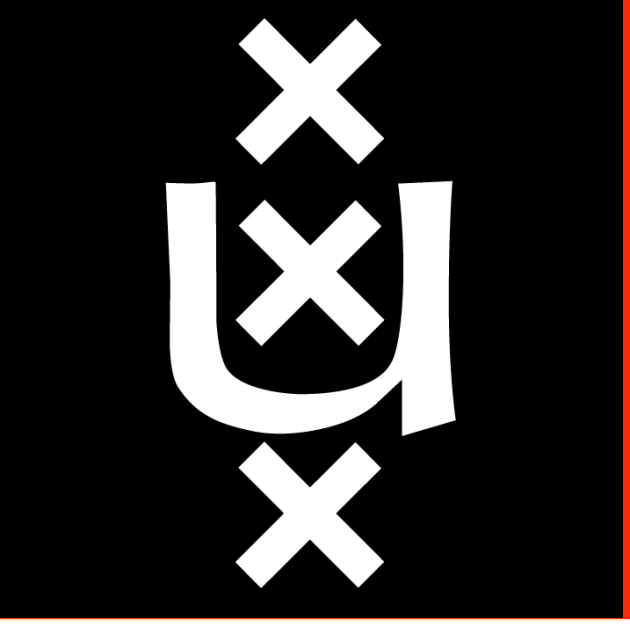


Christof Monz

Applied Language Technology
Language Modeling for SMT

Today's Class

- Role of LMs in SMT
- Kneser-Ney smoothing
- Large language models
- Interpolation of LMs
- Current research developments in language modeling (for SMT)

The Role of LM in SMT

- Translation models map source phrases to target phrases
  - Translation probabilities should reflect the degree to which the meaning of the source phrase is preserved by the target phrase (adequacy)
  - source: "Der Mann hat einen Hund gekauft."
    monotone translation: "The man has a dog bought."
    Translation preserves the meaning but is not fluent
- Language models compute the probability of a string
  - $p(\text{the man has a dog bought.}) < p(\text{the man has bought a dog.})$
  - Language model probabilities do not necessarily correlate with grammaticality: $p(\text{green ideas sleep furiously.})$ is likely to be small
  - During translation language model scores of translation hypotheses are compared to each other

The Role of LM in SMT

- In SMT the language model is mainly used for two purposes
- Ensuring fluency: Disfluencies can arise from
  - phrase translation candidates: (hat,did), (hat,has), and (gekauft,bought)
    $p(\text{the man did bought}) < p(\text{the man has bought})$
  - source to target reorderings:
    $p(\text{the man has a dog bought.}) < p(\text{the man has bought a dog.})$
- Contextual disambiguation:
  - "Er schoss ein Tor." where (Tor,gate) and (Tor,goal)
    $p(\text{he scored a gate.}) < p(\text{he scored a goal.})$
  - Disambiguation is limited to local context (n-gram window)

The Role of LM in SMT

- The language model is one of the most important models in SMT
- Substantial improvements in translation quality can be gained from carefully trained language models
- Decades of research (and engineering) in language modeling for Automated Speech Recognition (ASR)
  - Many insights can be transferred to SMT
  - Types of causes for disfluencies differ between both areas
    ASR: $p(\text{We won't I scream}) < p(\text{We want ice cream})$
    SMT: $p(\text{Get we ice cream}) < p(\text{We want ice cream})$
  - Reordering does not play a role in ASR

N-gram Language Modeling

- N-gram language model compute the probability of a string as the product of probabilities of the consecutive n-grams:
  - $p(<\text{s}>\ \text{the man has a dog bought .}\ </\text{s}>)$ $= p(<\text{s}>\ \text{the}) \cdot p(<\text{s}>\ \text{the man}) \cdot p(\text{the man has}) \cdot p(\text{man has a}) \cdot p(\text{has a dog}) \cdot p(\text{a dog bought}) \cdot p(\text{dog bought .}) \cdot p(\text{bought .}\ </\text{s}>)$
  - Generally: $p(w_1^N) = \prod_{i=1}^{N} p(w_i | w_{i-n+1}^{i-1})$, for order n
  - Problem: if one n-gram probability is zero, e.g., $p(\text{dog bought .}) = 0$, then the probability of the entire product is zero
  - Solution: smoothing

Language Model Smoothing

- A number of smoothing approaches have been developed for language modeling
- Jelinek-Mercer smoothing
  - Weighted linear interpolation of conditional probabilities of different orders
- Katz smoothing
  - Back-off to lower-order probabilities and counts are discounted
- Witten-Bell smoothing
  - Linear interpolation where lower-order probabilities are weighted by the number of contexts of the history
- Kneser-Ney smoothing
  - Weight lower-order probabilities by the number of contexts in which they occur

Kneser-Ney Smoothing

$$p_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i})), 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

- Original backoff-style formulation of Kneser-Ney smoothing
  - Closer to representation found in ARPA style language models
  - Can be re-formulated as linear interpolation (see Chen and Goodman 1999)

Kneser-Ney Smoothing

if $\text{length}(w_i|w_{i-n+1}^{i-1}) = n$:

$$p_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i})), 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\text{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

$$D(c) = \begin{cases} 0 & \text{if } c = 0 \\ D_1 & \text{if } c = 1 \\ D_2 & \text{if } c = 2 \\ D_{3+} & \text{if } c \geq 3 \end{cases} \quad \text{and} \quad \begin{aligned} D_1 &= 1 - 2\frac{n_1}{n_1+2n_2}\frac{n_2}{n_1} \\ D_2 &= 2 - 3\frac{n_1}{n_1+2n_2}\frac{n_3}{n_2} \\ D_{3+} &= 3 - 4\frac{n_1}{n_1+2n_2}\frac{n_4}{n_3} \end{aligned}$$

$$\gamma(w_{i-n+1}^{i-1}) = \frac{D_1 N_1(w_{i-n+1}^{i-1}\bullet) + D_2 N_2(w_{i-n+1}^{i-1}\bullet) + D_{3+} N_{3+}(w_{i-n+1}^{i-1}\bullet)}{\sum_{w_i} c(w_{i-n+1}^{i})}$$

Kneser-Ney Smoothing

if $\text{length}(w_i|w_{i-n+1}^{i-1}) < n$:

$$p_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{N_{1+}(\bullet w_{i-n+2}^{i}) - D(c(w_{i-n+1}^{i})), 0\}}{\sum_{w_i} N_{1+}(\bullet w_{i-n+2}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\text{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

For lower order n-grams p_{KN} is not based on the token counts but on the type counts (number of different words, regardless of their frequency) of the first word in the history

Kneser-Ney Smoothing

- Assume an n-gram language model has been built based on these n-gram counts:

$$p_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i})), 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

Kneser-Ney Smoothing

- Assume an n-gram language model has been built based on these n-gram counts:

$$p_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i})), 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\text{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

n-gram	count
has buy not	0
has buy	0
buy not	0
buy	> 0
not	> 0

Kneser-Ney Smoothing

- Assume an n-gram language model has been built based on these n-gram counts:

$$p_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i})), 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\text{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

n-gram	count
has buy not	0
has buy	0
buy not	0
buy	> 0
not	> 0

$p_{\text{KN}}(\text{not} \mid \text{has buy})$

Kneser-Ney Smoothing

- Assume an n-gram language model has been built based on these n-gram counts:

$$p_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i})), 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\text{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

n-gram	count
has buy not	0
has buy	0
buy not	0
buy	> 0
not	> 0

$$p_{\text{KN}}(\text{not} \mid \text{has buy}) = \gamma(\text{has buy})$$

Kneser-Ney Smoothing

- Assume an n-gram language model has been built based on these n-gram counts:

$$p_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i})), 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\text{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

n-gram	count
has buy not	0
has buy	0
buy not	0
buy	> 0
not	> 0

$$\begin{aligned} & p_{\text{KN}}(\text{not} \mid \text{has buy}) \\ = \quad & \gamma(\text{has buy}) \\ = \quad & 1 \end{aligned}$$

Kneser-Ney Smoothing

- Assume an n-gram language model has been built based on these n-gram counts:

$$p_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i})), 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\text{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

n-gram	count
has buy not	0
has buy	0
buy not	0
buy	> 0
not	> 0

$$\begin{aligned} & p_{\text{KN}}(\text{not} \mid \text{has buy}) \\ = \; & \gamma(\text{has buy}) \\ = \; & 1 \cdot p_{\text{KN}}(\text{not} \mid \text{buy}) \end{aligned}$$

Kneser-Ney Smoothing

- Assume an n-gram language model has been built based on these n-gram counts:

$$p_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i})), 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

n-gram	count
has buy not	0
has buy	0
buy not	0
buy	> 0
not	> 0

$$\begin{aligned} & p_{\mathrm{KN}}(\text{not} \mid \text{has buy}) \\ = \; & \gamma(\text{has buy}) \\ = \; & 1 \cdot p_{\mathrm{KN}}(\text{not} \mid \text{buy}) \\ & = \gamma(\text{buy}) \end{aligned}$$

Kneser-Ney Smoothing

- Assume an n-gram language model has been built based on these n-gram counts:

$$p_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i})), 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\text{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

n-gram	count
has buy not	0
has buy	0
buy not	0
buy	> 0
not	> 0

$$\begin{aligned} & p_{\text{KN}}(\text{not} \mid \text{has buy}) \\ = \; & \gamma(\text{has buy}) \\ = \; & 1 \cdot p_{\text{KN}}(\text{not} \mid \text{buy}) \\ & = \gamma(\text{buy}) \cdot p_{\text{KN}}(\text{not}) \end{aligned}$$

Kneser-Ney Smoothing

- Assume an n-gram language model has been built based on these n-gram counts:

$$p_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i})), 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\text{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

n-gram	count
has buy not	0
has buy	0
buy not	0
buy	> 0
not	> 0

$$\begin{aligned} & p_{\text{KN}}(\text{not} \mid \text{has buy}) \\ = \; & \gamma(\text{has buy}) \\ = \; & 1 \cdot p_{\text{KN}}(\text{not} \mid \text{buy}) \\ & = \gamma(\text{buy}) \cdot p_{\text{KN}}(\text{not}) \end{aligned}$$

$$p_{\text{KN}}(\text{not} \mid \text{has buy}) = \gamma(\text{buy}) \cdot p_{\text{KN}}(\text{not})$$

LM Smoothing in SMT

- Does the choice of smoothing method matter for SMT?
  - Kneser-Ney smoothing typically yields results with the lowest perplexity
  - Correlation between perplexity and MT metrics (such a BLEU) is low
  - Few comparative studies, but Kneser-Ney smoothing yields small gains over Witten-Bell smoothing
- Kneser-Ney smoothing is the de facto standard for SMT (and ASR)
- Recent SMT research combines Witten-Bell smoothing with Kneser-Ney smoothing

ARPA Format

- Most language models store the language model file in the ARPA format:

```
-1.917789    banned for
-1.053701    banned from   -0.09642049
-2.533886    banned fur
-0.8801329   banned in     -0.1573689
```

- The columns are
  1. probability of n-gram
  2. n-gram
  3. backoff costs of n-gram
- Not all n-grams have backoff costs
  - $n_{1+}(\text{banned for } \bullet) = 0$
  - Can be due to actual data or removal of low-count n-grams

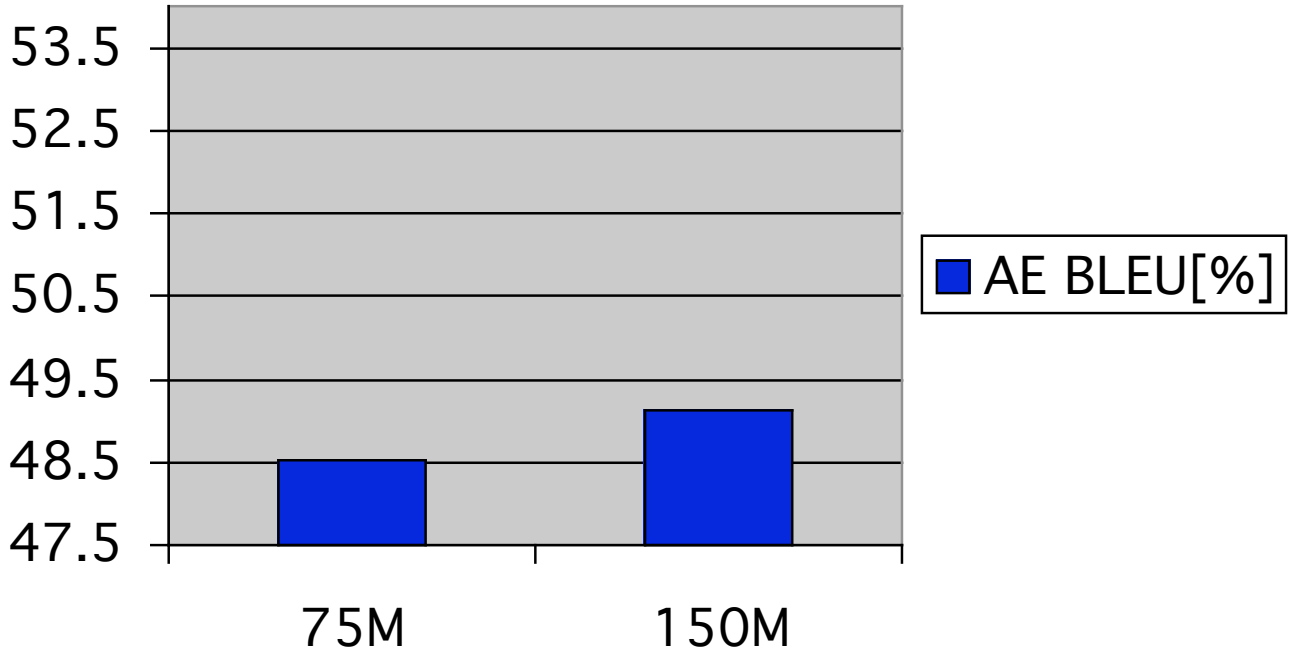
Language Modeling Data

- Translation modeling requires parallel data, which tends to be scarce
- Language modeling requires just monolingual data in the target language
  - The amount of available monolingual data is several orders of magnitude larger than the amount of available parallel data!
- Preprocessing of monolingual data
  - Detect sentence boundaries
  - Tokenize sentences
  - Normalize certain words, e.g., $1993 \rightarrow 6666$ and $27.3 \rightarrow 66.6$
  - Replace out-of-vocabulary words by <unk>
  - Remove duplicate sentences

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:



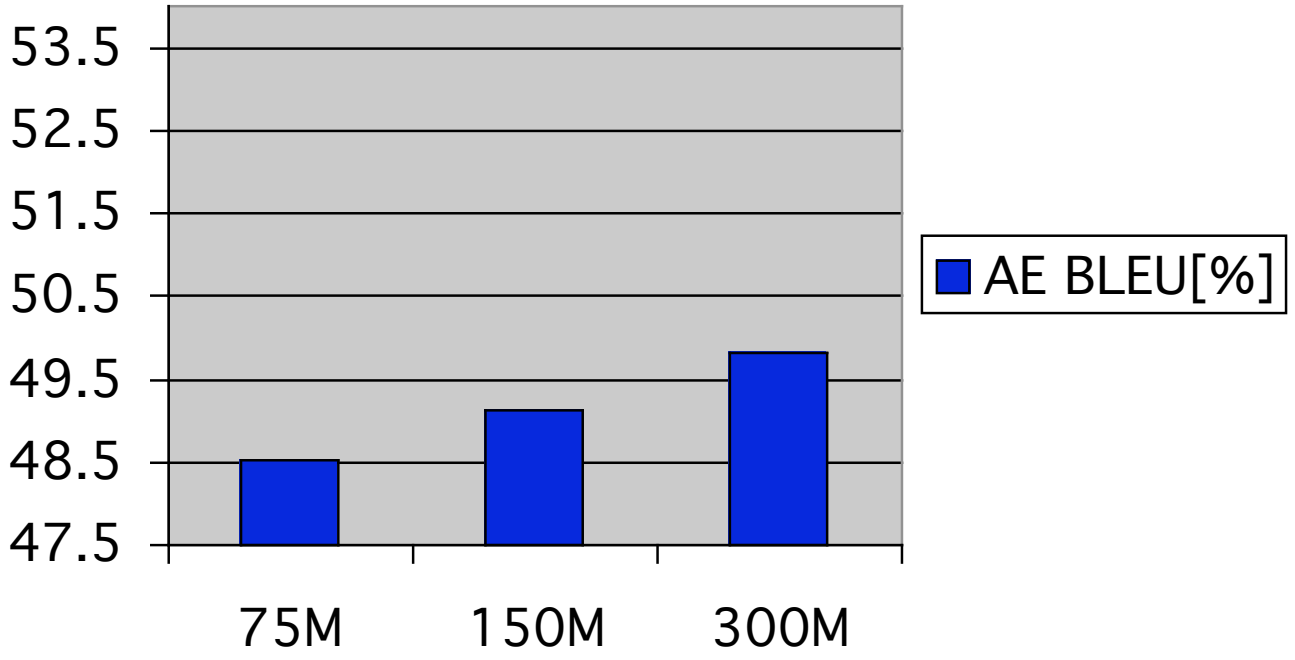
9

Google

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:



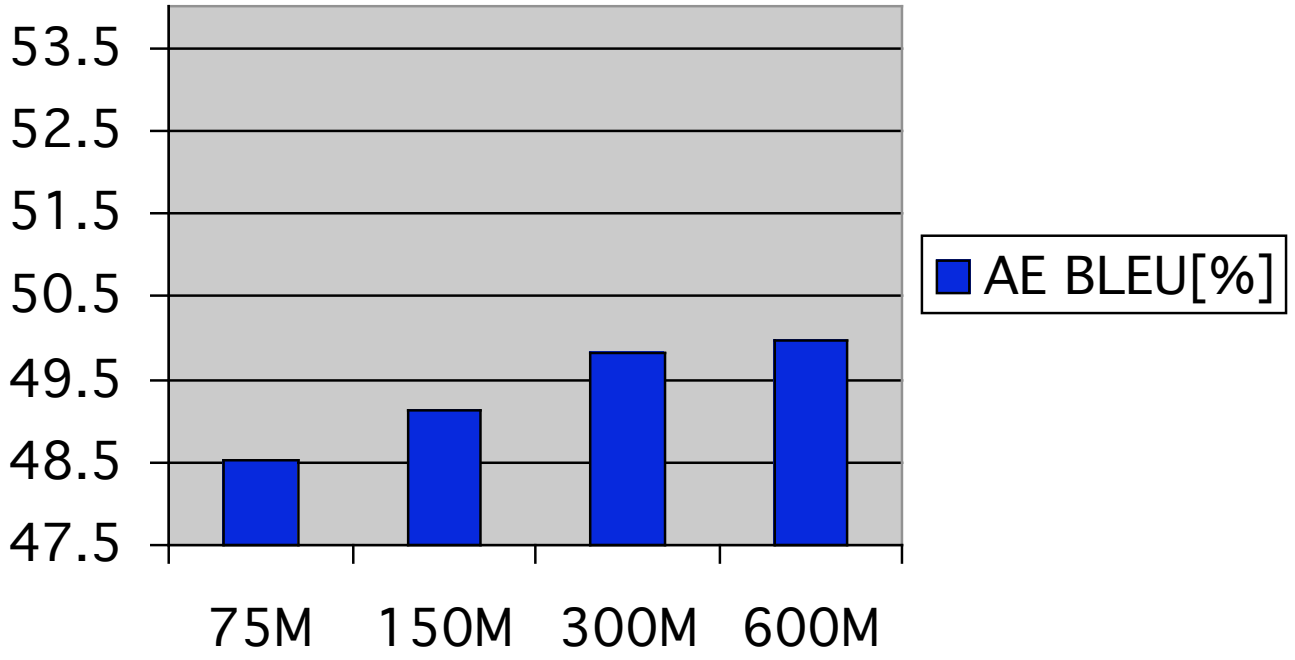
10

Google

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:



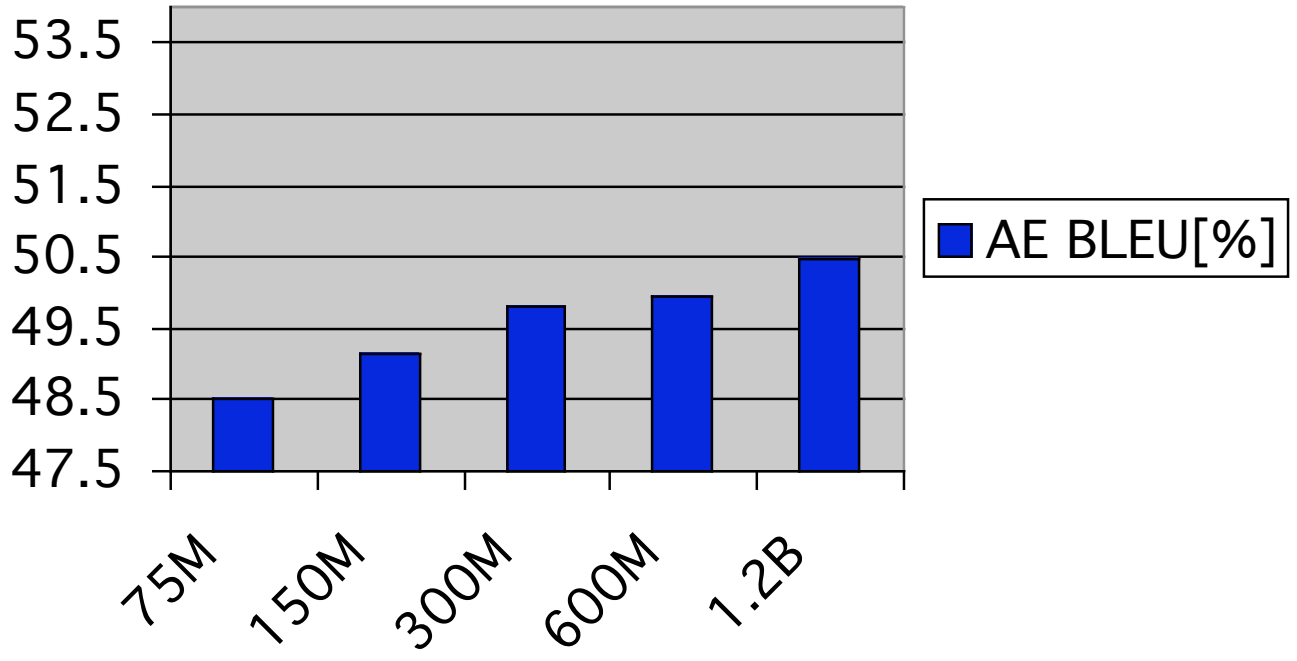
11

Google

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:



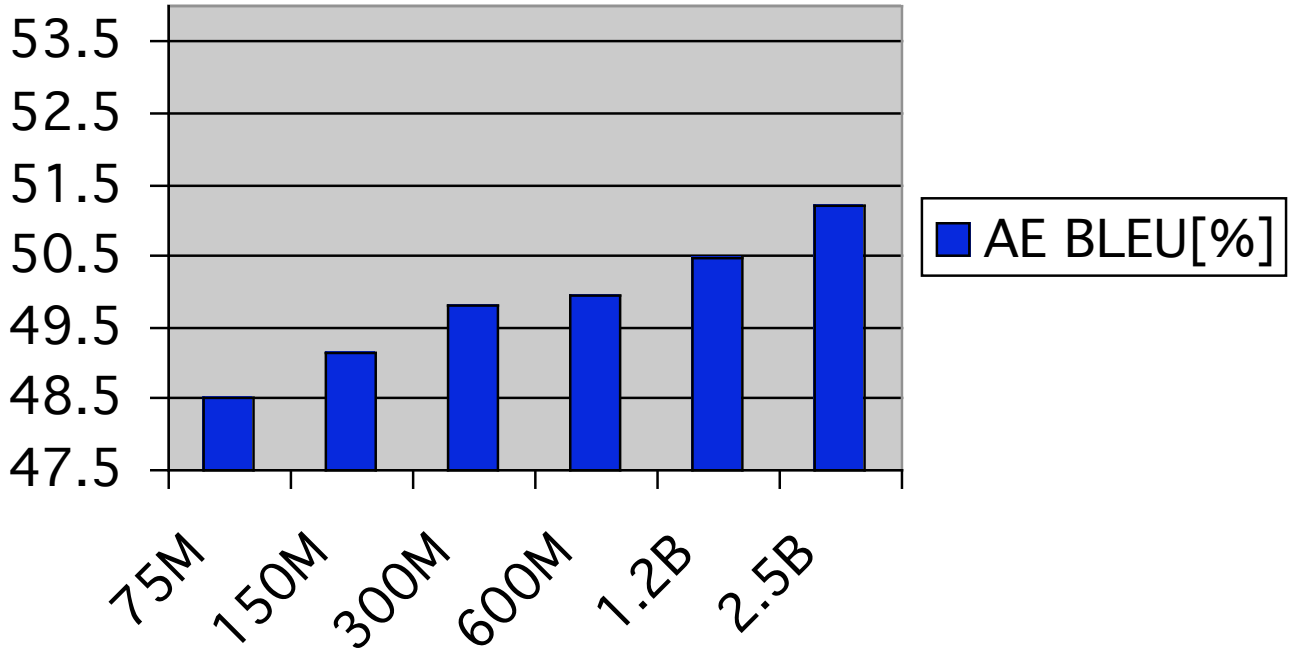
12

Google

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:



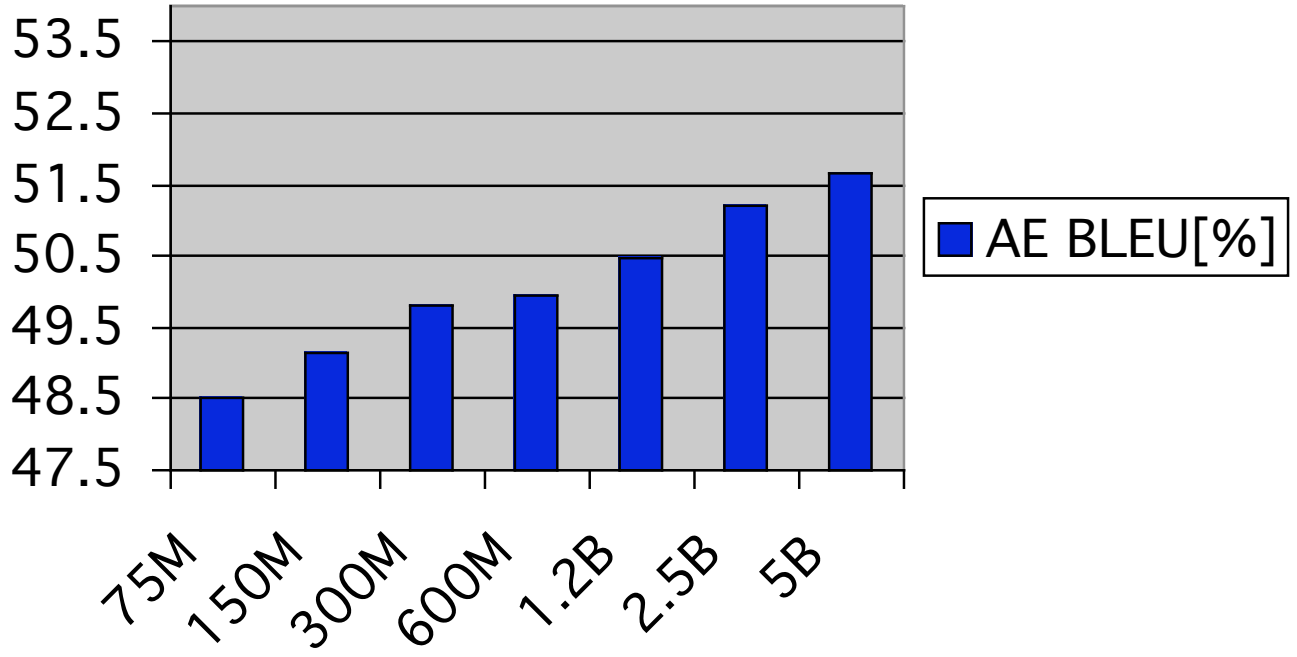
13

Google

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:



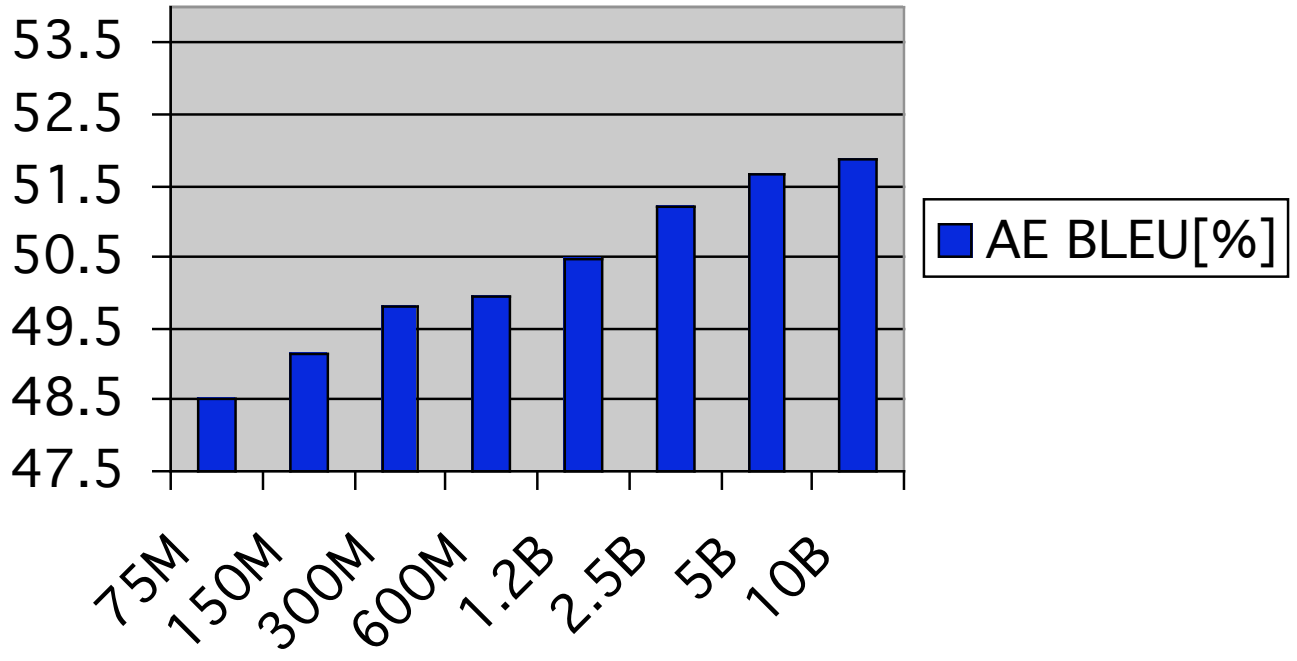
14

Google

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:



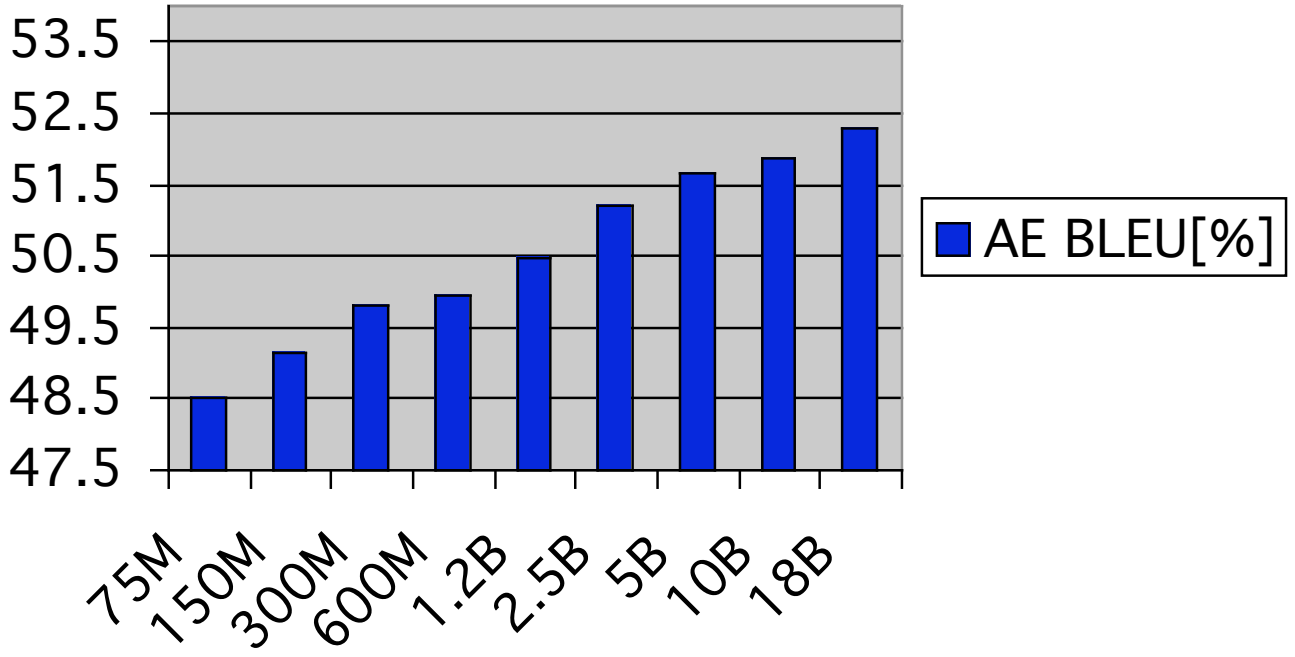
15

Google

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:



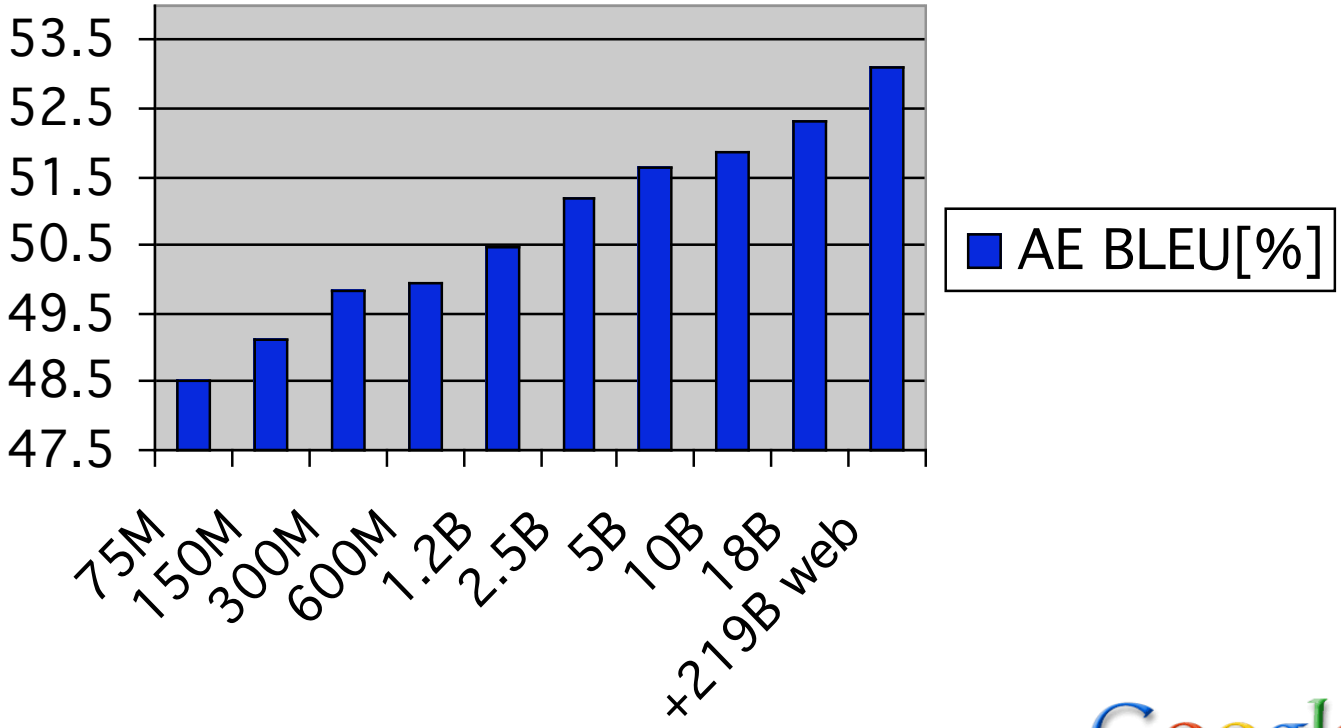
16

Google

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:



17

Google

Building Large LMs

- Large amounts of training data can soon become too big to build
  - Keeping all counts in memory intractable (even with today's machines)
- For simple maximum likelihood estimation

  $$p(w_{i-n+1}^{i}) = \frac{c(w_{i-n+1}^{i})}{c(w_{i-n+1}^{i-1})}$$

  we only need to keep in memory the counts for all continuations of a given history $\{w_i : c(w_{i-n+1}^{i-1} w_i) > 0\}$
- Keep data out of RAM and on disk as much as possible
  - write all n-gram counts from data to file
  - sort counts by history on disk (e.g., unix sort = merge sort)
  - compute probabilities for n-grams with same history in RAM
- This cannot be done so easily for smoothing estimates

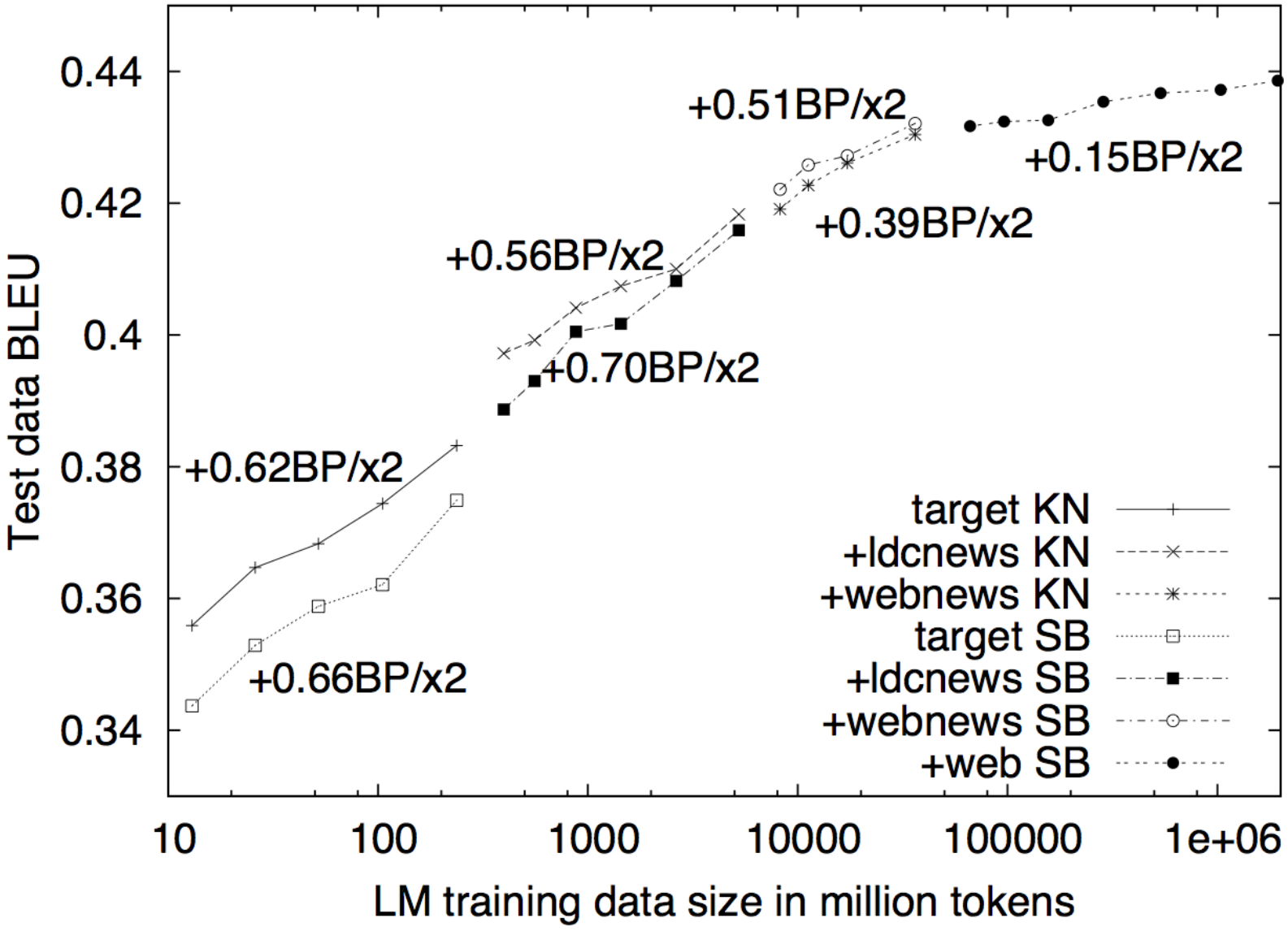
Stupid Backoff

- Proper smoothing, such as Kneser-Ney smoothing, becomes intractable for large amounts of data due to memory demands
- Combine maximum likelihood estimation with a trivial backoff strategy
  - Brants et al. (EMNLP 2009) coined their approach "stupid backoff"

$$S(w_{i-n+1}^{i}) = \begin{cases} \frac{c(w_{i-n+1}^{i})}{c(w_{i-n+1}^{i-1})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \alpha S(w_{i-n+2}^{i}) & \text{otherwise} \end{cases}$$

$\alpha = 0.4$ (set heuristically)
α is not parameterized in any way

Stupid Backoff



Language Model Servers

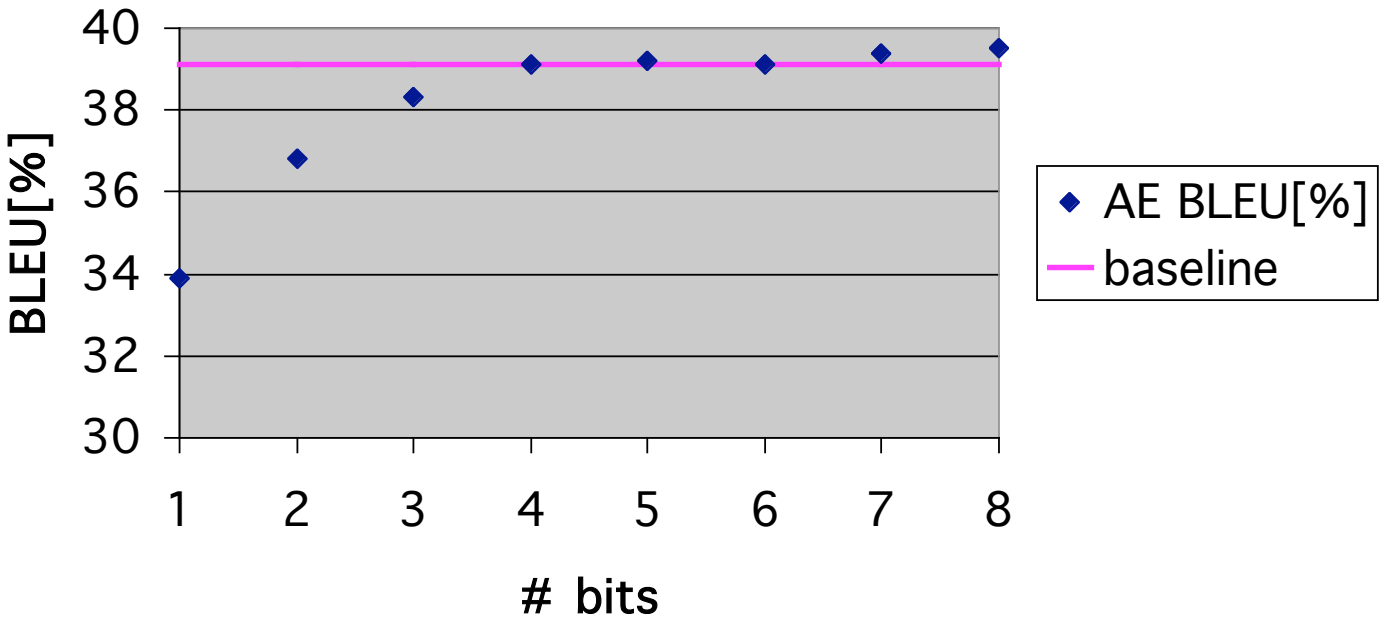
- Methods such as stupid backoff make it possible to train large language models
- During decoding we still have to keep the entire language model in RAM
- Solution: distribute portions of the language model over a number of servers
  - Each server can compute all required probabilities for a subset of n-grams
  - For example server-1 maintains all n-gram statistics where the last word begins with letters a to m, and server-2 n-grams n-z
- Servers are queried through network (bottleneck!)
  - Delay n-gram queries during decoding
  - Compute partial model costs (i.e., modulo LM costs) for a number of partial translations
  - Send a batch query for all relevant n-grams and recompute actual costs

Quantitization

- How precise do language model probabilities have to be?
  - We are not so much interested the actual probability but its ability to discriminate between degrees of fluency
- Log probabilities are typically stored as floats (4–8 bytes)
- Quantitization approximates actual probabilities by using a lower bit representation
- Two approaches to quantitization:
  - Lloyd's algorithm: Cluster values using K-Means clustering, distance is absolute numerical difference, keep only k centroids
  - Binning: Partition values into uniformly populated intervals
    - Numerically sort all unique values and partition into k non-overlapping intervals
    - The k intervals are represented by respective mean
- Pointers to approximated probabilities can be stored in $\lceil \log_2 k \rceil$ bits

Impact of Quantitization

How many bits are needed to store probabilities?

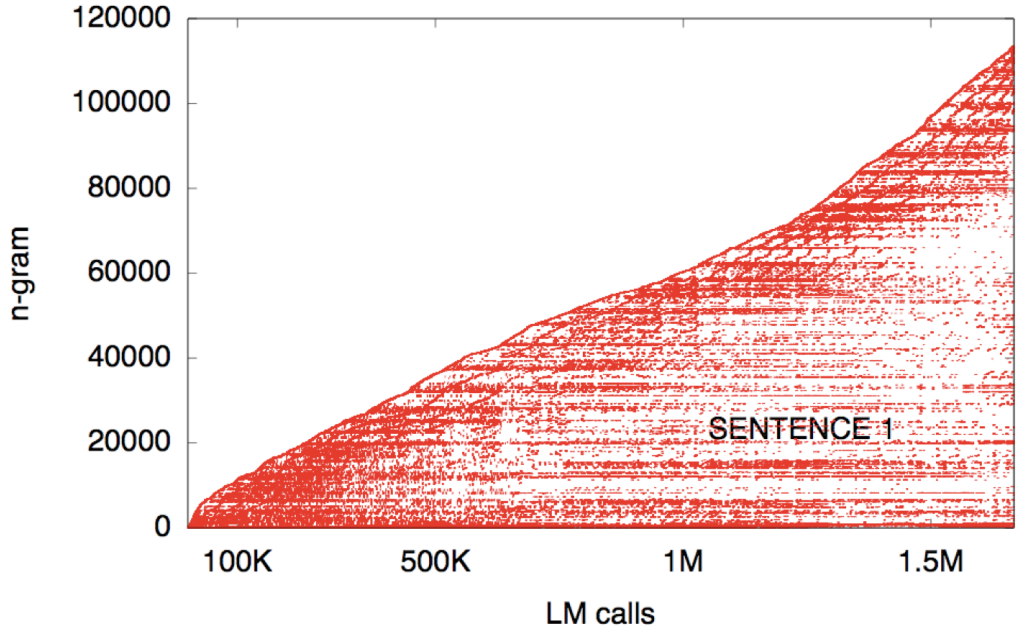


18

Google

Cost of Language Model Calls

- Calls to the language model are computationally not cheap
  - Despite efficient trie data structure
  - Several backoff steps need to carried out
- Large number of calls to LM during decoding



Cube Pruning

- Cube pruning (Huang and Chiang 2007): only compute language model costs for most 'promising' partial translation hypotheses
  - Compute model costs m' (modulo language model) for each hypothesis
  - Rank hypothesis by m'
  - Recompute actual model costs m (including language model) for top k hypotheses only
- Language model computation is not only delayed (as in the case of batch queries to LM servers) but restricted
- More details in lecture on decoding

Combining Multiple Language Models

- Instead of building one large language model using all available monolingual data one can also build multiple language models for different portions of the data
- Partitioning of the data can be based on multiple criteria
  - source: newswire, blogs, general web crawl
  - date: date window
  - target side of bitext vs rest
- Language models can be trained separately
  - mitigates the memory requirements when building large models
- Two ways to use multiple language models for translation
  - Use separate language models during decoding, each with a log-linear weight that can be tuned wrt translation quality
  - Linear interpolation of models wrt held out data
    - During decoding one interpolated model is used
- Experimental results show that linear interpolation consistently outperforms log-linear combination

LM Interpolation

- A separate language model has been built for each data portion $d \in D$
- Set aside some held-out data H (disjunct from the training and testing data)
  - Ideally in the same domain and genre as the test data (if known beforehand)
- Collect n-gram counts from H
- Estimate the optimal $(\lambda_1^*, \ldots, \lambda_{|D|}^*)$ by expectation maximization (EM)

LM Linear Interpolation with EM

- Initialize $\lambda_d^{(0)}$, such that $\forall d : 0 < \lambda_d^{(0)}$ and $\sum_{d=1}^{|D|} \lambda_d^{(0)} = 1$
  Iteration counter $t = 0$
- Update: $\forall d : e_d = \sum_{w_{i-n+1}^i \in H} \frac{c(w_{i-n+1}^i)\lambda_d^{(t)} p_d(w_{i-n+1}^i)}{\sum_{d=1}^{|D|} \lambda_d^{(t)} p_d(w_{i-n+1}^i)}$

  $\forall d : \lambda_d^{(t+1)} = \frac{e_d}{\sum_{d=1}^{|D|} e_d}$
- Convergence check:

  $\ell(\lambda^{(t)}) = \sum_{w_{i-n+1}^i \in H} c(w_{i-n+1}^i) \log \sum_{d=1}^{|D|} \lambda_d^{(t)} p_d(w_{i-n+1}^i)$

  $\ell(\lambda^{(t+1)}) = \sum_{w_{i-n+1}^i \in H} c(w_{i-n+1}^i) \log \sum_{d=1}^{|D|} \lambda_d^{(t+1)} p_d(w_{i-n+1}^i)$
- Stop if for some small value ε: $\frac{\ell(\lambda^{(t+1)}) - \ell(\lambda^{(t)})}{|\ell(\lambda^{(t+1)})|} \leq \varepsilon$

  Else $t = t + 1$
- $(\lambda_1^*, \ldots, \lambda_{|D|}^*) = (\lambda_1^{(t)}, \ldots, \lambda_{|D|}^{(t)})$

Anecdote in LM Interpolation

- When interpolating language models using different sources of Gigaword and target side of bitext I noticed
  - $\lambda^*_{\text{bitext}} = 0.3$ ($\approx$ 50M gzipped)
  - $\lambda^*_{\text{AFP}} = 0.15$ ($\approx$ 1.2G gzipped)
  - $\vdots$
  - $\lambda^*_{\text{XIN}} = 0.4$ ($\approx$ 600M gzipped)
  - $\lambda^*_{\text{UN}} = 0.07$ ($\approx$ 250M gzipped)
- One can note that the sources with the highest interpolation weight are the bitext and the Xinhua news corpus
  - Why?

Excerpt from Xinhua News

Thai Prime Minister Thaksin Shinawatra has explained the situation in the country's deep South to leaders of Jodan and Oman during his visits there, local newspaper reported Sunday.

In his weekly radio address, Thaksin said the two leaders were concerned with the development of the southern violence, and had pledged to explain to other Arab states about the real situation in the region.

The prime minister noted many Muslim countries misinterpretered the situation in the deep South through information provided by certain sources which reported it in a negative way.

Thai people should be against those trouble-makers in the South in respond to the queen's call for peaceful unity, he was quoted by Bangkok Post newspaper as saying.

Translationese LMs

- It is known within the area of translation studies that translated texts are significantly different from texts that are originally authored in the target language
  - Structurally different
  - Different word choices
  - Dependent on the original language
- Lembersky et al. (EMNLP 2011) investigated whether this fact can be exploited for SMT
- Build different language models
  - Only data authored in the target language
  - Only data translated from the source language into the target language
  - Only data translated from a different source language into the target language
  - Mixture of the above
  - All data sets were down-sampled to roughly the same size

Translationese LMs

DE to EN	
LM	**BLEU**
MIX	21.95
O-EN	*21.35*
T-DE	**22.42**
T-FR	21.47
T-IT	21.79
T-NL	21.59

IT to EN	
LM	**BLEU**
MIX	26.79
O-EN	*25.69*
T-DE	25.86
T-FR	26.56
T-IT	**27.28**
T-NL	25.77

FR to EN	
LM	**BLEU**
MIX	25.43
O-EN	*24.85*
T-DE	25.03
T-FR	**25.91**
T-IT	25.44
T-NL	25.17

NL to EN	
LM	**BLEU**
MIX	25.17
O-EN	*24.46*
T-DE	25.12
T-FR	24.79
T-IT	24.93
T-NL	**25.73**

Class-Based LMs

- So far we have only considered language models built using actual surface word forms
  - Issues of data sparsity are addressed by smoothing
- Another way to address data sparsity and achieve better generalizability is to use less parameterized representations with a more densely populated event space
- Build language models on sequences of classes instead of words
- Classes can be
  - Part of speech tags
  - Induced word clusters
  - Semantic roles
  - etc.
- While there are $100,000^5$ possible word 5-grams, there are only 50^5 possible PoS 5-grams

Class-Based LMs

- Simple class-based LMs can be built (and smoothed) just like word-based models
  - Can be used in combination with surface models
  - Separate models or interpolated
- Further possibilities by mixing classes and words:
  $p(\text{Tuesday}|\text{party on})$
  $=p(\text{WEEKDAY}|\text{party on}) \cdot p(\text{Tuesday}|\text{party on WEEKDAY})$
- In general:
  $p(w_i|w_{i-2}w_{i-1}) = p(W_i|w_{i-2}w_{i-1}) \cdot p(w_i|w_{i-2}w_{i-1}W_i)$

Class-Based LMs

- $p(W_i|w_{i-2}w_{i-1}) \cdot p(w_i|w_{i-2}w_{i-1}W_i)$

$$= \frac{p(w_{i-2}w_{i-1}W_i)}{p(w_{i-2}w_{i-1})} \cdot \frac{p(w_{i-2}w_{i-1}W_iw_i)}{p(w_{i-2}w_{i-1}W_i)}$$

$$= \frac{p(w_{i-2}w_{i-1}W_iw_i)}{p(w_{i-2}w_{i-1})} \qquad (1)$$

Class-Based LMs

- $p(W_i|w_{i-2}w_{i-1}) \cdot p(w_i|w_{i-2}w_{i-1}W_i)$

  $= \frac{p(w_{i-2}w_{i-1}W_i)}{p(w_{i-2}w_{i-1})} \cdot \frac{p(w_{i-2}w_{i-1}W_iw_i)}{p(w_{i-2}w_{i-1}W_i)}$

  $= \frac{p(w_{i-2}w_{i-1}W_iw_i)}{p(w_{i-2}w_{i-1})}$ (1)

- Since each word belongs to exactly 1 class: $p(W_i|w_i) = 1$

Class-Based LMs

- $p(W_i|w_{i-2}w_{i-1}) \cdot p(w_i|w_{i-2}w_{i-1}W_i)$

  $= \frac{p(w_{i-2}w_{i-1}W_i)}{p(w_{i-2}w_{i-1})} \cdot \frac{p(w_{i-2}w_{i-1}W_iw_i)}{p(w_{i-2}w_{i-1}W_i)}$

  $= \frac{p(w_{i-2}w_{i-1}W_iw_i)}{p(w_{i-2}w_{i-1})}$ (1)
- Since each word belongs to exactly 1 class: $p(W_i|w_i) = 1$
- $p(w_{i-2}w_{i-1}W_iw_i) = p(w_{i-2}w_{i-1}w_i) \cdot p(W_i|w_{i-2}w_{i-1}w_i)$

  $= p(w_{i-2}w_{i-1}w_i) \cdot p(W_i|w_i)$

  $= p(w_{i-2}w_{i-1}w_i)$ (2)
- Substituting (2) into (1) we get

  $p(W_i|w_{i-2}w_{i-1}) \cdot p(w_i|w_{i-2}w_{i-1}W_i)$

  $= \frac{p(w_{i-2}w_{i-1}w_i)}{p(w_{i-2}w_{i-1})} = p(w_i|w_{i-2}w_{i-1})$

Class-Based LMs

- Since we can prove the strict equality

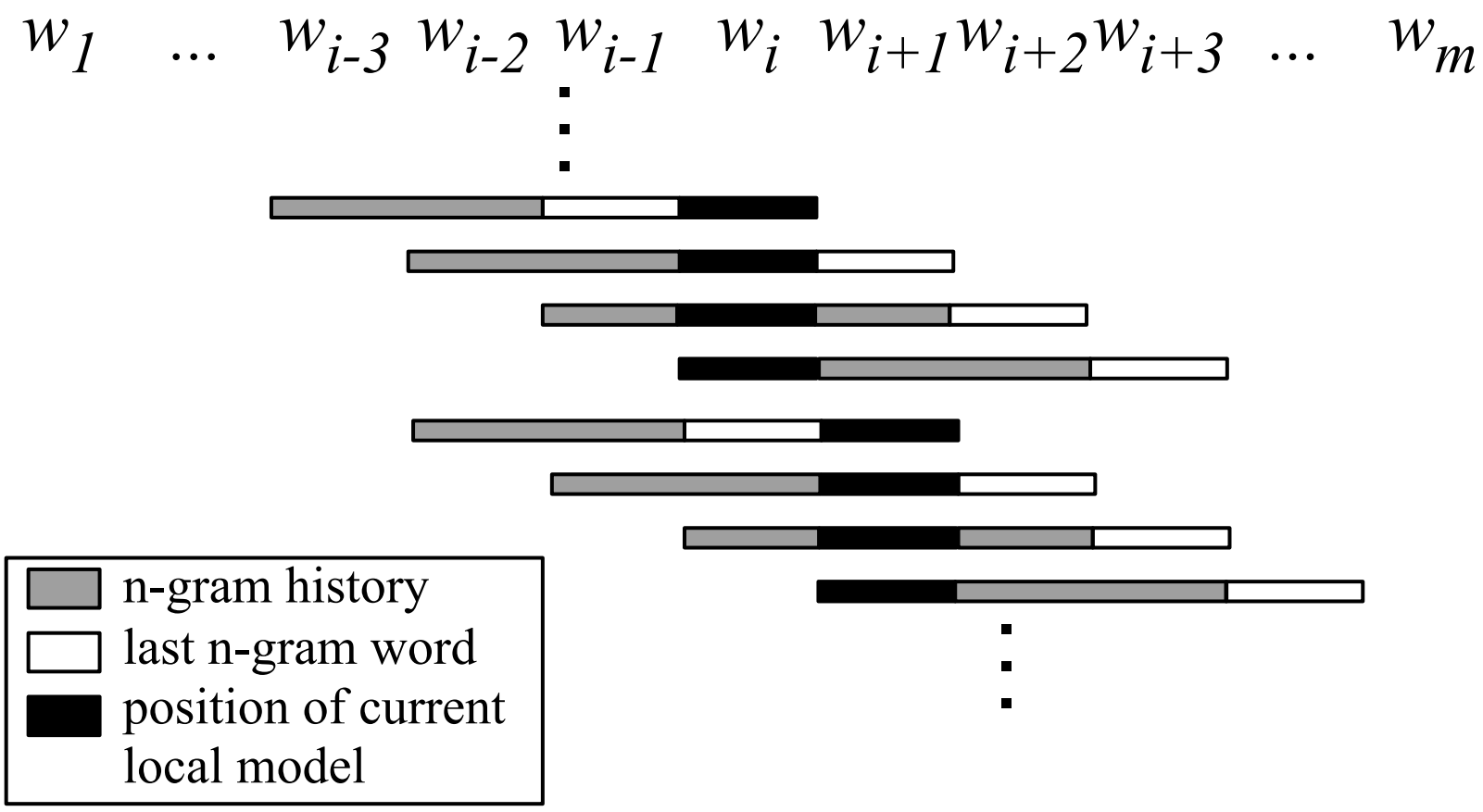
  $$p(W_i|w_{i-2}w_{i-1}) \cdot p(w_i|w_{i-2}w_{i-1}W_i) = p(w_i|w_{i-2}w_{i-1})$$

  why bother with classes?
- Both forms can still behave differently in combination with smoothing
- Imagine we have never seen "party on Tuesday" during training, but we have seen "party on Wednesday" and "party on Friday", and therefore "party on WEEKDAY"
  - $p(\text{WEEKDAY}|\text{party on})$ will be relatively high
  - Even though we have not observed "party on WEEKDAY Tuesday" we can backing off to or interpolate with $p(\text{Tuesday}|\text{on WEEKDAY})$

Local LMs

- So far we have consider cases of having one large language model or a few language models per source, date, class-type
- Another form of mixing classes with lexicalized information is to use a separate language class model (local model) for each word (Monz, EMNLP 2011)
- Probabilities are of the form: $p_c(c_3 : +2 | c_1 : -1,\ c_2 : +1)$
- A separate probability distribution is associated with each word (local model)
- The probability of a word occurrence is based on all local models in its context

Local LMs


w_1 ... w_{i-3} w_{i-2} w_{i-1} w_i w_{i+1} w_{i+2} w_{i+3} ... w_m
n-gram history
last n-gram word
position of current local model

Local LMs

position	0	1	2	3	4	5	6
token	<s>	Cuba	frees	more	dissidents	.	</s>
POS tag	<s>	NNP	VBZ	JJR	NNS	.	</s>

$$p(\text{cuba} : \text{NNS}) = p_{<\text{s}>:<\text{s}>}(\text{NNS} : +1)$$

$$p(\text{frees} : \text{VBZ}) = p_{<\text{s}>:<\text{s}>}(\text{VBZ} : +2 | \text{NNP} : +1) \cdot p_{\text{cuba:NNP}}(\text{VBZ} : +1 | <\text{s}> : -1)$$

$$p(\text{more} : \text{JJR}) = p_{<\text{s}>:<\text{s}>}(\text{JJR} : +3 | \text{NNP} : +1 \ \text{VBZ} : +2) \cdot p_{\text{cuba:NNP}}(\text{JJR} : +2 | <\text{s}> : -1 \ \text{VBZ} : +1) \cdot p_{\text{frees:VBZ}}(\text{JJR} : +1 | <\text{s}> : -2 \ \text{NNP} : -1)$$

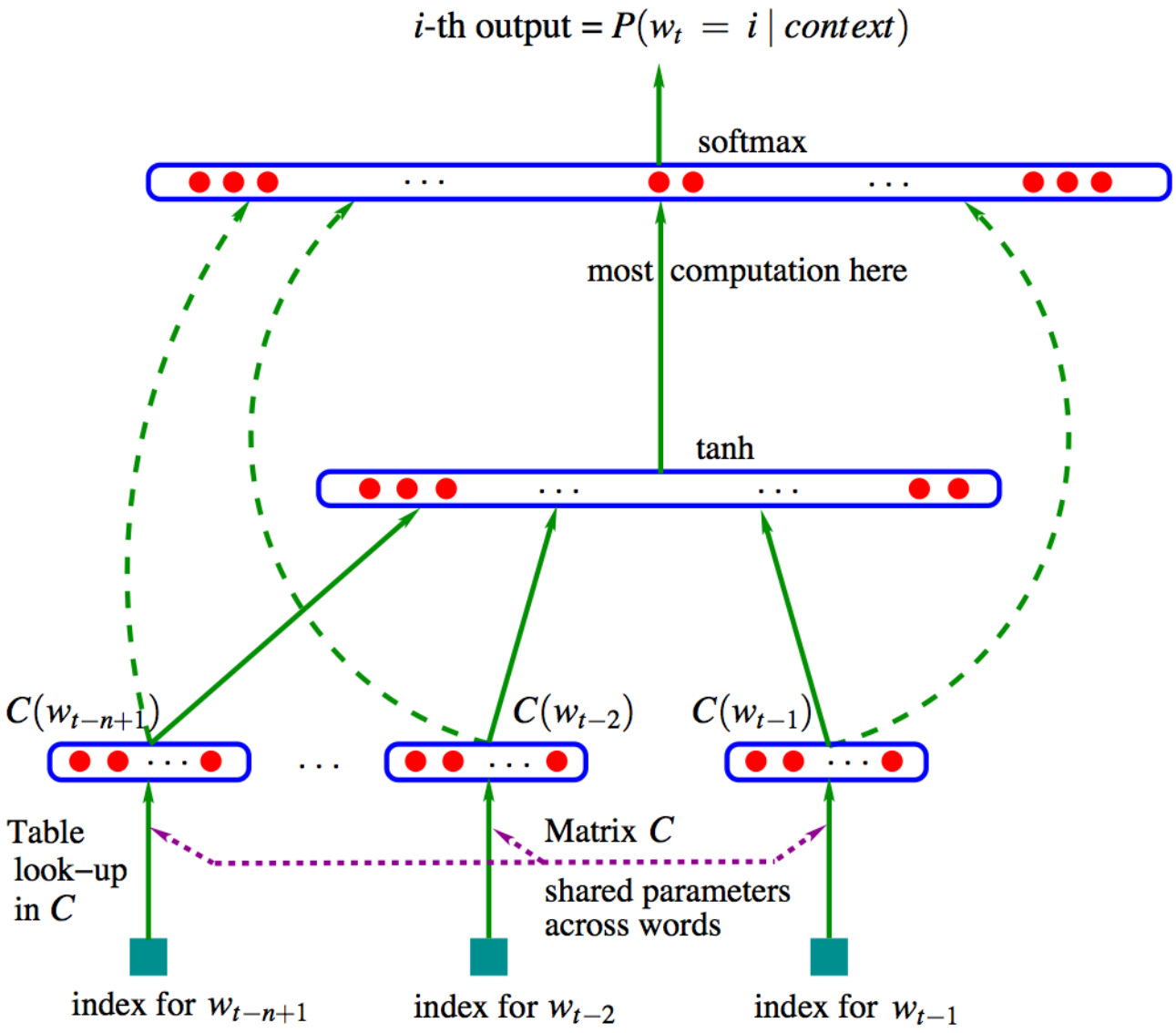
$$p(\text{dissidents} : \text{NNS}) = p_{\text{cuba:NNS}}(\text{NNS} : +3 | \text{VBZ} : +1 \ \text{JJR} : +2) \cdot p_{\text{frees:VBZ}}(\text{NNS} : +2 | \text{NNP} : -1 \ \text{JJR} : +1) \cdot p_{\text{more:JJR}}(\text{NNS} : +1 | \text{NNP} : -2 \ \text{VBZ} : -1)$$

...

Probabilistic Neural Network LMs

- Both word- and class-based models use discrete parameters as elements of the event space
- The current word+history n-gram has not been seen during training or it has not been seen (binary decision)
  - Smoothing results in a more relaxed matching criterion
- Probabilistic Neural Network LMs (Bengio et al. JMLR 2003) use a distributed real-valued representation of words and contexts
- Each word in the vocabulary is mapped to a m-dimensional real-valued vector
  - $C(w) \in \mathbb{R}^m$, typical values for m are 50, 100, 150
  - A hidden layer capture the contextual dependencies between words in an n-gram
  - The output layer is a $|V|$-dimensional vector describing the probability distribution of $p(w_i|w_{i-n+1}^{i-1})$

Probabilistic Neural Network LMs

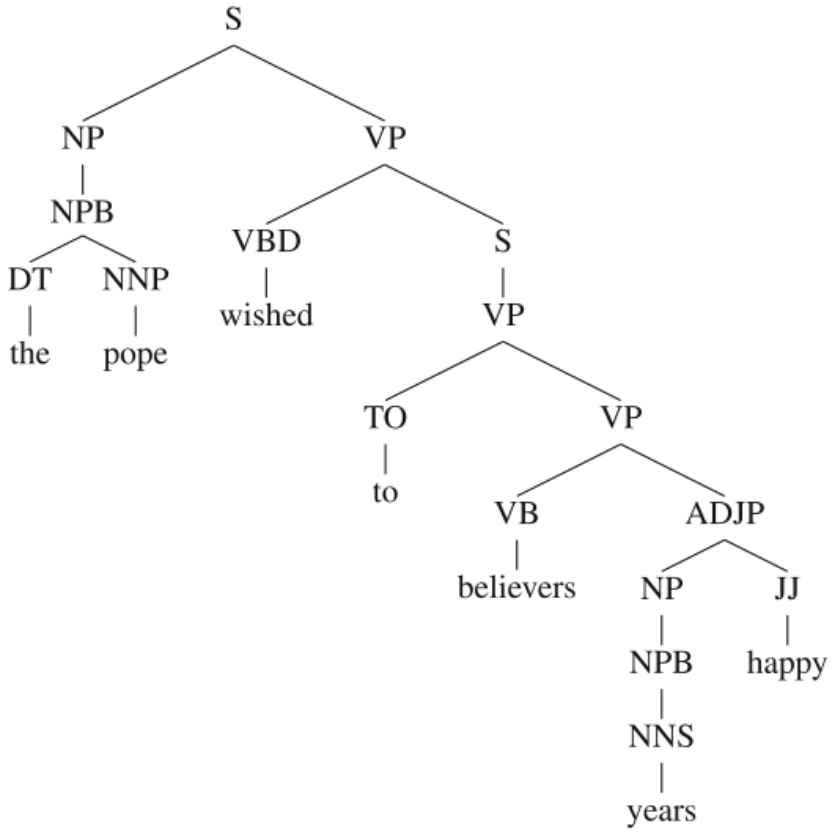


Parsing as Language Modeling

- So far fluency was estimated by using very local representations (n-grams)
  - Difficult to detect long-distance disfluencies
    "The **advisors** to the department of commerce in Washington, D.C., **has** quit."
  - Would require a 12-gram language model!
- Why don't we use syntactic parser to score partial translations?
  - Parse probabilities do not correlate that well with fluency
  - Introduces another latent variable that needs to be maximized
  - $\text{trans}(f) = \text{argmax}_e \sum_{m=1}^{M} \lambda_m h_m(e,f)$

    $\text{trans}(f) = \text{yield}(\text{argmax}_d \sum_{m=1}^{M} \lambda_m h_m(\text{yield}(d),f))$
    - where $\text{yield}(d)$ is the target sentence corresponding to derivation d
  - $D(e) = \{d : \text{yield}(d) = e\} \quad T(e) = \{t : \text{yield}(t) = e\}$
  - $\bigcup_{t \in T(e)} D(t) \gg D(e)$

Parsing as Language Modeling



Type	Sentence	LM4	CM2
SMT	the pope wished to believers years happy	**-26.61**	-5.5
REF 1	pope wishes worshipers happy new year	-22.75	**-7.13**
REF 2	pope wishes believers happy year .	-24.21	-6.01
REF 3	pope wishes the faithful a happy year	-25.86	-5.72
REF 4	pope wishes happy year for the faithful	-23.3	-5.92

Discriminative LMs

- In standard n-gram language modeling (same applies to Probabilistic NNLMs) we score each n-gram in a given sequence

  $$p(w_1^N) = \prod_{i=1}^{N} p(w_i | w_{i-n+1}^{i-1})$$

- Alternatively, we can model certain phenomena that affect the probability of a sentence as global features

  ```
  advisors ...  has
  advisors ...  have
  NNS ...  have
  NNS ...  VBZ
  advisors to
  advisors britain
  ```

- Features can be anything! (almost)

  $p_d(w_1^N) = \exp(\sum_{k=1}^{K} \lambda_k l_k(w_1^N))$

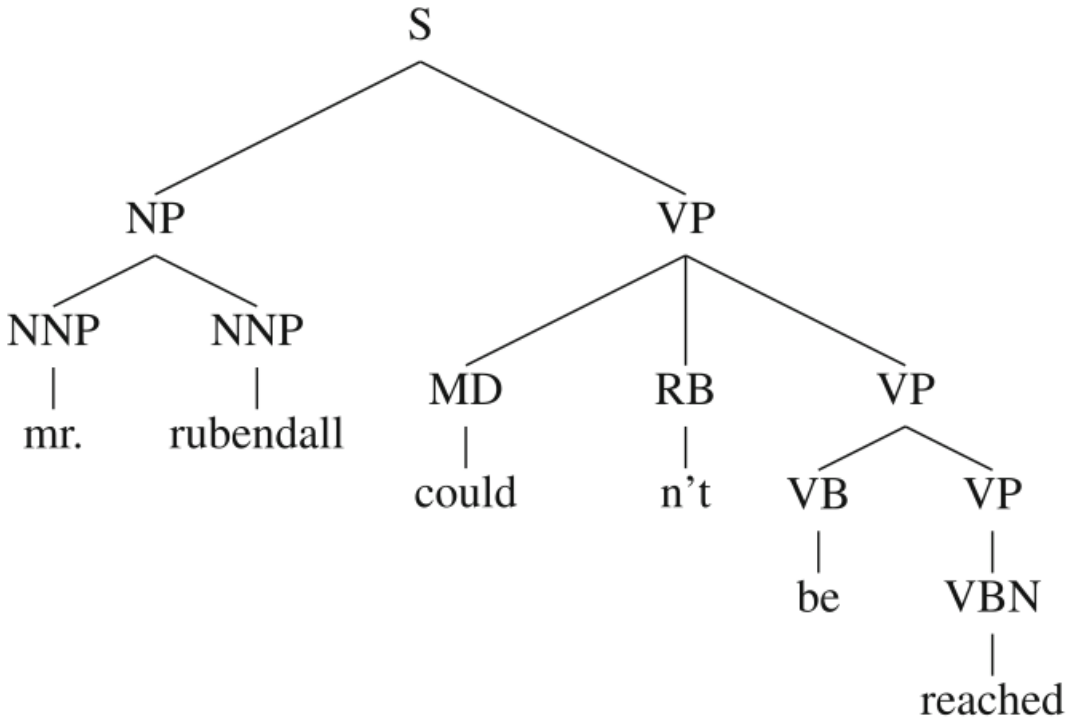
  Applied to MT by Carter and Monz (MTJ 2012)

Discriminative LMs

- How do we estimate the K feature weights in

  $p_d(w_1^N) = \exp(\sum_{k=1}^{K} \lambda_k l_k(w_1^N))$?
- Discriminative learning based on machine translation nbest-lists and oracle translations
  - E.g., perceptron based learning: pairwise comparison of translations (t) and the oracle (o)
  - If a translation is not the oracle update the feature weights s.t. they approximate the feature weights of the oracle
- During decoding use the optimized feature weights
- Some (long range) features are difficult to apply during decoding
  - Restricted to n-best rescoring (where whole-sentence information is available)

Discriminative LMs



(POS)

mr/NNP rubendall/NNP could/MD
n't/RB be/VB reached/VBN

(SEQ-B)

mr/NP^b rubendall/NP^c could/VP^b
n't/VP^c be/VP^b reached/VP^b

(SEQ-C)

mr/NNP-NP^b rubendall/NNP-NP^c
could/MD-VP^b n't/RB-VP^c
be/VB-VP^b reached/VBN-VP^b

Other LMs

- Backward LMs
  - Similar to standard language models, but the language model probability is computed from right to left
    $p(w_1^N) = \prod_{i=1}^{N} p(w_i|\text{reverse}(w_{i+1}^{i+n-1}))$
  - Simply trained by applying $\text{reverse}(w_1^N)$ to all sentences in the training data
- Skip LMs
  - Some phenomena are beyond the context that can be modeled by n-grams of order n
  - Drop elements in the history

    $p(w_i|w_{i-4}w_{i-3}w_{i-1}$ or $p(w_i|w_{i-4}w_{i-2}w_{i-1})$
    Retains n-grams of order n
    but stretches to dependencies beyond n $(w_{i-4} \rightsquigarrow w_i)$
  - Problem: many choices for dropping words

Other LMs

- Trigger LMs
  - Certain words are unlikely to appear in general, e.g., "shoe-bomber"
  - Wider context can increase the probabilities of these low frequency items
    - "airplane", "Al-Qaeda", "terrorist", "'plot", etc.
  - The trigger probability of a word w in context C can be computed as $p(w|C) = \sum_{w' \in C} p(w|w')$
    - C can be the sentence, part of the document, or the entire document

Bilingual LMs

- So far all language model have been restricted to one language (target language)
- Bilingual LMs combine information from the source and target side (Casacuberta and Vidal, CL 2004)
  - Trained on parallel corpora and the corresponding word-alignment
- A bilingual token is a tuple of the form (f_j, e_i) and n-grams are sequences of tuples:
  $p_{BLM}((f_j, e_i) \mid (f_{a(i-2)}, e_{i-2}) (f_{a(i-1)}, e_{i-1}))$
- Technically just like a language model but it captures information relating to
  - Target string probability (just like target LMs)
  - Translation probabilities (in context)
  - Reordering

LM Implementations

- A number of smoothing approaches have been developed for language modeling
- SRILM
  - Most commonly used toolkit, many functionalities
- BerkeleyLM
  - Limited functionality, lossy quantitization for low-memory consumption
- KenLM
  - Limited functionality, focus on speed and low-memory consumption
- MITLM
  - Limited functionality, focus on accurate model estimation
- IRSTLM
  - Limited functionality, specifically designed for low-memory consumption

Recap

- Role of LMs in SMT
  - Fluency, word translation choice, reordering
- Kneser-Ney smoothing
- Large language models
  - Building large LMs
  - Using large LMs
- Interpolation of LMs
- Current developments in language modeling
  - Class-based LMs
  - Local LMs
  - Probabilistic Neural Network LMs
  - N-gram LMs vs. parsing
  - Discriminative LMs
  - Bilingual LMs