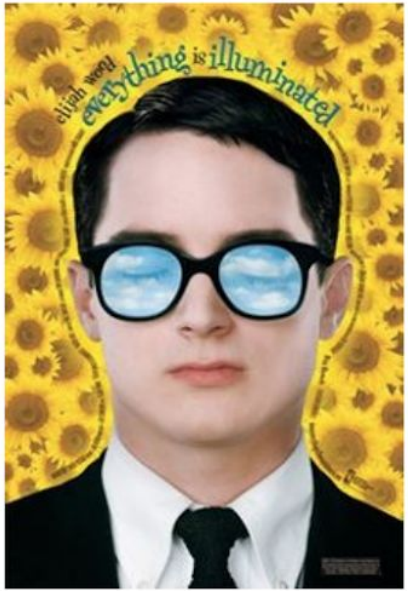
# Algorithms for NLP

## 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:

$$w^* = \arg\max_w P(w|a)$$

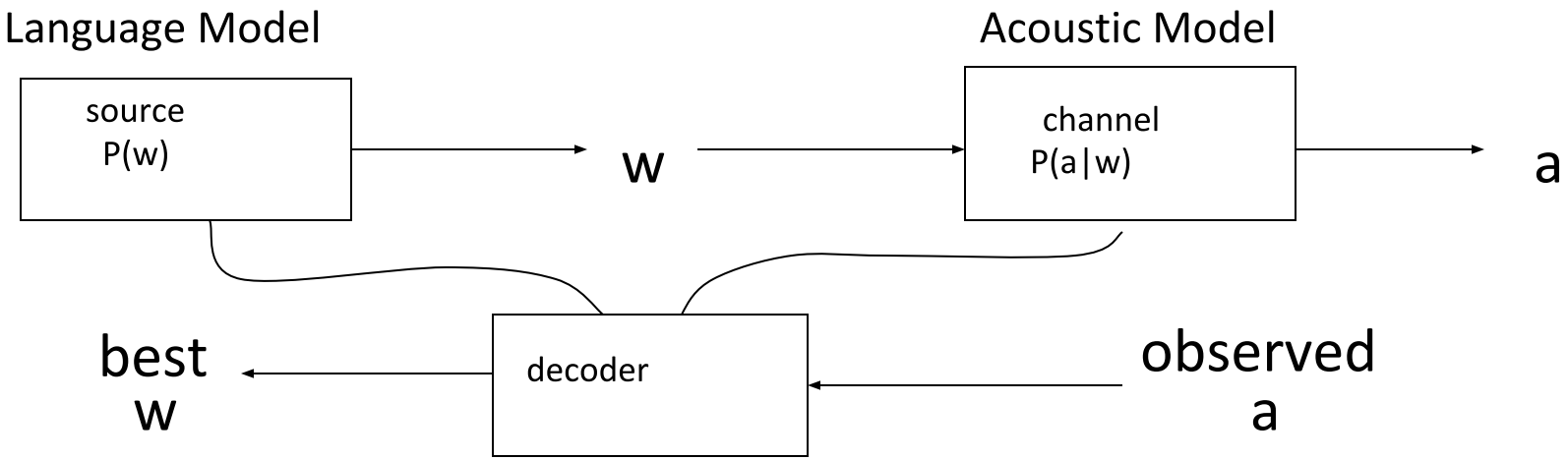
- The noisy-channel approach:

$$w^* = \arg\max_w P(w|a)$$
$$= \arg\max_w P(a|w)P(w)/P(a)$$
$$= \arg\max_w P(a|w)P(w)$$

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

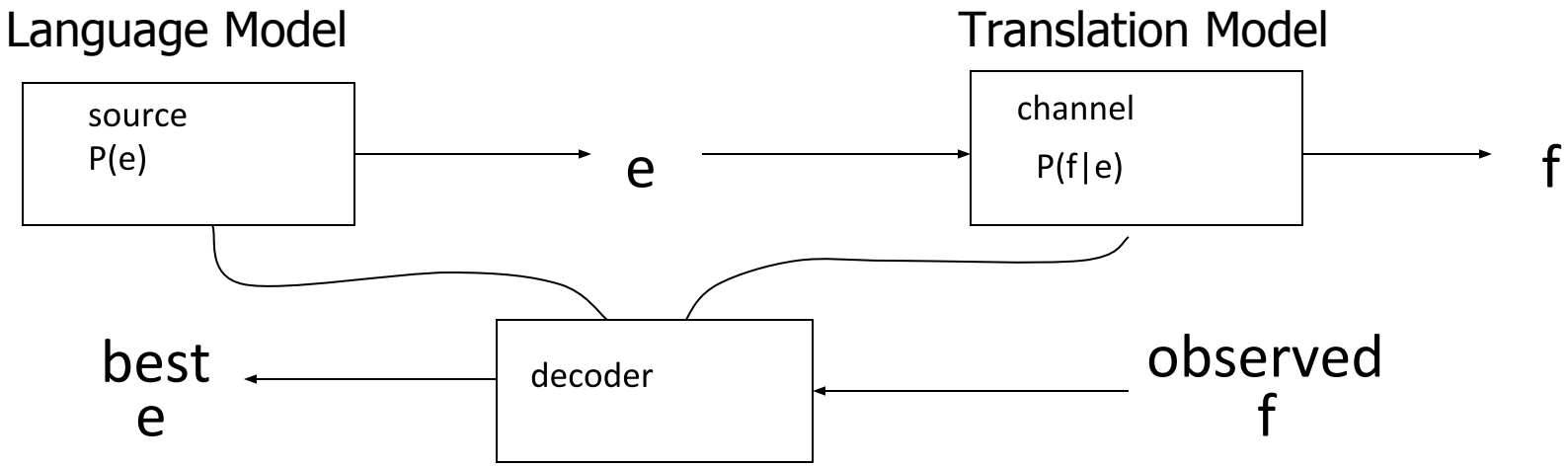
Prior
Language model: Distributions over sequences of words (sentences)

# ASR Components




$$\underset{w}{\operatorname{argmax}}\, P(w|a) = \underset{w}{\operatorname{argmax}}\, P(a|w)P(w)$$

# MT System Components




$$\underset{e}{\operatorname{argmax}}\, P(e|f) = \underset{e}{\operatorname{argmax}}\, P(f|e)P(e)$$

# Acoustic Confusions

| | |
|---|---|
| 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

$$P(w_1 \dots w_n)$$

- 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

- All orders recursively discount and back-off:

$$P_k(w|prev_{k-1}) = \frac{max(c'(prev_{k-1}, w) - d, 0)}{\Sigma_v c'(prev_{k-1}, v)} + \alpha_{k-1} P_{k-1}(w|prev_{k-2})$$

- 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 ___."

# Long-distance Predictions

## 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]

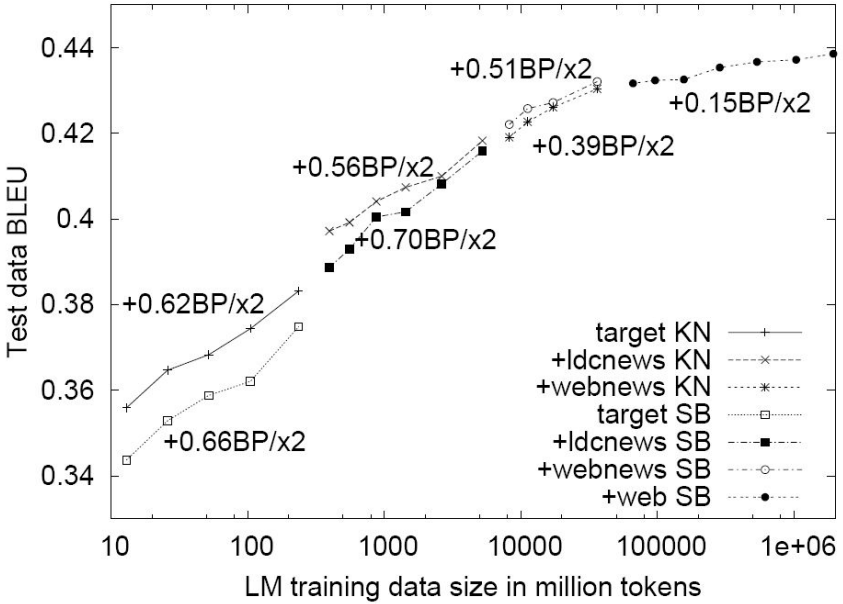
# Other Techniques?

- 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!



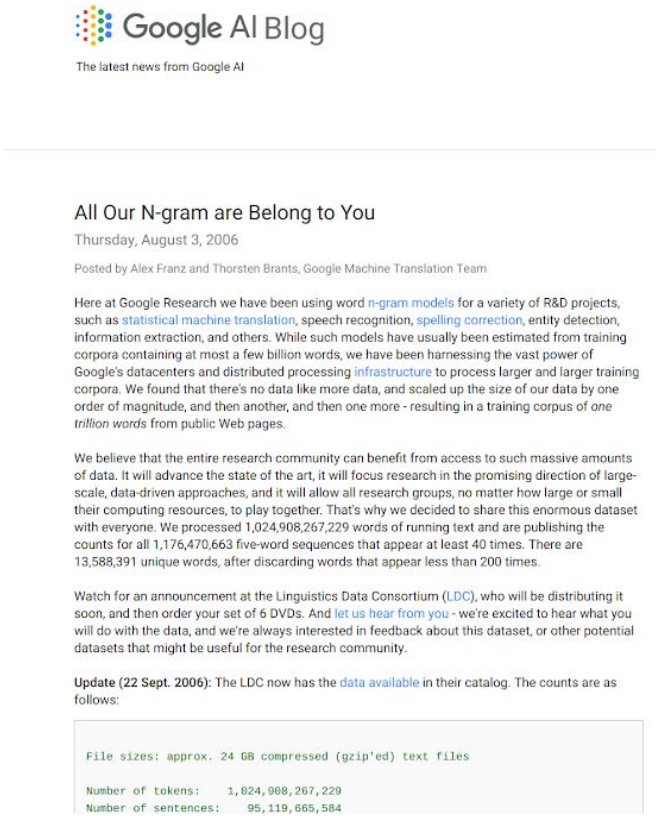

[Brants et al, 2007]

# Storing Counts

- Key function: map from n-grams to counts

| | |
|---|---|
| … | |
| 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 AI Blog

The latest news from Google AI

## 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's 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 this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 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 6 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. 2006):** 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:    1,024,908,267,229
Number of sentences:    95,119,665,584
```

# 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

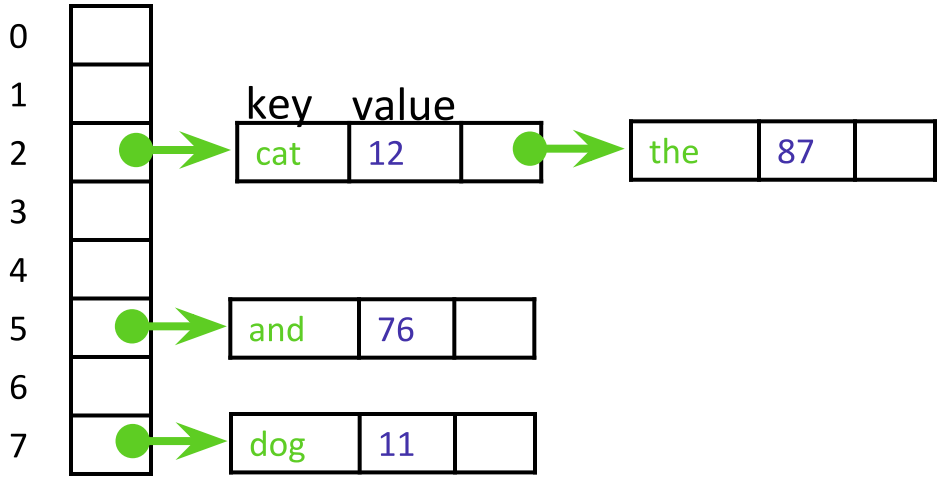
c(cat) = 12 hash(cat) = 2

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?

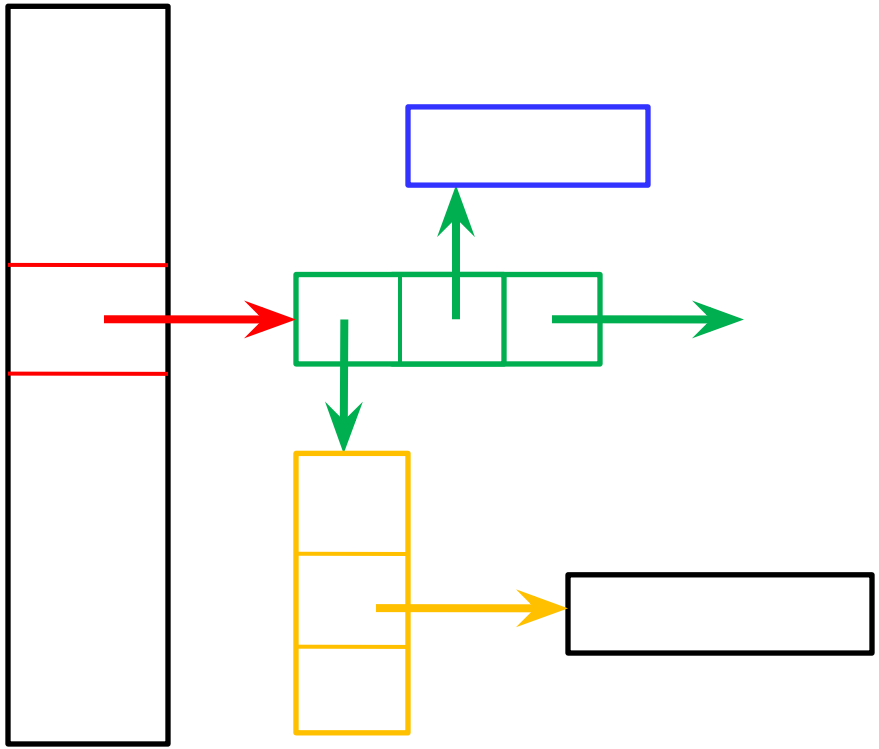
```
HashMap<String, Long> ngram_counts;



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;



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

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

# 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

| | key | value |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |

# Open Address Hashing

c(cat) = 12

c(the) = 87

c(and) = 76

c(dog) = 11

c(have) = ?

hash(cat) = 2

hash(the) = 2

hash(and) = 5

hash(dog) = 7

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

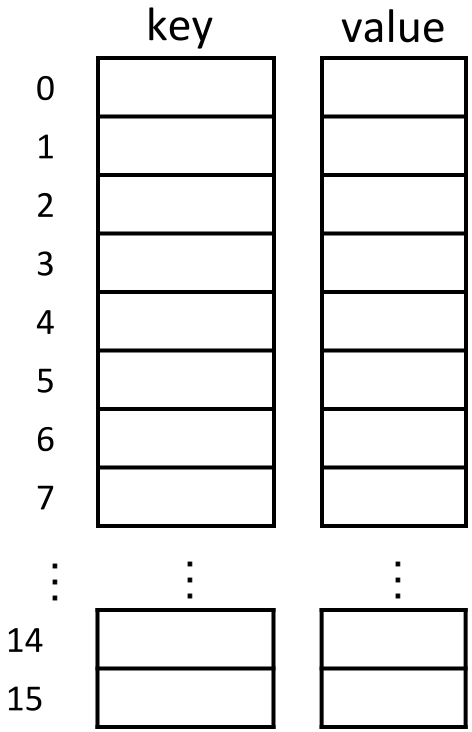
c(dog) = 11

hash(cat) = 2

hash(the) = 2

hash(and) = 5

hash(dog) = 7

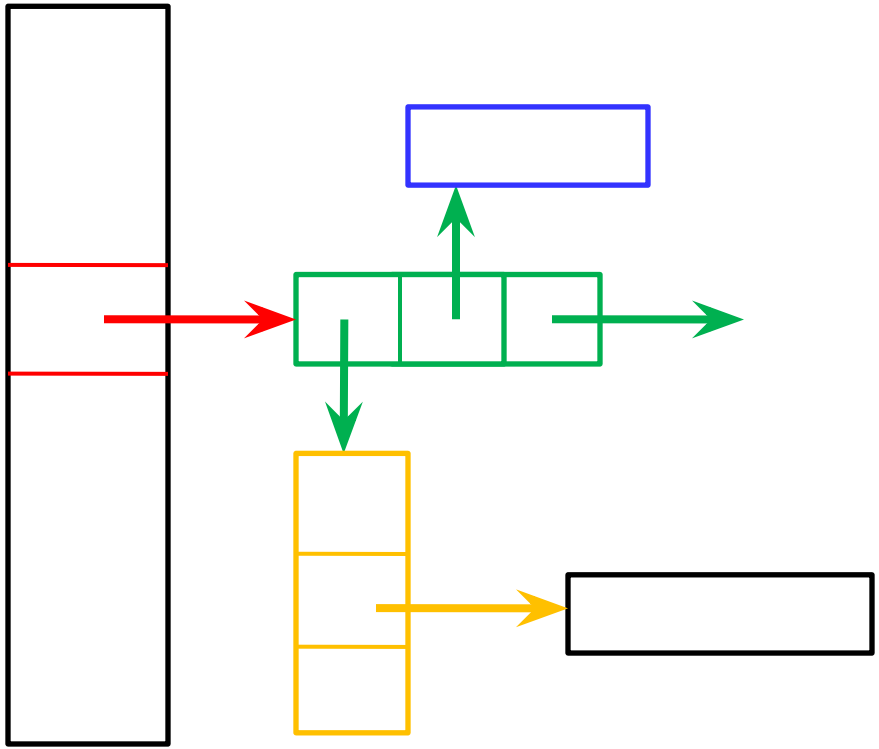


# Efficient Hashing

- 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?

```
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)

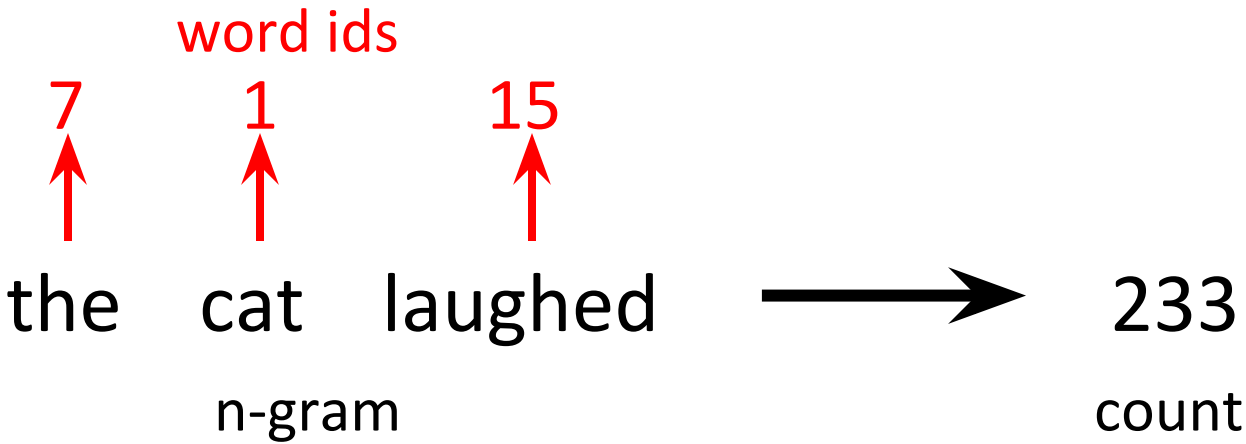
1 String[] = 8 bytes (obj) +
+ 3x8 bytes (pointers)

… at best Strings are canonicalized

Total: > 88 bytes

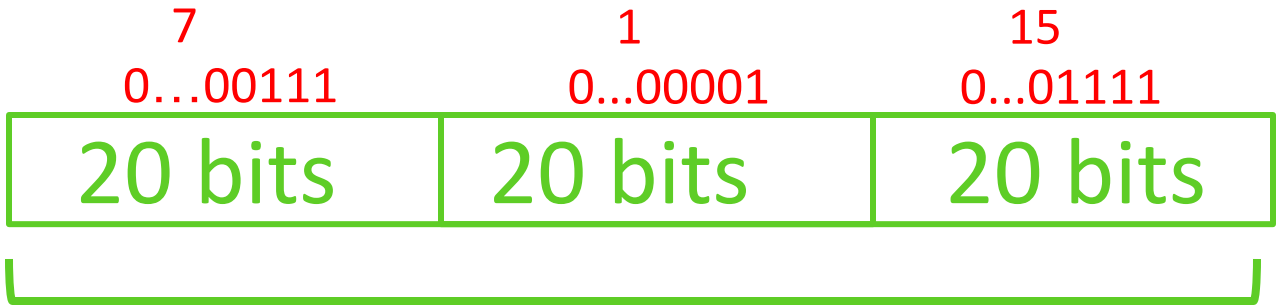
Obvious alternatives:
- Sorted arrays
- Open addressing

# Integer Encodings



# Bit Packing

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




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

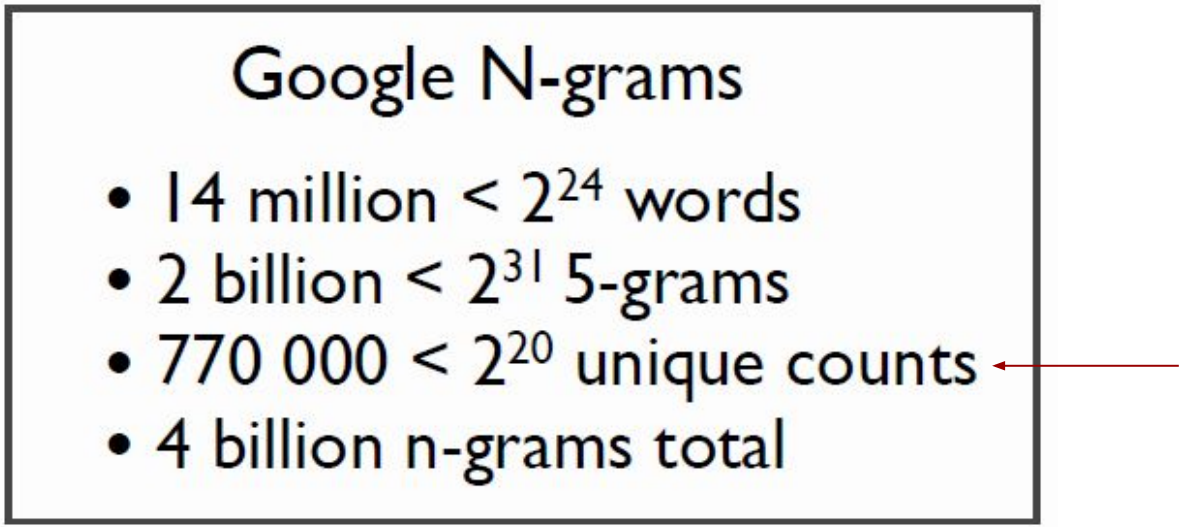
32 bytes → 8 bytes

# Rank Values

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

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

# Example: Google N-Grams




- 24GB compressed
- 6 DVDs

# Rank Values

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

20 bits to represent ranks of all counts


| rank | count |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 51 |
| 3 | 233 |

# So Far

## Vocabulary

| word | id |
|---|---|
| cat | 0 |
| the | 1 |
| was | 2 |
| ran | 3 |

## Counts lookup

| rank | freq |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 51 |
| 3 | 233 |

## N-gram encoding scheme

unigram: f(id) = id

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

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

## Count DB

| unigram | | bigram | | trigram | |
|---|---|---|---|---|---|
| 16078820 | 0381 | 16078820 | 0381 | 16078820 | 0381 |
| 15176595 | 0051 | 15176595 | 0051 | 15176595 | 0051 |
| 15176583 | 0076 | 15176583 | 0076 | 15176583 | 0076 |
| — | — | — | — | — | — |
| 16576628 | 0021 | 16576628 | 0021 | 16576628 | 0021 |
| — | — | — | — | — | — |
| 15176600 | 0018 | 15176600 | 0018 | 15176600 | 0018 |
| 16089320 | 0171 | 16089320 | 0171 | 16089320 | 0171 |
| 15176583 | 0039 | 15176583 | 0039 | 15176583 | 0039 |
| 14980420 | 0030 | 14980420 | 0030 | 14980420 | 0030 |
| — | — | — | — | — | — |
| 15020330 | 0482 | 15020330 | 0482 | 15020330 | 0482 |

# Hashing vs Sorting

Sorting

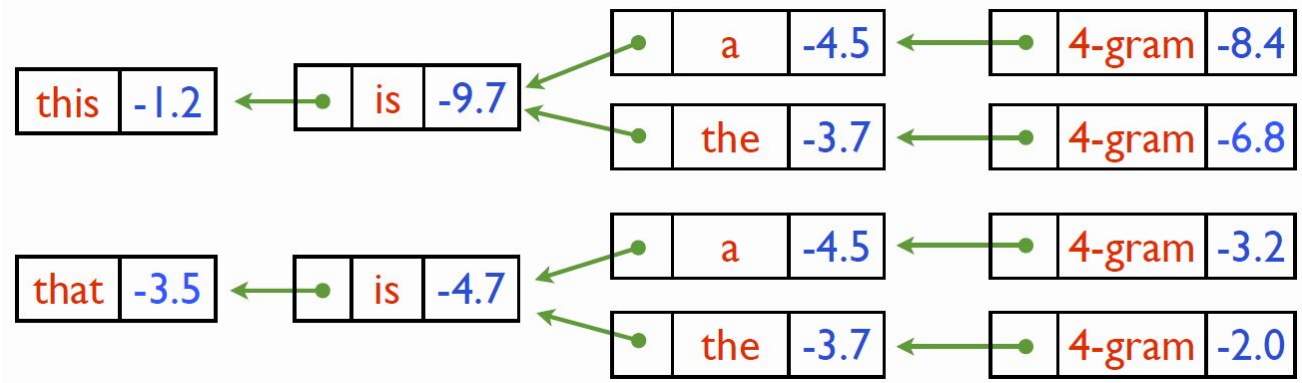
| c | val |
|---|---|
| 15176583 | 0076 |
| 15176595 | 0051 |
| 15176600 | 0018 |
| 16078820 | 0381 |
| 16089320 | 0171 |
| 16576628 | 0021 |
| 16980420 | 0030 |
| 17020330 | 0482 |
| 17176583 | 0039 |

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

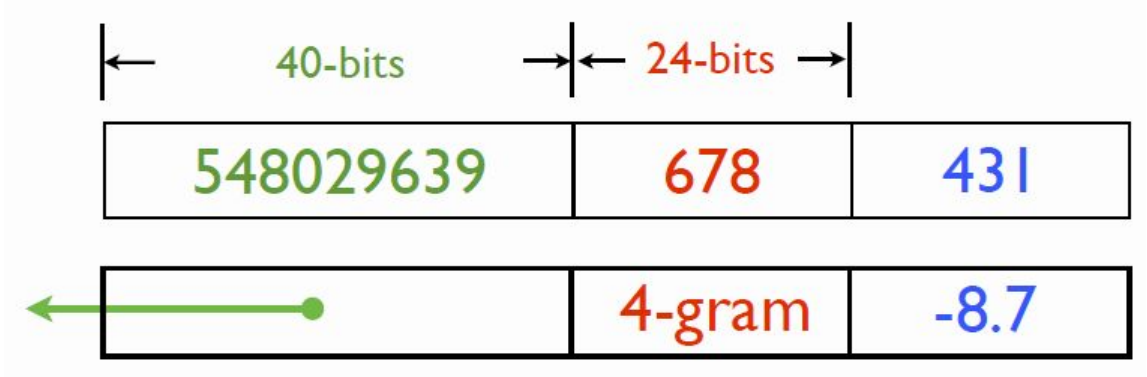


# Tries




[Hsu and Glass 2008]

# Context Encodings

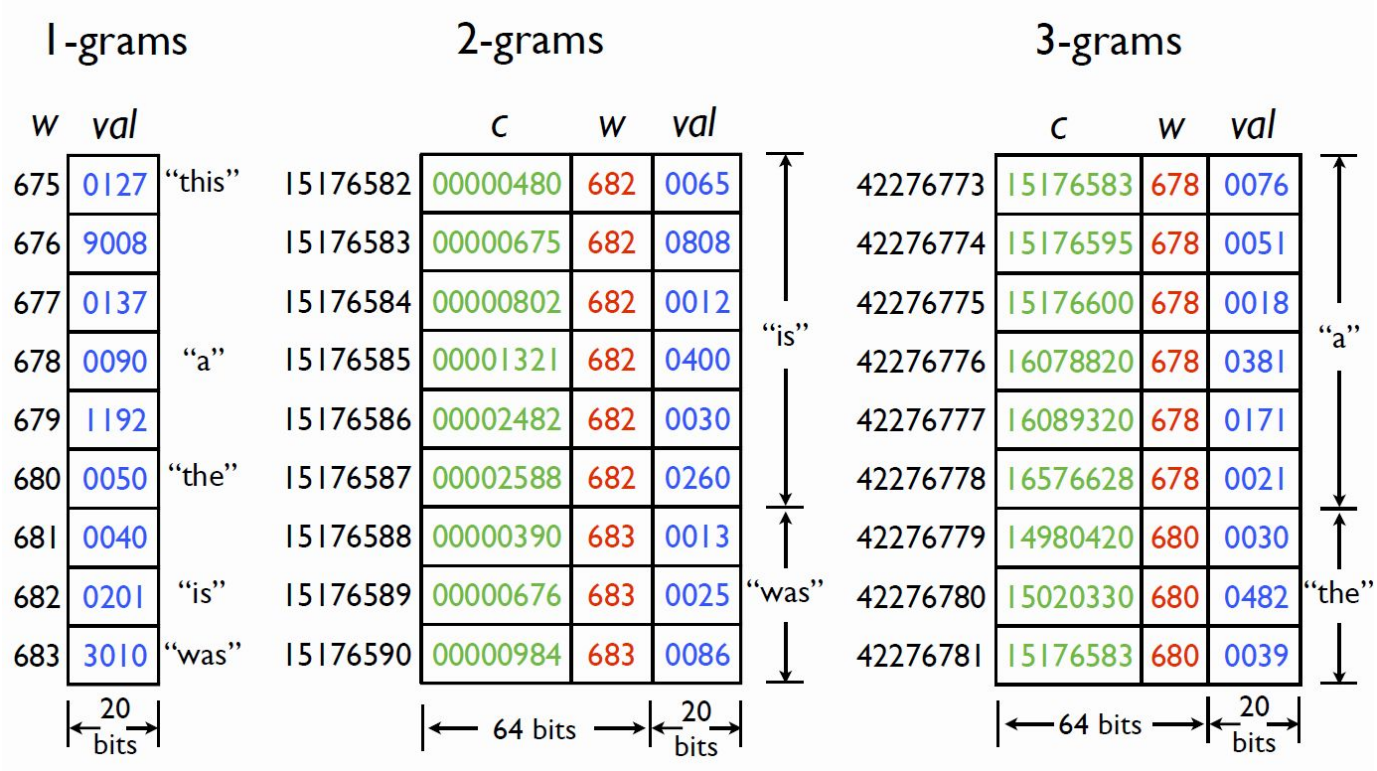



Google N-grams

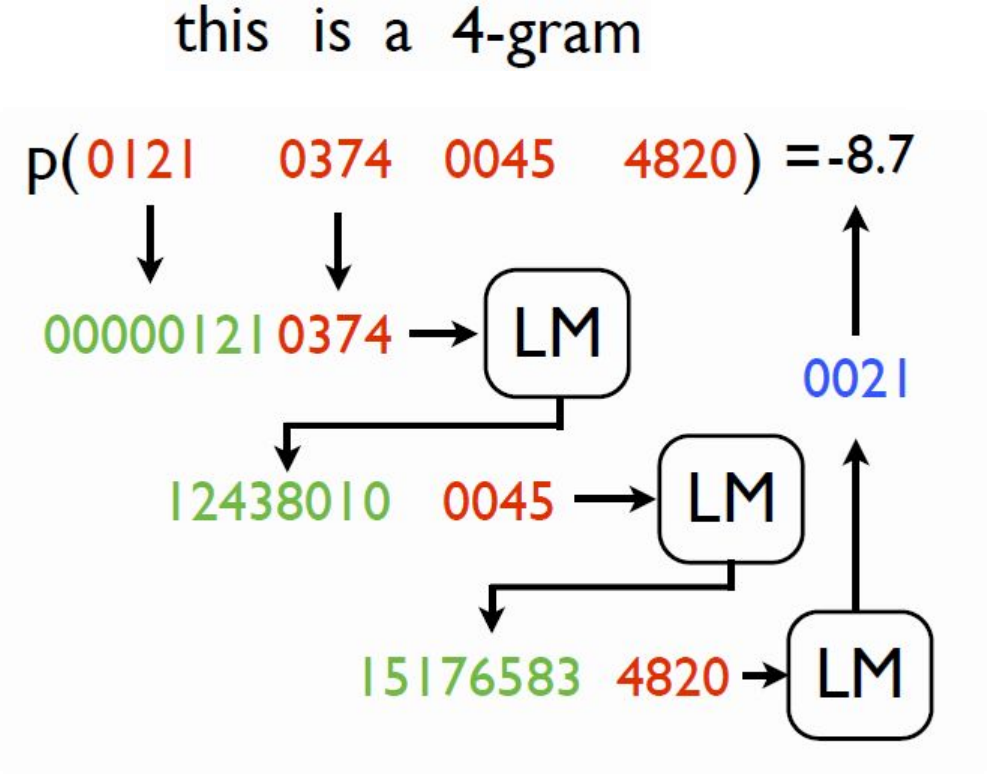
- 10.5 bytes/n-gram
- 37 GB total

[Many details from Pauls and Klein, 2011]

# Context Encodings



# 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”

000 1001

Length in Unary

Number in Binary

Google N-grams

- 2.9 bytes/n-gram
- 10 GB total

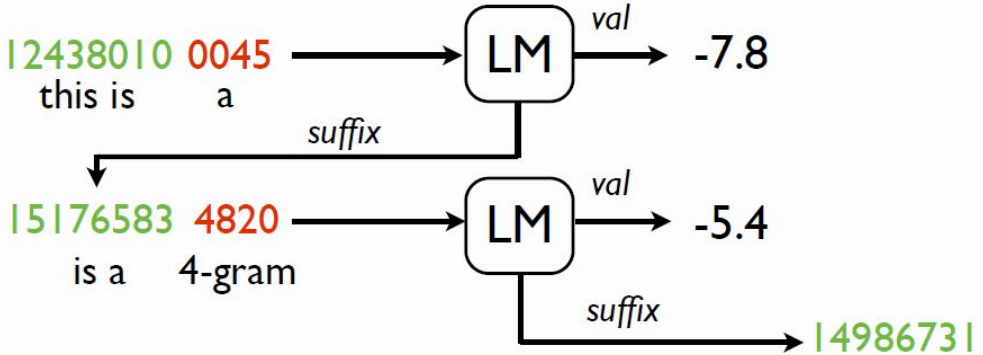
[Elias, 75]

# Speed-Ups

# Rolling Queries

this is + a 4-gram

12438010 0045 4820




| 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 |

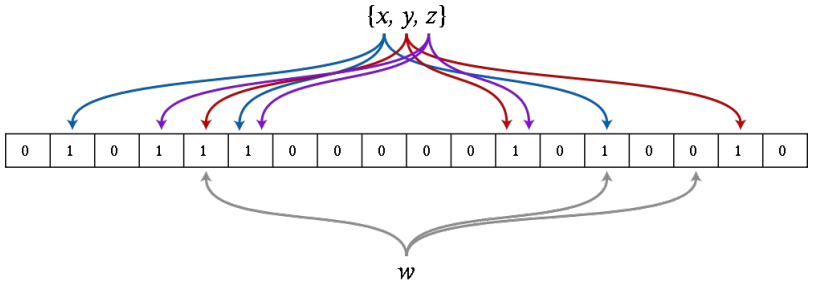
hash( 124 80 42 1243 ) =0

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



# Homework 1 Overview