Exploring Web Scale Language Models for Search Query Processing

Jianfeng Gao, MSR
(Joint work with Jian Huang, Jiangbo Miao, Xiaolong Li, Kuansan Wang and Fritz Behr)
Outline

• N-gram language model (LM) ABC
• N-gram LM at Microsoft
  – Bing-It-On-Ngram
• Building Web scale N-gram LM
• Three search query processing tasks
  – Query spelling correction
  – Query bracketing
  – Long query segmentation
• Conclusion
Word n-gram model

• Compute the probability of a word string using **chain rule** on its *history* (=preceeding words)

\[
P(\text{the dog of our neighbor barks}) = P(\text{the} \mid <s>) \\
\times P(\text{dog} \mid <s>, \text{the}) \\
\times P(\text{of} \mid <s>, \text{the, dog}) \\
\ldots \\
\times P(\text{barks} \mid <s>, \text{the, dog, of, our, neighbor}) \\
\times P(</s> \mid <s>, \text{the, dog, or, our, neighbor, barks})
\]

\[
P(w_1, w_2 \ldots w_n) = P(w_1 \mid <s>) \\
\times P(w_2 \mid <s> w_1) \\
\times P(w_3 \mid <s> w_1 w_2) \\
\ldots \\
\times P(w_n \mid <s> w_1 w_2 \ldots w_{n-1}) \\
\times P(</s> \mid <s> w_1 w_2 \ldots w_n)
\]
Word n-gram model

- Markov independence assumption
  - A word depends only on N-1 preceding words
  - N=3 → word trigram model
- Reduce the number of parameters in the model
  - By forming equivalence classes
- Word trigram model

\[
P(w_i | <s> \, w_1 \, w_2 \, ... \, w_{i-2} \, w_{i-1}) = P(w_i | w_{i-2} \, w_{i-1})
\]

\[
P(w_1, w_2, ... \, w_n) = P(w_1 | <s>) \times P(w_2 | <s> \, w_1) \times P(w_3 | w_1 \, w_2) \times ...
\]

\[
\times P(w_n | w_{n-2} \, w_{n-1}) \times P(</s> | w_{n-1} \, w_n)
\]
Example: input method editor (IME)

- Software to convert keystrokes (Pinyin) to text output
LM Evaluation

- Perplexity – quality of LM
  - Geometric average inverse probability
  - Branching factor of a doc: predicting power of LM
  - Lower perplexities are better
  - Character perplexity for Chinese/Japanese

  \[ pplx = 2^H \quad \text{where} \quad H = \frac{1}{|W|} \log P(W) \]

- Better to use task-specific evaluation, e.g.,
  - Character error rate (CER) – quality of IME
  - Test set (A, W*)
  - CER = edit distance between converted W and W*
MLE for trigram LM

• $P_{ML}(w_3 | w_1 w_2) = \frac{\text{Count}(w_1 w_2 w_3)}{\text{Count}(w_1 w_2)}$
• $P_{ML}(w_2 | w_1) = \frac{\text{Count}(w_1 w_2)}{\text{Count}(w_1)}$
• $P_{ML}(w) = \frac{\text{Count}(w)}{N}$
• It is easy – let us get real text and start counting

$$P_{ML}(\text{barked} | \text{the, dog}) = \frac{\text{Count}(\text{the, dog, barked})}{\text{Count}(\text{the, dog})}$$

• But why is this the MLE solution?
The derivation of MLE for N-gram

• Homework 😊

• Hints
  – This is a constrained optimization problem
  – Use log likelihood as objective function
  – Assume a multinomial distribution of LM
  – Introduce Lagrange multiplier for the constraints
    • $\sum_{x \in \mathcal{X}} P(x) = 1$, and $P(x) \geq 0$
Sparse data problems

- Say our vocabulary size is $|V|$
- There are $|V|^3$ parameters in the trigram LM
  - $|V| = 20,000 \Rightarrow 20,000^3 = 8 \times 10^{12}$ parameters
- Most trigrams have a zero count even in a large text corpus
  - $\text{Count}(w_1 w_2 w_3) = 0$
  - $P_{ML}(w_3 | w_1 w_2) = \frac{\text{Count}(w_1 w_2 w_3)}{\text{Count}(w_1 w_2)} = 0$
  - $P(W) = P_{ML}(w_1) P_{ML}(w_2 | w_1) \prod_i P_{ML}(w_i | w_{i-2} w_{i-1}) = 0$
  - $W = \arg\max_W P(A|W)P(W) = \ldots \text{ oops}$
Smoothing: backoff

- Backoff trigram to bigram, bigram to unigram

\[ P(w_3|w_1, w_2) = \begin{cases} 
\frac{C(w_1, w_2, w_3) - D}{C(w_1, w_2)}, & \text{if } C(w_1, w_2, w_3) > 0 \\
\alpha(w_1, w_2)P(w_3|w_2), & \text{if } C(w_1, w_2, w_3) = 0
\end{cases} \]

- \( D \in (0,1) \) is a discount constant – absolute discount
- \( \alpha \) is calculated so probabilities sum to 1 (homework😊)

\[ 1 = \sum_{(w_1, w_2)} P(w_3|w_1, w_2) \]
Smoothing: improved backoff

• Allow $D$ to vary
  – Different $D$’s for different N-gram
  – Value of $D$’s as a function of Count(.)
  – Modified absolute discount
• Optimizing $D$’s on dev data using e.g., Powell search

$$D = \text{argmax}_D \sum_{(w_1,w_2,w_3) \text{ in dev data}} \log P(w_3|w_1w_2)$$

• Using word type probabilities rather than token probability for backoff models
  • Kneser-Ney smoothing
What is the best smoothing?

• It varies from task to task
  – Chen and Goodman (1999) gives a very thorough evaluation and descriptions of a number of methods

• My favorite smoothing methods
  – Modified absolute discount (MAD, Gao et al., 2001)
    • Simple to implement and use
    • Good performance across many tasks, e.g., IME, SMT, ASR, Speller
  – Interpolated Kneser-Ney
    • Recommended by Chen and Goodman (1999)
    • Only slightly better than MAD on SMT (more expensive to train, though)
N-gram LM at Microsoft

• Contextual speller in WORD
  – 1-5 MB trigram
  – LM compression (Church et al. 2007)

• Chinese ASR and IME
  – 20-100 MB trigram
  – Training data selection and LM compression (Gao et al. 2002a)

• Japanese IME
  – 30-60 MB trigram
  – Capture language structure – headword trigram (Gao et al. 2002b)

• MS-SMT (MSRLM)
  – 1-20 GB tri/four-gram
  – LM compression, training data selection, runtime (client/server)

• Bing Search, e.g., query speller/segmentation, ranking
  – Terabyte 5-gram
  – Model building and runtime via cloud-based platform
LM research

• Research communities (speech/IR/NLP)
  – Make LM smarter via
    • Using better smoothing
    • Using word class
    • Capturing linguistic structure, etc.

• Industry: data is smarter!
  – Make LM simpler and more scalable, e.g.,
    • Google’s “stupid smoothing” model
    • Don’t do research until you run out of data (Eric Brill)

• Bridge the gap btw academic/industry research
  – Bing-It-On-Ngram service hosted by MS
    (http://research.microsoft.com/web-ngram)
Microsoft Web N-gram Services

Access petabytes of data via the Web N-gram services (Beta version).

The Web N-gram services provide you access to:
- Content types: Document Body, Document Title, Anchor Texts
- Model types: Counts and smoothed models
- N-gram availability: unigram, bigram, trigram, N-gram with N=4, 5.
- Training size (Body): All documents indexed by Bing
- Access: Hosted Services by Microsoft
- Updates: Periodical updates

Currently, this program is a "private beta." We are working with a small number of academic researchers who previously collaborated with External Research. In this early phase of the program, we are collaborating to verify the use of Web services tools to support research on large data sets.

Web N-gram is brought to you by Microsoft Research in partnership with Microsoft Bing.

Related Links
- Web N-gram FAQ
- Web N-gram Community Site
- Microsoft External Research Home
- Computer Science Home
- Microsoft Research Facebook Site
Outline

• N-gram language model (LM) ABC
• N-gram LM at Microsoft
  – Bing-It-On-Ngram
• **Building Web scale N-gram LM**
• Three search query processing tasks
  – Query spelling correction
  – Query bracketing
  – Long query segmentation
• Conclusion
Everyone can count. Why is it so difficult?

Count all bigrams of a given text
- Input: tokenized text
- Output: a list of <bigram, freq> pairs

Counting alg.
- Use hash if text is small
- Sort and merge (used in most SLM toolkits)
  - Could be very slow on large amounts of text data

Probability/backoff estimation often requires sorting n-grams in different orders
- This is why KN-smoothed LM is expensive to train
A cloud-based n-gram platform
Cloud infrastructure

- Build programs using a *script* language
  - SQL-like language, easy to use
  - User-defined func (operators) written in C#
  - Map “serial” code to “parallel” execution plan automatically
Example: n-gram count in cloud

Node 1
- Web Pages
  - Parsing
  - Tokenize
  - Counting
  - Local Hash

Node 2
- Web Pages
  - Parsing
  - Tokenize
  - Counting
  - Local Hash

... (Repeat for Node N)

Recursive Reducer

Output
Script language: 5-gram counting

Raw = EXTRACT docId, htmlDoc
FROM @"webChunks"
USING WebChunkExtractor
WHERE language=="en";

ParsedDoc=PROCESS Raw
PRODUCE docId, TokenizedDoc
USING HTMLPageProcessor(-featureSet BodyStream);

Ngram=PROCESS ParsedDoc
PRODUCE Ngram, NGcount
USING NGramCountProcessor(-stream TokenizedDoc -
order 5 -bufferSize 20000000);

NGramCount=REDUCE Ngram
ON Ngram
PRODUCE Ngram, NGcount
USING NGramCountReducer
;
OUTPUT TO @"Body-5-gram-count.txt";
Web pages

• Web page is a multi-field text
  – Content fields: URL/Title/Body
  – Popularity fields: anchor/query-click (Gao et al. 2009)

<table>
<thead>
<tr>
<th></th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>msn web</td>
<td>0.6675749</td>
</tr>
<tr>
<td>Webmenssenger</td>
<td>0.6621253</td>
</tr>
<tr>
<td>msn online</td>
<td>0.6403270</td>
</tr>
<tr>
<td>windows web messanger</td>
<td>0.6321526</td>
</tr>
<tr>
<td>talking to friends on msn</td>
<td>0.6130790</td>
</tr>
<tr>
<td>school msn</td>
<td>0.5994550</td>
</tr>
<tr>
<td>msn anywhere</td>
<td>0.5667575</td>
</tr>
<tr>
<td>web message msn com</td>
<td>0.5476839</td>
</tr>
<tr>
<td>msn messenger</td>
<td>0.5313351</td>
</tr>
<tr>
<td>hotmail web chat</td>
<td>0.5231608</td>
</tr>
<tr>
<td>messenger web version</td>
<td>0.5013624</td>
</tr>
<tr>
<td>instant messager msn</td>
<td>0.4550409</td>
</tr>
<tr>
<td>browser based messenger</td>
<td>0.3814714</td>
</tr>
<tr>
<td>im messenger sign in</td>
<td>0.2997275</td>
</tr>
<tr>
<td>msn web browser download</td>
<td>0.0926431</td>
</tr>
<tr>
<td>install msn toolbar</td>
<td>0.0027248</td>
</tr>
</tbody>
</table>

Figure 1: A fragment of the query click field for the page http://webmessenger.msn.com.
# Web scale n-gram models (updated)

<table>
<thead>
<tr>
<th>Data Stream</th>
<th>Body</th>
<th>Title</th>
<th>Anchor</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total tokens</td>
<td>1,304,852,468,065</td>
<td>10,997,887,458</td>
<td>257,239,274,617</td>
<td>28,066,741,474</td>
</tr>
<tr>
<td>#unigrams</td>
<td>1,249,422,634</td>
<td>60,348,698</td>
<td>150,492,565</td>
<td>251,563,970</td>
</tr>
<tr>
<td>#bigrams</td>
<td>11,676,998,598</td>
<td>464,068,083</td>
<td>1,052,685,241</td>
<td>1,273,688,682</td>
</tr>
<tr>
<td>#trigrams</td>
<td>60,053,100,858</td>
<td>1,434,529,993</td>
<td>3,190,230,168</td>
<td>3,087,737,584</td>
</tr>
<tr>
<td>#four-grams</td>
<td>148,489,737,533</td>
<td>2,269,474,926</td>
<td>5,093,949,036</td>
<td>4,553,892,689</td>
</tr>
<tr>
<td>#five-grams</td>
<td>237,982,422,226</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Total entries</td>
<td>459,451,681,849</td>
<td>4,228,421,700</td>
<td>9,487,357,010</td>
<td>9,166,882,925</td>
</tr>
<tr>
<td>Size on disk</td>
<td>15.8 TB</td>
<td>183 GB</td>
<td>395 GB</td>
<td>393 GB</td>
</tr>
</tbody>
</table>
Perplexity results on a query set (updated)

- Query/anchor/content are different languages
- Web corpus is an aligned multi-lingual corpus
Outline

- N-gram language model (LM) ABC
- N-gram LM at Microsoft
  - Bing-It-On-Ngram
- Building Web scale N-gram LM
- Three search query processing tasks
  - Query spelling correction
  - Query bracketing
  - Long query segmentation
- Conclusion
Query Spelling Correction

• What does speller do?
• How does it work?
• What is the role of LM?
• Results
What does speller do

- Provide suggestion for misspelled query
What does speller do

- Alter the original query to improve relevance
WORD Speller vs. Bing Speller

• 1990’s: spellers built by hand
  – Dictionaries + heuristic rules used to identify misspellings
  – Typically, suggest only words sanctioned by dictionary
  – No suggestions for unknown words (e.g., names, new words)
  – Runs client-side

• 2010: spellers learned from user-generated data
  – Search query speller
    • Spelling modeled as a statistical translation problem: translate *bad* queries to *good* queries
    • Models trained on billions of query-suggestion pairs from search logs
  – Correct suggestions learned automatically from data
  – Runs on cluster: large models provide better suggestions
How query speller works

Input Query, \( q \)  
“for eggsample”

Candidates, \( \text{GEN}(q) \)
\( t_1 = “for eggsample” \)
\( t_2 = “for egg sample” \)
\( t_3 = “for example” \)
\( t_4 = “for eggs ample” \)

Speller A  
(Edit distance)
examplw \( \rightarrow \) example

Speller B  
(Phonetic mistake)
eggsample \( \rightarrow \) example

Speller C  
(Word breaker)
eggsample \( \rightarrow \) egg sample

Candidates, \( \text{GEN}(q) \)
\( t_1 = “for eggsample” \)
\( t_2 = “for egg sample” \)
\( t_3 = “for example” \)
\( t_4 = “for eggs ample” \)

Ranking results
“for example” 0.23
“for egg sample” 0.06
“for eggs ample” 0.03
“for eggsample” 0.01

Feature extractor
\( f_0: \text{N-gram prob.} \)
\( f_1: \text{Length} \)
\( f_2: \text{ED\textunderscore Bin} \)
\( f_i: \ldots \)

Ranker
\( \text{Score}(q, t) = \lambda f(q, t) \)
Speller workflow

Candidate Generation

Path Filtering

Ranking

Auto-correction

Generate candidate for each token in the query
- Typographic generator
- Phonetic generator
- Wordbreak generator
- Concatenation Generator

{Britnay Spears Vidios}
Speller workflow

Candidate Generation

Path Filtering

Ranking

Auto-correction

Use a small bigram model to pick the 20 best paths.

Britnay [-16.2] Spears [-11.0] Vidios [-25.2]
Britney [-10.7] Shears [-25.3] Videos [-17.1]
Bridney [-19.1] Speaks [-25.8] Vidoes [-25.1]
Birtney [-17.1] Vidies [-30.9]
Speller workflow

Candidate Generation

Path Filtering

Ranking

Auto-correction

Extract ~200 features.
Return the path with highest score.
Speller workflow

- Candidate Generation
- Path Filtering
- Ranking

Auto-correction

Determine whether we should alter the original query

Original query = britnay spears vidios
Altered query = \texttt{word:(britnay britney) spears word:(vidios videos)}
Roles of LM in a ranker-based speller

• A light decoder
  – Only uses a small bigram model trained on query log
  – Run Viterbi/A* to produce top-20 suggestions
  – A component of candidate generator

• A heavy ranker
  – Feature extraction
    • Derived from the models that generate candidates
    • Additional features (defined by domain experts)
  – (Non-)linear model ranker (with complex features)
    • uses ~200 features, including
    • 4 Web scale 4-gram language models (amount to 2 terabyte)
**Search Query Speller Accuracy**

1. Noisy-channel model trained on query logs
2. Ranker-based speller trained on query/session logs
3. 2 + word-based translation model trained on session logs
4. 3 + phrase-based translation model trained on query/session logs
5. 3 + phrase-based translation model trained on 1-m session logs
6. 3 + phrase-based translation model trained on 3-m session logs
7. 6 + TB language models trained on the Web collection
Figure 4: Precision of the query speller using language models of similar sizes from different sources with different orders.
Query Bracketing

• Task: given a three-word NP, determine sub-NP structure either as left or right bracketing

  Left bracketing: [sore gum] treatment
  Right bracketing: sore [gum treatment]

• Methods: compare word associations btw \( w_1w_2 \) and \( w_2w_3 \) (or btw \( w_1w_2 \) and \( w_1w_3 \)).

• Word association metrics
  
  – PMI based on raw counts or smoothed prob

\[
\text{PMI}(w_i, w_j) = \log \frac{P(w_iw_j)}{P(w_i)P(w_j)} \quad \text{PMI}^{[MLE]}(w_i, w_j) = \log \frac{C(w_iw_j)/N}{C(w_i)/N \cdot C(w_j)/N} \quad \propto \log \frac{C(w_iw_j)}{C(w_i)C(w_j)}
\]
## Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Association</th>
<th>anchor</th>
<th>title</th>
<th>body</th>
<th>query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong</td>
<td>[Baseline]</td>
<td></td>
<td></td>
<td></td>
<td>66.3%</td>
</tr>
<tr>
<td></td>
<td>PMI</td>
<td>92.0%</td>
<td>90.9%</td>
<td>93.3%</td>
<td>91.0%</td>
</tr>
<tr>
<td></td>
<td>PMI[^{MLE}]</td>
<td>86.3%</td>
<td>84.0%</td>
<td>86.2%</td>
<td>84.7%</td>
</tr>
<tr>
<td></td>
<td>$\chi^2$</td>
<td>90.6%</td>
<td>90.0%</td>
<td>91.1%</td>
<td>90.7%</td>
</tr>
<tr>
<td></td>
<td>Cond. Prob.</td>
<td>88.1%</td>
<td>86.9%</td>
<td>86.9%</td>
<td>85.9%</td>
</tr>
<tr>
<td>Strong + Weak</td>
<td>[Baseline]</td>
<td></td>
<td></td>
<td></td>
<td>60.8%</td>
</tr>
<tr>
<td></td>
<td>PMI</td>
<td>90.2%</td>
<td>89.5%</td>
<td>90.9%</td>
<td>88.9%</td>
</tr>
<tr>
<td></td>
<td>PMI[^{MLE}]</td>
<td>84.5%</td>
<td>83.1%</td>
<td>84.7%</td>
<td>83.0%</td>
</tr>
<tr>
<td></td>
<td>$\chi^2$</td>
<td>88.6%</td>
<td>88.3%</td>
<td>89.6%</td>
<td>89.3%</td>
</tr>
<tr>
<td></td>
<td>Cond. Prob.</td>
<td>86.3%</td>
<td>85.2%</td>
<td>84.5%</td>
<td>84.1%</td>
</tr>
</tbody>
</table>
Long Query Segmentation

• Task

• Method: best-first search based on SPMI

\[ n^* = \arg \min_{n \in C} \min_{2 \leq t \leq |q(n)|} \text{SPMI}(q(n), t) \]

\[ \text{SPMI}(q, t) = \log \frac{P(q_t q_r)}{P(q_t)P(q_r)} \]
Results

Figure 7: Tradeoff between the exact match rate and the violation rate for long query segmentation using different n-gram language models.
Conclusion

• Web page is a multi-field text
  – Web corpus is an aligned multi-lingual corpus
• We can build large and smart models
• Performance of a LM depends on
  – Language (style), model size, model order and smoothing
• Web as baseline for IR/NLP research
  – Bing-It-On-Ngram
Language Models to Drive Innovation

Natural Languages are used in specifying intents in queries; in expressing information in documents

Language Models leading technology in Search, Machine Translation, Speech, Learning

P(next word | N-1 preceding words)
Better “understanding” = model predicts better

Search applications
Query processing: alterations, expansions, suggestions, spelling
Document processing: classification, clustering

Petabytes of data via Cloud-based Services

Content types: Document Body, Document Title, Anchor Texts
Model types: Smoothed models
N-gram availability: up to 5-grams (no cut off)
Training size (Body): All documents indexed by Bing
Access: Microsoft Hosted Services
Updates: Periodical updates

Web Scale Language Models for Search

Monthly new N-grams
Query perplexity and OOV mas with Good-Tuning
Perplexities among various fields with respect to queries

We highlight Microsoft unique offering w.r.t. language models

DATA
Web documents

Web N-GRAM Service

INFORMATION
Language Models

“Raleigh Serengeti!” recognised as an entity using Anchor Text and Document Title, and unlike using Body

Implicit Search

Web N-GRAM SEGMENTATION DEMO

Making sense of hash tags to follow the most popular Tweets on the web

Internal: http://webgram
External: http://research.microsoft.com/webgram

Microsoft Research
K. Wang, C. Thrasher, P. Hsu, X. Li
J. Gao
E. Viegas

MSR ISRC
MSR NLP
MSR ER

F. Behr, Z. Zheng
OSD Bing

Microsoft Research