Methods for Efficient N-Gram Representation

Anthony Dickson
dican732@student.otago.ac.nz
Department of Computer Science
University of Otago

ABSTRACT

N-grams are used in a variety of tasks across fields such as Information Retrieval, Natural Language Processing, and Machine Learning. Handling large datasets of n-grams requires an efficient method of representing these n-grams, in terms of both memory usage and retrieval speed. In this literature review I undertake a brief survey of the current state of the art for efficient n-gram representations and language models, and then look at a trie-based method whose authors report to be competitive with the current state of the art in terms of both time and space complexity.

1 INTRODUCTION

N-grams are sequences of n tokens, and they are used in things like query auto-completion and machine translation. For example, "I like apples" would be considered a trigram (or alternatively a 3-gram). Some common n-gram datasets are Europarl and the Google n-gram datasets. These contain around 100 million and 11 billion n-grams respectively. You can imagine that in the case of the Google n-gram dataset, it would be impossible to fit a language model based on this dataset into memory without some sort of compression and for querying such a collection an efficient method for retrieving language model statistics would be indispensable.

N-grams alone are not very helpful for the tasks they are used in, we also need to store some statistics about the collection of n-grams as a language model. Typically, this is the frequency count of n-grams in a collection. For example, one such frequency count could be how often the trigram "University of Otago" appears in a given document. Language models typically employ some method of smoothing to improve the quality of the language model. Kneser-Ney smoothing [6], and more recently its modified version [2], are considered as the most effective methods currently available. Hence, most modern language models employ some variation of Kneser-Ney smoothing.

In this literature review I will look at three efficient n-gram representations: one method that modifies the Bloom filter and applies it to language modelling and machine translation, another method demonstrating the effectiveness of a pre-existing succinct trie data structure coupled with various compression techniques, and a recently published method that makes extensive use of integer encoding to allow for a compact and fast trie implementation.

2 RANDOMISED LANGUAGE MODELLING FOR STATISTICAL MACHINE TRANSLATION

In this 2007 paper Talbot and Osborne [12] investigate ways to apply the space-efficient Bloom filter to language modelling. Their main contributions are:

- a method for including approximate frequency statistics efficiently within a Bloom filter
- a method for reducing the error rate of the Bloom filter based language models.

The Bloom filter is a hash-based method that can be used to check membership of an element in a set [1]. It consists of a bit string of a given length, and the filter is trained by iterating over each element in a set (e.g. a collection of n-grams) and hashing the element with a series of hash functions. For each element we hash it k times with k different hash functions, and the corresponding k bits in the bit string are set to one. To check if an element is a member of the set the element is hashed with all of the hash functions and the values of the corresponding bits are checked. If the corresponding bits are all one then the element is considered to be a member of the set, otherwise if any of the corresponding bits are zero the element is considered to not exist in the set.

The defining features of the Bloom filter are that it is simple, has a small memory footprint, and its space requirements do not necessarily depend on the number of elements in the set it represents. However, one disadvantage of the Bloom filter is that it may give false positives. Due to its construction there is no guarantee that for the k bits that an arbitrary element w hashes to, there exists a corresponding element in the training set that hashes to the same k bits. That is to say, an arbitrary element w may indeed hash to k bits that do not map to a single element in the training set (see Figure 1). In this case the element w is erroneously reported to be a member of the set by the Bloom filter. However, since the Bloom filter acts much like a hash table, we can increase the size of the Bloom filter. This reduces the probability of collisions occurring and subsequently the false positive rate. It should be noted that this error is one-sided, that is to say that while the Bloom filter may erroneously report that a given element is a member of the set, it is always correct when it says an element is not a member of the set.

The authors propose a novel method of applying the Bloom filter to language modelling. They first quantise the language model statistics (the frequency count of a given n-gram) using a logarithmic codebook:

$$qc(x) = 1 + \lfloor log_b c(x) \rfloor$$

where $qc(x)$ is the quantised frequency count of a given n-gram x, $_b$ is the base of the logarithm that controls the granularity of the quantisation, and $c(x)$ is the frequency count of x. Each n-gram is then hashed by itself with its quantised frequency count appended to it. However, due to the nature of the Bloom filter, using it to store the exact quantised frequency is liable to errors. The authors take advantage of the one-sided error and hash each n-gram appended with the numbers one through to $qc(x)$. For example,

[1] Figure from https://commons.wikimedia.org/wiki/File:Bloom_filter.svg and caption adapted from https://en.wikipedia.org/wiki/Bloom_filter#/media/File:Bloom_filter.svg.

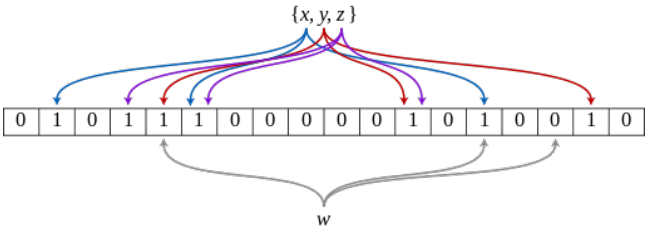


Figure 1: An example of a Bloom filter, representing the set $\{x, y, z\}$. The coloured arrows show the positions in the bit array that each set element is mapped to. The element w is not in the set $\{x, y, z\}$, because it hashes to one bit-array position containing 0. For this figure, the bit array has a length of 18 and 3 hash functions are used.[1]

if the bigram "Computer Science" has a quantised frequency of three, then the entries "Computer Science1", "Computer Science2", "Computer Science3" will be hashed into the Bloom filter. When retrieving the quantised frequency for the bigram, the bigram is suffixed with the numbers one through N, where N is the maximum quantised frequency, stopping at the first occurrence of zero. In the case of the example, we would get zero at "Computer Science4" meaning the quantised frequency for the bigram is at most three. As we can see this method gives a upper bound on the quantised frequency for a given n-gram, sacrificing precision.

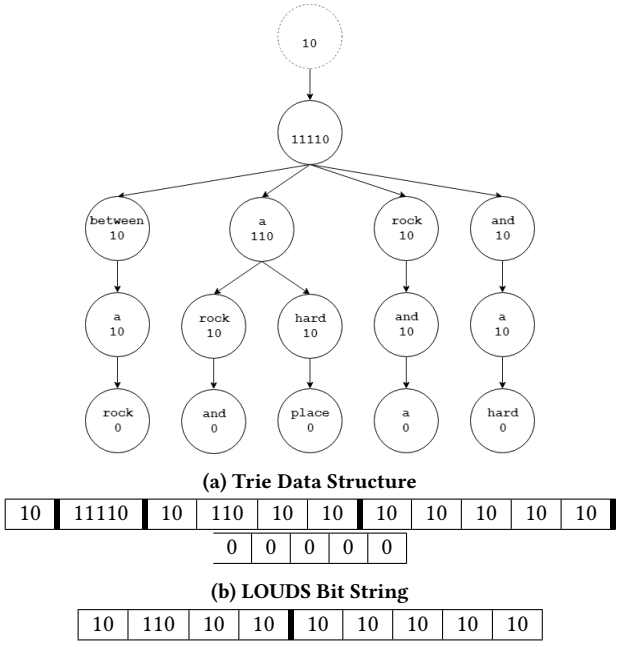
Putting all of this together, Talbot and Osborne construct a simple yet memory efficient n-gram language model based on the Bloom filter. However, due to the construction of the Bloom filter it is possible that false positives occur. The authors approach this problem by leveraging the one-sided error property of the filter and opt for giving the upper bound for the quantised frequency of a n-gram. Their method is also configurable such that it can be optimised for higher precision and lower error rate, or lower memory usage according to what is needed. One thing the authors do not go into is how fast their method is.

3 A SUCCINCT N-GRAM LANGUAGE MODEL

In 2009 Watanabe et al. [13] investigated ways of further improving the memory efficiency of the LOUDS tree structure representation [5, 10]. Their main contributions are:

- modifications to the original LOUDS method that reduces its memory usage
- experiments showing how much further memory can be saved by using variable-length coding and block-wise compression (`zlib` as used in GNU `gzip`) on the data.

Level-order unary degree sequence (LOUDS) is a compact representation that uses a $2M + 1$ bit string to represent a tree structure with M nodes. Rather than using explicit pointers which can use up quite a bit of space in large data structures, it uses this bit string for traversal of the tree structure. It is constructed by traversing the tree in breadth-first order. The degree of each node (i.e. the number of child nodes) is encoded as a unary bit string and appended to the bit string. The original author adds a *super-root* node in order to maintain the "one **1** per node" property [5]. An example is shown in Figure 2.



(a) Trie Data Structure

(b) LOUDS Bit String

(c) Watanabe et al. LOUDS Bit String

Figure 2: An example of using a trie for representing a n-gram collection (top), the corresponding LOUDS bit string representation of the tree structure (middle), and an example of the optimised version of the LOUDS bit string that Watanabe et al. propose (bottom). Figure 2a illustrates a trie consisting of trigrams generated from the phrase "between a rock and a hard place". The node with the dotted outline is the *super-root* node. Written in each node is the word it represents and its degree in unary. Figure 2b shows the resulting LOUDS bit string broken up over two lines. The thick lines show where the levels of the trie stop and start.

Two optimisations that the authors apply to the LOUDS representation are removing the bits representing the root node and the leaf nodes. These are unnecessary if the starting position of the bigrams and the highest order n-grams (i.e. the leaf nodes in the trie) are recorded as integers instead. Using these optimisations we can save a number of bits equal to the number of unigrams and highest order n-grams in the dataset. However, these optimisations have little effect in the grand scheme of things. They report their results on the Web 1T 5-gram dataset which is about 24GB when compressed with gzip. In the dataset there about 13.5 million unigrams and 1.1 billion 5-grams. Thus, in this case Watanabe et al. save about 13.5 million bits (about 1.7MB) by omitting the root node from the LOUDS bit string, and about 1.1 billion `0` bits (about 138MB) by omitting the highest order n-grams. However, even their most compact representation of the Web 1T 5-gram dataset is about 9.8GB, so the advantage gained by these optimisations is only about 1%. So assuming that these approximate calculations are correct, it would seem that these optimisations have very little effect, and are unnecessary unless you really need to save those few hundred megabytes of memory.

Watanabe et al. decrease the memory usage of the word IDs and counts by applying a variable length encoding and block-wise compression. The variable length encoding that they introduce is almost identical to variable byte encoding. The only difference is instead of storing the bit indicating when an integer has ended within the byte sequence, they store these bits in a separate bit string. With this integer encoding they are able to save about 6GB (or about 9.2%) over the method proposed in [11] which they use as one of the baselines. They further compress the data by compressing it in 8KB blocks using `zlib` and facilitate random access by recording the compressed blocks' starting offsets. With this they are able to reduce the total memory usage by about 33%.

Compared to the work of Talbot and Osborne, the advantage of the work of Watanabe et al. is that their compression is lossless and yet it provides a comparable compression rate. However, Watanabe et al. do not run experiments evaluating the speed of their methods and interestingly enough they simply conclude that their representation of the n-grams is "practical enough".

Fortunately, the authors of [9] perform extensive experiments to evaluate the performance of various n-gram representations, including this one. In these experiments the authors use the publicly available code of Watanabe et al., and Talbot and Osborne's work (henceforth referred to as Expgram and RandLM respectively). On the Europarl dataset[2] Expgram uses about 13.8% more memory than RandLM (about 2.06 bytes per n-gram for Expgram) however on average it was about 36.2% faster than RandLM (2.8 μsec/query for Expgram). Interestingly, Expgram does not scale well and when tested on the YahooV2 dataset[3] it loses its speed advantage and ends up about 81.6% percent slower (9.23 μsec/query for Expgram). Furthmore, the authors of [9] could not get Expgram to work on the GoogleV2 dataset[4], whereas RandLM worked fine.

So while the work of Watanabe et al. in their 2007 paper looks promising and provides better performance than Talbot and Osborne's work on smaller datasets, it struggles to scale well with larger scale datasets such as the YahooV2 and GoogleV2 datasets.

4 HANDLING MASSIVE N-GRAM DATASETS EFFICIENTLY

This paper [9] was recently published in the 2019 ACM Transactions on Information Systems (TOIS). Pibiri and Venturini explore methods of improving n-gram representations to produce a method that is compact, fast and lossless. Their main contributions are:

- a compressed trie data structure in which each level of the trie is modelled as a monotonic non-decreasing integer sequence that is encoded with Elias-Fano which allows constant time access
- a method for further decreasing the memory usage of the trie data structure by reducing the magnitude of the integers that represent the trie's structure.
- an improvement over the state of the art implementation of Kneser-Ney smoothing that reduces I/O and increases performance more than two-fold.

[2]http://www.statmt.org/europarl

[3]http://webscope.sandbox.yahoo.com/catalog.php?datatype=l

[4]http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

Their main result is the Elias-Fano trie. This formulation is similar to that of LOUDS in the way logical pointers are used instead of physical pointers in order to save space. Whereas LOUDS uses a unary bit string to represent the tree structure, the trie structure proposed by Pibiri and Venturini uses Elias-Fano encoded [3, 4] integer sequences. The main advantage of the Elias-Fano encoding is that it allows constant time random access without costly decoding. This encoding scheme relies on the integer sequence being monotonic non-decreasing, so the authors transform the word IDs in each level of the trie to satisfy this constraint. One disadvantage of the Elias-Fano encoding is that does not take advantage of sequences of integers that are all close together (i.e. have small deltas) – something that occurs often in data structures such as inverted indexes. There is a variant of the Elias-Fano encoding called partitioned Elias-Fano encoding that can take advantage of this property [7]. In this variant the integer sequence is partitioned into chunks. The representation is further split into two levels: the first level contains the last element of each chunk and the second level contains the encoding of the chunks themselves. Essentially, the partitioned Elias-Fano encoding sacrifices a bit of speed for a more compact representation.

In Pibiri and Venturini's trie each level is represented as a set of three arrays: an array of word IDs, an array of pointers (integer offsets) indicating where the successor/child nodes of a given node start and finish in the next level, and an array of frequency counts. By construction the pointer array is sorted and immediately Elias-Fano encodable, however the word ID array is not. If we consider the set of unigrams {A, B, C, D} and the set of bigrams {AA, AC, BB, BC, BD, CA, CD, DB, DD} from the example in Figure 3, and given that the unigrams have the word IDs {0, 1, 2, 3}, the word ID array of the second level would look like this: `{0, 2, 1, 2, 3, 0, 3, 1, 3}`. As we can see this is not monotonically non-decreasing as the Elias-Fano encoding requires. The authors solve this problem by transforming the word IDs with a range-wise prefix sum. Initially this sum is set to zero. Then for each node in the level we add the prefix sum to the word IDs of the nodes that have the same parent (i.e. are sibling nodes) and then update the prefix sum to be equal to the maximum word ID of the sibling nodes. In the above example the first set of nodes have the word IDs {0, 2}, since the initial prefix sum is zero there is no change to the word IDs and the prefix sum is set to the word ID of the right-most sibling. The next set of sibling nodes is {1, 2, 3} so we add 2 to each word ID giving the new word IDs {3, 4, 5} and we update the prefix sum to 5 and repeat this for the rest of the nodes in the level. This results in the word ID sequence `{0, 2, 3, 4, 5, 5, 8, 9, 11}` as shown in the example in Figure 3. With this the authors are able to apply Elias-Fano encoding to the word ID and pointer arrays, allowing them to represent the trie structure compactly whilst still allowing for contstant time random access.

Another important optimisation that Pibiri and Venturini add is array-rank encoding of the frequency counts. This method takes advantage of certain properties of n-gram collections. One such property is that the distribution of frequency counts is Zipfian, there are a few n-grams that appear frequently and the rest (i.e. the majority) only appear occasionally. Another property is that if we

[5]Figure and caption adapted from the paper [9].

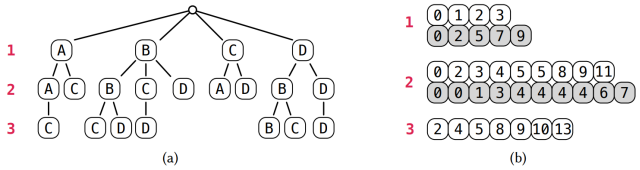


Figure 3: On the left (a): example of a trie of order 3, representing the set of grams {A, AA, AAC, AC, B, BB, BBC, BBD, BC, BCD, BD, CA, CD, DB, DBB, DBC, DDD}. On the right (b): the sorted-array representation of the Elias-Fano trie. Light-grey arrays represent the word IDs transformed into monotonic sequences using range-wise prefix sums and the dark-grey arrays represent the position in the next level where the successors of each node start and finish.[5]

look at the unique frequency counts we notice that the number of these is much smaller than the range of the frequency counts. These properties can be taken advantage of to reduce the space requirements. First, we store the unique frequency counts in an array meaning less frequency counts need to be stored. As mentioned before, each level of the trie uses three arrays, one of which stores the frequency counts. Rather than storing the frequency counts directly, which could be inefficient in space due to the possible large range of frequency counts requiring a large number of bits per frequency count, we instead store the corresponding rank of the frequency count in the array of unique frequency counts. These sequences of rank values are not monotonic non-decreasing so they cannot be Elias-Fano encoded, however the values are highly repetitive so instead they are coded with a binary codebook such that frequency counts that occur often are assigned shorter codes. With this optimisation the authors are able further reduce the space requirements of the n-gram language model.

Overall we can see that the optimisations implemented by Pibiri and Venturini leads to a n-gram representation that is both compact and fast, while remaining lossless. They use a few other tricks to further improve their n-gram representation but I will not go into these due to space restrictions. Their method is also competitive with previous state-of-the-art methods, including Talbot and Osborne's and Watanabe et al's work, in both space and time. Pibiri and Venturini's extensive experimental analysis shows that their PEF (partioned Elias-Fano) trie is more compact and faster than a collection of the recent state-of-the-art methods [8, 12, 13], and even more compact than the lossy method introduced by Talbot and Osborne. While PEF trie is still slower than the fastest hash table based method they tested (by about 1.5μsec/query or 81.5% slower), PEF trie was close to 4× more space efficient than the most compact hash table based method tested. Another advantage that PEF trie has over current methods is that it scales better and gains more of an advantage in both space and time efficiency over the other methods as the dataset gets larger. Also, many of the tested methods failed to run on the GoogleV2 dataset. So if you need to use large n-gram datasets then it seems like Pibiri and Venturini's PEF trie is the way to go.

5 FUTURE WORK

Query throughput is important especially when a large volume of queries need to be served at any given moment. It would be interesting to see how the time-space trade-off would play out between PEF trie and the hash table based methods. The hash table based methods require many times more memory but are up to many times faster than PEF trie. To store the same amount of data the hash table based methods would require many times more RAM but would offer a higher query throughput. Conversely, PEF trie would require many times less RAM but in order to offer equivalent query throughput would require multiple instances of PEF trie to be run in parallel. So it would be interesting to investigate the cost of these two approaches in terms of the hardware required to provide equivalent query throughput.

The papers discussed in this review apply existing methods in interesting ways or introduce novel methods that give impressive results in creating efficient n-gram language models. It would be interesting to see if the lessons learned from these papers could be, or have already been, applied to other tasks.

6 CONCLUSION

In this review we have covered three papers that take different approaches to creating efficient n-gram representations. Talbot and Osborne apply Bloom filters to language modelling and devise a compact representation that trades off space and precision. Watanabe et al. propose improvements to the LOUDS tree representation and demonstrate the effectiveness of LOUDS in conjunction with variable-length integer encoding. Finally, Pibiri and Venturini apply optimisations to the trie-based n-gram language model representation that allows for extensive use of Elias-Fano encoding. They further close the gap between hash-based and non hash-based methods in terms of speed while providing one the most compact n-gram representations to date.

REFERENCES

[1] Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426.

[2] Stanley F Chen and Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language* 13, 4 (1999), 359–394.

[3] Peter Elias. 1974. Efficient storage and retrieval by content and address of static files. *Journal of the ACM (JACM)* 21, 2 (1974), 246–260.

[4] Robert Mario Fano. 1971. *On the number of bits required to implement an associative memory.* Massachusetts Institute of Technology, Project MAC.

[5] Guy Jacobson. 1989. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science*. IEEE, 549–554.

[6] Hermann Ney, Ute Essen, and Reinhard Kneser. 1994. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language* 8, 1 (1994), 1–38.

[7] Giuseppe Ottaviano and Rossano Venturini. 2014. Partitioned elias-fano indexes. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. ACM, 273–282.

[8] Adam Pauls and Dan Klein. 2011. Faster and smaller n-gram language models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 258–267.

[9] Giulio Ermanno Pibiri and Rossano Venturini. 2019. Handling Massive N-Gram Datasets Efficiently. *ACM Transactions on Information Systems (TOIS)* 37, 2 (2019), 25.

[10] Naila Rahman, Rajeev Raman, et al. 2006. Engineering the LOUDS succinct tree representation. In *International Workshop on Experimental and Efficient Algorithms*. Springer, 134–145.

[11] Bhiksha Raj and Edward WD Whittaker. 2003. Lossless compression of language model structure and word identifiers. In *2003 IEEE International Conference on*

Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03)., Vol. 1. IEEE, I–I.

[12] David Talbot and Miles Osborne. 2007. Randomised language modelling for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. 512–519.

[13] Taro Watanabe, Hajime Tsukada, and Hideki Isozaki. 2009. A succinct n-gram language model. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*. Association for Computational Linguistics, 341–344.