# Random Indexing Re-Hashed

**Erik Velldal**
Department of Informatics
University of Oslo, Norway
erikve@ifi.uio.no

## Abstract

This paper introduces a modified version of Random Indexing, a technique for dimensionality reduction based on random projections. We here describe how RI can be efficiently implemented using the notion of universal hashing. This eliminates the need to store any random vectors, replacing them instead with a small number of hash-functions, thereby dramatically reducing the memory footprint. We dub this reformulated version of the method Hashed Random Indexing (HRI).

## 1 Introduction

*Random indexing* (RI) is a technique for dimensionality reduction that was initially introduced by Kanerva et al. (2000) for constructing compact word-by-context vector spaces for modeling the semantic similarity of words. The method is related to a family of techniques based on *random projections*, but comes with several particular advantages in terms of computational simplicity and incrementality. In this paper we introduce a modified version of RI which we call *hashed random indexing* (HRI), where we replace the so-called index vectors by a small set of *hash functions*.

We will start by describing the basic methodology of random indexing as introduced by Kanerva et al. (2000), also highlighting the relation to random projections. In Section 3 we then describe the notion of *universal families of hash functions*, and in particular *multiplicative universal hashing* (Dietzfelbinger et al., 1997), before finally showing how RI can be efficiently implemented using hash functions drawn from such families. In Section 4 we also include some general caveats regarding dimensionality reduction based on random projections. Finally, Section 5 provides an overview of other related work based on feature hashing, as well as discussing possible future directions.

## 2 Random Indexing

Random indexing was initially introduced by Kanerva et al. (2000) for reducing the dimensionality of word-by-document vector spaces, modeling the semantic similarity of words in terms of their contextual distribution. Proving an attractive alternative (or complement) to more costly dimensionality reduction techniques such as singular value decomposition (SVD), it has since been extensively used for constructing compact semantic space models (Karlgren and Sahlgren, 2001; Sahlgren, 2005; Widdows and Ferraro, 2009). Beyond this setting of distributional word similarity, Velldal (2010) applied RI as a general means of reducing the feature space of a classification problem. Working with SVM-based uncertainty detection, Velldal (2010) showed that RI could compress the feature space by two orders of magnitude without sacrificing classifier performance.

As the current paper has more of a theoretical focus, we will not assume any particular type of data or application. We will, however, assume that each data item is represented by a $d$-dimensional feature vector $\vec{f_i} \in \Re^d$. Given $n$ examples and $d$ features, the vectors can be thought of as rows in a matrix $F \in \Re^{n \times d}$. The purpose of RI is to avoid working with the original (possibly huge) feature matrix $F \in \Re^{n \times d}$, replacing it instead with a smaller matrix $G \in \Re^{n \times k}$ where $k \ll d$. The RI method constructs this compressed representation of the data in $G$ by *incrementally accumulating* so-called *index vectors* assigned to each of the $d$ features (Sahlgren, 2005). The process can be described by the following two simple steps:

- When a new feature is instantiated, it is assigned a randomly generated vector of a fixed dimensionality $k$, consisting of a small number of $-1$s and $+1$s (the remaining elements being 0). This is then the so-called *index vector* or *random label* of the feature.

**Bolette Sandford Pedersen, Gunta Nešpore and Inguna Skadiņa (Eds.)**
**NODALIDA 2011 Conference Proceedings, pp. 224–229**

- The vector representing a given training example (the $j$th row of $G$ represents the $j$th example) is then constructed by simply summing the random index vectors of its features.

The parameters of RI that need to be specified are the number of non-zeros ($\epsilon$) and the dimensionality ($k$) of the ternary index vectors. As noted by Sahlgren (2005), if the index vectors had been specified to consist of only one position of value 1 for each feature, i.e. *orthogonal* vectors of $k$=$d$ dimensions, the two steps above would have produced the standard feature matrix $F$. With RI, one instead uses randomly initialized $k$-dimensional index vectors. As observed by Hecht-Nielsen (1994), high-dimensional vectors having random directions are very likely to be *close to orthogonal*, and the matrix $G$ can in this sense be viewed as an approximation of $F$ (in terms of the relative distances of rows). Moreover, we can expect this approximation to be better the higher we set $k$.

Note that, in the traditional setting of semantic space modeling, RI is applied on the *type level*, accumulating global context vectors that represent the aggregated distribution of words across a corpus. When used more generally for compressing the feature space of a learning problem as in (Velldal, 2010), RI can also be applied at the *token level*, producing a compact representation of each training instance.

Mathematically, RI can be seen as part of a larger family of dimension reduction techniques based on *random projections*, and in particular the "neuronal" version described by Vempala (2004). Such methods work by multiplying the feature matrix $F \in \Re^{n \times d}$ by a random matrix $R \in \Re^{d \times k}$, for $k \ll d$, thereby reducing the number of dimensions from $d$ to $k$:

$$FR = G \in \Re^{n \times k} \qquad \text{with } k \ll d \qquad (1)$$

Given that $k$ is sufficiently high (logarithmic in $n$), the Johnson-Lindenstrauss lemma (Johnson and Lindenstrauss, 1984) tells us that the pairwise distances in $F$ can be preserved with high probability within the lower-dimensional space $G$ (Li et al., 2006). While much work on random projections for producing such so-called JL-embeddings assume $r_{ij}$ having a standard normal distribution, Achlioptas (2001) shows that the only requirement on $r_{ij}$ is that they are i.i.d. with zero mean and unit variance.

With RI, the index vector of the $i$th feature corresponds to the $i$th row of $R$ in Equation 1. Moreover, for $\epsilon$ non-zeros, the entries of the index vectors $r_{ij} \in \{+1, 0, -1\}$ will then be distributed with probabilities corresponding to $\{\frac{\epsilon/2}{k}, \frac{k-\epsilon}{k}, \frac{\epsilon/2}{k}\}$.

One important advantage of the particular random indexing approach is that the full $n \times d$ feature matrix $F$ never needs to be explicitly computed or represented (Karlgren and Sahlgren, 2001). As described above, with RI we construct the representation of the data in $G$ by *incrementally accumulating* the index vectors assigned to each feature. This means that the dimension reduction is only *implicit*, in the sense that the compressed representation in $G$ is constructed directly. With the introduction of *hashed* random indexing below, we also eliminate the need to explicitly represent $R$.

## 3 Indexing by Universal Hashing

In the traditional implementation of RI, a randomly generated index vector is assigned to each feature as it is first encountered. If the same feature is encountered again later, that same index vector is simply retrieved by look-up. For each of the $d$ distinct features (which for typical NLP problems can number hundreds of thousands or millions), a separate index vector must be stored.

An attractive feature of RI is that is it allows us to think about dimensionality reduction in terms of data structures, rather than as a separate process. In fact, taking a step back, the accumulated index vectors in the reduced space $G$ are reminiscent of probabilistic data structures like *Bloom filters* and various *sketch* representations. These are *hash-based* data structures, however, designed for compactly representing things like set membership and frequency counts in an approximate way. In a similar fashion, we here propose to save resources in RI by using a small set of *hash functions* to implicitly represent the index vectors. As we shall see, this eliminates the need for storing the random vectors in $R \in \Re^{d \times k}$.

Let $h$ be a hash function that maps from a set of hash keys $U$ into some smaller set of hash codes $C$. We here assume both $U$ and $C$ to be *integers* (without loss of generality as strings can be converted to integers). More precisely, $U$ will correspond to the dimensions of the original input space $F$, and $C$ will correspond to dimensions in the lower-dimensional index vectors. Given that the RI method relies on a one-to-many mapping from features into index vector positions, we need to use *multiple distinct hash functions*. For this purpose, the notion of *universal families of hash functions* comes in handy. This concept, introduced by Carter and Wegman (1979), refers to a method for randomly generating hash functions $h_i : U \rightarrow C$ from a family of functions $H$ that also comes with certain guarantees on the probability of *collisions* (i.e. the chances of $h_i$ mapping two distinct keys into the same code). There exists several approaches to defining such universal classes, typically based on computations seeded by some large prime, as in the original proposal by Carter and Wegman (1979). For the implementation of HRI, however, we propose to instead select functions from the particularly simple class of *multiplicative universal hashing* introduced by Dietzfelbinger et al. (1997), which provides mappings that can be evaluated very efficiently.

More precisely, Dietzfelbinger et al. (1997) define a universal family of mappings from *$l$-bit keys* to *$m$-bit indices*. Let $U = \{0, \ldots, 2^l - 1\}$ and $C = \{0, \ldots, 2^m - 1\}$. Furthermore, let $A$ be the set of positive odd $l$-bit numbers, i.e. $A = \{a \,|\, 0 < a < 2^l \text{ and } a \text{ is odd}\}$. This then defines a family of universal hash functions $H_{l,m} = \{h_a | a \in A\}$, with $h_a$ computed as:

$$h_a(x) = (ax \bmod 2^l) \operatorname{div} 2^{l-m} \quad \text{for } x \in U \tag{2}$$

where $\bmod$ refers to the modulo operation and $\operatorname{div}$ means integer division. By randomly picking a number $a \in A$, we generate a new hash function $h_a$ from the set of $2^{l-1}$ distinct hash functions in the class $H_{l,m}$. Note that, as the bits of the keys are typically assumed fixed to some value like $l = 32$, we will write $H_{l,m}$ as simply $H_m$.

As noted by Dietzfelbinger et al. (1997), the fact that the arithmetic involved in Equation 2 is based on powers of two, allows the mapping to be efficiently implemented at the level of simple bitwise operations.[1] Dietzfelbinger et al. (1997) also prove that $H_m$ is so-called *2-universal*. This means that, for two distinct keys $x$ and $y$, the function $h_a$ obeys the following lemma:

$$Prob\,(h_a(x) = h_a(y)) \leq \frac{1}{2^{m-1}} \tag{3}$$

Returning to the method of random indexing, we now have a principled way of very efficiently computing and representing the random index vectors. Any set of $d$ index vectors $R$ in $k$=$2^m$ dimensions[2] and with $\epsilon$ non-zero elements can now be *implicitly represented* by a set of hash functions $H^\epsilon = \{h_{a^1}, \ldots, h_{a^\epsilon}\} \subset H_m$. Each hash function $h_a \in H^\epsilon$ computes one non-zero element position. We randomly pick $\frac{\epsilon}{2}$ of the functions to indicate $-1$s, and label the remaining half $+1$s. Let $\sigma_k$ correspondingly evaluate to $+1$ or $-1$, depending on whether $h_k$ is selected to map into index positions with positive or negative values. In the compressed representation $G$, an entry corresponding to the $j$th dimension for the $i$th data item can then be formally described as

$$G_{ij} = \sum_{h_k \in H^\epsilon} \; \sum_{l : h_k(l) = j} \sigma_k \, f_{il} \tag{4}$$

Procedurally speaking, $G$ is constructed in the incremental fashion described in Section 2, and without the need to first construct the full feature count matrix $F$: As a feature is instantiated, we update $G$ according to the values and positions indicated by the hash functions. Evaluating the hash functions is very cheap and done in constant time. Most importantly, however; storing the full set of index vectors—corresponding to the $d \times k$ random matrix $R$ in Equation 1—now simply amounts to storing the few random seed numbers (the $a$s) for generating the hash functions. In terms of this implicit representation of $R$, no extra overhead is associated with increasing the dimensionality of the feature space ($d$) or the reduced space ($k$), or the number of training examples ($n$).

In theory, implementing the random index vectors in terms of universal hashing means introducing some new dependencies in our mappings. The lemma in Equation 3 gives the probability of a given hash-function mapping two different given keys to the same value, i.e. $h_i(x) = h_i(y)$. However, given that each index assignment assumes a one-to-many mapping, we would also like to avoid that two different functions map the same key to the same value, i.e. $h_i(x) = h_j(x)$. In practice, this proves to be a minimal concern, with collisions of this sort occurring with an observed probability of roughly $\frac{1}{2^m}$.

[1] The modulo operation can be computed as bit-wise AND (e.g. $y \bmod 2^l = y$ AND $(2^l - 1)$), and the integer division can be done by simple bit-shifting (e.g. $y \operatorname{div} 2^x = y$ RIGHT-SHIFT-BY $x$).

[2] Note that the dimensionality $k$ will always be specified as a power of two, given the definition of the multiplicative hash functions in Equation 2.

Note that, the incrementality of RI means that the method generally lends itself well to *parallelization* or *stream processing*. For example, to combine or update different context vectors accumulated for a given word, we simply add them together. This presupposes that the assignment of index vectors is shared and known across subtasks or machines, however (to ensure coherent mappings). With HRI, parallelization is even simpler, as the only knowledge that needs to be shared is the seed numbers for the hash functions.

So far we have occupied ourselves with a particular method for *computing* a reduced space $G$. In the next section we shift focus slightly and turn to look at some potential pitfalls of *working within* such a reduced space, as produced by random projection based methods in general.

## 4 Caveats

Although methods based on random projections, such as random indexing, typically come with the promise of reducing memory load and computational cost, there are certain limitations and possible caveats that are worth bearing in mind and that are often overlooked in the literature.

First, if the original input space $F$ is very *sparse*, as is indeed often the case in NLP settings, the reduced space $G$ will typically be much more dense—the exact degree depending on the number of non-zero entries specified in $R$. In other words, the absolute number of non-zero elements will then be higher for the reduced space than the original space. In practice, the cost of storing a given matrix depends not on its dimensionality alone, but its number of non-zero elements. The reason, of course, is that any zero-valued element can simply be ignored. This means that storing the reduced space might actually end up requiring *more* memory than storing the original non-reduced space.

Naturally, the same line of argument also applies to computational aspects. Assuming a non-naive implementation, the computational cost of many vector operations depends less on the total number dimensions and more on the number of non-zero elements of the vectors. This means that certain common operations such as dot-products, euclidean distance, etc., might take *longer* to compute in the reduced (but more dense) space compared to the non-reduced (but more sparse) space.

## 5 #Hashing—A Trending Topic

There has recently been a series of papers in the NLP and ML literature on the use of hashing for constructing faster and more compact models. Several authors have explored the use of hash-based randomized data structures for storing approximate frequency counts for large data sets, such as the generalized notion of a *Bloom filter* used by Talbot and Osborne (2007) and Durme and Lall (2009) in the context of language modeling, or the *Count-Min sketch* used by Goyal et al. (2010) for computing web-scale distributional similarities. More closely related to the work presented in the current paper perhaps, is the notion of *hash kernels* introduced by Shi et al. (2009) and Weinberger et al. (2009) in the context of SVM-based spam filtering and topic categorization. With hash kernels, high-dimensional input vectors are compressed using a single hash function that maps the original features into a smaller range of indices, and dot-products are then computed between these hash maps. For an input vector $f \in \Re^d$, and for some hash functions $h : U \rightarrow C$ and $\xi : U \rightarrow \{\pm 1\}$, the $i$th element of a hash map $\phi(f)$ is defined as

$$\phi_i^{(h,\xi)}(f) = \sum_{j:h(j)=i} \xi(j) f_j \qquad (5)$$

Each element in the hash map is given by a signed sum of all coordinates with the same hash code (Weinberger et al., 2009). Shi et al. (2009) argue that by using only a single hash function, hash kernels *preserve sparsity*. However, to reduce the information loss caused by collisions in the hash map, the original features are explicitly *duplicated* prior to hashing (Weinberger et al., 2009). Each level of duplication will incur a doubling in the dimensionality $d$ of the original feature space $F$. In contrast, with the RI methodology, the same effect is achieved by simply increasing the ratio of non-zeros in the index vectors. On this background, we see that the signed sum in the hash kernel of Weinberger et al. (2009) is essentially equivalent to the result of adding ternary index vectors in the random indexing approach of Kanerva et al. (2000). Moreover, when comparing hash kernels to the general approach of random projections, Weinberger et al. (2009) note that one advantage of the former is that there is no need for storing the random matrices. With hashed RI, however, we need not store neither the random projection matrix $R$ nor the original feature matrix $F$.

Another line of work bearing resemblance to (H)RI is the use of *Locality Sensitive Hashing* (LSH) for identifying semantically similar words by Ravichandran et al. (2005). LSH is a method for fast but approximate nearest neighbor search based on compact *bit signatures* created for each data point or vector. These signatures are created by applying multiple binary hash functions to each point in a way so that close items are hashed to the same buckets with a high probability. The cosine similarity of the original word vectors is then approximated by the hamming distance of their bit signatures. In the work of Ravichandran et al. (2005), the value of each hash function ($\{0,1\}$) is defined by the sign of the dot product between each word vector and a random vector. The bit signatures produced by LSH can be viewed as similar to the reduced representations produced by HRI, although the underlying perspective on the process itself can at first seem rather different: While the LSH approach of Ravichandran et al. (2005) defines hash functions over points in terms of random projections, HRI defines random projections in terms of hash functions over dimensions. Moreover, a modified version of projection-based LSH is presented by Van Durme and Lall (2010) for online generation of bit signatures for data *streams*. Taking advantage of the fact that the operations in the dot products between the data vectors and the random "hash vectors" are linear, Van Durme and Lall (2010) replace the dot products with individual additions corresponding to the random values associated with each feature as it is encountered in the stream, thereby taking a step towards the incrementality that we have several times pointed out in relation to random indexing above.

The final example of related work that we will be discussing is the feature hashing approach of Ganchev and Dredze (2008). Targeting NLP applications on resource constrained devices, Ganchev and Dredze (2008) suggest *eliminating the symbol-table* (also known as the alphabet, dictionary, etc.), replacing it instead with a hash function. Tests on a range of tasks (sentiment analysis, spam detection, topic labeling, etc.) shows "tolerable" degradation of performance relative to savings in storage (Ganchev and Dredze, 2008). A similar approach was taken by Bohnet (2010), who uses feature hashing for speeding up the feature handling in a data-driven dependency parser. It is important to note that, rather than being primarily aimed at dimensionality reduction, the approach of Ganchev and Dredze (2008) aims to save resources by discarding the symbol-table. In fact, in order to reduce the chances of collisions, the assumed dimensionality is instead sometimes greatly *increased* using this approach, as in the dependency parsing experiments of Bohnet (2010).

An interesting direction for future work would be to combine HRI with the feature hashing approach of Ganchev and Dredze (2008), i.e. applying HRI on the symbolic feature representations directly. It should be noted that when learning a model that is to be applied to unseen test examples, the expected savings in terms of storage would likely come at the cost of reduced accuracy. The reason is that the space of possible features instantiated by our feature templates is typically not closed, in the sense that we might expect to instantiate features during testing that were not observed during training (e.g. unseen $n$-grams). Usually such unseen features will be filtered out and discarded as they will not correspond to an entry in the model's symbol-table. However, when by-passing the symbol-table and applying the feature hashing directly on the string level, we risk introducing some noise by mapping such previously unknown features into our feature vectors. If, on the other hand, we are to work within the "closed" vector space itself (for example, searching for nearest neighbors among given points in a semantic space model, as opposed to using the space as input for estimating a classifier), such worries would not arise. Although it would mean giving up the possibility to assign meaning to specific dimensions, that is something we have already done when applying random indexing in first place.

## 6 Conclusion

While random indexing (RI) is a well-established technique for dimensionality reduction, this paper has described a novel reformulation of the method, dubbed hashed random indexing (HRI), that eliminates the need to store any random vectors, thereby substantially reducing the memory footprint of the method. This is accomplished by replacing the so-called index vectors or random labels with a set of hash-functions. We furthermore suggest that these functions are drawn from the family of multiplicative universal hash functions described by Dietzfelbinger et al. (1997). Finally, we have also noted some general caveats regarding dimensionality reduction methods based on random projections, random indexing included, as well as discussed the relation of (H)RI to other approaches employing various notions of feature hashing.

## References

Dimitris Achlioptas. 2001. Database-friendly random projections. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Santa Barbara, USA.

Bernd Bohnet. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, Beijing, China.

Larry Carter and Mark N Wegman. 1979. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2).

Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. 1997. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1).

Benjamin Van Durme and Ashwin Lall. 2009. Probabilistic counting with randomized storage. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, Pasadena, USA.

Kuzman Ganchev and Mark Dredze. 2008. Small statistical models by random feature mixing. In *Proceedings of the ACL-2008 Workshop on Mobile Language Processing*, pages 19 – 20, Columbus, USA.

Amit Goyal, Jagadeesh Jagarlamudi, Hal Daumé III, and Suresh Venkatasubramanian. 2010. Sketch techniques for scaling distributional similarity to the web. In *GEometrical Models of Natural Language Semantics Workshop (GEMS) at ACL*, pages 51 – 56, Uppsala, Sweden.

Robert Hecht-Nielsen. 1994. Context vectors: General purpose approximate meaning representations self-organized from raw data. In J. M. Zurada, R. J. Marks II, and C. J. Robinson, editors, *Computational Intelligence: Imitating Life*. IEEE Press, Orlando, USA.

William Johnson and Joram Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26.

Pentti Kanerva, Jan Kristoferson, and Anders Holst. 2000. Random indexing of text samples for latent semantic analysis. In *Proceedings of the 22nd Annual Conference of the Cognitive Science Society*, Pennsylvania.

Jussi Karlgren and Magnus Sahlgren. 2001. From words to understanding. In Y. Uesaka, P. Kanerva, and H. Asoh, editors, *Foundations of Real-World Intelligence*. CSLI Publications, Stanford.

Ping Li, Trevor Hastie, and Kenneth Church. 2006. Very sparse random projections. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, USA.

Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *Proceedings of the 43rd Meeting of the Association for Computational Linguistics*, Michigan, USA.

Magnus Sahlgren. 2005. An introduction to random indexing. In *Proceedings of the Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering (TKE)*, Copenhagen, Denmark.

Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, Alex Strehl, and Vishy Vishwanathan. 2009. Hash kernels. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, Florida, April.

David Talbot and Miles Osborne. 2007. Smoothed Bloom Filter language models: Tera-scale LMs on the cheap. In *Proceedings of the 2007 Conference on Empirical Methods in Natural Language Processing*, Prague, Czech Republic.

Benjamin Van Durme and Ashwin Lall. 2010. Online generation of locality sensitive hash signatures. In *Proceedings of the 48th Meeting of the Association for Computational Linguistics*, Uppsala, Sweden.

Erik Velldal. 2010. Detecting uncertainty in biomedical literature: A simple disambiguation approach using sparse random indexing. In *Proceedings of the Fourth International Symposium on Semantic Mining in Biomedicine (SMBM)*, Cambridgeshire, UK.

Santosh S. Vempala. 2004. *The Random Projection Method*, volume 65 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, RI, USA.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, Montreal, Canada.

Dominic Widdows and Kathleen Ferraro. 2009. Semantic vectors: a scalable open source package and online technology management application. In *Proceedings of the 6th International Conference on Language Resources and Evaluation*, Marrakech, Morocco, May.