A Succinct N-gram Language Model

Taro Watanabe **Hajime Tsukada** **Hideki Isozaki**
NTT Communication Science Laboratories
2-4 Hikaridai Seika-cho Soraku-gun Kyoto 619-0237 Japan
{taro,tsukada,isozaki}@cslab.kecl.ntt.co.jp

Abstract

Efficient processing of tera-scale text data is an important research topic. This paper proposes *lossless* compression of N-gram language models based on LOUDS, a succinct data structure. LOUDS succinctly represents a trie with M nodes as a $2M + 1$ bit string. We compress it further for the N-gram language model structure. We also use '*variable length coding*' and '*block-wise compression*' to compress *values* associated with nodes. Experimental results for three large-scale N-gram compression tasks achieved a significant compression rate *without any loss*.

1 Introduction

There has been an increase in available N-gram data and a large amount of web-scaled N-gram data has been successfully deployed in statistical machine translation. However, we need either a machine with hundreds of gigabytes of memory or a large computer cluster to handle them.

Either *pruning* (Stolcke, 1998; Church et al., 2007) or *lossy randomizing approaches* (Talbot and Brants, 2008) may result in a compact representation for the application run-time. However, the lossy approaches may reduce accuracy, and tuning is necessary. A lossless approach is obviously better than a lossy one if other conditions are the same. In addtion, a lossless approach can easly combined with pruning. Therefore, lossless representation of N-gram is a key issue even for lossy approaches.

Raj and Whittaker (2003) showed a general N-gram language model structure and introduced a lossless algorithm that compressed a sorted integer vector by recursively shifting a certain number of bits and by emitting index-value inverted vectors. However, we need more compact representation.

In this work, we propose a succinct way to represent the N-gram language model structure based on LOUDS (Jacobson, 1989; Delpratt et al., 2006). It was first introduced by Jacobson (1989) and requires only a small space close to the information-theoretic lower bound. For an M node ordinal trie, its information-theoretical lower bound is $2M - O(\lg M)$ bits ($\lg(x) = \log_2(x)$)

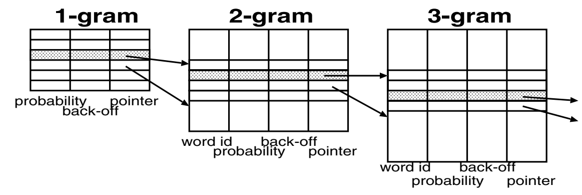


Figure 1: Data structure for language model

and LOUDS succinctly represents it by a $2M + 1$ bit string. The space is further reduced by considering the N-gram structure. We also use *variable length coding* and *block-wise compression* to compress the *values* associated with each node, such as word ids, probabilities or counts.

We experimented with English Web 1T 5-gram from LDC consisting of 25 GB of gzipped raw text N-gram counts. By using 8-bit floating point quantization [1], N-gram language models are compressed into 10 GB, which is comparable to a *lossy* representation (Talbot and Brants, 2008).

2 N-gram Language Model

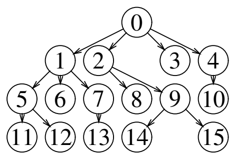
We assume a back-off N-gram language model in which the conditional probability $Pr(w_n|w_1^{n-1})$ for an arbitrary N-gram $w_1^n = (w_1, ..., w_n)$ is recursively computed as follows.

$$\begin{array}{ll} \alpha(w_1^n) & \text{if } w_1^n \text{ exists.} \\ \beta(w_1^{n-1})Pr(w_n|w_2^{n-1}) & \text{if } w_1^{n-1} \text{ exists.} \\ Pr(w_n|w_2^{n-1}) & \text{otherwise.} \end{array}$$

$\alpha(w_1^n)$ and $\beta(w_1^n)$ are smoothed probabilities and back-off coefficients, respectively.

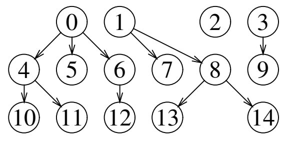
The N-grams are stored in a trie structure as shown in Figure 1. N-grams of different orders are stored in different tables and each row corresponds to a particular w_1^n, consisting of a word id for w_n, $\alpha(w_1^n)$, $\beta(w_1^n)$ and a pointer to the first position of the succeeding $(n + 1)$-grams that share the same prefix w_1^n. The succeeding $(n+1)$-grams are stored in a contiguous region and sorted by the word id of w_{n+1}. The boundary of the region is determined by the pointer of the next N-gram in the

[1] The compact representation of the floating point is out of the scope of this paper. Therefore, we use the term *lossless* even when using floating point quantization.


(a) Trie structure

node id		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
bit position	0 1	2 3 4 5 6	7 8 9 10	11 12 13	14	15 16	17 18 19	20	21 22	23	24 25 26	27	28	29	30	31	32
LOUDS bit	**1 0**	**1 1 1 1 0**	**1 1 1 0**	**1 1 0**	**0**	**1 0**	**1 1 0**	**0**	**1 0**	**0**	**1 1 0**	**0**	**0**	**0**	**0**	**0**	**0**

(b) Corresponding LOUDS bit string


(c) Trie structure for N-gram

node id	0	1	2	3	4	5	6	7	8	9
bit position	0 1 2 3	4 5 6	7	8 9	10 11 12	13	14 15	16	17 18 19	20
LOUDS bit	**1 1 1 0**	**1 1 0**	**0**	**1 0**	**1 1 0**	**0**	**1 0**	**0**	**1 1 0**	**0**

(d) Corresponding N-gram optimized LOUDS bit string

Figure 2: Optimization of LOUDS bit string for N-gram data

row. When an N-gram is traversed, binary search is performed N times. If each word id corresponds to its node position in the unigram table, we can remove the word ids for the first order.

Our implementation merges across different orders of N-grams, then separates into multiple tables such as word ids, smoothed probabilities, back-off coefficients, and pointers. The starting positions of different orders are memorized to allow access to arbitrary orders. To store N-gram counts, we use three tables for word ids, counts and pointers. We share the same tables for word ids and pointers with additional probability and back-off coefficient tables.

To support distributed computation (Brants et al., 2007), we further split the N-gram data into "shards" by hash values of the first bigram. Unigram data are shared across shards for efficiency.

3 Succinct N-gram Structure

The table of pointers described in the previous section represents a trie. We use a succinct data structure LOUDS (Jacobson, 1989; Delpratt et al., 2006) for compact representation of the trie.

For an M node ordinal trie, there exist $\frac{1}{2M+1}\binom{2M+1}{M}$ different tries. Therefore, its information-theoretical lower bound is $\lg\left\lceil\frac{1}{2M+1}\binom{2M+1}{M}\right\rceil \approx 2M - O(\lg M)$ bits. LOUDS represents a trie with M nodes as a $2M + O(M)$ bit string.

The LOUDS bit string is constructed as follows. Starting from the root node, we traverse a trie in level order. For each node with $d \geq 0$ children, the bit string $\mathbf{1}^d\mathbf{0}$ is emitted. In addition, **10** is prefixed to the bit string emitted by an imaginary super-root node pointing to the root node. Figure 2(a) shows an example trie structure. The nodes are numbered in level order, and from left to right. The corresponding LOUDS bit string is shown in Figure 2(b). Since the root node 0 has four child nodes, it emits four **1**s followed by **0**, which marks the end of the node. Before the root node, we assume an imaginary super root node emits **10** for its only child, i.e., the root node. After the root node, its first child or node 1 follows. Since $(M+1)$**0**s and M**1**s are emitted for a trie with M nodes, LOUDS occupies $2M+1$ bits.

We define a basic operation on the bit string. $\mathrm{sel}_1(i)$ returns the position of the i-th **1**. We can also define similar operations over zero bit strings, $\mathrm{sel}_0(i)$. Given sel_b, we define two operations for a node x. $\mathrm{parent}(x)$ gives x's parent node and $\mathrm{firstch}(x)$ gives x's first child node:

$$\begin{aligned} \mathrm{parent}(x) &= \mathrm{sel}_1(x+1) - x - 1, & (1)\\ \mathrm{firstch}(x) &= \mathrm{sel}_0(x+1) - x. & (2) \end{aligned}$$

To test whether a child node exists, we simply check $\mathrm{firstch}(x) \neq \mathrm{firstch}(x+1)$. Similarly, the child node range is determined by $[\mathrm{firstch}(x), \mathrm{firstch}(x+1))$.

3.1 Optimizing N-gram Structure for Space

We propose removing redundant bits from the baseline LOUDS representation assuming N-gram structures. Since we do not store any information in the root node, we can safely remove the root so that the imaginary super-root node directly points to unigram nodes. The node ids are renumbered and the first unigram is 0. In this way, 2 bits are saved.

The N-gram data structure has a fixed depth N and takes a flat structure. Since the highest order N-grams have no child nodes, they emit $\mathbf{0}^{\mathcal{N}_N}$ in the tail of the bit stream, where $\mathcal{N}_n$ stands for the number of n-grams. By memorizing the starting position of the highest order N-grams, we can completely remove $\mathcal{N}_N$ bits.

The imaginary super-root emits $\mathbf{1}^{\mathcal{N}_1}\mathbf{0}$ at the beginning of the bit stream. By memorizing the bigram starting position, we can remove the $\mathcal{N}_1 + 1$ bits.

Finally, $\mathrm{parent}(x)$ and $\mathrm{firstch}(x)$ are rewritten as

integer seq.	52	156	260		364	
coding	0x34	0x9c	0x01	0x04	0x01	0x6c
boundary	**1**	**1**	**0**	**1**	**0**	**1**

Figure 3: Example of variable length coding

follows:

$$\mathrm{parent}(x) = \mathrm{sel}_1(x + 1 - \mathcal{N}_1) + \mathcal{N}_1 - x, \quad (3)$$

$$\mathrm{firstch}(x) = \mathrm{sel}_0(x) + \mathcal{N}_1 + 1 - x. \quad (4)$$

Figure 2(c) shows the N-gram optimized trie structure ($N = 3$) from Figure 2 with $\mathcal{N}_1 = 4$ and $\mathcal{N}_3 = 5$. The parent of node 8 is found by $\mathrm{sel}_1(8+1-4) = 5$ and $5+4-8 = 1$. The first child is located by $\mathrm{sel}_0(8) = 16$ and $16+4+1-8 = 13$.

When accessing the N-gram data structure, $\mathrm{sel}_b(i)$ operations are used extensively. We use an auxiliary dictionary structure proposed by Kim et al. (2005) and Jacobson (1989) that supports an efficient $\mathrm{sel}_1(i)$ ($\mathrm{sel}_0(i)$) with the dictionary. We omit the details due to lack of space.

3.2 Variable Length Coding

The above method compactly represents pointers, but not associated *values*, such as word ids or counts. Raj and Whittaker (2003) proposed *integer compression* on each range of the word id sequence that shared the same N-gram prefix.

Here, we introduce a simple but more effective variable length coding for integer sequences of word ids and counts. The basic idea comes from encoding each integer by the smallest number of required bytes. Specifically, an integer within the range of 0 to 255 is coded as a 1-byte integer, the integers within the range of 256 to 65,535 are stored as 2-byte integers, and so on. We use an additional bit vector to indicate the boundary of the byte sequences. Figure 3 presents an example integer sequence, 52, 156, 260 and 364 with coded integers in hex decimals with boundary bits.

In spite of the length variability, the system can *directly access* a value at index i as bytes in $[\mathrm{sel}_1(i) + 1, \mathrm{sel}_1(i + 1) + 1)$ by the efficient sel_1 operation assuming that $\mathrm{sel}_1(0)$ yields -1. For example, the value 260 at index 2 in Figure 3 is mapped onto the byte range of $[\mathrm{sel}_1(2) + 1, \mathrm{sel}_1(3) + 1) = [2, 4)$.

3.3 Block-wise Compression

We further compress every 8K-byte data block of all tables in N-grams by using a generic compression library, `zlib`, employed in UNIX `gzip`. We treat a sequence of 4-byte floats in the probability table as a byte stream, and compress every 8K-byte block. To facilitate random access to the compressed block, we keep track of the compressed block's starting offsets. Since the offsets are in sorted order, we can apply *sorted integer compression* (Raj and Whittaker, 2003). Since N-gram language model access preserves some locality, N-gram with block compression is still practical enough to be usable in our system.

4 Experiments

We applied the proposed representation to 5-gram trained by "*English Gigaword* 3rd Edition," "*English Web 1T* 5-gram" from LDC, and "*Japanese Web 1T* 7-gram" from GSK. Since their tendencies are the same, we only report in this paper the results on English Web 1T 5-gram, where the size of the count data in gzipped raw text format is 25GB, the number of N-grams is 3.8G, the vocabulary size is 13.6M words, and the number of the highest order N-grams is 1.2G.

We implemented an N-gram indexer/estimator using MPI inspired by the MapReduce implementation of N-gram language model indexing/estimation pipeline (Brants et al., 2007).

Table 1 summarizes the overall results. We show the initial indexed counts and the final language model size by differentiating compression strategies for the pointers, namely the 4-byte raw value (Trie), the sorted integer compression (Integer) and our succinct representation (Succinct). The "block" indicates block compression. For the sake of implementation simplicity, the sorted integer compression used a fixed 8-bit shift amount, although the original paper proposed recursively determined optimum shift amounts (Raj and Whittaker, 2003). 8-bit quantization was performed for probabilities and back-off coefficients using a simple binning approach (Federico and Cettolo, 2007).

N-gram counts were reduced from 23.59GB to 10.57GB by our succinct representation with block compression. N-gram language models of 42.65GB were compressed to 18.37GB. Finally, the 8-bit quantized N-gram language models are represented by 9.83GB of space.

Table 2 shows the compression ratio for the pointer table alone. Block compression employed on raw 4-byte pointers attained a large reduction that was almost comparable to sorted integer compression. Since large pointer value tables are sorted, even a generic compression algorithm could achieve better compression. Using our succinct representation, 2.4 bits are required for each N-gram. By using the "flat" trie structure, we approach closer to its information-theoretic lower bound beyond the LOUDS baseline. With block compression, we achieved 1.8 bits per N-gram.

Table 3 shows the effect of *variable length coding* and *block compression* for the word ids, counts, probabilities and back-off coefficients. After *variable-length coding*, the word id is almost half its original size. We assign a word id for each

		w/o block	w/ block
Counts	Trie	23.59 GB	12.21 GB
	Integer	14.59 GB	11.18 GB
	Succinct	12.62 GB	**10.57 GB**
Language model	Trie	42.65 GB	20.01 GB
	Integer	33.65 GB	18.98 GB
	Succinct	31.67 GB	**18.37 GB**
Quantized language model	Trie	24.73 GB	11.47 GB
	Integer	15.73 GB	10.44 GB
	Succinct	13.75 GB	**9.83 GB**

Table 1: Summary of N-gram compression

	total	per N-gram
4-byte Pointer	12.04 GB	27.24 bits
+block compression	2.42 GB	5.48 bits
Sorted Integer	3.04 GB	6.87 bits
+block compression	1.39 GB	3.15 bits
Succinct	1.06 GB	2.40 bits
+block compression	0.78 GB	1.76 bits

Table 2: Compression ratio for pointers

word according to its reverse sorted order of frequency. Therefore, highly frequent words are assigned smaller values, which in turn occupies less space in our variable length coding. With *block compression*, we achieved further 1 GB reduction in space. Since the word id sequence preserves local ordering for a certain range, even a generic compression algorithm is effective.

The most frequently observed count in N-gram data is *one*. Therefore, we can reduce the space by the variable length coding. Large compression rates are achieved for both probabilities and backoff coefficients.

5 Conclusion

We provided a succinct representation of the N-gram language model *without any loss*. Our method approaches closer to the information-theoretic lower bound beyond the LOUDS baseline. Experimental results showed our succinct representation drastically reduces the space for the pointers compared to the sorted integer compression approach. Furthermore, the space of N-grams was significantly reduced by *variable length coding* and *block compression*. A large amount of N-gram data is reduced from unindexed gzipped 25 GB text counts to 10 GB of indexed language models. Our representation is practical enough though we did not experimentally investigate the runtime efficiency in this paper. The proposed representation enables us to utilize a web-scaled N-gram in our MT competition system (Watanabe et al., 2008). Our succinct representation will encourage new research on web-scaled N-gram data without requiring a larger computer cluster or hundreds of gigabytes of memory.

	total	per N-gram
word id size (4 bytes)	14.09 GB	31.89 bits
+variable length	6.72 GB	15.20 bits
+block compression	5.57 GB	12.60 bits
count size (8 bytes)	28.28 GB	64.00 bits
+variable length	4.85 GB	10.96 bits
+block compression	4.22 GB	9.56 bits
probability size (4 bytes)	14.14 GB	32.00 bits
+block compression	9.55 GB	21.61 bits
8-bit quantization	3.54 GB	8.00 bits
+block compression	2.64 GB	5.97 bits
backoff size (4 bytes)	9.76 GB	22.08 bits
+block compression	2.48 GB	5.61 bits
8-bit quantization	2.44 GB	5.52 bits
+block compression	0.85 GB	1.92 bits

Table 3: Effects of block compression

Acknowledgments

We would like to thank Daisuke Okanohara for his open source implementation and extensive documentation of LOUDS, which helped our original coding.

References

T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean. 2007. Large language models in machine translation. In *Proc. of EMNLP-CoNLL 2007*.

K. Church, T. Hart, and J. Gao. 2007. Compressing trigram language models with Golomb coding. In *Proc. of EMNLP-CoNLL 2007*.

O. Delpratt, N. Rahman, and R. Raman. 2006. Engineering the LOUDS succinct tree representation. In *Proc. of the 5th International Workshop on Experimental Algorithms*.

M. Federico and M. Cettolo. 2007. Efficient handling of n-gram language models for statistical machine translation. In *Proc. of the 2nd Workshop on Statistical Machine Translation*.

G. Jacobson. 1989. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science*, Nov.

D. K. Kim, J. C. Na, J. E. Kim, and K. Park. 2005. Efficient implementation of rank and select functions for succinct representation. In *Proc. of the 5th International Workshop on Experimental Algorithms*.

B. Raj and E. W. D. Whittaker. 2003. Lossless compression of language model structure and word identifiers. In *Proc. of ICASSP 2003*, volume 1.

A. Stolcke. 1998. Entropy-based pruning of backoff language models. In *Proc. of the ARPA Workshop on Human Language Technology*.

D. Talbot and T. Brants. 2008. Randomized language models via perfect hash functions. In *Proc. of ACL-08: HLT*.

T. Watanabe, H. Tsukada, and H. Isozaki. 2008. NTT SMT system 2008 at NTCIR-7. In *Proc. of the 7th NTCIR Workshop*, pages 420–422.