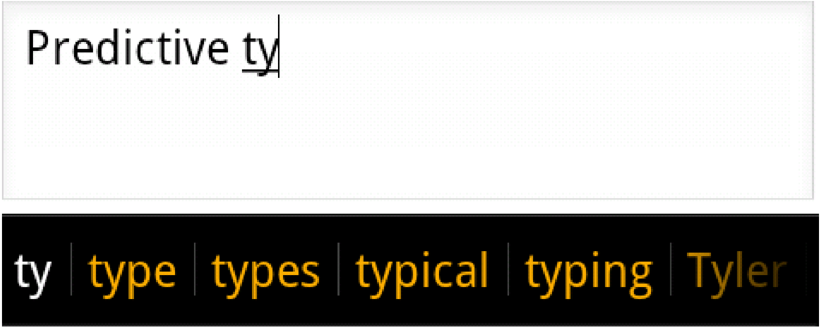
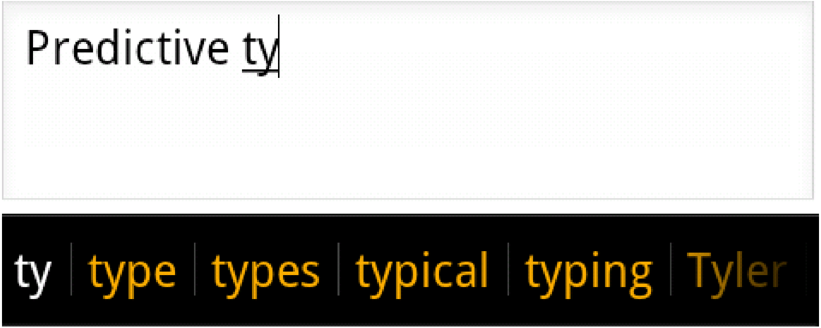
# Language Modeling

Kenneth Heafield

University of Edinburgh


Predictive ty
ty
type
types
typical
typing
Tyler




$$p(\text{type} \mid \text{Predictive}) > p(\text{Tyler} \mid \text{Predictive})$$

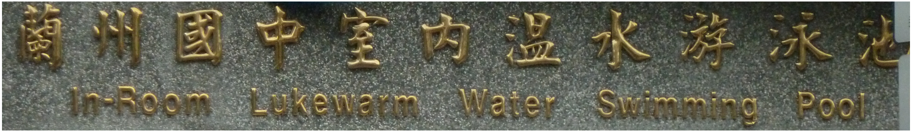
Win or luse, it was a great game.

Win or lose, it were a great game.

Win or loose, it was a great game.

$$p(\text{lose} \mid \text{Win or}) \gg p(\text{loose} \mid \text{Win or})$$

[Church et al, 2007]




Heated indoor swimming pool

Chambre

↓

Bedroom

présidente de la Chambre des représentants

chairwoman of the Bedroom of Representatives

présidente de la Chambre des représentants

↓

chairwoman of the House of Representatives

présidente de la Chambre des représentants

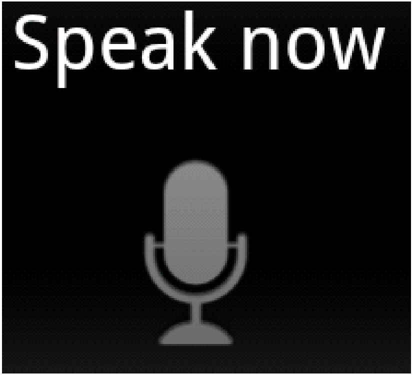
↓

chairwoman of the House of Representatives

$p$(chairwoman of the House of Representatives)

$>$

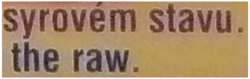
$p$(chairwoman of the Bedroom of Representatives)




$$p(\text{Another one bites the dust.})$$
$$>$$
$$p(\text{Another one rides the bus.})$$


Prediction


Spelling




Translation

Speech

# Essential Component: **Language Model**

$$p(\text{in the raw}) = ?$$

# Language model: fluency of output

✗ How well it translates the source

✗ Ratio to source sentence

✓ Length

✓ Ratio of letter "z" to letter "e"

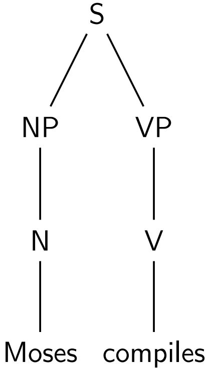
# Language model: fluency of output

- ✗ How well it translates the source
- ✗ Ratio to source sentence

- ✓ Length
- ✓ Ratio of letter "z" to letter "e"
- ✓ Parsing
- ✓ **Sequence Models**

# Parsing

$$p(\text{Moses compiles}) =$$




$$p(\text{S} \rightarrow \text{NP VP})$$

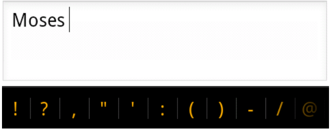
$$\cdot p(\text{NP} \rightarrow \text{N}) p(\text{VP} \rightarrow \text{V})$$

$$\cdot p(\text{N} \rightarrow \text{Moses}) p(\text{V} \rightarrow \text{compiles})$$

# Sequence Models

Chain Rule

$$p(\text{Moses compiles}) = p(\text{Moses})\,p(\text{compiles} \mid \text{Moses})$$



# Sequence Model

$$
\begin{array}{rll}
 & \log p(\text{iran} & \mid <\text{s}> ) \\
 & \log p(\text{is} & \mid <\text{s}> \text{ iran} ) \\
 & \log p(\text{one} & \mid <\text{s}> \text{ iran is} ) \\
 & \log p(\text{of} & \mid <\text{s}> \text{ iran is one} ) \\
 & \log p(\text{the} & \mid <\text{s}> \text{ iran is one of} ) \\
 & \log p(\text{few} & \mid <\text{s}> \text{ iran is one of the} ) \\
 & \log p(\text{countries} & \mid <\text{s}> \text{ iran is one of the few} ) \\
 & \log p(. & \mid <\text{s}> \text{ iran is one of the few countries} ) \\
+ & \log p(</\text{s}> & \mid <\text{s}> \text{ iran is one of the few countries .} ) \\
\hline
= & \log p(<\text{s}> \text{ iran is one of the few countries . } </\text{s}> ) &
\end{array}
$$

# Sequence Model

$$
\begin{array}{rll}
 & \log p(\text{iran} & \mid \text{<s>} \;) \\
 & \log p(\text{is} & \mid \text{<s> iran} \;) \\
 & \log p(\text{one} & \mid \text{<s> iran is} \;) \\
 & \log p(\text{of} & \mid \text{<s> iran is one} \;) \\
 & \log p(\text{the} & \mid \text{<s> iran is one of} \;) \\
 & \log p(\text{few} & \mid \text{<s> iran is one of the} \;) \\
 & \log p(\text{countries} & \mid \text{<s> iran is one of the few} \;) \\
 & \log p(. & \mid \text{<s> iran is one of the few countries} \;) \\
+ & \log p(\text{</s>} & \mid \text{<s> iran is one of the few countries .} \;) \\
\hline
= & \log p(\text{<s> iran is one of the few countries . </s>} \;) &
\end{array}
$$

Explicit begin and end of sentence.

# Sequence Model

$$
\begin{array}{rll}
 & \log p(\text{iran} & \mid \text{<s>} & ) = & -3.33437 \\
 & \log p(\text{is} & \mid \text{<s> iran} & ) = & -1.05931 \\
 & \log p(\text{one} & \mid \text{<s> iran is} & ) = & -1.80743 \\
 & \log p(\text{of} & \mid \text{<s> iran is one} & ) = & -0.03705 \\
 & \log p(\text{the} & \mid \text{<s> iran is one of} & ) = & -0.08317 \\
 & \log p(\text{few} & \mid \text{<s> iran is one of the} & ) = & -1.20788 \\
 & \log p(\text{countries} & \mid \text{<s> iran is one of the few} & ) = & -1.62030 \\
 & \log p(\text{.} & \mid \text{<s> iran is one of the few countries} & ) = & -2.60261 \\
+ & \log p(\text{</s>} & \mid \text{<s> iran is one of the few countries .} & ) = & -0.04688 \\
\hline
= & \log p(\text{<s> iran is one of the few countries . </s>} & & ) = & -11.79900
\end{array}
$$

Where do these probabilities come from?

# Probabilities from Text

Model

$p(\text{raw} \mid \text{in the})$

# Estimating from Text

help in the search for an answer .
Copper burned in the raw wood .
put forward in the paper
Highs in the 50s to lower 60s .

⋮

$\Longrightarrow$

$$p(\text{raw} \mid \text{in the}) \approx \frac{1}{4}$$

# Estimating from Text

help in the search for an answer .
Copper burned in the raw wood .
put forward in the paper
Highs in the 50s to lower 60s .

⋮

$$\Longrightarrow$$

$$p(\text{raw} \mid \text{in the}) \approx \frac{1}{4}$$

$$p(\text{Ugrasena} \mid \text{in the}) \approx 0$$

# Estimating from Text

help in the search for an answer .
Copper burned in the raw wood .
put forward in the paper
Highs in the 50s to lower 60s .

⋮

$$\Longrightarrow \quad p(\text{raw} \mid \text{in the}) \approx \frac{1}{6}$$

$$p(\text{Ugrasena} \mid \text{in the}) \approx \frac{1}{1000}$$

# Problem

"in the Ugrasena" was not seen, but could happen.

$$p(\text{Ugrasena} \mid \text{in the}) = \frac{\text{count(in the Ugrasena)}}{\text{count(in the)}} = 0?$$

# Problem

"in the Ugrasena" was not seen, but could happen.

$$p(\text{Ugrasena} \mid \text{in the}) = \frac{\text{count}(\text{in the Ugrasena})}{\text{count}(\text{in the})} = 0?$$

$$= \frac{\text{count}(\text{the Ugrasena})}{\text{count}(\text{the})} = 2.07 \cdot 10^{-9}$$

# Problem

"in the Ugrasena" was not seen, but could happen.

$$p(\text{Ugrasena} \mid \text{in the}) = \frac{\text{count(in the Ugrasena)}}{\text{count(in the)}} = 0?$$

$$= \frac{\text{count(the Ugrasena)}}{\text{count(the)}} = 2.07 \cdot 10^{-9}$$

Stupid Backoff: Drop context until count is non-zero
[Brants et al, 2007]

Can we be less stupid?

# Smoothing

“in the Ugrasena” was not seen, but could happen.

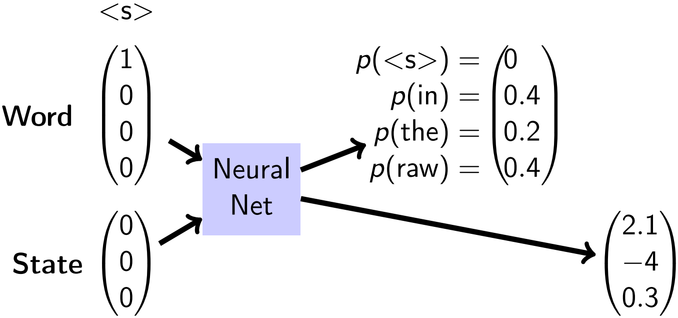
1 **Neural Networks**: classifier predicts next word
2 **Backoff**: maybe “the Ugrasena” was seen?

# Turning Words into Vectors

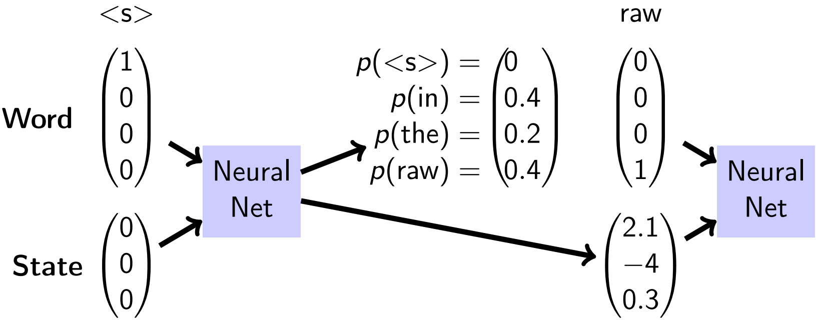
| <s> | in | the | raw |
|---|---|---|---|
| $\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$ | $\begin{pmatrix}0\\1\\0\\0\end{pmatrix}$ | $\begin{pmatrix}0\\0\\1\\0\end{pmatrix}$ | $\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$ |

Assign each word a unique row.

# Recurrent Neural Network



# Recurrent Neural Network



# Recurrent Neural Network Properties

Treat language modeling as a classification problem:
Predict the next word.

**State** uses the *entire* context back to the beginning.

# Turning Words into Vectors

| <s> | in | the | raw |
|---|---|---|---|
| $\begin{pmatrix} -3 \\ 1.5 \\ 6.2 \end{pmatrix}$ | $\begin{pmatrix} 2.2 \\ 7.5 \\ -.8 \end{pmatrix}$ | $\begin{pmatrix} -.1 \\ 0.8 \\ 9.1 \end{pmatrix}$ | $\begin{pmatrix} 1.1 \\ 7.0 \\ -.2 \end{pmatrix}$ |

Vectors from a recurrent neural network

. . . or your favorite ACL paper.

# Neural N-gram Models

$$p(\text{raw} \mid \text{Vector(in)}, \text{Vector(the)})$$

Vectors for context words
$\rightarrow$ neural network classifier
$\rightarrow$ probability distribution over words

# Backoff Smoothing

"in the Ugrasena" was not seen $\rightarrow$ try "the Ugrasena"

$$p(\text{Ugrasena} \mid \text{in the}) \approx p(\text{Ugrasena} \mid \text{the})$$

# Backoff Smoothing

"in the Ugrasena" was not seen $\rightarrow$ try "the Ugrasena"

$$p(\text{Ugrasena} \mid \text{in the}) \approx p(\text{Ugrasena} \mid \text{the})$$

"the Ugrasena" was not seen $\rightarrow$ try "Ugrasena"

$$p(\text{Ugrasena} \mid \text{the}) \approx p(\text{Ugrasena})$$

# Backoff Smoothing

"in the Ugrasena" was not seen $\to$ try "the Ugrasena"

$$p(\text{Ugrasena} \mid \text{in the}) = p(\text{Ugrasena} \mid \text{the})b(\text{in the})$$

"the Ugrasena" was not seen $\to$ try "Ugrasena"

$$p(\text{Ugrasena} \mid \text{the}) = p(\text{Ugrasena})b(\text{the})$$

Backoff *b* is a penalty for not matching context.

# Example Language Model

**Unigrams**

| **Words** | **log $p$** | **log $b$** |
|---|---|---|
| <s> | $-\infty$ | $-2.0$ |
| iran | $-4.1$ | $-0.8$ |
| is | $-2.5$ | $-1.4$ |
| one | $-3.3$ | $-0.9$ |
| of | $-2.5$ | $-1.1$ |

**Bigrams**

| **Words** | **log $p$** | **log $b$** |
|---|---|---|
| <s> iran | $-3.3$ | $-1.2$ |
| iran is | $-1.7$ | $-0.4$ |
| is one | $-2.0$ | $-0.9$ |
| one of | $-1.4$ | $-0.6$ |

**Trigrams**

| **Words** | **log $p$** |
|---|---|
| <s> iran is | $-1.1$ |
| iran is one | $-2.0$ |
| is one of | $-0.3$ |

# Example Language Model

**Unigrams**

| **Words** | **log *p*** | **log *b*** |
|---|---|---|
| <s> | $-\infty$ | $-2.0$ |
| iran | $-4.1$ | $-0.8$ |
| is | $-2.5$ | $-1.4$ |
| one | $-3.3$ | $-0.9$ |
| of | $-2.5$ | $-1.1$ |

**Bigrams**

| **Words** | **log *p*** | **log *b*** |
|---|---|---|
| <s> iran | $-3.3$ | $-1.2$ |
| iran is | $-1.7$ | $-0.4$ |
| is one | $-2.0$ | $-0.9$ |
| one of | $-1.4$ | $-0.6$ |

**Trigrams**

| **Words** | **log *p*** |
|---|---|
| <s> iran is | $-1.1$ |
| iran is one | $-2.0$ |
| is one of | $-0.3$ |

## Query

$$\log p(\text{is} \mid \text{<s> iran}) = -1.1$$

# Example Language Model

**Unigrams**

| Words | log $p$ | log $b$ |
|---|---|---|
| <s> | $-\infty$ | $-2.0$ |
| iran | $-4.1$ | $-0.8$ |
| is | $-2.5$ | $-1.4$ |
| one | $-3.3$ | $-0.9$ |
| of | $-2.5$ | $-1.1$ |

**Bigrams**

| Words | log $p$ | log $b$ |
|---|---|---|
| <s> iran | $-3.3$ | $-1.2$ |
| iran is | $-1.7$ | $-0.4$ |
| is one | $-2.0$ | $-0.9$ |
| one of | $-1.4$ | $-0.6$ |

**Trigrams**

| Words | log $p$ |
|---|---|
| <s> iran is | $-1.1$ |
| iran is one | $-2.0$ |
| is one of | $-0.3$ |

$$\text{Query}: \; p(\text{of} \mid \text{iran is})$$

| | |
|---|---|
| $\log p(\text{of})$ | $-2.5$ |
| $\log b(\text{is})$ | $-1.4$ |
| $\log b(\text{iran is})$ | $+\;-0.4$ |
| $\log p(\text{of} \mid \text{iran is})$ | $=\;-4.3$ |

# Close words matter more.

Though long-distance matters:

- Grammatical structure
- Topical coherence
- Words tend to repeat
- Cross-sentence dependencies

Alternative: skip over words in the context

[Pickhardt et al, ACL 2014]

# Language Modeling

# Where do $p$ and $b$ come from?
# Text!

**Kneser-Ney**
Witten-Bell
Good-Turing

## Common high-quality smoothing

1. Adjust
2. Normalize
3. Interpolate

# Adjusted counts are:

Trigrams Count in the text.

Others Number of unique words to the left.

Lower orders are used when a trigram did not match.
How freely does the text associate with new words?

# Adjusted counts are:

**Trigrams** Count in the text.

**Others** Number of unique words to the left.

Lower orders are used when a trigram did not match.
How freely does the text associate with new words?

Input

| Trigam | Count |
|---|---|
| are one of | 1 |
| is one of | 5 |
| are two of | 3 |

→

Output

| 1-gram | Adjusted |
|---|---|
| of | 2 |

Output

| 2-gram | Adjusted |
|---|---|
| one of | 2 |
| two of | 1 |

# Discounting and Normalization

Save mass for unseen events

$$\text{pseudo}(w_n | w_1^{n-1}) = \frac{\text{adjusted}(w_1^n) - \text{discount}_n(\text{adjusted}(w_1^n))}{\sum_x \text{adjusted}(w_1^{n-1} x)}$$

Normalize

# Discounting and Normalization

Save mass for unseen events

$$\text{pseudo}(w_n|w_1^{n-1}) = \frac{\text{adjusted}(w_1^n) - \text{discount}_n(\text{adjusted}(w_1^n))}{\sum_x \text{adjusted}(w_1^{n-1}x)}$$

Normalize

Input

| 3-gram | Adjusted |
|---|---|
| are one of | 1 |
| are one that | 2 |
| is one of | 5 |

→

Output

| 3-gram | Pseudo |
|---|---|
| are one of | 0.26 |
| are one that | 0.47 |
| is one of | 0.62 |

# Interpolate: Sparsity vs. Specificity

Interpolate unigrams with the uniform distribution.

$$p(\text{of}) = \text{pseudo}(\text{of}) + \text{backoff}(\epsilon)\frac{1}{|\text{vocabulary}|}$$

# Interpolate: Sparsity vs. Specificity

Interpolate unigrams with the uniform distribution,

$$p(\text{of}) = \text{pseudo}(\text{of}) + \text{backoff}(\epsilon)\frac{1}{|\text{vocabulary}|}$$

Interpolate bigrams with unigrams, etc.

$$p(\text{of}|\text{one}) = \text{pseudo}(\text{of} \mid \text{one}) + \text{backoff}(\text{one})p(\text{of})$$

# Interpolate: Sparsity vs. Specificity

Interpolate unigrams with the uniform distribution,

$$p(\text{of}) = \text{pseudo}(\text{of}) + \text{backoff}(\epsilon)\frac{1}{|\text{vocabulary}|}$$

Interpolate bigrams with unigrams, etc.

$$p(\text{of}|\text{one}) = \text{pseudo}(\text{of} \mid \text{one}) + \text{backoff}(\text{one})p(\text{of})$$

Input

| $n$-gram | pseudo | interpolation weight |
|---|---|---|
| of | 0.1 | backoff( $\epsilon$) = 0.1 |
| one of | 0.2 | backoff( one) = 0.3 |
| are one of | 0.4 | backoff(are one) = 0.2 |

→

Output

| $n$-gram | $p$ |
|---|---|
| of | 0.110 |
| one of | 0.233 |
| are one of | 0.447 |

# Kneser-Ney Intuition

**Adjust** Measure association with new words.
**Normalize** Leave space for unseen events.
**Interpolate** Handle sparsity.

How do we implement it?

# Language Modeling

"LM queries often account for more than 50% of the CPU"
[Green et al, WMT 2014]

500 billion entries in my largest model

Need speed and memory efficiency

# Counting *n*-grams

<s> Australia is one of the few

| 5-gram | Count |
|---|---|
| <s> Australia is one of | 1 |
| Australia is one of the | 1 |
| is one of the few | 1 |

Hash table?

# Counting $n$-grams

<s> Australia is one of the few

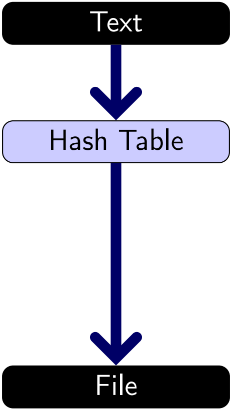
↓

| **5-gram** | **Count** |
|---|---|
| <s> Australia is one of | 1 |
| Australia is one of the | 1 |
| is one of the few | 1 |

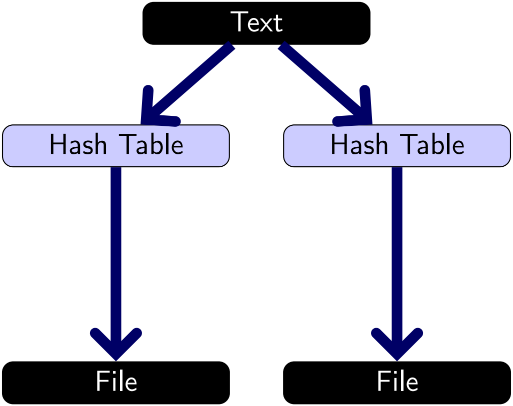
Hash table?

Runs out of RAM.

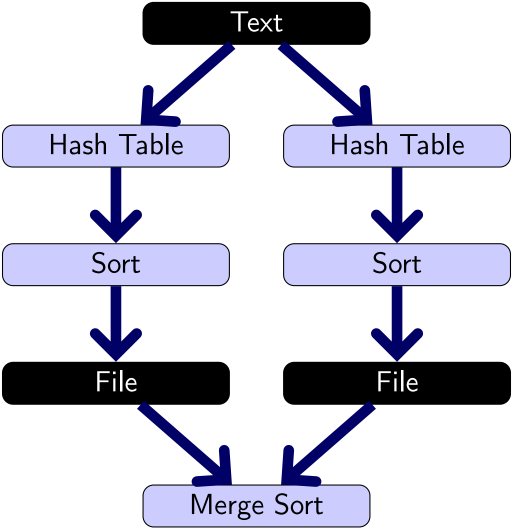
# Spill to Disk When RAM Runs Out



# Split Data



# Split and Merge



# Training Problem:
Batch process large number of records.

# Solution: Split/merge
Stupid backoff in one pass
Kneser-Ney in three passes

# Training Problem:
Batch process large number of records.

# Solution: Split/merge
Stupid backoff in one pass
Kneser-Ney in three passes

Training is designed for mutable batch access.
What about queries?

# Query

$$\text{stupid}(w_n \mid w_1^{n-1}) = \begin{cases} \dfrac{\text{count}(w_1^n)}{\text{count}(w_1^{n-1})} & \text{if count}(w_1^n) > 0 \\ 0.4\text{stupid}(w_n \mid w_2^{n-1}) & \text{if count}(w_1^n) = 0 \end{cases}$$

**stupid(few | is one of the)**

count(is one of the few) = 5 ✓

count(is one of the) = 12

# Query

$$\text{stupid}(w_n \mid w_1^{n-1}) = \begin{cases} \dfrac{\text{count}(w_1^n)}{\text{count}(w_1^{n-1})} & \text{if count}(w_1^n) > 0 \\ 0.4\,\text{stupid}(w_n \mid w_2^{n-1}) & \text{if count}(w_1^n) = 0 \end{cases}$$

**stupid(periwinkle | is one of the)**

count(is one of the periwinkle) = 0 ✗
count(one of the periwinkle) = 0 ✗
count(of the periwinkle) = 0 ✗
count(the periwinkle) = 3 ✓

count(the) = 1000

# Save Memory: Forget Keys

Giant hash table with *n*-grams as keys and counts as values.

Replace the *n*-grams with 64-bit hashes:
Store hash(is one of) instead of "is one of".
Ignore collisions.

# Save Memory: Forget Keys

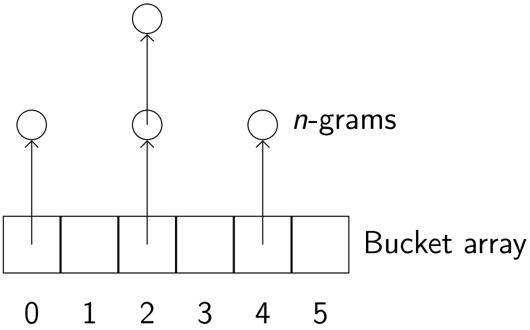
Giant hash table with *n*-grams as keys and counts as values.

Replace the *n*-grams with 64-bit hashes:
Store hash(is one of) instead of “is one of”.
Ignore collisions.

Birthday attack: $\sqrt{2^{64}} = 2^{32}$.
$\Longrightarrow$ Low chance of collision until $\approx$ 4 billion entries.

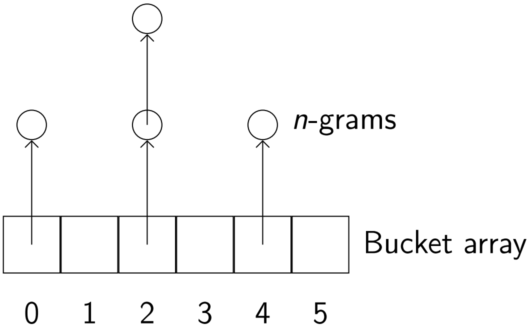
# Default Hash Table

`boost::unordered_map` and `__gnu_cxx::hash_map`



# Default Hash Table

`boost::unordered_map` and `__gnu_cxx::hash_map`




Lookup requires two random memory accesses.

# Linear Probing Hash Table

- 1.5 buckets/entry (so buckets = 6).
- Ideal bucket = hash mod buckets.
- Resolve *bucket* collisions using the next free bucket.

**Bigrams**

| Words | Ideal | Hash | Count |
|---|---|---|---|
| iran is | 0 | 0x959e48455f4a2e90 | 3 |
| | | 0x0 | 0 |
| is one | 2 | 0x186a7caef34acf16 | 5 |
| one of | 2 | 0xac66610314db8dac | 2 |
| <s> iran | 4 | 0xf0ae9c2442c6920e | 1 |
| | | 0x0 | 0 |

# Minimal Perfect Hash Table

Maps every *n*-gram to a unique integer $[0, |n - grams|)$
$\rightarrow$ Use these as array offsets.

# Minimal Perfect Hash Table

Maps every *n*-gram to a unique integer $[0, |n - grams|)$
$\rightarrow$ Use these as array offsets.

Entries not in the model get assigned offsets
$\rightarrow$ Store a fingerprint of each *n*-gram

# Minimal Perfect Hash Table

Maps every *n*-gram to a unique integer $[0, |n - grams|)$
$\rightarrow$ Use these as array offsets.

Low memory, but potential for false positives

# Less Memory: Sorted Array

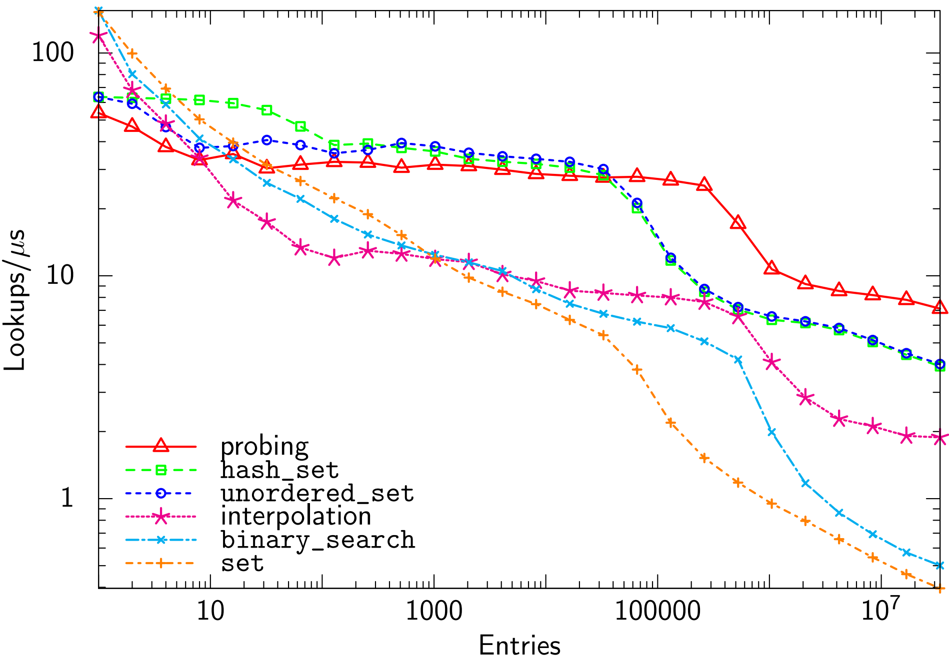
Look up "zebra" in a dictionary.

## Binary search

Open in the middle. $O(n \log n)$ time.

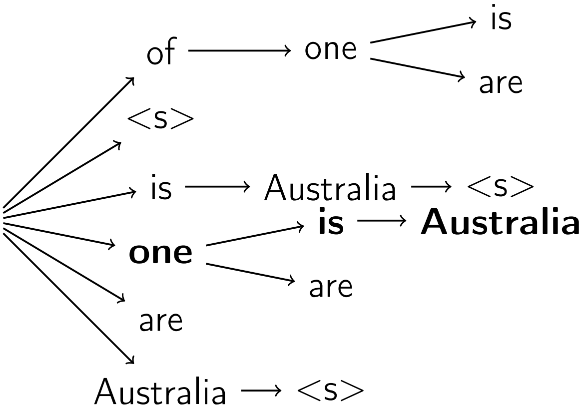
## Interpolation search

Open near the end. $O(n \log \log n)$ time.


Lookups/$\mu$s
100
10
1
10
1000
100000
$10^7$
Entries
probing
hash_set
unordered_set
interpolation
binary_search
set

# Trie

Reverse *n*-grams, arrange in a trie.



# Saving More RAM

- Quantization: store approximate values
- Collapse probability and backoff

# Implementation Summary

Implementation involves sparse mapping

- Hash table
- Probing hash table
- Minimal perfect hash table
- Sorted array with binary or interpolation search

# Conclusion

Language models measure fluency.

Neural networks and backoff are the dominant formalisms.

Efficient implementation needs good data structures.