Algorithms for NLP

Language Modeling II

Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein – UC Berkeley

Announcements

- Should be able to really start project after today's lecture
- Get familiar with bit-twiddling in Java (e.g. &, |, <<, >>)
- No external libraries / code
- We will go over KN again in recitation – edge cases
- Tentative office hours:
  - Me:
  - Maria:
  - Hieu:
  - Akshay:

Language Models

- Language models are distributions over sentences

$$P(w_1 \dots w_n)$$

- N-gram models are built from local conditional probabilities

$$P(w_1 \dots w_n) = \prod_i P(w_i | w_{i-k} \dots w_{i-1})$$

- 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

- Kneser-Ney smoothing combines two ideas
  - Discount and reallocate like absolute discounting
  - In the backoff model, word probabilities are proportional to context fertility, not frequency

$$P(w) \propto |\{w' : c(w', w) > 0\}|$$

- Theory and practice
  - Practice: KN smoothing has been repeatedly proven both effective and efficient
  - Theory: KN smoothing as approximate inference in a hierarchical Pitman-Yor process [Teh, 2006]

Kneser-Ney Edge Cases

- All orders recursively discount and back-off:

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

- The unigram base case does not need to discount (though it can)
- Alpha is computed to make the probability normalize (but if context count is zero, then fully back-off)
- 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):

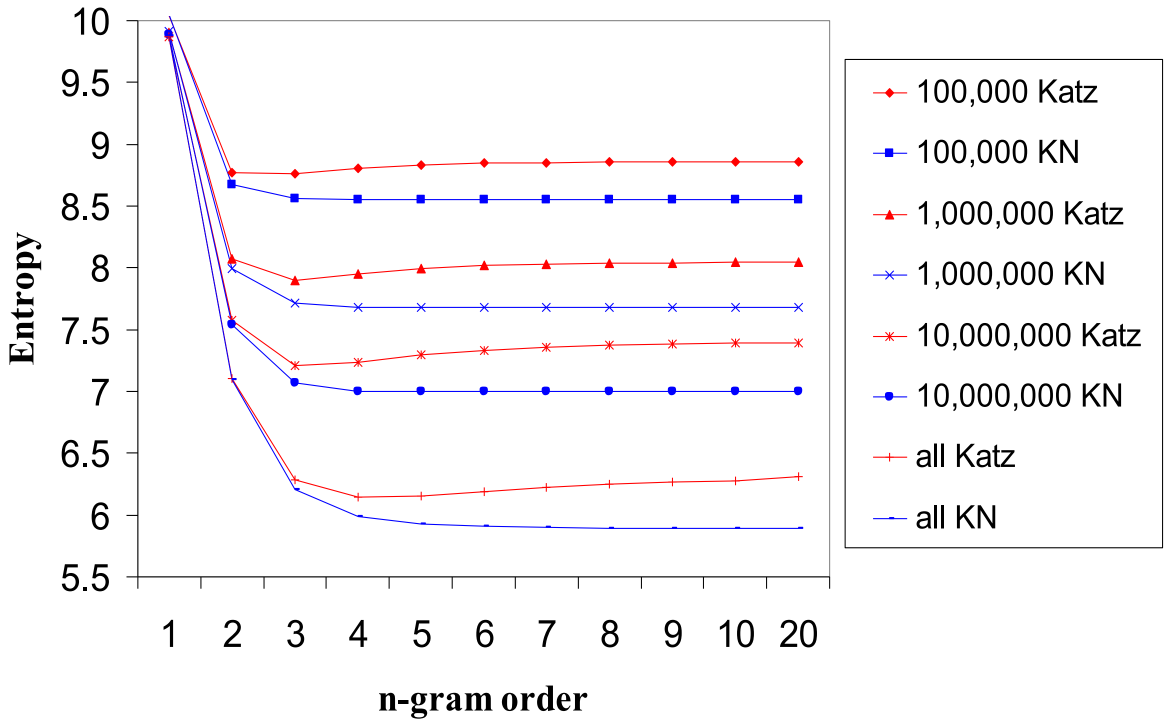
$$c'(x) = |\{u : c(u,x) > 0\}|$$

Idea 4: Big Data

There's no data like more data.

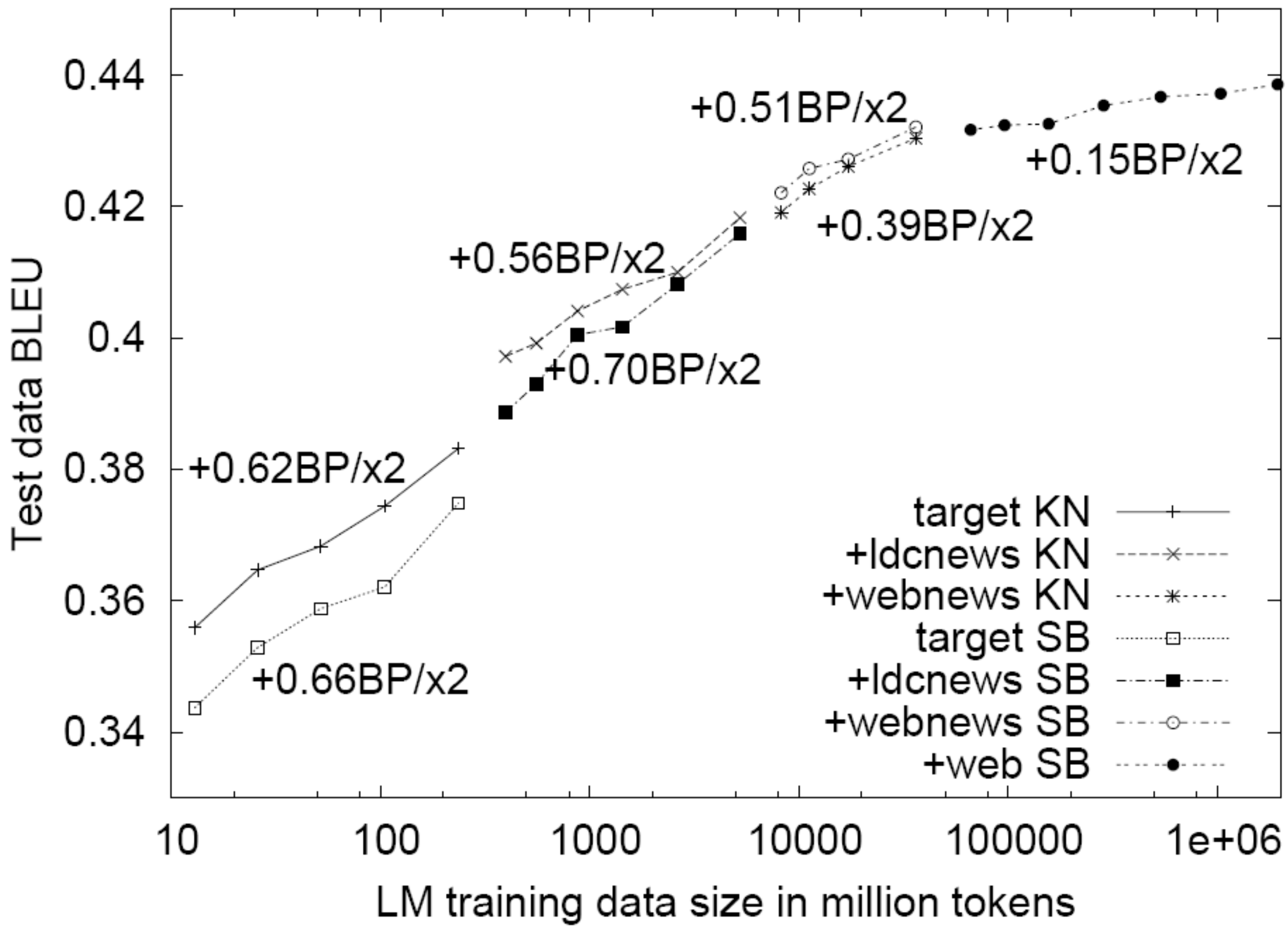
Data >> Method?

- Having more data is better…



- … but so is using a better estimator
- Another issue: N > 3 has huge costs in speech recognizers

Tons of Data?



[Brants et al, 2007]

What about…

Unknown Words?

- What about totally unseen words?
- Most LM applications are closed vocabulary
  - ASR systems will only propose words that are in their pronunciation dictionary
  - MT systems will only propose words that are in their phrase tables (modulo special models for numbers, etc)
- In principle, one can build open vocabulary LMs
  - E.g. models over character sequences rather than word sequences
  - Back-off needs to go down into a “generate new word” model
  - Typically if you need this, a high-order character model will do

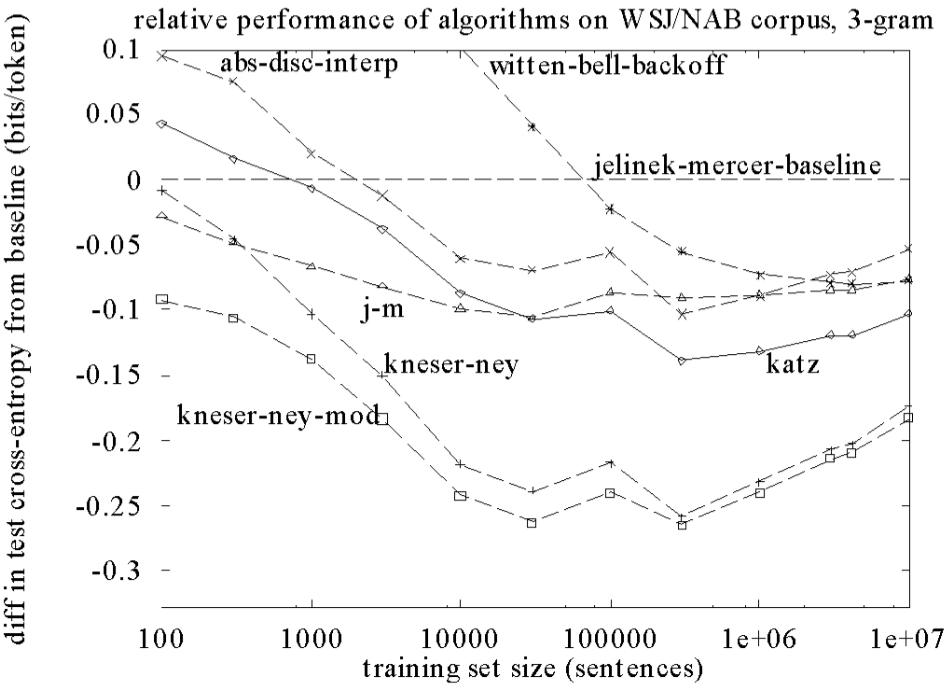
What's in an N-Gram?

- 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 ___."

What Actually Works?

- Trigrams and beyond:
  - Unigrams, bigrams generally useless
  - Trigrams much better
  - 4-, 5-grams and more are really useful in MT, but gains are more limited for speech
- Discounting
  - Absolute discounting, Good-Turing, held-out estimation, Witten-Bell, etc…
- Context counting
  - Kneser-Ney construction of lower-order models
- See [Chen+Goodman] reading for tons of graphs…



[Graph from Joshua Goodman]

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 ___.”

Linguistic Pain?

- The N-Gram assumption hurts one's inner linguist!
  - Many linguistic arguments that language isn't regular
    - Long-distance dependencies
    - Recursive structure
- Answers
  - N-grams only model local correlations, but they get them all
  - As N increases, they catch even more correlations
  - N-gram models scale much more easily than structured LMs
- Not convinced?
  - Can build LMs out of our grammar models (later in the course)
  - Take any generative model with words at the bottom and marginalize out the other variables

What Gets Captured?

- Bigram model:
  - [texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen]
  - [outside, new, car, parking, lot, of, the, agreement, reached]
  - [this, would, be, a, record, november]

- PCFG model:
  - [This, quarter, ‘s, surprisingly, independent, attack, paid, off, the, risk, involving, IRS, leaders, and, transportation, prices, .]
  - [It, could, be, announced, sometime, .]
  - [Mr., Toseland, believes, the, average, defense, economy, is, drafted, from, slightly, more, than, 12, stocks, .]

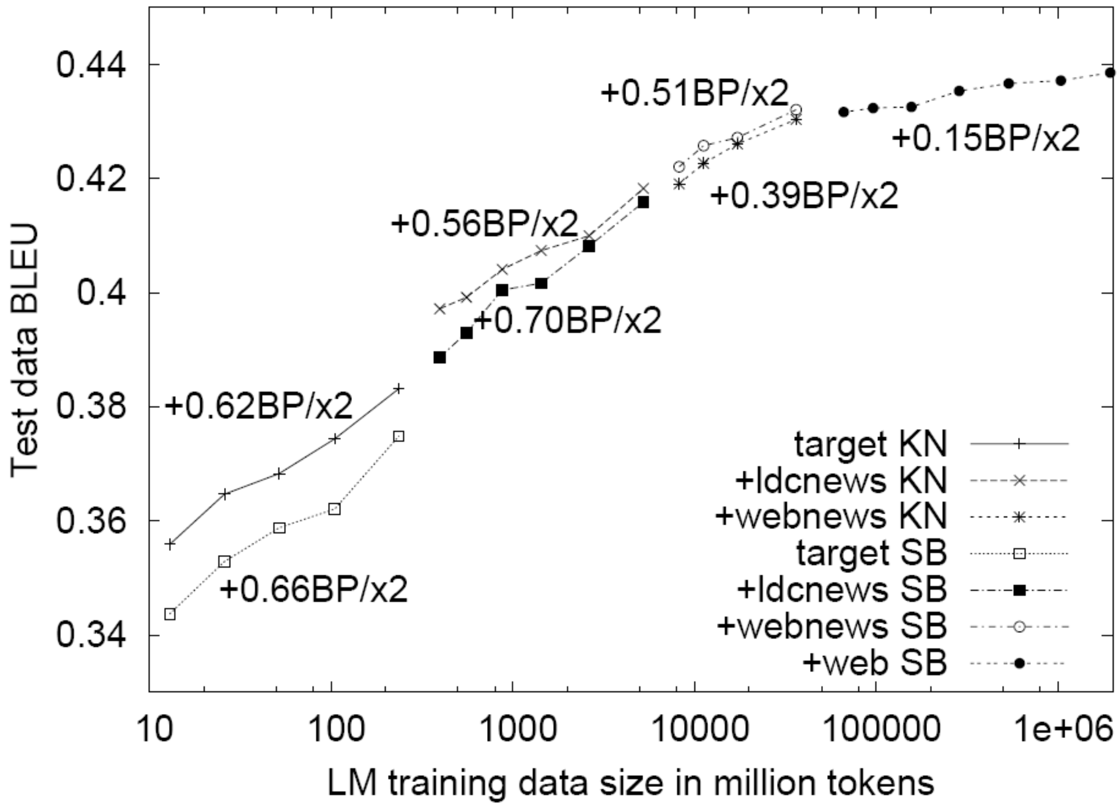
Other Techniques?

- 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

…	
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

Naïve Approach

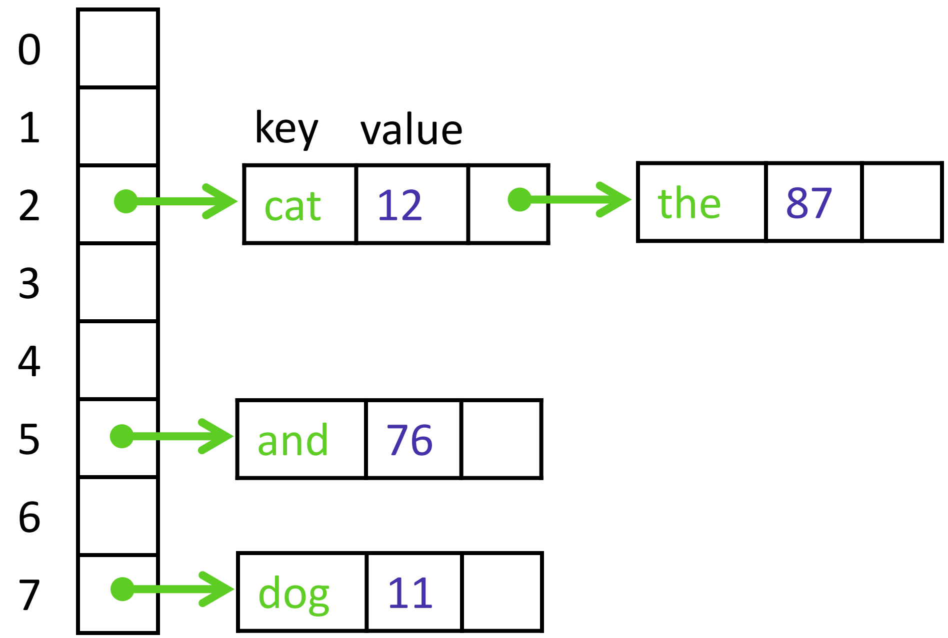
c(cat) = 12 hash(cat) = 2

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

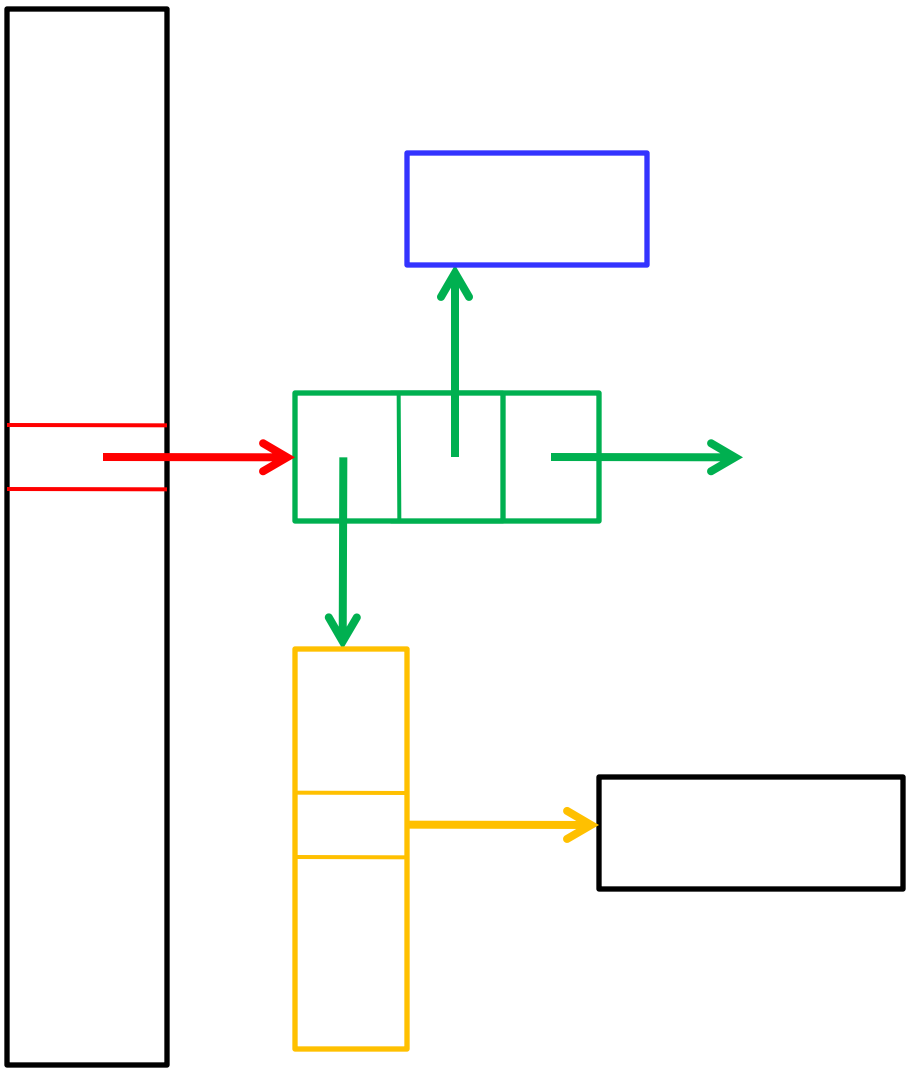
c(and) = 76 hash(and) = 5

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

c(have) = ? hash(have) = 2



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

Open Address Hashing

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

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

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

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

	key	value
0		
1		
2		
3		
4		
5		
6		
7		

Open Address Hashing

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

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

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

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

c(have) = ? hash(have) = 2

	key	value
0		
1		
2	cat	12
3	the	87
4		
5	and	5
6		
7	dog	7

Open Address Hashing

c(cat) = 12

c(the) = 87

c(and) = 76

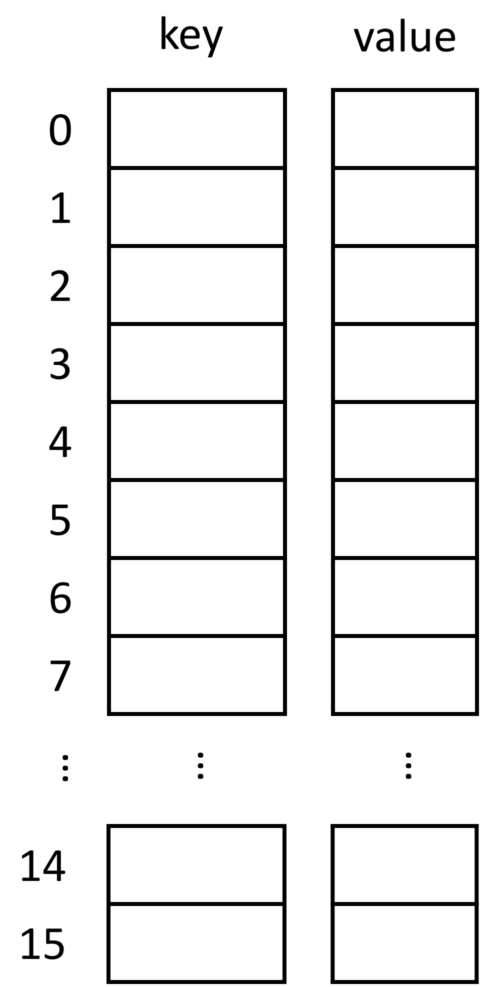
c(dog) = 11

~~hash(cat) = 2~~

~~hash(the) = 2~~

~~hash(and) = 5~~

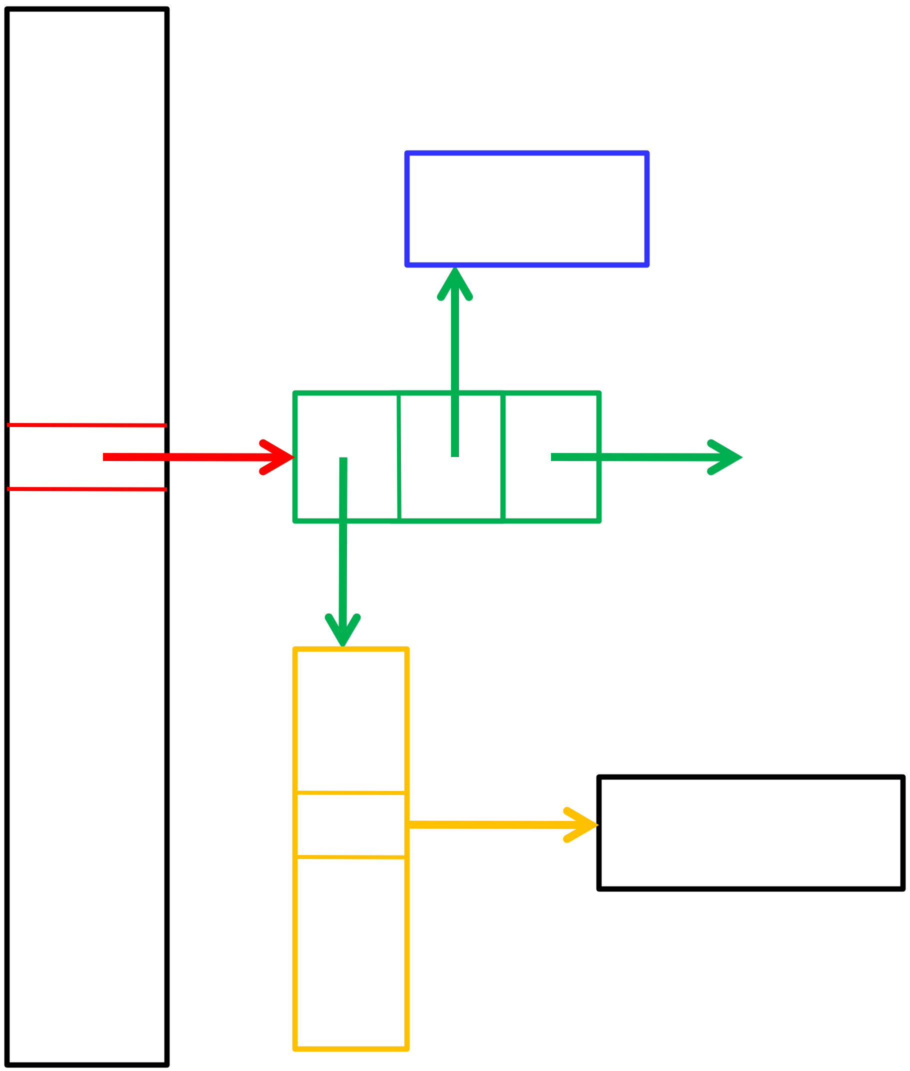
~~hash(dog) = 7~~



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

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

word ids

7 1 15

the cat laughed → 233

n-gram count

Bit Packing

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

7	1	15
0…00111	0…00001	0…01111
20 bits	20 bits	20 bits

Fits in a primitive 64-bit long

Integer Encodings

n-gram encoding

15176595 = | 20 bits | 20 bits | 20 bits |

~~the cat laughed~~ → 233

n-gram count

Rank Values

$$c(\text{the}) = 23135851162 < 2^{35}$$

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

$$\underbrace{\overbrace{15176595}^{\text{60 bits}}}_{\text{n-gram encoding}} \longrightarrow \underbrace{\overbrace{233}^{\text{35 bits}}}_{\text{count}}$$

Rank Values

\# unique counts = 770000 < 2^{20}

20 bits to represent ranks of all counts

$$\underbrace{\overbrace{15176595}^{\text{60 bits}}}_{\text{n-gram encoding}} \longrightarrow \underbrace{\overbrace{3}^{\text{20 bits}}}_{\text{rank}}$$

rank	freq
0	1
1	2
2	51
3	233

So Far

Word indexer

word	id
cat	0
the	1
was	2
ran	3

N-gram encoding scheme

unigram: $f(id) = id$

bigram: $f(id_1, id_2) = ?$

trigram: $f(id_1, id_2, id_3) = ?$

Rank lookup

rank	freq
0	1
1	2
2	51
3	233

Count DB

unigram

16078820	0381
15176595	0051
15176583	0076
—	—
16576628	0021
—	—
15176600	0018
16089320	0171
15176583	0039
14980420	0030
—	—
15020330	0482

bigram

16078820	0381
15176595	0051
15176583	0076
—	—
16576628	0021
—	—
15176600	0018
16089320	0171
15176583	0039
14980420	0030
—	—
15020330	0482

trigram

16078820	0381
15176595	0051
15176583	0076
—	—
16576628	0021
—	—
15176600	0018
16089320	0171
15176583	0039
14980420	0030
—	—
15020330	0482

Hashing vs Sorting

Sorting

c	val
15176583	0076
15176595	0051
15176600	0018
16078820	0381
16089320	0171
16576628	0021
16980420	0030
17020330	0482
17176583	0039

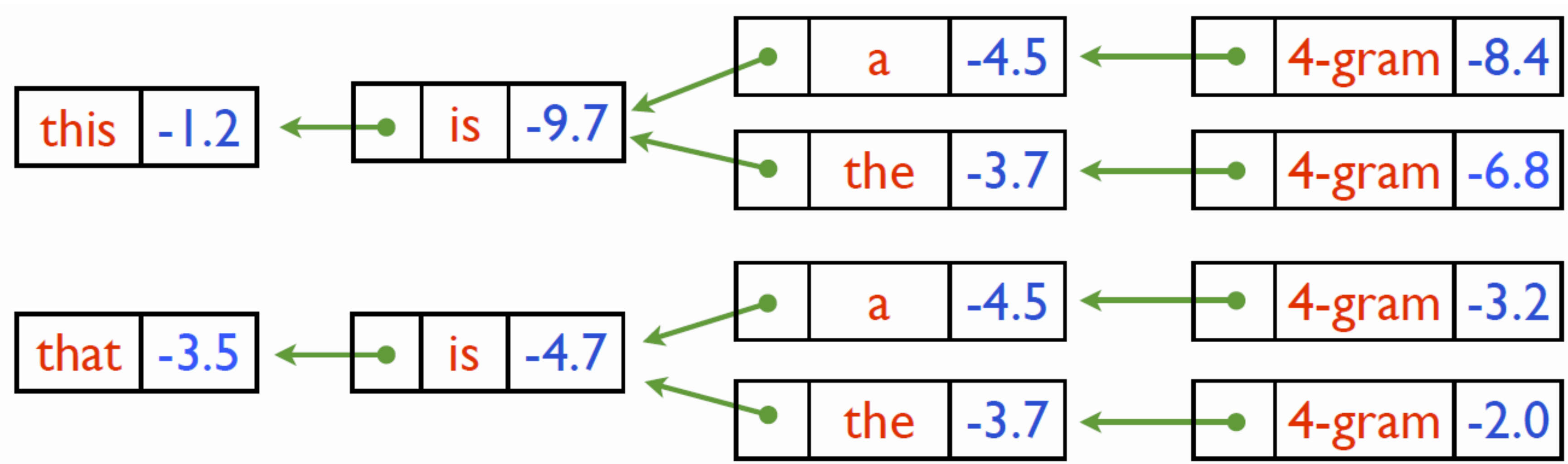
query: 15176595

Hashing

c	val
16078820	0381
15176595	0051
15176583	0076
—	—
16576628	0021
—	—
15176600	0018
16089320	0171
15176583	0039
14980420	0030
—	—
15020330	0482

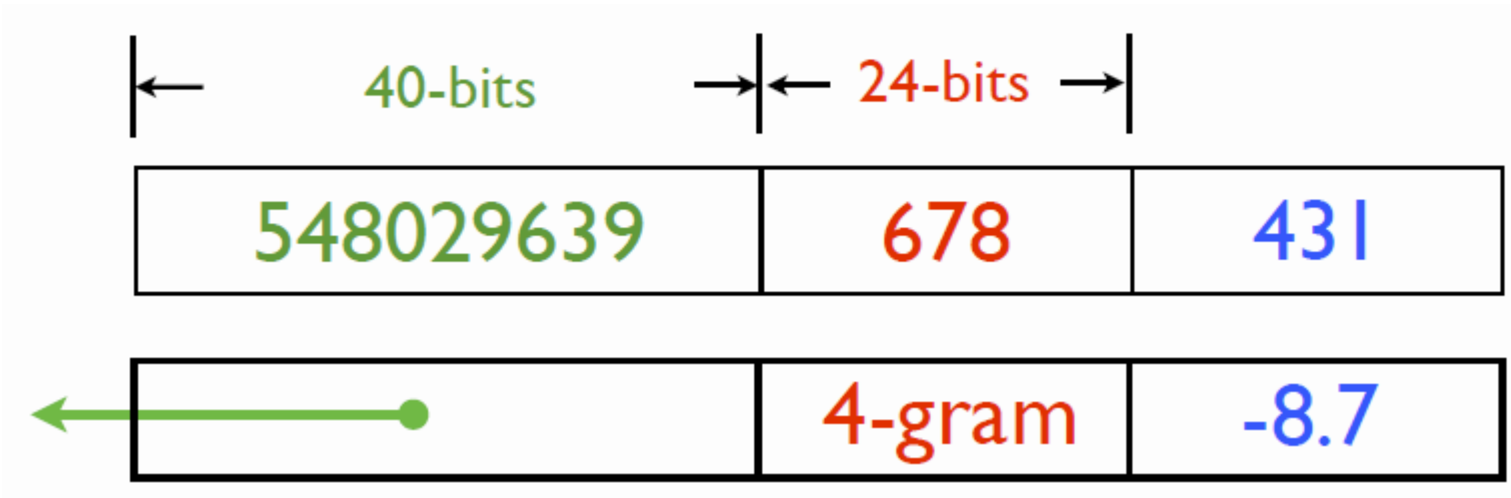
Context Tries

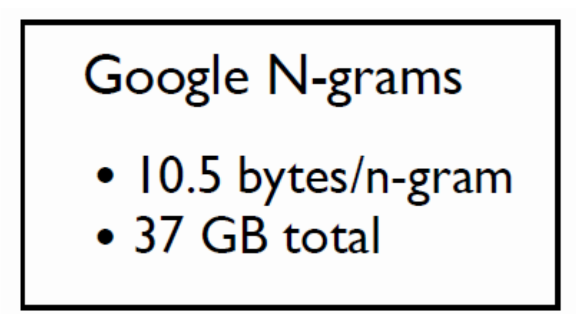
Tries



[Hsu and Glass 2008]

Context Encodings





[Many details from Pauls and Klein, 2011]

Context Encodings

1-grams

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"

val: 20 bits

2-grams

	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"

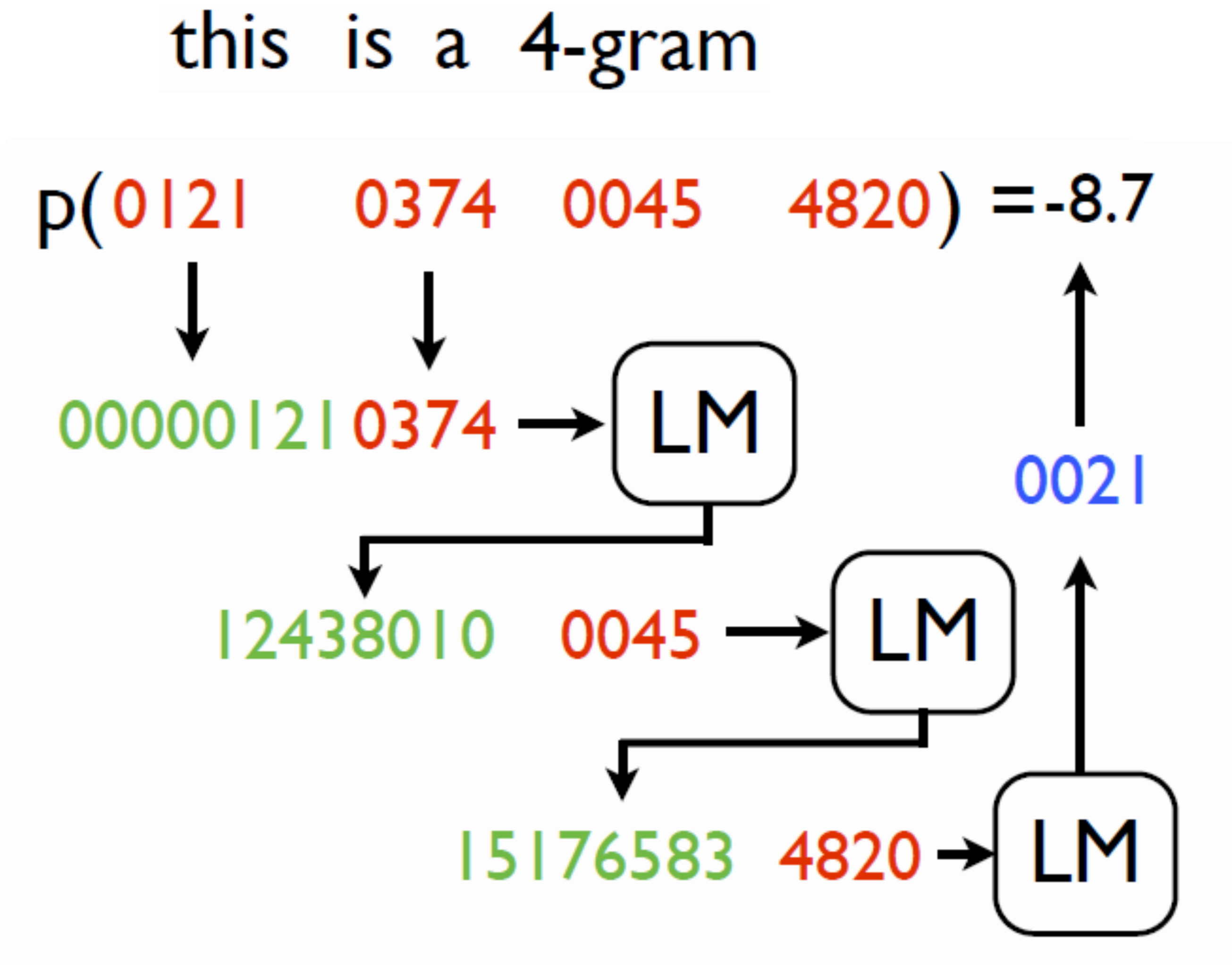
c + w: 64 bits; val: 20 bits

3-grams

	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"

c + w: 64 bits; val: 20 bits

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

$\lvert\Delta w\rvert$	$\lvert\Delta c\rvert$	$\lvert val\rvert$
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

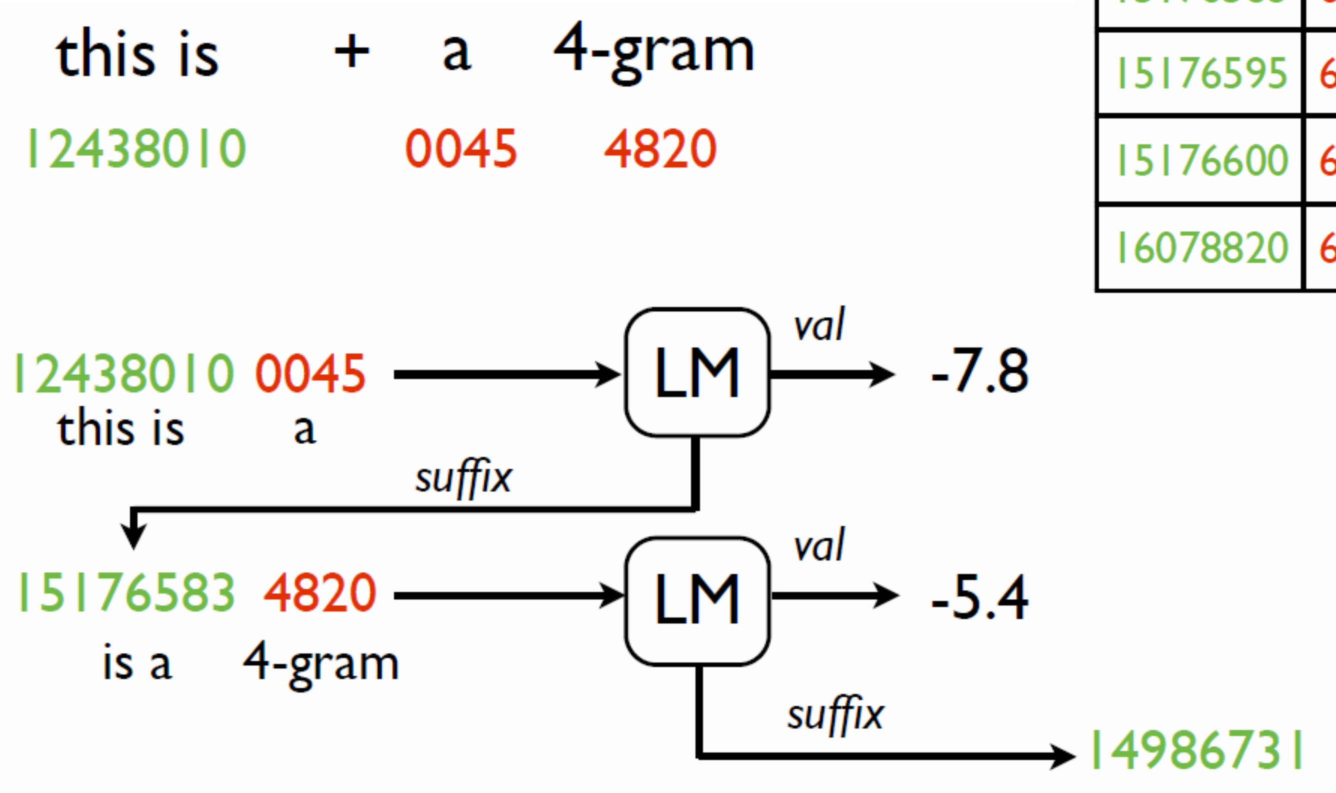
- 2.9 bytes/n-gram
- 10 GB total

[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



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

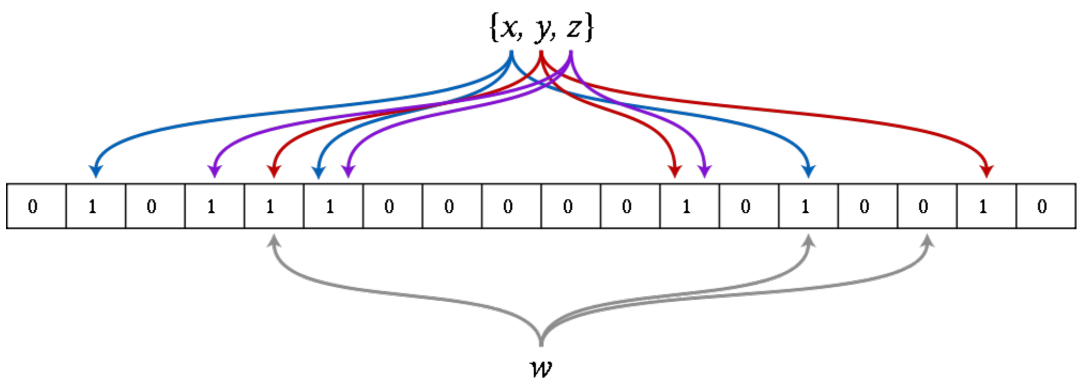
hash(1423 43 42 400) =1

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:

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

- 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:

$$\mathsf{P}(\mathbf{y}|\mathbf{x},\mathbf{w}) = \frac{\exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}))}{\sum_{\mathbf{y}'} \exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}'))}$$

Make positive

Normalize

  - Maximize the (log) conditional likelihood of training data

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

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

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

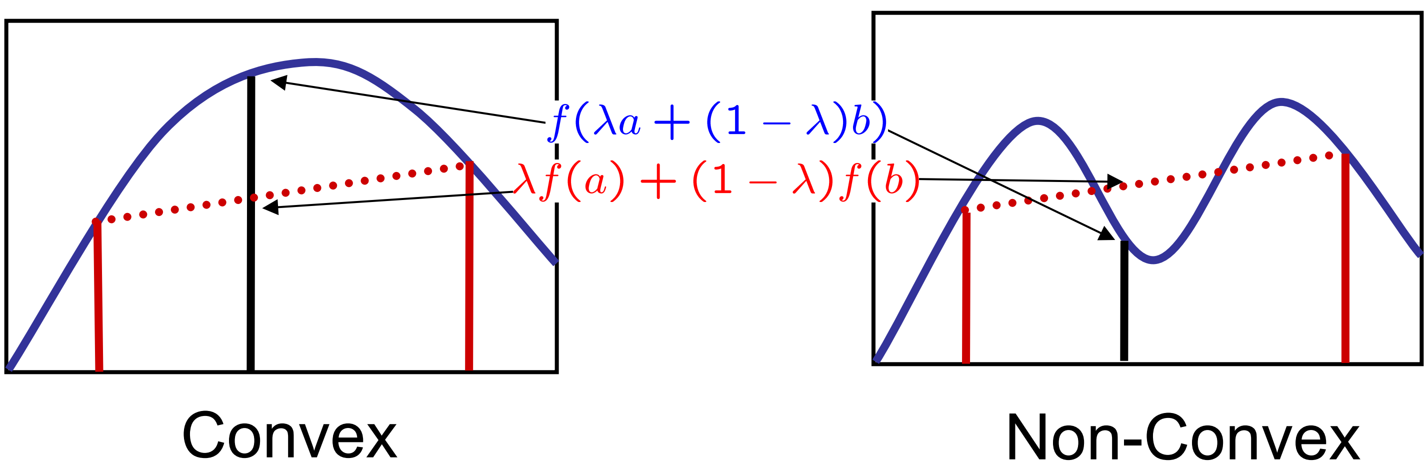
Expected feature vector over possible candidates

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*)

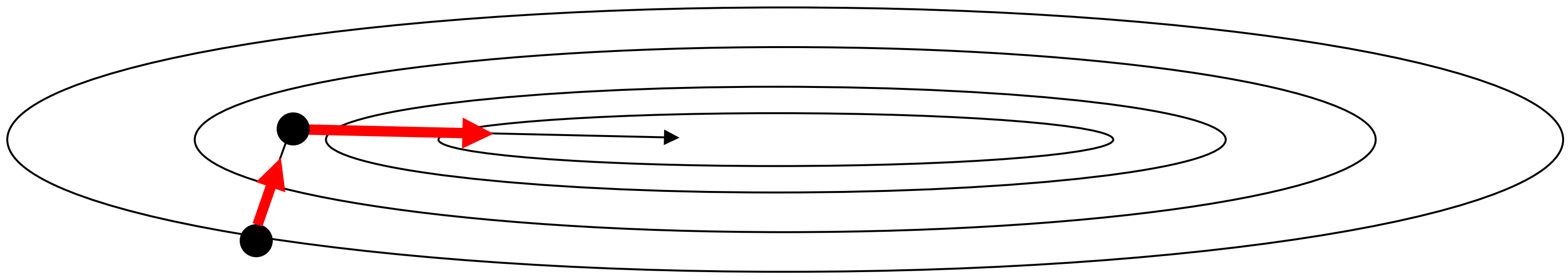
$$f(\lambda a + (1-\lambda)b) \geq \lambda f(a) + (1-\lambda) f(b)$$



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
- Online methods (e.g. AdaGrad) now very popular

Implicit Representation