Algorithms for NLP

Language Modeling II
Yulia Tsvetkov – CMU

Slides: Taylor Berg-Kirkpatrick – CMU/UCSD
Dan Klein – UC Berkeley
My legal name is Alexander Perchov. But all of my many friends dub me Alex, because that is a more flaccid-to-utter version of my legal name. Mother dubs me Alexi-stop-spleening-me!, because I am always spleening her. If you want to know why I am always spleening her, it is because I am always elsewhere with friends, and disseminating so much currency, and performing so many things that can spleen a mother. Father used to dub me Shapka, for the fur hat I would don even in the summer month. He ceased dubbing me that because I ordered him to cease dubbing me that. It sounded boyish to me, and I have always thought of myself as very potent and generative.
The Noisy-Channel Model

- We want to predict a sentence given acoustics:
  \[ w^* = \arg \max_w P(w|a) \]

- The noisy-channel approach:
  \[ w^* = \arg \max_w P(w|a) \]
  \[ = \arg \max_w P(a|w)P(w)/P(a) \]
  \[ = \arg \max_w P(a|w)P(w) \]

Likelihood
Acoustic model: HMMs over word positions with mixtures of Gaussians as emissions

Prior
Language model: Distributions over sequences of words (sentences)
ASR Components

Language Model

source \( P(w) \)

\[ \text{best } w \]

Acoustic Model

channel \( P(a|w) \)

\[ \text{observed } a \]

\[ \text{argmax } P(w|a) = \text{argmax } P(a|w)P(w) \]
MT System Components

Language Model

source
P(e)

Translation Model

cchannel
P(f|e)

encoder

decoder

best
e

argmax P(e|f) = argmax P(f|e)P(e)
the station signs are in deep in english -14732
the stations signs are in deep in english -14735
the station signs are in deep into english -14739
the station 's signs are in deep in english -14740
the station signs are in deep in the english -14741
the station signs are indeed in english -14757
the station 's signs are indeed in english -14760
the station signs are indians in english -14790
the station signs are indian in english -14799
the stations signs are indians in english -14807
the stations signs are indians and english -14815
Language Models

- A language model is a distribution over sequences of words (sentences)

\[ P(w) = P(w_1 \ldots w_n) \]

- What’s \( w \)? (closed vs open vocabulary)
- What’s \( n \)? (must sum to one over all lengths)
- Can have rich structure or be linguistically naive

- Why language models?
  - Usually the point is to assign high weights to plausible sentences (cf acoustic confusions)
  - This is not the same as modeling grammaticality
Language Models

- Language models are distributions over sentences
  \[ P(w_1 \ldots w_n) \]

- N-gram models are built from local conditional probabilities
  \[ P(w_1 \ldots w_n) = \prod_i P(w_i|w_{i-k} \ldots w_{i-1}) \]

- The methods we’ve seen are backed by corpus n-gram counts
  \[ \hat{P}(w_i|w_{i-1}, w_{i-2}) = \frac{c(w_{i-2}, w_{i-1}, w_i)}{c(w_{i-2}, w_{i-1})} \]
Kneser-Ney Smoothing

- All orders recursively discount and back-off:

\[
P_k(w|\text{prev}_{k-1}) = \frac{\max(c'(\text{prev}_{k-1}, w) - d, 0)}{\sum_v c'(\text{prev}_{k-1}, v)} + \alpha_{k-1} P_{k-1}(w|\text{prev}_{k-2})
\]

- Alpha is a function computed to make the probability normalize (see if you can figure out an expression).

- For the highest order, \(c'\) is the token count of the n-gram. For all others it is the context fertility of the n-gram: (see Chen and Goodman p. 18)

\[
c'(w) = |\{w_{k-1} : c(w_{k-1}, w) > 0\}|
\]

- The unigram base case does not need to discount.
- Variants are possible (e.g. different \(d\) for low counts)
What’s in an N-Gram?

▪ Just about every local correlation!
  ▪ Word class restrictions: “will have been ___”
  ▪ Morphology: “she ___”, “they ___”
  ▪ Semantic class restrictions: “danced the ___”
  ▪ Idioms: “add insult to ___”
  ▪ World knowledge: “ice caps have ___”
  ▪ Pop culture: “the empire strikes ___”

▪ But not the long-distance ones
  ▪ “The computer which I had just put into the machine room on the fifth floor ___.”
The LAMBADA dataset

Context: “Why?” “I would have thought you’d find him rather dry,” she said. “I don’t know about that,” said Gabriel. “He was a great craftsman,” said Heather. “That he was,” said Flannery.

Target sentence: “And Polish, to boot,” said _______.

Target word: Gabriel

[Paperno et al. 2016]
Other Techniques?

- Lots of other techniques
  - Maximum entropy LMs (soon)
  - Neural network LMs (soon)
  - Syntactic / grammar-structured LMs (much later)
How to Build an LM
Tons of Data

- Good LMs need lots of n-grams!

[Brants et al, 2007]
Storing Counts

- Key function: map from n-grams to counts

<table>
<thead>
<tr>
<th>n-gram</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>searching for the best</td>
<td>192593</td>
</tr>
<tr>
<td>searching for the right</td>
<td>45805</td>
</tr>
<tr>
<td>searching for the cheapest</td>
<td>44965</td>
</tr>
<tr>
<td>searching for the perfect</td>
<td>43959</td>
</tr>
<tr>
<td>searching for the truth</td>
<td>23165</td>
</tr>
<tr>
<td>searching for the &quot;</td>
<td>19086</td>
</tr>
<tr>
<td>searching for the most</td>
<td>15512</td>
</tr>
<tr>
<td>searching for the latest</td>
<td>12670</td>
</tr>
<tr>
<td>searching for the next</td>
<td>10120</td>
</tr>
<tr>
<td>searching for the lowest</td>
<td>10080</td>
</tr>
<tr>
<td>searching for the name</td>
<td>8402</td>
</tr>
<tr>
<td>searching for the finest</td>
<td>8171</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
All Our N-gram are Belong to You
Thursday, August 3, 2006
Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training corpora containing at most a few billion words, we have been harnessing the vast power of Google’s datacenters and distributed processing infrastructure to process larger and larger training corpora. We found that there is no data like more data, and scaled up the size of our data by one order of magnitude, and then another, and then one more – resulting in a training corpus of one trillion words from public Web pages.

We believe that the entire research community can benefit from access to such massive amounts of data. It will advance the state of the art, it will focus research in the promising direction of large-scale, data-driven approaches, and it will allow all research groups, no matter how large or small their computing resources, to play together. That’s why we decided to share the enormous dataset with everyone. We processed 1,924,656,767,229 words of running text and are publishing the counts for all 1,116,407,664 five-word sequences that appear at least 40 times. There are 13,988,301 unique words, after discarding words that appear less than 200 times.

Watch for an announcement at the Linguistics Data Consortium (LDC), who will be distributing it soon, and then order your set of DVDs. And let us hear from you – we’re excited to hear what you will do with the data, and we’re always interested in feedback about this dataset, or other potential datasets that might be useful for the research community.

Update (22 Sept. 2004): The LDC now has the data available in their catalog. The counts are as follows:

- File sizes: approx. 24 GB compressed (gzip’ed) text files
- Number of tokens: 1,604,906,767,229
- Number of sentences: 85,139,665,584
Example: Google N-Grams

Google N-grams

- 14 million \(< 2^{24}\) words
- 2 billion \(< 2^{31}\) 5-grams
- 770,000 \(< 2^{20}\) unique counts
- 4 billion n-grams total

- 24GB compressed
- 6 DVDs
Efficient Storage
Naïve Approach

c(cat) = 12 \quad \text{hash(cat)} = 2

c(the) = 87 \quad \text{hash(the)} = 2

c(and) = 76 \quad \text{hash(and)} = 5

c(dog) = 11 \quad \text{hash(dog)} = 7

c(have) = ? \quad \text{hash(have)} = 2
HashMap<String, Long> ngram_counts;

String ngram1 = "I have a car";
String ngram2 = "I have a cat";

ngram_counts.put(ngram1, 123);
ngram_counts.put(ngram2, 333);
HashMap<String[], Long> ngram_counts;

String[] ngram1 = {"I", "have", "a", "car"};
String[] ngram2 = {"I", "have", "a", "cat"};

ngram_counts.put(ngram1, 123);
ngram_counts.put(ngram2, 333);
A Simple Java Hashmap?

```
HashMap<String[], Long> ngram_counts;
```

Per 3-gram:
1. Pointer = 8 bytes
2. Map.Entry = 8 bytes (obj) + 3x8 bytes (pointers)
3. Long = 8 bytes (obj) + 8 bytes (long)
4. String[] = 8 bytes (obj) + 3x8 bytes (pointers)

… at best Strings are canonicalized

Total: > 88 bytes

Obvious alternatives:
- Sorted arrays
- Open addressing
# Open Address Hashing

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

- $c(\text{cat}) = 12$  
  hash(\text{cat}) = 2
- $c(\text{the}) = 87$  
  hash(\text{the}) = 2
- $c(\text{and}) = 76$  
  hash(\text{and}) = 5
- $c(\text{dog}) = 11$  
  hash(\text{dog}) = 7
### Open Address Hashing

<table>
<thead>
<tr>
<th>word</th>
<th>hash function</th>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the</td>
<td>2</td>
<td></td>
<td>cat</td>
</tr>
<tr>
<td>and</td>
<td>5</td>
<td></td>
<td>the</td>
</tr>
<tr>
<td>dog</td>
<td>7</td>
<td></td>
<td>and</td>
</tr>
<tr>
<td>have</td>
<td>2</td>
<td></td>
<td>dog</td>
</tr>
</tbody>
</table>

- $c(\text{cat}) = 12$, $\text{hash(cat)} = 2$
- $c(\text{the}) = 87$, $\text{hash(the)} = 2$
- $c(\text{and}) = 76$, $\text{hash(and)} = 5$
- $c(\text{dog}) = 11$, $\text{hash(dog)} = 7$
- $c(\text{have}) = ?$, $\text{hash(have)} = 2$
Open Address Hashing

c(cat) = 12  hash(cat) = 2

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

c(the) = 87  hash(the) = 2

c(and) = 76  hash(and) = 5

c(dog) = 11  hash(dog) = 7
Efficient Hashing

- Closed address hashing
  - Resolve collisions with chains
  - Easier to understand but bigger

- Open address hashing
  - Resolve collisions with probe sequences
  - Smaller but easy to mess up

- Direct-address hashing
  - No collision resolution
  - Just eject previous entries
  - Not suitable for core LM storage
HashMap<String[], Long> ngram_counts;

Per 3-gram:
- 1 Pointer = 8 bytes
- 1 Map.Entry = 8 bytes (obj) + 3x8 bytes (pointers)
- 1 Long = 8 bytes (obj) + 8 bytes (long)
- 1 String[] = 8 bytes (obj) + 3x8 bytes (pointers)

... at best Strings are canonicalized

Total: > 88 bytes

Obvious alternatives:
- Sorted arrays
- Open addressing
Got 3 numbers under $2^{20}$ to store?

7
0...00111

1
0...00001

15
0...01111

20 bits  20 bits  20 bits

Fits in a primitive 64-bit long
n-gram encoding

15176595 = 20 bits 20 bits 20 bits

the cat laughed 233
count

32 bytes → 8 bytes
c(the) = 23135851162 < 2^{35}

35 bits to represent integers between 0 and 2^{35}

60 bits
\[15176595\] n-gram encoding

35 bits
\[233\] count
Example: Google N-Grams

Google N-grams

- 14 million < $2^{24}$ words
- 2 billion < $2^{31}$ 5-grams
- 770,000 < $2^{20}$ unique counts
- 4 billion n-grams total

- 24GB compressed
- 6 DVDs
# unique counts = 770000 < 2^{20}

20 bits to represent ranks of all counts

<table>
<thead>
<tr>
<th>rank</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>51</td>
</tr>
<tr>
<td>3</td>
<td>233</td>
</tr>
</tbody>
</table>

15176595 n-gram encoding → 60 bits

3 ranks → 20 bits
So Far

### Vocabulary

<table>
<thead>
<tr>
<th>word</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>0</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>ran</td>
<td>3</td>
</tr>
</tbody>
</table>

### N-gram encoding scheme

- **unigram:** \( f(id) = id \)
- **bigram:** \( f(id_1, id_2) = ? \)
- **trigram:** \( f(id_1, id_2, id_3) = ? \)

### Counts lookup

<table>
<thead>
<tr>
<th>rank</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>51</td>
</tr>
<tr>
<td>3</td>
<td>233</td>
</tr>
</tbody>
</table>

### Count DB

- **unigram**
  - 1607820 0381
  - 15176595 0051
  - 15176583 0076
  - 16576628 0021
  - 15176600 0018
  - 16089320 0171
  - 15176583 0039
  - 14980420 0030
  - 15020330 0482

- **bigram**
  - 1607820 0381
  - 15176595 0051
  - 15176583 0076
  - 16576628 0021
  - 15176600 0018
  - 16089320 0171
  - 15176583 0039
  - 14980420 0030
  - 15020330 0482

- **trigram**
  - 1607820 0381
  - 15176595 0051
  - 15176583 0076
  - 16576628 0021
  - 15176600 0018
  - 16089320 0171
  - 15176583 0039
  - 14980420 0030
  - 15020330 0482
# Hashing vs Sorting

<table>
<thead>
<tr>
<th>Sorting</th>
<th>query: 15176595</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c )</td>
<td>( val )</td>
</tr>
<tr>
<td>15176583</td>
<td>0076</td>
</tr>
<tr>
<td>15176595</td>
<td>0051</td>
</tr>
<tr>
<td>15176600</td>
<td>0018</td>
</tr>
<tr>
<td>16078820</td>
<td>0381</td>
</tr>
<tr>
<td>16089320</td>
<td>0171</td>
</tr>
<tr>
<td>16576628</td>
<td>0021</td>
</tr>
<tr>
<td>16980420</td>
<td>0030</td>
</tr>
<tr>
<td>17020330</td>
<td>0482</td>
</tr>
<tr>
<td>17176583</td>
<td>0039</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hashing</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( c )</td>
<td>( val )</td>
</tr>
<tr>
<td>16078820</td>
<td>0381</td>
</tr>
<tr>
<td>15176595</td>
<td>0051</td>
</tr>
<tr>
<td>15176583</td>
<td>0076</td>
</tr>
<tr>
<td>16576628</td>
<td>0021</td>
</tr>
<tr>
<td>15176600</td>
<td>0018</td>
</tr>
<tr>
<td>16089320</td>
<td>0171</td>
</tr>
<tr>
<td>15176583</td>
<td>0039</td>
</tr>
<tr>
<td>14980420</td>
<td>0030</td>
</tr>
<tr>
<td>15020330</td>
<td>0482</td>
</tr>
</tbody>
</table>
Context Tries
Tries

[Image of a diagram showing the structure of a trie with examples of words and their associated values]

[Reference: Hsu and Glass 2008]
Context Encodings

Google N-grams
- 10.5 bytes/n-gram
- 37 GB total

[Many details from Pauls and Klein, 2011]
## Context Encodings

<table>
<thead>
<tr>
<th>1-grams</th>
<th>2-grams</th>
<th>3-grams</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>w</strong></td>
<td><strong>val</strong></td>
<td><strong>c</strong></td>
</tr>
<tr>
<td>675</td>
<td>0127</td>
<td>15176582</td>
</tr>
<tr>
<td>676</td>
<td>9008</td>
<td>15176583</td>
</tr>
<tr>
<td>677</td>
<td>0137</td>
<td>15176584</td>
</tr>
<tr>
<td>678</td>
<td>0090</td>
<td>15176585</td>
</tr>
<tr>
<td>679</td>
<td>1192</td>
<td>15176586</td>
</tr>
<tr>
<td>680</td>
<td>0050</td>
<td>15176587</td>
</tr>
<tr>
<td>681</td>
<td>0040</td>
<td>15176588</td>
</tr>
<tr>
<td>682</td>
<td>0201</td>
<td>15176589</td>
</tr>
<tr>
<td>683</td>
<td>3010</td>
<td>15176590</td>
</tr>
</tbody>
</table>

**Notes:**
- **w** represents the word.
- **val** represents the value.
- **c** represents the context.
- **20 bits** indicates the length of the context.
- **64 bits** indicates the length of the word.
- **20 bits** indicates the length of the context.
- **42276773** represents the index of the word in the encoding.
N-Gram Lookup

this is a 4-gram

$$p(0121\ 0374\ 0045\ 4820) = -8.7$$
Compression
Idea: Differential Compression

<table>
<thead>
<tr>
<th>$c$</th>
<th>$w$</th>
<th>$val$</th>
</tr>
</thead>
<tbody>
<tr>
<td>15176585</td>
<td>678</td>
<td>3</td>
</tr>
<tr>
<td>15176587</td>
<td>678</td>
<td>2</td>
</tr>
<tr>
<td>15176593</td>
<td>678</td>
<td>1</td>
</tr>
<tr>
<td>15176613</td>
<td>678</td>
<td>8</td>
</tr>
<tr>
<td>15179801</td>
<td>678</td>
<td>1</td>
</tr>
<tr>
<td>15176585</td>
<td>680</td>
<td>298</td>
</tr>
<tr>
<td>15176589</td>
<td>680</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\Delta c$</th>
<th>$\Delta w$</th>
<th>$val$</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>+0</td>
<td>2</td>
</tr>
<tr>
<td>+6</td>
<td>+0</td>
<td>1</td>
</tr>
<tr>
<td>+40</td>
<td>+0</td>
<td>8</td>
</tr>
<tr>
<td>+188</td>
<td>+0</td>
<td>1</td>
</tr>
<tr>
<td>+2</td>
<td>+0</td>
<td>1</td>
</tr>
</tbody>
</table>

| $|\Delta w|$ | $|\Delta c|$ | $|val|$ |
|-------------|-------------|-------|
| 40          | 24          | 3     |
| 3           | 2           | 3     |
| 3           | 2           | 3     |
| 9           | 2           | 6     |
| 12          | 2           | 3     |
| 36          | 4           | 15    |
| 6           | 2           | 3     |

| 15176585    | 678 | 563097887 | 956 | 3 | +2 | +0 | 2 | +6 | +0 | 1 | +40 | +2 | 8 | ... |
Variable Length Encodings

Encoding “9”

000 1001

Length in Unary  Number in Binary

Google N-grams
- 2.9 bytes/n-gram
- 10 GB total

[Elias, 75]
Speed-Ups
Rolling Queries

this is + a 4-gram
12438010 0045 4820

12438010 0045 a
this is a

15176583 4820
is a 4-gram

<table>
<thead>
<tr>
<th>c</th>
<th>w</th>
<th>val</th>
<th>suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>15176583</td>
<td>682</td>
<td>0065</td>
<td>00000480</td>
</tr>
<tr>
<td>15176595</td>
<td>682</td>
<td>0808</td>
<td>00000675</td>
</tr>
<tr>
<td>15176600</td>
<td>682</td>
<td>0012</td>
<td>00000802</td>
</tr>
<tr>
<td>16078820</td>
<td>682</td>
<td>0400</td>
<td>00001321</td>
</tr>
</tbody>
</table>

LM \[ val \] -7.8
LM \[ val \] -5.4

14986731
Idea: Fast Caching

<table>
<thead>
<tr>
<th>n-gram</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>124 80 42 1243</td>
</tr>
<tr>
<td>1</td>
<td>37 2435 243 21</td>
</tr>
<tr>
<td>2</td>
<td>804 42 4298 43</td>
</tr>
</tbody>
</table>

hash(124 80 42 1243) = 0

hash(1423 43 42 400) = 1

LM can be more than 10x faster w/ direct-address caching
Approximate LMs

- Simplest option: hash-and-hope
  - Array of size $K \sim N$
  - (optional) store hash of keys
  - Store values in direct-address
  - Collisions: store the max
  - What kind of errors can there be?

- More complex options, like bloom filters (originally for membership, but see Talbot and Osborne 07), perfect hashing, etc
Homework 1 Overview