Natural Language Processing

Language Modeling II

Dan Klein – UC Berkeley

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

$$P(w_1 \ldots w_n) = \prod_i P(w_i | w_{i-1}, w_{i-2})$$

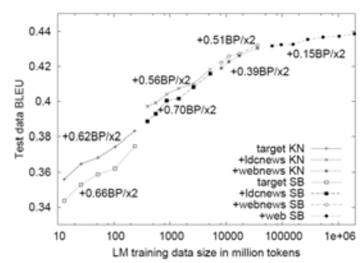
- The methods we've seen are backed by corpus n-gram counts

$$P(w_i | w_{i-1}, w_{i-2}) = \frac{c(w_{i-2}, w_{i-1}, w_i)}{c(w_{i-2}, w_{i-1})}$$

N-Gram Demo

Tons of Data

- Good LMs need lots of n-grams!



[Brants et al, 2007]

Storing Counts

- Key function: map from n-grams to counts

…	
searching for the best	192593
searching for the right	45805
searching for the cheapest	44965
searching for the perfect	43959
searching for the truth	23165
searching for the "	19086
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
…	

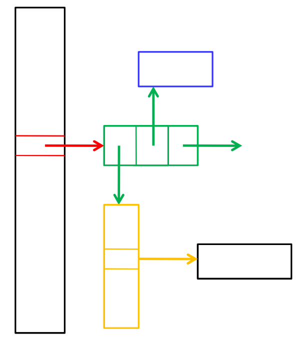
Example: Google N-Grams

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

Efficient Storage

A Simple Java Hashmap?



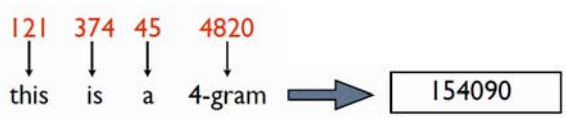
Per 3-gram:
- 1 Pointer = 8 bytes
- 1 Map.Entry = 8 bytes (obj) +3x8 bytes (pointers)
- 1 Double = 8 bytes (obj) + 8 bytes (double)
- 1 String[] = 8 bytes (obj) + + 3x8 bytes (pointers)

… at best Strings are canonicalized

Total: > 88 bytes

Obvious alternatives:
- Sorted arrays
- Open addressing

Integer Encodings



Bit Packing

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

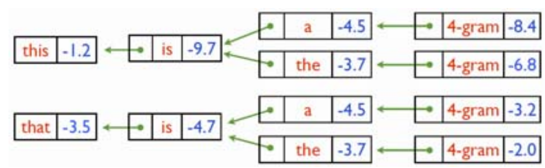
Rank Values

this is a 4-gram → -9.87

rank	prob	freq.
0	-8.7	8
1	-5.4	6
2	-7.6	3

Context Tries

Tries



[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

1-grams (w: 20 bits)

w	val	
675	0127	"this"
676	9008	
677	0137	
678	0090	"a"
679	1192	
680	0050	"the"
681	0040	
682	0201	"is"
683	3010	"was"

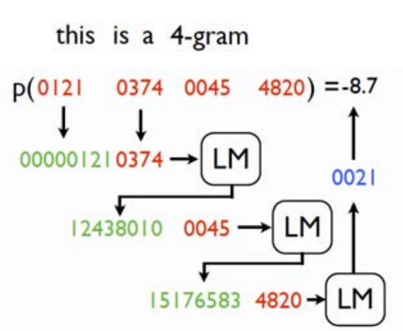
2-grams (c, w: 64 bits; val: 20 bits)

	c	w	val	
15176582	00000480	682	0065	"is"
15176583	00000675	682	0808	"is"
15176584	00000802	682	0012	"is"
15176585	00001321	682	0400	"is"
15176586	00002482	682	0030	"is"
15176587	00002588	682	0260	"is"
15176588	00000390	683	0013	"was"
15176589	00000676	683	0025	"was"
15176590	00000984	683	0086	"was"

3-grams (c, w: 64 bits; val: 20 bits)

	c	w	val	
42276773	15176583	678	0076	"a"
42276774	15176595	678	0051	"a"
42276775	15176600	678	0018	"a"
42276776	16078820	678	0381	"a"
42276777	16089320	678	0171	"a"
42276778	16576628	678	0021	"a"
42276779	14980420	680	0030	"the"
42276780	15020330	680	0482	"the"
42276781	15176583	680	0039	"the"

N-Gram Lookup



Compression

Idea: Differential Compression

c	w	val
15176585	678	3
15176587	678	2
15176593	678	1
15176613	678	8
15179801	678	1
15176585	680	298
15176589	680	1

Δc	Δw	val
15176583	678	3
+2	+0	2
+6	+0	1
+40	+0	8
+188	+0	1
15176585	+2	298
+4	+0	1

|Δw|	|Δc|	|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	0	+2	+0	2	+6	+0	1	+40	+2	8	. . .

Variable Length Encodings

Encoding "9"

$\underbrace{000}_{\text{Length in Unary}}\ \underbrace{1001}_{\text{Number in Binary}}$

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

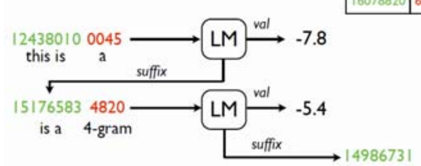
[Elias, 75]

Speed-Ups

Rolling Queries

c	w	val	suffix
15176583	682	0065	00000480
15176595	682	0808	00000675
15176600	682	0012	00000802
16078820	682	0400	00001321

this is + a 4-gram
12438010 0045 4820



Idea: Fast Caching

	n-gram	probability
0	124 80 42 1243	-7.034
1	37 2435 243 21	-2.394
2	804 42 4298 43	-8.008

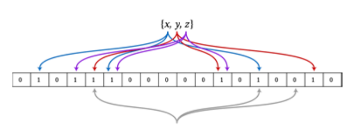
hash(124 80 42 1243) =0

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 ~ 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

  P(construction | After the demolition was completed, the)

- Here:
  - "the" gives syntactic constraint
  - "demolition" gives semantic constraint
  - Unlikely 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:

$$P(y|x) = P(\textit{ close the door } | \textit{ close the })$$

- 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	$\mathbf{x}_i$	*close the ____*
CANDIDATE SET	$\mathcal{Y}(\mathbf{x})$	*{close the door, close the table, …}*
CANDIDATES	$\mathbf{y}$	*close the table*
TRUE OUTPUTS	$\mathbf{y}_i^*$	*close the door*
FEATURE VECTORS	$\mathbf{f}_i(\mathbf{y})$	[0 0 0 0 1 0 1 0 0 0 0 0]

"close" in x ∧ v="door"

v_{-1}="the" ∧ v="door"

"door" in x and v

Linear Models: Maximum Entropy

- Maximum entropy (logistic regression)
  - Use the scores as probabilities:

$$P(\mathbf{y}|\mathbf{x},\mathbf{w}) = \frac{\exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}))}{\sum_{\mathbf{y}'} \exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}'))} \quad \begin{array}{l} \leftarrow \text{Make positive} \\ \leftarrow \text{Normalize} \end{array}$$

  - Maximize the (log) conditional likelihood of training data

$$L(\mathbf{w}) = \log \prod_i P(\mathbf{y}_i^*|\mathbf{x}_i,\mathbf{w}) = \sum_i \log \left(\frac{\exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*))}{\sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}))} \right)$$

$$= \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right)$$

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)

$$\max_{\mathbf{w}} \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right) - k||\mathbf{w}||^2$$

$$\min_{\mathbf{w}} k||\mathbf{w}||^2 - \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right)$$

Derivative for Maximum Entropy

$$L(\mathbf{w}) = -k||\mathbf{w}||^2 + \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}_i^*) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right)$$

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = -2k\mathbf{w} + \sum_i \left(\mathbf{f}_i(\mathbf{y}_i^*) - \sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}_i)\mathbf{f}_i(\mathbf{y}) \right)$$

Big weights are bad

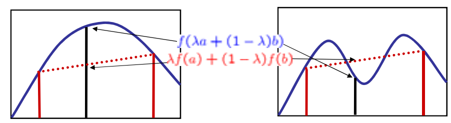
Expected feature vector
over possible candidates

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*)

$$f(\lambda a + (1-\lambda)b) \geq \lambda f(a) + (1-\lambda) f(b)$$

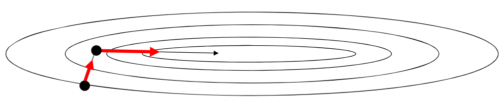


Convex Non-Convex

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