**Language Models**

- Language models are distributions over sentences
  \[ P(w) = P(w_1 \ldots w_n) \]
- N-gram models are built from local conditional probabilities
  \[ P(w_1 \ldots w_n) = \prod_{i=1}^{n} P(w_i | w_{i-1}, w_{i-2}) \]
- The methods we've seen are backed by corpus n-gram counts
  \[ P(w_i | w_{i-1}, w_{i-2}) = \frac{c(w_i, w_{i-1}, w_{i-2})}{c(w_{i-2}, w_{i-1})} \]

**N-Gram Demo**

**Tons of Data**

- Good LMs need lots of n-grams!

---

**Storing Counts**

- Key function: map from n-grams to counts

```
searching for the best 192593
searching for the right 45805
searching for the cheapest 43589
searching for the perfect 23165
searching for the 19836
searching for the most 15512
searching for the latest 12670
searching for the next 10120
searching for the lowest 10080
searching for the name 8402
searching for the finest 8171
...
```

---

**Example: Google N-Grams**

- 14 million < 2^{24} words
- 2 billion < 2^{31} 5-grams
- 770 000 < 2^{22} unique counts
- 4 billion n-grams total
Efficient Storage

A Simple Java Hashmap?

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

- at least Strings are canonicalized
- Total = 88 bytes

Obvious alternatives:
- Sorted arrays
- Open addressing

Integer Encodings

Got 3 numbers under $2^{20}$ to store?

- 20 bits
- 20 bits
- 20 bits

Fits in a primitive 64-bit long

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

Bit Packing

Rank Values
Context Tries

Tries

Context Encodings

Context Encodings

N-Gram Lookup

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>679</td>
<td>0</td>
</tr>
<tr>
<td>15176901</td>
<td>678</td>
<td>1</td>
</tr>
<tr>
<td>15176585</td>
<td>489</td>
<td>298</td>
</tr>
<tr>
<td>15176589</td>
<td>689</td>
<td>1</td>
</tr>
</tbody>
</table>

Δc | Δw | val |
---|----|-----|
-2 | +0 | 2   |
+6 | +0 | 1   |
+40 | +0 | 9   |
+180 | +0 | 1   |
+2 | +0 | 1   |
+2 | +0 | 2   |

<table>
<thead>
<tr>
<th>Δc</th>
<th>Δw</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>36</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

### Variable Length Encodings

<table>
<thead>
<tr>
<th>Encoding “9”</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 1001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>in Unary</td>
<td>in Binary</td>
</tr>
</tbody>
</table>

Google N-grams
- 5.9 bytes/gram
- 3.2 GB total

[Elia, 75]

### Speed-Ups

### 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 244 21</td>
</tr>
<tr>
<td>2</td>
<td>894 42 4290 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

### Rolling Queries

### Approximate LMs

- Simplest option: hash-and-hope
  - Array of size K = 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.
Maximum Entropy Models

Improving on N-Grams?
- N-grams don’t combine multiple sources of evidence well
- Here:
  - “the” gives syntactic constraint
  - “demolition” gives semantic constraint
- Unlike the interaction between these two has been densely observed
- We’d like a model that can be more statistically efficient

Maximum Entropy LMs
- Want a model over completions \( y \) given a context \( x \):
  \[
  P(y|x) = P(\text{close the door} | \text{close the})
  \]
- Want to characterize the important aspects of \( y = (v,x) \) using a feature function \( f \)
- \( F \) might include
  - Indicator of \( v \) (unigram)
  - Indicator of \( v \), previous word (bigram)
  - Indicator whether \( v \) occurs in \( x \) (cache)
  - Indicator of \( v \) and each non-adjacent previous word
  - …

Some Definitions
- INPUTS \( X_i \)
- CANDIDATE SET \( \mathcal{Y}(x) \)
- CANDIDATES \( \mathcal{Y} \)
- TRUE OUTPUTS \( \mathcal{Y}^* \)
- FEATURING VECTORS \( f_i(y) \)

Linear Models: Maximum Entropy
- Maximum entropy (logistic regression)
  - Use the scores as probabilities:
    \[
    P(y|x, w) = \frac{\exp(w^T f_i(y))}{\sum_y \exp(w^T f_i(y))}
    \]
  - Make positive
  - Normalize
- Maximize the (log) conditional likelihood of training data
  \[
  L(w) = \log \prod_i P(y_i|x_i, w) = \sum_i \log \left( \frac{\exp(w^T f_i(y_i))}{\sum_y \exp(w^T f_i(y))} \right)
  \]
  \[
  = \sum_i \left( w^T f_i(y_i) - \log \sum_y \exp(w^T f_i(y)) \right)
  \]

Maximum Entropy II
- Motivation for maximum entropy:
  - Connection to maximum entropy principle (sort of)
  - Might want to do a good job of being uncertain on noisy cases...
  - … in practice, though, posteriors are pretty peaked
- Regularization (smoothing)
  \[
  \min_w \sum_i \left( w^T f_i(y_i) - \log \sum_y \exp(w^T f_i(y)) \right) - E[|w|^2]
  \]
  \[
  \min_w \frac{1}{2} \sum_i \left( w^T f_i(y_i) - \log \sum_y \exp(w^T f_i(y)) \right)
  \]
Derivative for Maximum Entropy

\[ I(w) = -k||w||^2 + \sum_t \left( w^T E(x_t) - \log \sum_y w^T E(y) \right) \]

\[ \frac{\partial I(w)}{\partial w} = -2k w + \sum_t \left( e(x_t) - \sum_y e(y|x_t) E(y) \right) \]

Big weights are bad
Expected feature vector over possible candidates
Total count of feature \( n \) in correct candidates

Convexity

- The maxent objective is nicely behaved:
  - Differentiable (so many ways to optimize)
  - Convex (so no local optima*)

\[ f(\lambda a + (1 - \lambda) b) \geq \lambda f(a) + (1 - \lambda) f(b) \]

Convexity guarantees a single, global maximum value because any higher points are greedily reachable

Unconstrained Optimization

- Once we have a function \( f \), we can find a local optimum by iteratively following the gradient

- For convex functions, a local optimum will be global
- Basic gradient ascent isn’t very efficient, but there are simple enhancements which take into account previous gradients: conjugate gradient, L-BFGS, AdaGrad now popular
- There are special-purpose optimization techniques for maxent, like iterative scaling, but they aren’t better