Randomized Language Models via Perfect Hash Functions

David Talbot*
School of Informatics
University of Edinburgh
2 Buccleuch Place, Edinburgh, UK
d.r.talbot@sms.ed.ac.uk

Thorsten Brants
Google Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94303, USA
brants@google.com

Abstract

We propose a succinct randomized language model which employs a *perfect hash function* to encode *fingerprints* of n-grams and their associated probabilities, backoff weights, or other parameters. The scheme can represent any standard n-gram model and is easily combined with existing model reduction techniques such as entropy-pruning. We demonstrate the space-savings of the scheme via machine translation experiments within a distributed language modeling framework.

1 Introduction

Language models (LMs) are a core component in statistical machine translation, speech recognition, optical character recognition and many other areas. They distinguish plausible word sequences from a set of candidates. LMs are usually implemented as n-gram models parameterized for each distinct sequence of up to n words observed in the training corpus. Using higher-order models and larger amounts of training data can significantly improve performance in applications, however the size of the resulting LM can become prohibitive.

With large monolingual corpora available in major languages, making use of all the available data is now a fundamental challenge in language modeling. Efficiency is paramount in applications such as machine translation which make huge numbers of LM requests per sentence. To scale LMs to larger corpora with higher-order dependencies, researchers have considered alternative parameterizations such as class-based models (Brown et al., 1992), model reduction techniques such as entropy-based pruning (Stolcke, 1998), novel represention schemes such as suffix arrays (Emami et al., 2007), Golomb Coding (Church et al., 2007) and distributed language models that scale more readily (Brants et al., 2007).

In this paper we propose a novel randomized language model. Recent work (Talbot and Osborne, 2007b) has demonstrated that randomized encodings can be used to represent n-gram counts for LMs with signficant space-savings, circumventing information-theoretic constraints on lossless data structures by allowing errors with some small probability. In contrast the representation scheme used by our model encodes parameters directly. It can be combined with any n-gram parameter estimation method and existing model reduction techniques such as entropy-based pruning. Parameters that are stored in the model are retrieved without error; however, *false positives* may occur whereby n-grams not in the model are incorrectly 'found' when requested. The false positive rate is determined by the space usage of the model.

Our randomized language model is based on the Bloomier filter (Chazelle et al., 2004). We encode *fingerprints* (random hashes) of n-grams together with their associated probabilities using a *perfect hash function* generated at random (Majewski et al., 1996). Lookup is very efficient: the values of 3 cells in a large array are combined with the fingerprint of an n-gram. This paper focuses on machine translation. However, many of our findings should transfer to other applications of language modeling.

*Work completed while this author was at Google Inc.

Proceedings of ACL-08: HLT, pages 505–513,
Columbus, Ohio, USA, June 2008.

2 Scaling Language Models

In statistical machine translation (SMT), LMs are used to score candidate translations in the target language. These are typically n-gram models that approximate the probability of a word sequence by assuming each token to be independent of all but $n-1$ preceding tokens. Parameters are estimated from monolingual corpora with parameters for each distinct word sequence of length $l \in [n]$ observed in the corpus. Since the number of parameters grows somewhat exponentially with n and linearly with the size of the training corpus, the resulting models can be unwieldy even for relatively small corpora.

2.1 Scaling Strategies

Various strategies have been proposed to scale LMs to larger corpora and higher-order dependencies. *Model-based* techniques seek to parameterize the model more efficiently (e.g. latent variable models, neural networks) or to reduce the model size directly by pruning uninformative parameters, e.g. (Stolcke, 1998), (Goodman and Gao, 2000). *Representation-based* techniques attempt to reduce space requirements by representing the model more efficiently or in a form that scales more readily, e.g. (Emami et al., 2007), (Brants et al., 2007), (Church et al., 2007).

2.2 Lossy Randomized Encodings

A fundamental result in information theory (Carter et al., 1978) states that a random set of objects cannot be stored using constant space per object as the universe from which the objects are drawn grows in size: the space required to uniquely identify an object increases as the set of possible objects from which it must be distinguished grows. In language modeling the universe under consideration is the set of all possible n-grams of length n for given vocabulary. Although n-grams observed in natural language corpora are *not* randomly distributed within this universe no lossless data structure that we are aware of can circumvent this space-dependency on both the n-gram order and the vocabulary size. Hence as the training corpus and vocabulary grow, a model will require more space *per parameter*.

However, if we are willing to accept that occasionally our model will be unable to distinguish between distinct n-grams, then it is possible to store each parameter in constant space independent of both n and the vocabulary size (Carter et al., 1978), (Talbot and Osborne, 2007a). The space required in such a *lossy encoding* depends only on the range of values associated with the n-grams and the desired error rate, i.e. the probability with which two distinct n-grams are assigned the same fingerprint.

2.3 Previous Randomized LMs

Recent work (Talbot and Osborne, 2007b) has used lossy encodings based on Bloom filters (Bloom, 1970) to represent logarithmically quantized corpus statistics for language modeling. While the approach results in significant space savings, working with corpus statistics, rather than n-gram probabilities directly, is computationally less efficient (particularly in a distributed setting) and introduces a dependency on the smoothing scheme used. It also makes it difficult to leverage existing model reduction strategies such as entropy-based pruning that are applied to final parameter estimates.

In the next section we describe our randomized LM scheme based on perfect hash functions. This scheme can be used to encode any standard n-gram model which may first be processed using any conventional model reduction technique.

3 Perfect Hash-based Language Models

Our randomized LM is based on the Bloomier filter (Chazelle et al., 2004). We assume the n-grams and their associated parameter values have been precomputed and stored on disk. We then encode the model in an array such that each n-gram's value can be retrieved. Storage for this array is the model's only significant space requirement once constructed.[1]

The model uses randomization to map n-grams to *fingerprints* and to generate a *perfect hash function* that associates n-grams with their values. The model can erroneously return a value for an n-gram that was never actually stored, but will always return the correct value for an n-gram that is in the model. We will describe the randomized algorithm used to encode n-gram parameters in the model, analyze the probability of a false positive, and explain how we construct and query the model in practice.

[1] Note that we do not store the n-grams explicitly and therefore that the model's parameter set cannot easily be enumerated.

3.1 N-gram Fingerprints

We wish to encode a set of n-gram/value pairs

$$\mathcal{S} = \{(x_1, v(x_1)), (x_2, v(x_2)), \ldots, (x_N, v(x_N))\}$$

using an array A of size M and a perfect hash function. Each n-gram x_i is drawn from some set of possible n-grams $\mathcal{U}$ and its associated value $v(x_i)$ from a corresponding set of possible values $\mathcal{V}$.

We do not store the n-grams and their probabilities directly but rather encode a *fingerprint* of each n-gram $f(x_i)$ together with its associated value $v(x_i)$ in such a way that the value can be retrieved when the model is queried with the n-gram x_i.

A fingerprint hash function $f : \mathcal{U} \rightarrow [0, B-1]$ maps n-grams to integers between 0 and $B-1$.[2] The array A in which we encode n-gram/value pairs has addresses of size $\lceil \log_2 B \rceil$ hence B will determine the amount of space used per n-gram. There is a trade-off between space and error rate since the larger B is, the lower the probability of a false positive. This is analyzed in detail below. For now we assume only that B is at least as large as the range of values stored in the model, i.e. $B \geq |\mathcal{V}|$.

3.2 Composite Perfect Hash Functions

The function used to associate n-grams with their values (Eq. (1)) combines a composite perfect hash function (Majewski et al., 1996) with the fingerprint function. An example is shown in Fig. 1. The composite hash function is made up of k independent hash functions $h_1, h_2, \ldots, h_k$ where each $h_i : \mathcal{U} \rightarrow [0, M-1]$ maps n-grams to locations in the array A. The lookup function is then defined as $g : \mathcal{U} \rightarrow [0, B-1]$ by[3]

$$g(x_i) = f(x_i) \otimes \left(\bigotimes_{i=1}^{k} A[h_i(x_i)] \right) \qquad (1)$$

where $f(x_i)$ is the fingerprint of n-gram x_i and $A[h_i(x_i)]$ is the value stored in location $h_i(x_i)$ of the array A. Eq. (1) is evaluated to retrieve an n-gram's parameter during decoding. To encode our model correctly we must ensure that $g(x_i) = v(x_i)$ for all n-grams in our set $\mathcal{S}$. Generating A to encode this function for a given set of n-grams is a significant challenge described in the following sections.

[2]The analysis assumes that all hash functions are random.

[3]We use $\otimes$ to denote the exclusive bitwise OR operator.

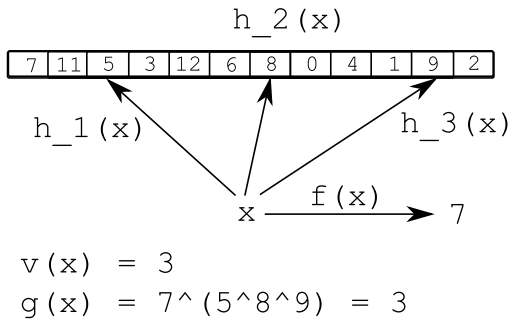

Figure 1: Encoding an n-gram's value in the array.

3.3 Encoding n-grams in the model

All addresses in A are initialized to zero. The procedure we use to ensure $g(x_i) = v(x_i)$ for all $x_i \in \mathcal{S}$ updates a single, unique location in A for each n-gram x_i. This location is chosen from among the k locations given by $h_j(x_i), j \in [k]$. Since the composite function $g(x_i)$ depends on the values stored at all k locations $A[h_1(x_i)], A[h_2(x_i)], \ldots, A[h_k(x_i)]$ in A, we must also ensure that once an n-gram x_i has been encoded in the model, these k locations are not subsequently changed since this would invalidate the encoding; however, n-grams encoded later may reference earlier entries and therefore locations in A can effectively be 'shared' among parameters.

In the following section we describe a randomized algorithm to find a suitable order in which to enter n-grams in the model and, for each n-gram x_i, determine which of the k hash functions, say h_j, can be used to update A without invalidating previous entries. Given this ordering of the n-grams and the choice of hash function h_j for each $x_i \in \mathcal{S}$, it is clear that the following update rule will encode x_i in the array A so that $g(x_i)$ will return $v(x_i)$ (cf. Eq.(1))

$$A[h_j(x_i)] = v(x_i) \otimes f(x_i) \otimes \bigotimes_{i=1 \cap i \neq j}^{k} A[h_i(x_i)]. \qquad (2)$$

3.4 Finding an Ordered Matching

We now describe an algorithm (Algorithm 1; (Majewski et al., 1996)) that selects one of the k hash

functions $h_j, j \in [k]$ for each n-gram $x_i \in \mathcal{S}$ and an order in which to apply the update rule Eq. (2) so that $g(x_i)$ maps x_i to $v(x_i)$ for all n-grams in $\mathcal{S}$.

This problem is equivalent to finding an *ordered matching* in a bipartite graph whose LHS nodes correspond to n-grams in $\mathcal{S}$ and RHS nodes correspond to locations in A. The graph initially contains edges from each n-gram to each of the k locations in A given by $h_1(x_i), h_2(x_i), \ldots, h_k(x_i)$ (see Fig. (2)). The algorithm uses the fact that any RHS node that has degree one (i.e. a single edge) can be safely matched with its associated LHS node since no remaining LHS nodes can be dependent on it.

We first create the graph using the k hash functions $h_j, j \in [k]$ and store a list (`degree_one`) of those RHS nodes (locations) with degree one. The algorithm proceeds by removing nodes from `degree_one` in turn, pairing each RHS node with the unique LHS node to which it is connected. We then remove both nodes from the graph and push the pair $(x_i, h_j(x_i))$ onto a stack (`matched`). We also remove any other edges from the matched LHS node and add any RHS nodes that now have degree one to `degree_one`. The algorithm succeeds if, while there are still n-grams left to match, `degree_one` is never empty. We then encode n-grams in the order given by the stack (i.e., first-in-last-out).

Since we remove each location in A (RHS node) from the graph as it is matched to an n-gram (LHS node), each location will be associated with at most one n-gram for updating. Moreover, since we match an n-gram to a location only once the location has degree one, we are guaranteed that any other n-grams that depend on this location are already on the stack and will therefore only be encoded once we have updated this location. Hence dependencies in g are respected and $g(x_i) = v(x_i)$ will remain true following the update in Eq. (2) for each $x_i \in \mathcal{S}$.

3.5 Choosing Random Hash Functions

The algorithm described above is not guaranteed to succeed. Its success depends on the size of the array M, the number of n-grams stored $|\mathcal{S}|$ and the choice of random hash functions $h_j, j \in [k]$. Clearly we require $M \geq |\mathcal{S}|$; in fact, an argument from Majewski et al. (1996) implies that if $M \geq 1.23|\mathcal{S}|$ and $k = 3$, the algorithm succeeds with high probability. We use 2-universal hash functions (L. Carter and M. Wegman, 1979) defined for a range of size M via a prime $P \geq M$ and two random numbers $1 \leq a_j \leq P$ and $0 \leq b_j \leq P$ for $j \in [k]$ as

$$h_j(x) \equiv a_j x + b_j \mod P$$

taken modulo M. We generate a set of k hash functions by sampling k pairs of random numbers $(a_j, b_j), j \in [k]$. If the algorithm does not find a matching with the current set of hash functions, we re-sample these parameters and re-start the algorithm. Since the probability of failure on a single attempt is low when $M \geq 1.23|\mathcal{S}|$, the probability of failing multiple times is very small.

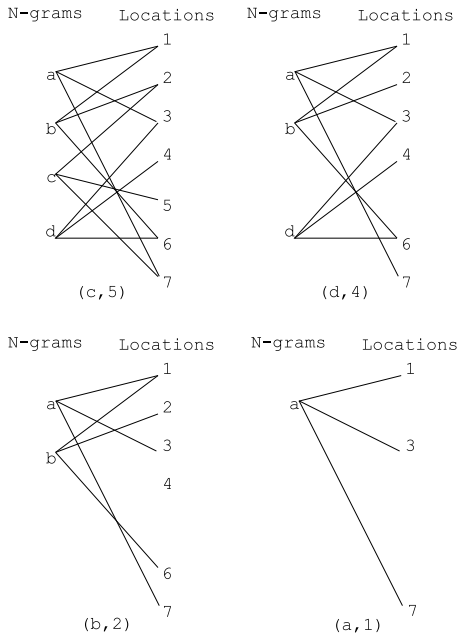

Figure 2: The ordered matching algorithm: `matched` $= [(a,1),(b,2),(d,4),(c,5)]$

3.6 Querying the Model and False Positives

The construction we have described above ensures that for any n-gram $x_i \in \mathcal{S}$ we have $g(x_i) = v(x_i)$, i.e., we retrieve the correct value. To retrieve a value given an n-gram x_i we simply compute the fingerprint $f(x_i)$, the hash functions $h_j(x_i), j \in [k]$ and then return $g(x_i)$ using Eq. (1). Note that unlike the constructions in (Talbot and Osborne, 2007b) and (Church et al., 2007) no errors are possible for n-grams stored in the model. Hence we will not make errors for common n-grams that are typically in $\mathcal{S}$.

Algorithm 1 Ordered Matching

Input : Set of n-grams $\mathcal{S}$; k hash functions $h_j, j \in [k]$; number of available locations M.
Output : Ordered matching `matched` or `FAIL`.

```
matched ⇐ [ ]
for all i ∈ [0, M − 1] do
  r2l_i ⇐ ∅
end for
for all x_i ∈ S do
  l2r_i ⇐ ∅
  for all j ∈ [k] do
    l2r_i ⇐ l2r_i ∪ h_j(x_i)
    r2l_{h_j(x_i)} ⇐ r2l_{h_j(x_i)} ∪ x_i
  end for
end for
degree_one ⇐ {i ∈ [0, M − 1] | |r2l_i| = 1}
while |degree_one| ≥ 1 do
  rhs ⇐ POP degree_one
  lhs ⇐ POP r2l_rhs
  PUSH (lhs, rhs) onto matched
  for all rhs' ∈ l2r_lhs do
    POP r2l_rhs'
    if |r2l_rhs'| = 1 then
      degree_one ⇐ degree_one ∪ rhs'
    end if
  end for
end while
if |matched| = |S| then
  return matched
else
  return FAIL
end if
```

On the other hand, querying the model with an n-gram that was not stored, i.e. with $x_i \in \mathcal{U} \setminus \mathcal{S}$ we may erroneously return a value $v \in \mathcal{V}$.

Since the fingerprint $f(x_i)$ is assumed to be distributed uniformly at random (u.a.r.) in $[0, B-1]$, $g(x_i)$ is also u.a.r. in $[0, B-1]$ for $x_i \in \mathcal{U} \setminus \mathcal{S}$. Hence with $|\mathcal{V}|$ values stored in the model, the probability that $x_i \in \mathcal{U} \setminus \mathcal{S}$ is assigned a value in $v \in \mathcal{V}$ is

$$\mathbf{Pr}\{g(x_i) \in \mathcal{V} | x_i \in \mathcal{U} \setminus \mathcal{S}\} = |\mathcal{V}|/B.$$

We refer to this event as a *false positive*. If $\mathcal{V}$ is fixed, we can obtain a false positive rate ϵ by setting B as

$$B \equiv |\mathcal{V}|/\epsilon.$$

For example, if $|\mathcal{V}|$ is 128 then taking $B = 1024$ gives an error rate of $\epsilon = 128/1024 = 0.125$ with each entry in A using $\lceil \log_2 1024 \rceil = 10$ bits. Clearly B must be at least $|\mathcal{V}|$ in order to distinguish each value. We refer to the additional bits allocated to each location (i.e. $\lceil \log_2 B \rceil - \log_2 |\mathcal{V}|$ or 3 in our example) as *error bits* in our experiments below.

3.7 Constructing the Full Model

When encoding a large set of n-gram/value pairs $\mathcal{S}$, Algorithm 1 will only be practical if the raw data and graph can be held in memory as the perfect hash function is generated. This makes it difficult to encode an extremely large set $\mathcal{S}$ into a single array A. The solution we adopt is to split $\mathcal{S}$ into t smaller sets $S'_i, i \in [t]$ that are arranged in lexicographic order.[4] We can then encode each subset in a separate array $A'_i, i \in [t]$ in turn in memory. Querying each of these arrays for each n-gram requested would be inefficient and inflate the error rate since a false positive could occur on each individual array. Instead we store an index of the final n-gram encoded in each array and given a request for an n-gram's value, perform a binary search for the appropriate array.

3.8 Sanity Checks

Our models are consistent in the following sense

$$(w_1, w_2, \ldots, w_n) \in \mathcal{S} \Longrightarrow (w_2, \ldots, w_n) \in \mathcal{S}.$$

Hence we can infer that an n-gram can *not* be present in the model, if the $n-1$-gram consisting of the final $n-1$ words has already tested false. Following (Talbot and Osborne, 2007a) we can avoid unnecessary false positives by not querying for the longer n-gram in such cases.

Backoff smoothing algorithms typically request the longest n-gram supported by the model first, requesting shorter n-grams only if this is not found. In our case, however, if a query is issued for the 5-gram $(w_1, w_2, w_3, w_4, w_5)$ when only the unigram (w_5) is present in the model, the probability of a false positive using such a backoff procedure would not be ϵ as stated above, but rather the probability that we fail to avoid an error on *any* of the four queries performed prior to requesting the unigram, i.e. $1-(1-\epsilon)^4 \approx 4\epsilon$. We therefore query the model first with the unigram working up to the full n-gram requested by the decoder only if the preceding queries test positive. The probability of returning a false positive for any n-gram requested by the decoder (but not in the model) will then be at most ϵ.

[4] In our system we use subsets of 5 million n-grams which can easily be encoded using less than 2GB of working space.

4 Experimental Set-up

4.1 Distributed LM Framework

We deploy the randomized LM in a distributed framework which allows it to scale more easily by distributing it across multiple language model servers. We encode the model stored on each languagage model server using the randomized scheme.

The proposed randomized LM can encode parameters estimated using any smoothing scheme (e.g. Kneser-Ney, Katz etc.). Here we choose to work with *stupid backoff* smoothing (Brants et al., 2007) since this is significantly more efficient to train and deploy in a distributed framework than a *context-dependent* smoothing scheme such as Kneser-Ney. Previous work (Brants et al., 2007) has shown it to be appropriate to large-scale language modeling.

4.2 LM Data Sets

The language model is trained on four data sets:
target: The English side of Arabic-English parallel data provided by LDC (132 million tokens).
gigaword: The English Gigaword dataset provided by LDC (3.7 billion tokens).
webnews: Data collected over several years, up to January 2006 (34 billion tokens).
web: The Web 1T 5-gram Version 1 corpus provided by LDC (1 trillion tokens).[5]

An initial experiment will use the Web 1T 5-gram corpus only; all other experiments will use a log-linear combination of models trained on each corpus. The combined model is pre-compiled with weights trained on development data by our system.

4.3 Machine Translation

The SMT system used is based on the framework proposed in (Och and Ney, 2004) where translation is treated as the following optimization problem

$$\hat{\mathbf{e}} \quad = \quad \arg\max_{\mathbf{e}} \sum_{i=1}^{M} \lambda_i \Phi_i(\mathbf{e}, \mathbf{f}). \qquad (3)$$

Here $\mathbf{f}$ is the source sentence that we wish to translate, $\mathbf{e}$ is a translation in the target language, $\Phi_i, i \in [M]$ are feature functions and $\lambda_i, i \in [M]$ are weights. (Some features may not depend on $\mathbf{f}$.)

[5] N-grams with count < 40 are not included in this data set.

	Full Set	Entropy-Pruned
# 1-grams	13,588,391	13,588,391
# 2-grams	314,843,401	184,541,402
# 3-grams	977,069,902	439,430,328
# 4-grams	1,313,818,354	407,613,274
# 5-grams	1,176,470,663	238,348,867
Total	3,795,790,711	1,283,522,262

Table 1: Num. of n-grams in the Web 1T 5-gram corpus.

5 Experiments

This section describes three sets of experiments: first, we encode the Web 1T 5-gram corpus as a randomized language model and compare the resulting size with other representations; then we measure false positive rates when requesting n-grams for a held-out data set; finally we compare translation quality when using conventional (lossless) languages models and our randomized language model.

Note that the standard practice of measuring perplexity is not meaningful here since (1) for efficient computation, the language model is not normalized; and (2) even if this were not the case, quantization and false positives would render it unnormalized.

5.1 Encoding the Web 1T 5-gram corpus

We build a language model from the Web 1T 5-gram corpus. Parameters, corresponding to negative logarithms of relative frequencies, are quantized to 8-bits using a uniform quantizer. More sophisticated quantizers (e.g. (S. Lloyd, 1982)) may yield better results but are beyond the scope of this paper.

Table 1 provides some statistics about the corpus. We first encode the full set of n-grams, and then a version that is reduced to approx. 1/3 of its original size using entropy pruning (Stolcke, 1998).

Table 2 shows the total space and number of bytes required per n-gram to encode the model under different schemes: "LDC gzip'd" is the size of the files as delivered by LDC; "Trie" uses a compact trie representation (e.g., (Clarkson et al., 1997; Church et al., 2007)) with 3 byte word ids, 1 byte values, and 3 byte indices; "Block encoding" is the encoding used in (Brants et al., 2007); and "randomized" uses our novel randomized scheme with 12 error bits. The latter requires around 60% of the space of the next best representation and less than half of the com-

	size (GB)	bytes/n-gram
Full Set		
LDC gzip'd	24.68	6.98
Trie	21.46	6.07
Block Encoding	18.00	5.14
Randomized	10.87	3.08
Entropy Pruned		
Trie	7.70	6.44
Block Encoding	6.20	5.08
Randomized	3.68	3.08

Table 2: Web 1T 5-gram language model sizes with different encodings. "Randomized" uses 12 error bits.

monly used trie encoding. Our method is the only one to use the same amount of space per parameter for both full and entropy-pruned models.

5.2 False Positive Rates

All n-grams explicitly inserted into our randomized language model are retrieved without error; however, n-grams not stored may be incorrectly assigned a value resulting in a false positive. Section (3) analyzed the theoretical error rate; here, we measure error rates in practice when retrieving n-grams for approx. 11 million tokens of previously unseen text (news articles published after the training data had been collected). We measure this separately for all n-grams of order 2 to 5 from the same text.

The language model is trained on the four data sources listed above and contains 24 billion n-grams. With 8-bit parameter values, the model requires 55.2/69.0/82.7 GB storage when using 8/12/16 error bits respectively (this corresponds to 2.46/3.08/3.69 bytes/n-gram).

Using such a large language model results in a large fraction of known n-grams in new text. Table 3 shows, e.g., that almost half of all 5-grams from the new text were seen in the training data.

Column (1) in Table 4 shows the number of false positives that occurred for this test data. Column (2) shows this as a fraction of the number of unseen n-grams in the data. This number should be close to 2^{-b} where b is the number of error bits (i.e. 0.003906 for 8 bits and 0.000244 for 12 bits). The error rates for bigrams are close to their expected values. The numbers are much lower for higher n-gram orders due to the use of sanity checks (see Section 3.8).

	total	seen	unseen
2gms	11,093,093	98.98%	1.02%
3gms	10,652,693	91.08%	8.92%
4gms	10,212,293	68.39%	31.61%
5gms	9,781,777	45.51%	54.49%

Table 3: Number of n-grams in test set and percentages of n-grams that were seen/unseen in the training data.

	(1) false pos.	(2) $\frac{\text{false pos}}{\text{unseen}}$	(3) $\frac{\text{false pos}}{\text{total}}$
8 error bits			
2gms	376	0.003339	0.000034
3gms	2839	0.002988	0.000267
4gms	6659	0.002063	0.000652
5gms	6356	0.001192	0.000650
total	16230	0.001687	0.000388
12 error bits			
2gms	25	0.000222	0.000002
3gms	182	0.000192	0.000017
4gms	416	0.000129	0.000041
5gms	407	0.000076	0.000042
total	1030	0.000107	0.000025

Table 4: False positive rates with 8 and 12 error bits.

The overall fraction of n-grams requested for which an error occurs is of most interest in applications. This is shown in Column (3) and is around a factor of 4 smaller than the values in Column (2). On average, we expect to see 1 error in around 2,500 requests when using 8 error bits, and 1 error in 40,000 requests with 12 error bits (see "total" row).

5.3 Machine Translation

We run an improved version of our 2006 NIST MT Evaluation entry for the Arabic-English "Unlimited" data track.[6] The language model is the same one as in the previous section.

Table 5 shows baseline translation BLEU scores for a lossless (non-randomized) language model with parameter values quantized into 5 to 8 bits. We use MT04 data for system development, with MT05 data and MT06 ("NIST" subset) data for blind testing. As expected, results improve when using more bits. There seems to be little benefit in going beyond

[6] See http://www.nist.gov/speech/tests/mt/2006/doc/

bits	dev MT04	test MT05	test MT06
5	0.5237	0.5608	0.4636
6	0.5280	0.5671	0.4649
7	0.5299	0.5691	0.4672
8	0.5304	0.5697	0.4663

Table 5: Baseline BLEU scores with lossless n-gram model and different quantization levels (bits).

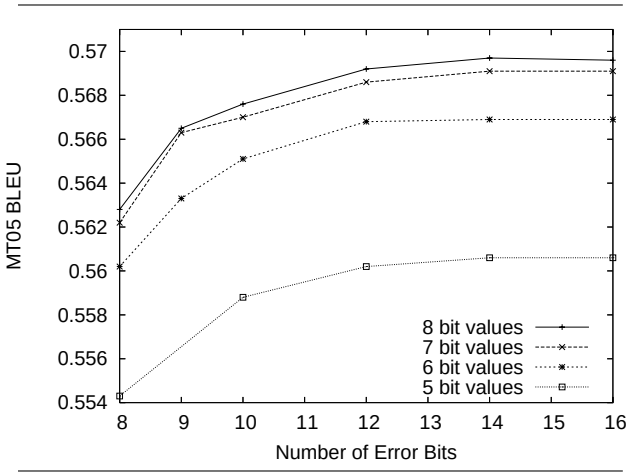

Figure 3: BLEU scores on the MT05 data set.

8 bits. Overall, our baseline results compare favorably to those reported on the NIST MT06 web site.

We now replace the language model with a randomized version. Fig. 3 shows BLEU scores for the MT05 evaluation set with parameter values quantized into 5 to 8 bits and 8 to 16 additional 'error' bits. Figure 4 shows a similar graph for MT06 data. We again see improvements as quantization uses more bits. There is a large drop in performance when reducing the number of error bits from 10 to 8, while increasing it beyond 12 bits offers almost no further gains with scores that are almost identical to the lossless model. Using 8-bit quantization and 12 error bits results in an overall requirement of $(8+12) \times 1.23 = 24.6$ bits = 3.08 bytes per n-gram.

All runs use the sanity checks described in Section 3.8. Without sanity checks, scores drop, e.g. by 0.002 for 8-bit quantization and 12 error bits.

Randomization and entropy pruning can be combined to achieve further space savings with minimal loss in quality as shown in Table (6). The BLEU score drops by between 0.0007 to 0.0018 while the model is reduced to approx. 1/4 of its original size.

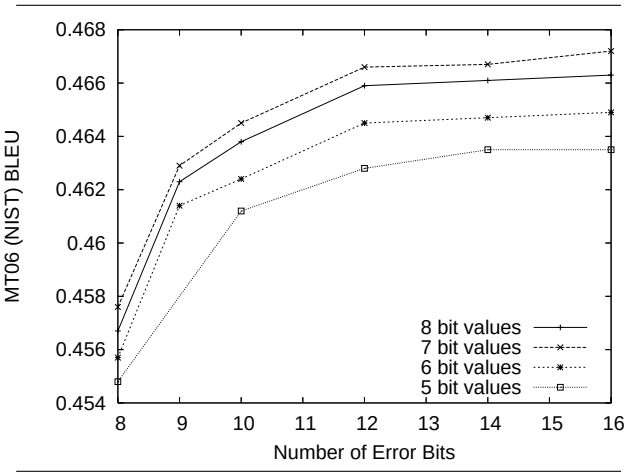

Figure 4: BLEU scores on MT06 data ("NIST" subset).

LM	size GB	dev MT04	test MT05	test MT06
unpruned block	116	0.5304	0.5697	0.4663
unpruned rand	69	0.5299	0.5692	0.4659
pruned block	42	0.5294	0.5683	0.4665
pruned rand	27	0.5289	0.5679	0.4656

Table 6: Combining randomization and entropy pruning. All models use 8-bit values; "rand" uses 12 error bits.

6 Conclusions

We have presented a novel randomized language model based on perfect hashing. It can associate arbitrary parameter types with n-grams. Values explicitly inserted into the model are retrieved without error; false positives may occur but are controlled by the number of bits used per n-gram. The amount of storage needed is independent of the size of the vocabulary and the n-gram order. Lookup is very efficient: the values of 3 cells in a large array are combined with the fingerprint of an n-gram.

Experiments have shown that this randomized language model can be combined with entropy pruning to achieve further memory reductions; that error rates occurring in practice are much lower than those predicted by theoretical analysis due to the use of runtime sanity checks; and that the same translation quality as a lossless language model representation can be achieved when using 12 'error' bits, resulting in approx. 3 bytes per n-gram (this includes one byte to store parameter values).

References

B. Bloom. 1970. Space/time tradeoffs in hash coding with allowable errors. *CACM*, 13:422–426.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proceedings of EMNLP-CoNLL 2007*, Prague.

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jennifer C. Lai, and Robert L. Mercer. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

Peter Brown, Stephen Della Pietra, Vincent Della Pietra, and Robert Mercer. 1993. The mathematics of machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Larry Carter, Robert W. Floyd, John Gill, George Markowsky, and Mark N. Wegman. 1978. Exact and approximate membership testers. In *STOC*, pages 59–65.

L. Carter and M. Wegman. 1979. Universal classes of hash functions. *Journal of Computer and System Science*, 18:143–154.

Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. 2004. The Bloomier Filter: an efficient data structure for static support lookup tables. In *Proc. 15th ACM-SIAM Symposium on Discrete Algoritms*, pages 30–39.

Kenneth Church, Ted Hart, and Jianfeng Gao. 2007. Compressing trigram language models with golomb coding. In *Proceedings of EMNLP-CoNLL 2007*, Prague, Czech Republic, June.

P. Clarkson and R. Rosenfeld. 1997. Statistical language modeling using the CMU-Cambridge toolkit. In *Proceedings of EUROSPEECH*, vol. 1, pages 2707–2710, Rhodes, Greece.

Ahmad Emami, Kishore Papineni, and Jeffrey Sorensen. 2007. Large-scale distributed language modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2007*, Hawaii, USA.

J. Goodman and J. Gao. 2000. Language model size reduction by pruning and clustering. In *ICSLP'00*, Beijing, China.

S. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

B.S. Majewski, N.C. Wormald, G. Havas, and Z.J. Czech. 1996. A family of perfect hashing methods. British Computer Journal, 39(6):547–554.

Franz J. Och and Hermann Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417–449.

Andreas Stolcke. 1998. Entropy-based pruning of back-off language models. In *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, pages 270–274.

D. Talbot and M. Osborne. 2007a. Randomised language modelling for statistical machine translation. In *45th Annual Meeting of the ACL 2007, Prague*.

D. Talbot and M. Osborne. 2007b. Smoothed Bloom filter language models: Tera-scale LMs on the cheap. In *EMNLP/CoNLL 2007, Prague*.