$

- Modifications to the formula are in the paper
Language Model Issues

1. LMs become very sparse.
   - Obviously cannot use matrices ($|\mathcal{V}| \times |\mathcal{V}| \times |\mathcal{V}| > 10^{12}$)

2. LMs are very big

3. Lookup speed is crucial
Issues?
Reverse Trie Data structure

Used in several standard language modeling toolkits.
Efficiency: Hash Tables

- KenLM finds it more efficient to directly hash ngrams.
- All ngrams of a given weight are kept in a hash table.
- Fast linear-probing hash tables make this work well.
Quantization

- Memory issues are often a concern.
- Probabilities are kept in log space.
- KenLM quantizes from 32 bits to smaller (others use 8 bits).
- Quantization is done by sorting, binning, and averaging.
Finite State Automata
Conclusion

Today,

- Language Modeling
- Various ideas in smoothing
- Tricks for efficient implementation.

Next time,

- Neural Network Language models