Simple Principles of Cognitive Computation with Distributed Representations

Blerim Emruli
Simple Principles of
Cognitive Computation with
Distributed Representations

Blerim Emruli

Department of Computer Science, Electrical and Space Engineering
Luleå University of Technology
Luleå, Sweden

Supervisors:
Jerker Delsing, Lennart Gustafsson and Fredrik Sandin
To my family
Brains and computers represent and process sensory information in different ways. Bridging that gap is essential for managing and exploiting the deluge of unprocessed and complex data in modern information systems. The development of brain-like computers that learn from experience and process information in a non-numeric cognitive way will open up new possibilities in the design and operation of both sensor and information communication systems.

This thesis presents a set of simple computational principles with cognitive qualities, which can enable computers to learn interesting relationships in large amounts of data streaming from complex and changing real-world environments. More specifically, this work focuses on the construction of a computational model for analogical mapping and the development of a method for semantic analysis with high-dimensional arrays.

A key function of cognitive systems is the ability to make analogies. A computational model of analogical mapping that learns to generalize from experience is presented in this thesis. This model is based on high-dimensional random distributed representations and a sparse distributed associative memory. The model has a one-shot learning process and an ability to recall distinct mappings. After learning a few similar mapping examples, the model generalizes and performs analogical mapping of novel inputs. As a major improvement over related models, the proposed model uses associative memory to learn multiple analogical mappings in a coherent way.

Random Indexing (RI) is a brain-inspired dimension reduction method that was developed for natural language processing to identify semantic relationships in text. A generalized mathematical formulation of RI is presented, which enables N-way Random Indexing (NRI) of multidimensional arrays. NRI is an approximate, incremental, scalable, and lightweight dimension reduction method for large non-sparse arrays. In addition, it provides low and predictable storage requirements, and also enables the range of array indices to be further extended without modification of the data representation. Numerical simulations of two-way and ordinary one-way RI are presented that illustrate when the approach is feasible. In conclusion, it is suggested that NRI can be used as a tool to manage and exploit Big Data, for instance in data mining, information retrieval, social network analysis, and other machine learning applications.

Keywords: analogical mapping · Big Data · cognitive computation · data mining · dimension reduction · distributed representations · Random Indexing · semantic computation · Sparse Distributed Memory · stream mining · Vector Symbolic Architectures
## CONTENTS

### Part I

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1 – Introduction</td>
<td>3</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>5</td>
</tr>
<tr>
<td>1.2 Objectives</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Outline</td>
<td>5</td>
</tr>
<tr>
<td>Chapter 2 – Background</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Distributed representations</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Sparse Distributed Memory</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Random Indexing</td>
<td>13</td>
</tr>
<tr>
<td>2.4 N-way Random Indexing</td>
<td>13</td>
</tr>
<tr>
<td>Chapter 3 – Summary of appended papers</td>
<td>17</td>
</tr>
<tr>
<td>3.1 Paper A</td>
<td>17</td>
</tr>
<tr>
<td>3.2 Paper B</td>
<td>18</td>
</tr>
<tr>
<td>Chapter 4 – Conclusions</td>
<td>19</td>
</tr>
<tr>
<td>4.1 Contributions</td>
<td>19</td>
</tr>
<tr>
<td>4.2 A look ahead</td>
<td>20</td>
</tr>
</tbody>
</table>

### References

23

### Part II

<table>
<thead>
<tr>
<th>Paper</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper A</td>
<td>29</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>31</td>
</tr>
<tr>
<td>2 From mapping vectors to mapping memories</td>
<td>32</td>
</tr>
<tr>
<td>3 Model</td>
<td>35</td>
</tr>
<tr>
<td>4 Results</td>
<td>39</td>
</tr>
<tr>
<td>5 Related work</td>
<td>45</td>
</tr>
<tr>
<td>6 Conclusions</td>
<td>47</td>
</tr>
<tr>
<td>Paper B</td>
<td>53</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>55</td>
</tr>
<tr>
<td>2 Indifference property of high-dimensional ternary vectors</td>
<td>57</td>
</tr>
<tr>
<td>3 N-way Random Indexing</td>
<td>61</td>
</tr>
<tr>
<td>4 Simulation results</td>
<td>66</td>
</tr>
</tbody>
</table>
First, I would like to express my sincere gratitude to my supervisors Jerker Delsing, Lennart Gustafsson and Fredrik Sandin. I am grateful to Jerker Delsing for accepting me as his Ph.D. student and for his continuous support; to Lennart Gustafsson for believing that I am well prepared and strongly motivated to pursue research and education; and to Fredrik Sandin for being my day-to-day supervisor and for his interdisciplinary intellectual curiosity, with interests ranging from single neuron-firing to consciousness, which served as a catalyst for this thesis, from conception to completion.

Furthermore, I thank Tamas Jantvik for being my mentor, Magnus Sahlgren for sharing his knowledge and expertise in natural language processing, Mevludin Memedi for our friendship, Asad Khan for fruitful discussions, and Jan van Deventer for making my life a bit easier.

Finally, my deepest gratitude goes to my parents and my sister for their love and support throughout my life.

_Blerim Emruli_
_Luleå, May 2012_
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>Analogical Mapping Unit</td>
</tr>
<tr>
<td>BSC</td>
<td>Binary Spatter Code</td>
</tr>
<tr>
<td>HRR</td>
<td>Holographic Reduced Representation</td>
</tr>
<tr>
<td>HAL</td>
<td>Hyperspace Analogue to Language</td>
</tr>
<tr>
<td>LSA</td>
<td>Latent Semantic Analysis</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>NRI</td>
<td>N-way Random Indexing</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RI</td>
<td>Random Indexing</td>
</tr>
<tr>
<td>RPM</td>
<td>Raven’s Progressive Matrices</td>
</tr>
<tr>
<td>TASA</td>
<td>Touchstone Applied Science Associates</td>
</tr>
<tr>
<td>TOEFL</td>
<td>Test of English as a Foreign Language</td>
</tr>
<tr>
<td>SVD</td>
<td>Singular Value Decomposition</td>
</tr>
<tr>
<td>SDM</td>
<td>Sparse Distributed Memory</td>
</tr>
<tr>
<td>VSAs</td>
<td>Vector Symbolic Architectures</td>
</tr>
</tbody>
</table>
Part I
Perhaps paradoxically, the architecture of computers is different from that of the human brains that invented and designed them. Most computers operate sequentially and have a central clock that synchronizes their processing. These and other disparities in architecture (and processing speed) enabled computers to surpass humans in arithmetic and in many other similar tasks for which precise sequential and logical steps are involved, for example, executing algorithms and storing, retrieving and transmitting exact information.

Over time, some people realized that computers could be more than “good number crunchers”. Thus, in 1956, leading researchers from different fields, including mathematics, electrical engineering and psychology, gathered at Dartmouth College to make computers “artificially intelligent”; in a sense, they aimed to enable computers to produce intelligent behavior. The idea that computers could do anything a brain could do and that they could be programmed to produce intelligent behavior was at its peak (Hawkins and Blakeslee, 2004). This led to some pragmatic but limited algorithms, such as the A* search, branch and bound, simulated annealing, and genetic algorithms; and methodologies such as automated theorem proving, knowledge representation, expert systems, and logic programming. Some of these algorithms and methodologies are still useful and successful today. However, so far, they have been limited to a narrow domain, in contrast to what most of the AI pioneers believed. In 1958, Nobel laureate Herber Simon stated that “there are now in the world machines that think, that learn and that create ... In a visible future – the range of problems they can handle will be coextensive with the range to which the human mind has been applied” (Simon and Newell, 1958).

Shortcomings of the traditional approach, known as symbolic computation, where meaning-free symbols are manipulated according to pre-defined rules that a programmer has coded (Newell and Simon, 1976), led to investigations of more biologically inspired approaches such as neural networks, for which no pre-defined rules are needed. Neural network researchers, also referred to as connectionists, were initially interested in mimicking the human brain. Their models were based on loosely coupled neurons and could learn and adapt to a changing environment, thus enabling genuine improvements over
the previous symbolic models. With some minor modifications, these networks could perform a variety of tasks, such as pattern recognition, classification and prediction. At that time, the field was flourishing, and new visions about thinking machines started to surface.

Neural networks brought new and interesting properties, such as unsupervised learning, generalization and robustness, but they also had some evident shortcomings. First, the implicit “rules” that they statistically acquire in order to produce behavior remained unidentifiable to the human observer. Second, their inability to handle hierarchical structures and other systematic aspects of human cognition were subject to heavy criticism from proponents of the traditional symbolic school (Fodor and Pylyshyn, 1988). Third, and most important, as the field evolved, it became dominated by researchers who were more interested in producing intelligent behavior rather than making these networks more closely resemble biological systems (Hawkins and Blakeslee, 2004).

Today, the term neural network, or more appropriately artificial neural network, indicates a diverse set of models, some of which are biologically plausible and some of which are not.

In the field of artificial intelligence and cognitive science, the representations chosen are commonly considered to determine which types of tasks the system can handle (Churchland and Sejnowski, 1992, Valiant, 2000, Plate, 2003, Stewart and Eliasmith, 2012, Doumas and Hummel, 2012). From a different perspective, this is also known to engineers and computer theoreticians. For example, a representation that works well for addition also works fairly well for multiplication, whereas a representation that simplifies multiplication may be useless for addition (Kanerva, 2001). Neural networks typically use distributed representations, while symbolic models use localist representations (Stewart and Eliasmith, 2012, Doumas and Hummel, 2012). Studies based on neural networks have shown that distributed representations can capture semantic information and can enable learning and generalization, while localist representations can integrate hierarchical structures and, therefore, are presumably more suited for modeling high-level cognitive functions (Neumann, 2001).

Once in a while, a new insight comes along that changes the direction of thinking. This type of fundamental change happened in the past when humans evolved from using fingers and other body parts for counting, moving to the present day number system. A similar step in evolution has been seen with the advent of neural networks; unlike symbolic models, these networks learned about the environment in an unsupervised way. The excitement to contribute to the first step in a new direction of cognitive computation underpins this thesis. This work involves the development of non-numeric cognitive processing of distributed representations with inherited semantics.
1.1 Motivation

The motivation of this thesis is to explore simple principles of cognitive computation based on distributed representations with inherited semantics. These representations are high-dimensional, associative and evolve from experience rather than being determined by some pre-defined conventions. This allows meaningful instead of meaning-free symbolic manipulation, a descriptor that was previously used with symbolic models (Newell and Simon, 1976). Online learning is prioritized instead of offline training, and only a few examples are needed to generalize from experience. Structure and semantics are integrated to a degree similar to human cognition (Eliasmith and Thagard, 2001, Gentner and Forbus, 2011). The approach considered here is simpler than that in computational neuroscience and more transparent than that in neural networks. This approach is taken in an attempt to understand more about the basic mathematical structures and operations needed to implement cognitive behavior. Note that cognitive behavior is only a demonstration of the approach taken, and not its main characteristic.

1.2 Objectives

The objective of this thesis is to study and develop simple computational principles with cognitive qualities, which can be implemented in computers and networked embedded systems. These principles can enable such systems to autonomously learn and adapt to complex and changing real-world environments, and to deal with imprecise, incomplete and unstructured information. A key idea is to build on simple principles that are computationally efficient, and to avoid recurrent and recursive signaling. Moreover, the approach should be scalable and have a distributable structure that makes it possible to decentralize information processing. The specific questions investigated in the appended papers are:

- Is it possible to store mapping examples of hierarchical structures in a distributable associative memory, so that the memory can be used to make correct analogies from novel inputs?
- Is it possible to extend the traditional method of Random Indexing (RI) to handle matrices and higher-order arrays in the form of N-way Random Indexing (NRI), so that more complex semantic relationships can be analyzed?

1.3 Outline

This thesis is a compilation thesis that consists of two parts\(^1\). Part I of the thesis introduces the reader to some of the key ideas and concepts needed to understand the appended papers in Part II and is written with the intention of being easily understood.

---

\(^1\)A compilation thesis is different from monographs, in which research is presented as a single coherent text.
The appended papers have been submitted to international scientific journals for peer review.
Chapter 2

Background

This chapter briefly presents the theoretical background of this thesis and the appended papers in Part II, and mentions relevant previous work.

2.1 Distributed representations

There are significant differences between how a brain and how a computer represent and process sensory information. Here, representation means how information is represented in a physical medium, for example, in an optical disk, a computer memory or a neuromorphic chip. In a digital computer, and in particular in cognitive models, there are two common and distinct approaches to represent information, commonly referred to as localist and distributed representations. One of the differences between these two approaches is illustrated in Table 2.1. This section briefly describes and addresses the usefulness and limitations of distributed representations from the perspective of this work. There are many differences, similarities, and respective advantages of distributed representations over the localist representations, which are not covered in detail here. The interested reader may be referred to Plate (2003), Stewart and Eliasmith (2012), and Doumas and Hummel (2012) for more information.

The distributed representations used in this thesis are high-dimensional vectors of fixed size that consist of many elements, where each element acts as a neuron output or an input synapse weight. In distributed representations, each concept is represented over multiple vector elements, and each element participates in the representation of multiple concepts (Hinton, 1986).

All of the representations have a fixed dimensionality, such as roles, fillers and relations. The fixed length of the vectors implies that new concepts can be formed from simpler concepts without increasing the size of the vectors, at the cost of increasing the noise level. These properties provide a number of biologically realistic, mathematically

\footnote{The term concept here is referred to as an abstract idea used to capture any aspect of the world that may be useful in describing it (Valiant, 2000).}
Table 2.1: This example demonstrates one of the key differences between localist and distributed representations. \( l_1 \) is a localist representation that is based on the binary coding used in digital computers. \( l_1 \) represents the decimal number 1 and its “meaning” comes from a pre-defined character-encoding scheme known as ASCII. On the right-hand side, \( d_1 \) is a distributed representation, and it represents a geometric circle. However, its meaning does not come from a pre-defined convention and instead arises from previously generated representations. For example, if earlier we had encoded a representation for concepts such as geometry and circle, then a combination of these two representations will create the new representation and a meaning of geometric circle. The binding of these two representations will be computed as \((\text{geometry} + \text{circle})\). For example, if \( \text{geometry} = 1000 \ 0100 \) and \( \text{shape circle} = 0010 \ 1001 \), then the new representation geometric circle is \( d_1 = 1010 \ 1101 \). Distributed representations yield new representations that are based on previously encoded representations, by inheriting their meanings. In contrast to localist representations, they are not rigid, bounded under some pre-defined conventions and semantically brittle. This construct is important for handling noise and for facilitating learning and generalization. Note that distributed representations are usually high dimensional \((D > 1000)\); this characteristic in combination with a clean-up memory (Plate, 2003, p. 102) enables them to produce noise-free patterns from heavily distorted and incomplete representations.
challenging problem (Fodor and Pylyshyn, 1988). Moreover, these representations support holistic processing. The process of directly operating on a hierarchical structure and simultaneously accessing all of its constituents, without a need to decompose it, is called holistic processing (Neumann, 2001). For example, it is not necessary to chase pointers to access a constituent as in symbolic models (Plate, 2003). This process is further illustrated in Table 2.2 from the perspective of Paper A.

Binary Spatter Code (BSC) is one example of a distributed representation (Kanerva, 2000) that is used in Paper A. Another well-known example of a distributed representation is the Holographic Reduced Representation (HRR) (Plate, 2003). The term “holographic” in this context refers to a convolution-based binding operator, which resembles the mathematics of holography. Historical developments in this field include early holography-inspired models of associative memory (Reichardt, 1957, Gabor, 1968, Longuet-Higgins, 1968). Note that the BSC is mathematically related to frequency-domain HRR (Aerts et al., 2009), because the convolution-style operators of HRR unfold to element-wise operations in frequency space. The recent implementation of HRR in a network of integrate and fire neurons by Rasmussen and Eliasmith (2011), is one example how these cognitive models eventually may be unified with more realistic dynamical models of neural circuits.

In a BSC, roles, fillers, relations and hierarchical structures are represented by binary vectors, $x_k$, of dimensionality $D$

$$x_k \in B^D, \quad x_k = (x_{k,1}, x_{k,2}, x_{k,3}, \ldots, x_{k,D}).$$

(2.1)

The elements of these vectors are populated randomly with an equal probability of zeros and ones. The operator $\otimes$ is referred to as the binding operator and is defined as the element-wise binary XOR operation. Bundling is another operator, which bundles multiple vectors $x_k$ and is defined as an element-wise binary average

$$\langle \sum_{k=1}^{n} x_{k,i} \rangle = \Theta \left( \frac{1}{n} \sum_{k=1}^{n} x_{k,i} \right),$$

(2.2)

where $\Theta(x)$ is a binary threshold function

$$\Theta(x) = \begin{cases} 1 & \text{for } x > 0.5, \\ 0 & \text{for } x < 0.5, \\ \text{random otherwise.} \end{cases}$$

(2.3)

This construct is an element-wise majority rule (when an even number of vectors are bundled, there could be ties).

2.1.1 Encoding of distributed representations

The transformation of raw sensory input into a form that is suitable for interpretation of the environment is known as the encoding problem (Abbott and Dayan, 2005). The approaches used in the literature to encode distributed representations can be divided
Background

Relation Representation

| Circle is above the square | $\bullet \square = (a + a_1 \otimes \bullet + a_2 \otimes \square)$ |

Table 2.2: BSC representation of the hierarchical structure “circle is above the square” that is presented in Figure 2.1. Structures are typically constructed from randomly generated names, roles and fillers with the \textit{binding} and \textit{bundling} operators $(a + a_1 \otimes \bullet + a_2 \otimes \square)$, where $a$ is the \textit{relation name} (“above”), $a_1$ and $a_2$ are \textit{roles} of the relation, and the geometric shapes are \textit{fillers} indicating what is related. The following example shows how to directly operate on this structure to access one of its constituents $a_1 \otimes \bullet \square \sim \bullet$. Note that the obtained result is not identical; instead, it is similar to the original filler. The correct constituent can be identified because the obtained result is correlated with the original filler. This process can be realized with a clean-up memory, for example, where the representation of $\bullet$ is stored.

mainly into four categories: (1) hand-coded; (2) random; (3) encoded in a supervised way; and (4) encoded in an unsupervised way.

(1) Hand-coded representations are widely used; a classic example is the McClelland et al. (1986) past tense acquisition model. (2) In Paper A, most of the referenced works use a mixture of random and hand-coded representations (Plate, 1995, Kanerva, 2000, Neumann, 2001, Eliasmith and Thagard, 2001, Plate, 2003). The primary advantage of random representations over the hand-coded representations is that they allow simpler encoding for the smallest constituents, usually referred to as \textit{atoms}. (3) One interesting property of multilayer neural networks based on the backpropagation algorithm is to learn distributed representations in its hidden layer from input-output examples. This type of learning is called supervised learning, and one such example is the family tree network from Hinton (1986); a more recent method that addresses the same task is presented by Paccanaro and Hinton (2001). (4) Unsupervised neural networks have been widely used for feature extraction from images (Olshausen and Field, 1996, Hinton et al., 1997, Ranzato et al., 2011) and recently image sequences (Cadieu and Olshausen, 2012). Using methods from natural language processing (NLP), such as Latent Semantic Analysis (LSA) (Deerwester et al., 1990), Hyperspace Analogue to Language (HAL) (Lund and Burgess, 1996) and recently Random Indexing (RI) (Kanerva et al., 2000), distributed

Figure 2.1: The circle is above the square. Thus, this implies that the square is below the circle. The model presented in Paper A can learn this simple “above–below” relation from examples and can successfully apply it to novel representations and relations.
2.2 Sparse Distributed Memory

We learn by experience, by interacting with the world, engaging with different people and trying new things. This experience is accumulated in our brains as a record. Our brains relate to this record constantly, to make predictions of future events and to choose appropriate courses of action. In cognitive psychology, this process is called memorization.

Sparse Distributed Memory (SDM) is a biologically inspired and mathematically simple model of this process; for details, see Kanerva (1988). Kanerva developed SDM to model some characteristics of human long-term memory. In his study, he sought to answer two key questions:

- How do humans recall past experiences and distinguish the familiar from the unfamiliar?
- How can we construct a digital binary memory from associative neuron-like components that enables efficient storage and retrieval of a record?

SDM essentially consists of two parts, a set of binary **address vectors** and a set of integer **counter vectors**, with a fixed one-to-one link between the address vectors and the counter vectors. The address vectors have dimensionality $D$ and are populated randomly with equal probability of zeros and ones. Counter vectors have dimensionality $D$ and in the beginning are initialized to zeros. Note that address and counter vectors are usually high dimensional ($D > 1000$). The number of address vectors (and counter vectors), defines the size of the memory. When a memory is written to the SDM some counter vectors are updated, but addresses are static. It is typically sufficient to have six or seven bits of precision in the counter elements (Kanerva, 1988). The random nature of the representations makes saturation unlikely with that precision. SDM has a good storage capacity (Keeler, 1988).

SDM can operate as an auto-associative and a hetero-associative memory. In Paper A, SDM is used as a hetero-associative memory. For example, to store pattern $Y$, it uses pattern $X$ as an input to the address vectors, and it uses $Y$ as an input to the counter vectors. SDM operates as an auto-associative memory when pattern $X$ is stored using itself as an input for the address vectors as well as the counter vectors. This is illustrated in Figure 2.2.

SDM inherits some features from conventional random access memory (RAM). For example, there is a binary address space (where address vectors are located), and the memory is addressed through store and retrieve operations. However, the addressing operation is quite different - it activates not a single but many locations, which are sparsely distributed across the memory. Moreover, SDM is an associative memory and
Figure 2.2: SDM interpreted as a computer memory.

can recall complete patterns when given a noisy or distorted input. In addition, SDM can be considered also as a neural network. In particular, a simple feedforward neural network can implement an auto-associative SDM, see Figure 2.3. This interpretation helps to compare SDM to other neural network models.

Figure 2.3: SDM interpreted as a feedforward neural network with sparse activations.
SDM inherits the properties of distributed representations. For example, SDM is high-dimensional and semantically sensitive; it degrades smoothly, and each memory location stores multiple patterns. Moreover, the information is represented in relatively few and sparse active locations, a scheme that is widely referred to as sparse coding (Olshausen and Field, 1997). There is also evidence that SDM exhibits interesting psychological traits for modeling episodic memory (Baddeley et al., 2001). In Paper A, the presented computational model of analogical mapping incorporates SDM to learn multiple analogical mappings from experience and to generalize with novel inputs.

2.3 Random Indexing

RI is a dimension reduction method that was originally developed for NLP. In that context, RI is used to estimate semantic relationships by the statistical analysis of word usage in text. RI is an approximate dimension reduction method that enables incremental coding of high-value matrix elements, which represent significant or common relationships between features in large datasets. The idea of RI derives from Pentti Kanerva’s work on SDM and related work on brain-like information processing with distributed representations.

Latent Semantic Analysis (LSA) (Deerwester et al., 1990) and Hyperspace Analogue to Language (HAL) (Lund and Burgess, 1996) are two pioneering and successful vector space models that have been used to perform semantic analysis of text. Unfortunately, these methods first construct a large word-word or word-document matrix, a so-called co-occurrence matrix, and then apply computationally costly Singular Value Decomposition (SVD) to reduce the dimensionality. RI avoids those computationally intensive steps and directly generates vectors with reduced dimensionality. RI thus requires a fraction of the RAM and processing power of LSA and HAL (Cohen and Widdows, 2009).

2.4 N-way Random Indexing

"Think Big, Start Small, Scale Fast."

In 2007, the world passed the critical point at which more data were produced than could physically be stored (Baraniuk, 2011). As time passes, this gap widens rapidly. Big Data is the buzzword for this phenomenon. Note that Big Data should not be interpreted only in terms of volume. For example, IBM characterizes Big Data with three attributes: volume, velocity and variety. They refer to these attributes as “the 3 Vs”, or “V3” (Zikopoulos et al., 2011), see Figure 2.4.

A number of key challenges and opportunities are as follows: How can computers (1) process and archive useful, and discard useless information?; (2) process text and other “non-numeric” information in a meaningful way?; (3) process and exploit large amounts of information streaming from heterogeneous sources?; (4) detect and predict events quickly and accurately? and (5) manage and exploit ternary or higher-order relationships, which
Figure 2.4: IBM characterizes Big Data with three attributes: volume, velocity and variety. They refer to these attributes as “the 3 Vs”, or “V3”.

is a problem that requires substantially more storage space than binary relationships? Problems of that type are interesting and naturally appear in applications, for example, in the form of context- or time-dependent relationships.

There are a few well-known algorithms for dimension reduction of higher-order arrays (sometimes called tensors); see Kolda and Bader (2009) for a recent review. Tucker decomposition and the parallel factor model are well known and are most commonly used. Tucker decomposition is also known as N-mode PCA or N-mode SVD, which decomposes higher-order arrays, $c_{ijk...}$, according to the scheme

$$c_{ijk...} = \sum_\alpha \sum_\beta \sum_\gamma \cdots g_{\alpha\beta\gamma...} a_{i\alpha} b_{j\beta} c_{k\gamma} \cdots,$$

(2.4)

where $g_{\alpha\beta\gamma...}$ is the core high-order array and $\{a_{i\alpha}, b_{j\beta}, c_{k\gamma}\}$ are factor matrices.

Several methods to compute Tucker decomposition have been developed (Kolda and Bader, 2009). These methods are computationally expensive, because the goal of finding an “optimal” decomposition with side constraints on the factor matrices is nontrivial.

The method presented in Paper B, called NRI, is a generalized mathematical formulation of RI that is somewhat similar to Tucker decomposition, because it uses a mathematically equivalent expression for decoding the representation. NRI is different from Tucker decomposition in the sense that it is based entirely on random coding, and therefore, NRI avoids the computationally expensive process of calculating the factor matrices. In NRI, the factor matrices are randomly generated objects, so-called random
indices. NRI does not require the storage of the original data, and no computationally expensive analysis is needed to insert new data. Array elements are coded in a randomly distributed reduced representation so that elements with high accumulated values can be approximately distinguished from elements with low values. The array does not need to be sparse; it may be fully populated. The high-value elements can be accurately identified when the majority of elements have significantly lower values. The subset of high-value elements is allowed to change over time as a consequence of element update operations.

Standard methods, such as PCA and SVD, and other methods for higher-order arrays, such as Tucker decomposition, are more precise than RI and NRI, but those methods are also computationally complex and of limited use when working with large datasets. In contrast to some methods, RI improves both in efficiency and accuracy as the size of the datasets increases (Gorman and Curran, 2006). NRI is incremental, scalable and lightweight and can thus be used for the online processing and analysis of large streaming datasets. However, NRI is an approximative method and performs well only with high-dimensional arrays (high dimensionality is a prerequisite). In contrast, Tucker decomposition is “exact” and useful for low-dimensional problems because of the computational complexity of the method.
Chapter 3

Summary of appended papers

This thesis includes two journal manuscripts, which are presented in Part II. This chapter briefly summarizes and lists the authors’ contributions to each paper. The appended papers have been submitted to international scientific journals for peer review.

3.1 Paper A

Title: Analogical mapping with sparse distributed memory: a simple model that learns to generalize from experience

Authors: Blerim Emruli and Fredrik Sandin

Summary: This paper presents a computational model of analogical mapping that learns to generalize from experience. This model is based on high-dimensional random distributed representations and a sparse distributed associative memory, which is presented in Section 2.2. Former works demonstrating analogical mapping with BSC and HRR are limited to isolated mapping examples, which makes online learning difficult. The following research question motivates the study: Is it possible to store mapping examples of hierarchical structures in an associative memory, so that the memory can be used to make correct analogies from novel inputs? The answer to this question and additional relevant results are briefly summarized below. The model has a one-shot learning process, and it is able to recall distinct mappings. The model’s ability to generalize increases with the number of mapping examples learned. After learning many similar mapping examples, no specific mapping is recalled, and the model provides the correct mapping by analogy. The feasibility of the model is demonstrated with numerical simulations. The numerically estimated error rate suggests that the model performs well with binary mapping memories. A major improvement over related models is that associative memory is used to learn multiple analogical mappings from experience and to generalize to novel inputs. The model has only three exogenous parameters.
Author contributions: B.E. developed the first version of the model and evaluated the model on the basis of simulation results, studied the literature, and wrote the related work section. F.S. provided the initial idea, helped with the development of the subsequent versions of the model, suggested parts of the simulations, and provided further interpretation of the simulation results. B.E. and F.S. wrote the manuscript, and B.E. submitted the final manuscript for publication.

3.2 Paper B

Title: N-way random indexing of arrays: a simple method for identifying common relationships in large datasets and information streams

Authors: Fredrik Sandin, Blerim Emruli and Magnus Sahlgren

Summary: In this paper, a generalized mathematical formulation of RI is presented. RI is a dimension reduction approach that was originally developed for NLP in order to encode vector-semantic relationships through the statistical analysis of word usage in text. The former method is limited to one-dimensional arrays (vectors). Based on a review of the literature, the following research question is proposed: Is it possible to extend the traditional method of RI to handle matrices and higher-order arrays in the form of NRI? The proposed method, called NRI, is an approximate, incremental, scalable, and lightweight dimension reduction method for large non-sparse arrays, which can be used for identifying array elements that have relatively high accumulated values. This method is based on approximate and randomly distributed reduced representations of the array elements. In addition, the range of array indices can be extended without modification of the array representation, which is beneficial when the feature space is large and the number of features is unknown. Numerical simulations of two-way and ordinary one-way RI are presented that illustrate when the approach is feasible. In conclusion, it is suggested that NRI can be used as a tool to manage and exploit Big Data, for instance in data mining, information retrieval, social network analysis, and event detection and prediction.

Author contributions: F.S. developed the method and the C++ template with an optional Matlab interface, suggested most of the simulations, provided further interpretation of the simulation results, studied the literature, wrote most of the manuscript and submitted the final manuscript for publication. B.E. performed most of the simulations, suggested further refinements to the initial method, generated most of the figures, studied the literature, and wrote parts of the manuscript. M.S. contributed to the ideas on how to apply the method in NLP, wrote parts of the manuscript related to NLP and shared the TASA corpus and TOEFL test items, which were kindly provided by Professor Thomas Landauer of the University of Colorado.

Supplementary information: C++ and Matlab software is provided that supports arrays of dimensionality $\geq 1$. 
This thesis presents a set of simple principles for cognitive computation that can be implemented in computers and networked embedded systems. The key principles are high-dimensionality, randomness and non-numeric operations with cognitive qualities. These principles facilitate autonomous learning and generalization to novel inputs. Moreover, the presented approach is simple, computationally efficient, distributable, and scalable.

4.1 Contributions

At the time of the writing of this thesis, the main contributions to scientific knowledge are as following:

- A construction of a computational model for analogical mapping that incorporates associative memory to learn multiple analogical mappings from experience and to generalize to novel inputs.

- A demonstration that the model performs well with binary memories, and numerically confirming the analytical estimate by Kanerva (2000) that the optimal dimensionality for the hierarchical structures is of order $10^4$.

- An extension of the traditional method of RI to handle matrices and higher-order arrays in the form of NRI.

- A derivation of an analytical result for the approximate orthogonality of high-dimensional ternary vectors and a comparison with corresponding probabilities obtained from explicit numerical simulations.

- Numerical simulations of two-way RI that demonstrate key differences compared to the traditional method.
• A suggestion that measuring the similarity of two words with the Jaccard index of high-value vector elements leads to better results than the traditional cosine approach for the synonym identification task.

4.2 A look ahead

The work presented in this thesis prompts a number of questions for further research. The results presented in Paper A show that the model can perform analogical mapping of novel inputs. In this case, the generalization performance of the model has an optimum. For a specific choice of the sparseness, there is an optimal coding density at which the generalization takes the minimum error rate. However, the location of the optimum in parameter space depends on the structure of mappings that the model is learning. This problem is nontrivial and requires further research.

A different but interesting, achievable task, where the model has already shown promising results, is to provide a set of individual examples from which the model can infer a general rule that predicts the future items in a sequence. One such example is solving Raven’s Progressive Matrices (RPM) (Raven, 1962), which is a popular test in the field of intelligence testing. To solve the RPM, subjects are presented with a $3 \times 3$ matrix, in which each cell in the matrix, except for the last cell in the bottom right, contains different geometrical figures, see Figure 4.1. In this case, the model’s task is to predict which one of eight different alternative solutions will provide the correct answer. A solution for a similar RPM task based on a HRR has already been reported by Rasmussen and Eliasmith (2011). However, in contrast to Rasmussen and Eliasmith (2011), the model presented in Paper A is not based on HRR but instead is based on simple binary representations, called BSC (Kanerva, 2000). Moreover, instead of operating with individual mappings, this model incorporates an associative memory, called SDM (see Section 2.2 for more details), which stores multiple mappings in a coherent way and computes the answer.

A bit further into the future lies the idea of incorporating a visual processing unit that can encode suitable distributed representations for the interpretation of geometrical figures, which is widely referred to as the encoding problem (see Section 2.1.1). On a more technical and pragmatic level is to make the source code of the model available to a wider community under an open source license.

The method presented in Paper B, called NRI, suggests that higher-rank arrays could suffer less from high-dimensional reduction ratios than vectors and matrices. This follows from a surprising result of a simulation that shows that the performance of one-way and two-way RI is comparable at a dimension reduction of $64 : 1$. The nature of this trend could be related to an interesting scaling phenomenon, which is expected to become increasingly interesting for higher-order arrays. This issue is a matter of further investigation, mainly because of the high computational and processing cost. However, note that the provided prototype software supports NRI of higher-order arrays. The performance of the NRI on the TOEFL synonym test illustrates that the number of correctly identified synonyms versus the size of the word-word correlation sets (so called
4.2. A look ahead

Figure 4.1: A simple Raven’s Progressive Matrix task. The model processes the first two rows and then computes to predict the solution to the third row, which is then compared with the eight different alternatives.

top-list) varies, and it is not clear how to determine the optimal length. This problem is nontrivial and requires an analysis of the fidelity of the decoded array components, for example, by generalizing the developments in Kanerva (1988).

Another important goal of this thesis is to nurture applied research with novel and useful computational methods and principles. In this context, the next step is to apply the presented cognitive computing principles to solve real-world problems and then based on this experience to further develop the methodology. This can enable knowledge transfer and technology integration between industry and academia with deep impact to society and science. From the ideas discussed in this thesis, the method presented in Paper B is ready for practical application. NRI is an approximate, incremental, scalable, and lightweight dimension reduction method for large non-sparse arrays. Based on these properties it is suggested that NRI can be used as a tool to manage and exploit Big Data, for instance in data mining, information retrieval, social network analysis, and other machine learning applications.


Part II
Analogical mapping with sparse distributed memory

Authors:
Blerim Emruli and Fredrik Sandin

Submitted for publication in a journal.
Analogical mapping with sparse distributed memory: a simple model that learns to generalize from experience

Blerim Emruli and Fredrik Sandin

Abstract

We present a computational model for analogical mapping of hierarchical structures that learns from examples. This model is based on a sparse distributed memory and operates on high-dimensional binary spatter codes. In addition, it has a one-shot learning process and it is able to recall distinct mappings. After learning a few similar mapping examples the model generalizes correctly and can perform analogical mapping of novel inputs. The ability of the model to generalize increases with the number of mapping examples learned. After learning many similar mapping examples no specific mapping is recalled and the model provides the correct mapping by analogy. Simulation results that characterize this analogical mapping model are presented. The model has only three exogenous parameters.

1 Introduction

Computers have excellent quantitative information processing mechanisms, which can be programmed to execute algorithms on numbers at high rate. Technology is less evolved in terms of qualitative reasoning and learning by experience. In that context biology excel. Intelligent systems that interact with humans and the environment need to deal with unprecise information and learn from experience. A key function is analogy (Gentner, 1983, Minsky, 1988, 2006, Holyoak and Thagard, 1989, 1996, Hofstadter, 2001), the process of using knowledge from past similar experiences to solve a new problem without known solution. In this paper we outline a method of learning analogical mappings of hierarchical structures using an associative memory. Our model is based on an associative memory known as sparse distributed memory (SDM), which operates on high-dimensional binary vectors so that associative and episodic memories can be formed and retrieved (Kanerva, 1988, 1993, Anderson et al., 1993, Claridge-Chang et al., 2009, Linhares et al., 2011). High-dimensional vectors are useful representations of hierarchical structures (Plate, 1995, Kanerva, 1994, 1997, 2000, Neumann, 2002), for in-depth discussion, see Plate (1994, 2003), Neumann (2001) and Kanerva (2009). With an appropriate choice of operators it is possible to create, combine and extract hierarchical structures represented by such vectors. It is also possible to create mapping vectors that perform analogical mapping between hierarchical structures (Plate, 1995, Kanerva, 2000, Neumann, 2002). Such mapping vectors can generalize to structures composed of novel elements (Kanerva, 2000), and to structures of higher complexity than those in the
training set (Neumann, 2002). These ideas and results are interesting, because they are examples of simple mechanisms that enable computers to perform analogical mappings.

From our point of view there are two issues that have limited the further development of these analogical mapping techniques. One, is the “encoding problem” (Harnad, 1990, Barsalou, 2008), which in this context is the problem to encode hierarchical structures from low-level sensor information without the mediation of an external interpreter. This includes the extraction of elementary representations of objects and events, and representations of invariant features of object and event categories. Examples demonstrating the technique are based on hand-constructed structures, which makes the step to real-world applications non trivial. The second issue is that mapping vectors are explicitly represented and that there is no framework to learn multiple analogical mappings. The construction of an explicit vector for each analogical mapping in former studies is excellent for the purpose of demonstrating its properties. However, in practical applications the method needs to be generalized so that the system can learn and use multiple mappings in a simple way. That issue is addressed in this paper.

We outline a method to generalize this analogical mapping approach so that multiple mappings can be learned from experience. The basic idea is that mapping examples are fed to an SDM so that mapping vectors are successively taking shape in memory. After learning one or two examples the system is able to recall, but it typically cannot use the limited experience to generalize. With more similar examples stored in memory the ability of the system to generalize increases. This approach automates the process of creating, storing and retrieving analogical mappings. It enables encapsulation of complexity in a simple “analogical mapping unit” (AMU), which creates abstraction and enables visualization of the higher-level system architecture. In the following we describe this AMU and present numerical results that characterize some of its properties.

2 From mapping vectors to mapping memories

The AMU is an example of a vector-symbolic architecture (VSA) (Gayler, 2003), or perhaps more appropriately named a VSA “component” in the spirit of component-based software design. In general, a VSA is based on a set of operators on high-dimensional vectors of fixed dimensionality, so-called reduced descriptions of a full concept (Hinton, 1990). The fixed length of vectors implies that new hierarchical structures can be formed from simpler structures without increasing the size of the representations, at the cost of increasing the noise level. In a VSA all representations have fixed dimensionality, such as: roles, fillers and relations. These vectors are sometimes referred to as holistic vectors (Kanerva, 1997) and their processing as holistic (Hammerton, 1998, Neumann, 2002). Reduced representations in cognitive models are essentially manipulated with two operators, named binding and bundling. Binding is similar to the idea of neural binding. It combines two vectors into a new vector, which is indifferent (~orthogonal) to the two. The implementation of the binding operator and the assumptions about the nature of the vector elements are model specific. The bundling operator is analogous to superposition. It typically is the algebraic sum of vectors, which may or may not be normalized.
Bundling and binding are used for example to create hierarchical structures. Well-known examples of VSAs are the Holographic Reduced Representation (HRR) (Plate, 1994, 1995, 2003) and the Binary Spatter Code (BSC) (Kanerva, 1994, 1997, 2000). The term “holographic” in this context refers to a convolution-based binding operator, which resembles the mathematics of holography. Historical developments in this direction include early holography-inspired models of associative memory (Reichardt, 1957, Gabor, 1968, Longuet-Higgins, 1968). Note that the BSC is mathematically related to frequency-domain HRR (Aerts et al., 2009), because the convolution-style operators of HRR unfold to element-wise operations in frequency space.

The work in this area has been inspired by cognitive behavior in humans and the structure of specific brain circuits in the cerebellum and cortex, but these models are not intended to be accurate models of biology. In particular, these models typically discard the temporal dynamics of neural processing systems and instead use sequential processing of high-dimensional random distributions. The recent implementation of HRR in a network of integrate and fire neurons (Rasmussen and Eliasmith, 2011) is, however, one example of how these cognitive models eventually may be unified with more realistic dynamical models of neural circuits. The interesting analogical mapping mechanisms enabled by these simple models anyway motivate further exploration and development.

In the following we use the BSC, because it is straightforward to store BSC representations in an SDM and it is more simple than the HRR. It is not clear how to construct an associative memory that enables a similar approach with HRR, but we see no reason why that should be impossible. In a BSC, roles, fillers, relations and hierarchical structures, are represented by binary vector, \( x_k \), of dimensionality \( D \)

\[
x_k \in B^D, \quad x_k = (x_{k,1}, x_{k,2}, x_{k,3}, \ldots, x_{k,D}).
\]

(1)

The binding operator, \( \otimes \), is defined as the element-wise binary XOR operation. Bundling of multiple vectors \( x_k \) is defined as an element-wise binary average

\[
\langle \sum_{k=1}^n x_{k,i} \rangle = \Theta \left( \frac{1}{n} \sum_{k=1}^n x_{k,i} \right),
\]

(2)

where \( \Theta(x) \) is a binary threshold function

\[
\Theta(x) = \begin{cases} 
1 & \text{for } x > 0.5, \\
0 & \text{for } x < 0.5, \\
\text{random} & \text{otherwise}.
\end{cases}
\]

(3)

This is an element-wise majority rule. When an even number of vectors are bundled there may be ties. These elements are populated randomly with an equal probability for zeros and ones. Structures are typically constructed from randomly generated names, roles and fillers with the binding and bundling operators, \( \langle a + a_1 \otimes \bullet + a_2 \otimes \square \rangle \), where \( a \) is the relation name (“above”), \( a_1 \) and \( a_2 \) are roles of the relation, and the geometric shapes are fillers indicating what is related.
Figure 1: Circle is above the square. This implies that the square is below the circle. The analogical mapping unit (AMU) can learn this simple "above–below" relation from examples and successfully apply it to novel representations, see Section 3 and Section 4.

All terms and factors in this representation are high-dimensional vectors, as defined in (1). Vectors are typically initialized randomly, or they may be compositions of other random vectors. In a full-featured VSA application this encoding step is to be automated with other methods. A key thing to realize in this context is that the AMU, and VSAs in general, operate on distributed representations based on high-dimensional random distributions. This makes the VSA approach fairly robust to noise and in principle enables operation with non-ideal encoded hierarchical structures. The starting point for the development of the AMU is the idea of holistic mapping vectors (Kanerva, 2000, Neumann, 2002). In Kanerva (2000) a BSC mapping of the type “X is the mother of Y” \(\rightarrow\) “X is the parent of Y” is presented. This mapping is mathematically similar to the above–below relation illustrated in Figure 1, with the exception that the mother–parent mapping is unidirectional because a parent is not necessarily a mother. We think that the geometric example illustrated here is somewhat simpler to comprehend and we therefore use that. The key idea presented in Kanerva (2000) is that one can construct a generic mapping vector, \(M\), that performs a mapping of the type: “If the circle is above the square, then the square is below the circle”.

The representations of these particular descriptions are illustrated in Table 1 and are chosen to be mathematically identical to those in Kanerva (2000). A mapping vector

<table>
<thead>
<tr>
<th>Relation</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle is above the square</td>
<td>(\bullet, \square = (a + a_1 \otimes \bullet + a_2 \otimes \square))</td>
</tr>
<tr>
<td>Square is below the circle</td>
<td>(\square, \bullet = (b + b_1 \otimes \square + b_2 \otimes \bullet))</td>
</tr>
</tbody>
</table>

Table 1: Two different representations of the geometric composition presented in Figure 1. A mapping vector, \(M\), can be constructed that maps one of the representations into the other (Kanerva, 2000). When constructed from several mapping examples this mapping vector generalizes to novel representations, which in this case means that it can be successfully applied to novel geometric shapes and other objects or events with an associated “above-below” relation. That is an example of an analogical mapping where experience of above-below relations is used in a new situation. The AMU presented below can learn the analogical mapping between many different representations like these.
defined in this way
\[ M = G \uparrow I \downarrow \otimes I \downarrow G \uparrow , \] (4)
has the property that
\[ M \otimes G \uparrow I \downarrow = I \downarrow G \uparrow . \] (5)
This is to be expected, because the XOR-based binding operator is an involutary function.
A more remarkable property appears when bundling several mapping examples
\[ M = \langle G \uparrow I \downarrow \otimes I \downarrow G \uparrow + I \uparrow L \downarrow \otimes L \downarrow I \uparrow + \ldots \rangle , \] (6)
because in this case the mapping vector generalizes correctly to novel representations
\[ M \otimes \# \uparrow N \downarrow \sim N \downarrow \# \uparrow . \] (7)
The symbols \# and \( N \) have not been involved in the construction of \( M \), but the mapping anyway results in the correct analogical mapping. The left- and righthand side of (7) are similar, but not identical. It is possible to identify the correct mapping because the result is highly correlated with it. This process can be realized with a “clean-up memory” (Plate, 1995, chap. 3), for example where the representation of \( N \uparrow \# \) is stored.
Alternatively, the approximate mapping result is transmitted to another VSA component for further processing. Using a VSA operator known as probing it is possible to extract parts of a mapping result, which by themselves may be composite structures. Observe that the analogical mapping (7) is not simply a matter of permutation. The mapping vector maps one representation into another, nearly uncorrelated representation. This mechanism generalize to structures of higher complexity than those in the training set, see Neumann (2002) for one interesting example. These findings motivate the development of the AMU in the next section.

Note that (4) is symmetric in the sense that the mapping is bidirectional. It can perform the two mappings \( M \otimes \bullet_{1} \downarrow \equiv \bullet_{2} \downarrow \rightarrow I \downarrow \) and \( M \otimes I \downarrow \equiv \bullet_{2} \downarrow \rightarrow G \uparrow \) equally well. This is a consequence of the commutative property of the binding operator. In this particular example that does not pose a problem, because both mappings are true. In the parent–mother example in Kanerva (2000) it implies that “parent” is mapped to “mother”, which is not necessarily a good thing. A more severe problem with bidirectional mappings appear in the context of inference, which is a strictly unidirectional mapping. To enable robust analogical mapping and inference in general, the mapping direction of BSC mappings must be controlled. In the next section a simple solution to that problem is presented, which solves also the problem of how to organize multiple mappings in a simple way.

3 Model

To make practical use of mapping vectors we need a method to create, store and query them. In particular, this involves the difficulty of knowing how to bundle new mapping vectors with past experience. How to keep the experience organized without involving
Figure 2: Schematic illustration of a sparse distributed memory (SDM), an associative memory for randomly distributed high-dimensional binary vectors (Kanerva, 1988). The open circle denotes the connection for the input address. The open (solid) square denotes the connection for the input (output) vector in writing (reading) mode.

A homunculus, who could just as well do the mappings for us? Fortunately, a suitable framework has already been developed for another purpose. The development presented here rests on the idea that an SDM used in an appropriate way is a suitable memory for analogical mappings, which automatically bundles similar mapping memories into well-organized mapping memories.

The SDM model of associative and episodic memory is well described in the textbook by Kanerva (1988). Various modifications of the SDM have been proposed (Hely et al., 1997, Anwar and Franklin, 2003, Ratitch and Precup, 2004, Meng et al., 2009, Snaider and Franklin, 2012) but here we use the original SDM model (Kanerva, 1988). It essentially consists of two parts, a set of binary address vectors and a set of integer counter vectors, with a fixed one-to-one link between address vectors and counter vectors. The address vectors have dimensionality $D$ and are populated randomly with equal probability of zeros and ones. Counter vectors have dimensionality $D$ and are initialized to zero. The number of address vectors (and counter vectors), $S$, defines the size of the memory. When a memory is written to the SDM some counter vectors are updated, but addresses are static. It is typically sufficient to have six or seven bits of precision in the counter elements (Kanerva, 1988). The random nature of the representations make saturation unlikely with that precision. An SDM can be visualized as suggested in Figure 2, with two inputs and one output. One of the inputs is the query address, which is supplied both when storing and retrieving information. It is compared with the address vectors of the SDM; All counter vectors with an associated address vector that is within some given Hamming distance, $\delta$, from the query address are activated. In a writing operation the activated counter vectors are updated using the second input vector. For every 1 (0) of that input vector the corresponding activated counter elements are increased (decreased) by one. In a reading operation the activated counter vectors are summed and the resulting integer output vector, $s_{k,i}$, is converted to binary form with the rule

$$s_{k,i} \rightarrow x_{k,i} = \begin{cases} 1 & \text{for } s_{k,i} > 0, \\ 0 & \text{for } s_{k,i} < 0, \\ \text{random otherwise}. \end{cases}$$

(8)

An SDM query is therefore analogous to bundling (2) of all memories with addresses similar to the query address. Technically, the VSA operators and the SDM can be implemented in an alternative way if binary vectors are replaced with bipolar vectors according
to the mapping \( \{0 \rightarrow 1, \ 1 \rightarrow -1\} \). In that case the XOR binding operator is replaced with element-wise multiplication and vector sums have a simple polar interpretation. The number of counter vectors activated by any particular query depends on the proximity parameter \( r \). In this paper we choose to calculate this parameter in each query so that a given fraction, \( \chi \), of the memory is activated. In other words, the number of activated counter vectors is \( \chi S \) in each query. This convention implies that \( \chi \) represents the mutual overlap of different mapping memories. With a low value of \( \chi \) different mapping memories have low overlap, and vice versa. In total the AMU has three exogenous parameters, \( S, \chi \) and the dimensionality \( D \) of the VSA. Parameters of the model are summarized in Table 2. Next we show how mapping examples can be stored in an SDM in a training process, thereby forming analogical mappings that generalize correctly to novel examples.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S )</td>
<td>Memory size, number of address (and counter) vectors</td>
</tr>
<tr>
<td>( D )</td>
<td>Dimensionality of vector symbolic representations</td>
</tr>
<tr>
<td>( \chi )</td>
<td>Overlap of memory representations</td>
</tr>
</tbody>
</table>

Table 2: Summary of exogenous parameters of the AMU.

3.1 Learning of mapping examples

An SDM stores vectors in a process that is similar to the bundling of examples in (6), provided that the addresses of the mapping examples are similar. If the addresses are uncorrelated\(^1\) the individual mapping examples will be stored in different parts of the memory and no bundling takes place, which prevents generalization and analogical mapping. A simple approach is therefore to define the address of a mapping, \( x_k \rightarrow y_k \), as the variable \( x_k \). This implies that mappings with correlated \( x_k \) are, qualitatively speaking, bundled together within the SDM. The mapping vector \( x_k \otimes y_k \) is the input to the SDM, which is bundled to counter vectors at addresses similar to \( x_k \). A schematic diagram illustrating this learning circuit is shown in Figure 3.

This construction avoids the problem of bidirectionality discussed above, because \( y_k \) is uncorrelated with \( x_k \) in non-trivial examples and therefore activates different parts of the memory. If that activation pattern does not correspond to those of former training examples, the output is nonsense (noise). In other words, the reversed mapping \( y_k \rightarrow x_k \) is not implicitly learned. Note that different types of mappings have different \( x_k \) and therefore activate different parts of the memory. Given a sufficiently large memory it is therefore possible in principle to store several different types of mappings in one SDM. This is illustrated qualitatively with simulations in the next section.

---

\(^1\)The Hamming distance between addresses, \( \delta \), is related to the correlation coefficient (normalized covariance) by \( \rho = 1 - 2\delta \) (Kanerva, 1997).
3.2 Analogical mapping by experience

The learning mechanism in combination with the basic idea of analogical mapping (7) suggests that the mapping circuit should be defined as illustrated in Figure 4. This circuit binds the input, $x_k$, with the bundled mapping vector that combines the experience from similar mapping examples. With a number of similar mapping examples stored in the SDM, this circuit correctly generalizes to novel representations. This is illustrated with simulations in the next section.

3.3 The analogical mapping unit – AMU

The AMU includes an SDM and the learning and mapping circuits presented in the former two subsections, see Figure 5. It forms a computational unit for distributed representations that takes two input vectors and provides one output vector, much like the SDM does, but with a different result and interpretation. In principle the SDM could be shared with other VSA components, provided that the mappings are encoded in a suitable way. Next we present simulation results that characterize some properties of the AMU.

Figure 4: Schematic diagram of circuit for analogical mappings of type $x_k \rightarrow y_k$. Here $y'_k \sim y_k$ when $x_k \rightarrow y_k$ is in the training set. Otherwise $y'_k$ is an approximate analogical mapping of $x_k$. It is demonstrated in Section 4 that this mapping is the correct analogy, provided that sufficiently many similar examples have been learnt. This means that among all the alternative mappings that one could possibly come up with, the mapping result is mostly correlated with the correct alternative.
4 Results

An important measure for the quality of an intelligent system is its ability to generalize the acquired knowledge across different domains (Russell and Norvig, 2009). To test this, we introduce novel structures, that holds the “above-below” relation and has never been presented to the AMU, see Figure 6. This structure is encoded in a similar way as that outlined in Table 1, with the difference that the fillers are different, see Table 3.

Our mapping approach is analogous to that in Kanerva (2000), but differs from that in three ways. First, mapping examples are fed to an SDM so that multiple mapping memories are successively taking shape in memory, instead of defining explicit mapping vectors for each mapping. Second, the model uses binary spatter codes to represent mappings, which means that the AMU mapping vectors are binary and not integer. Third, the dimensionality $D$ is set at a somewhat low value of 1000 and we numerically illustrate the effect of higher dimensionality. We adopt the method in Kanerva (2000) to calculate the similarity between the output of the AMU and alternative mappings with the normalized covariance of two binary vectors, which results in the correlation coefficient, $\rho$. Randomly selected elements, such as: $a, b, a_1, a_2, b_1, b_2, \bullet, \blacksquare, \ldots, A, B$ are uncorrelated, $\rho \sim 0$. That is true also for $x_i$ and $y_i$, such as: $\bullet, \blacksquare, \ldots, \bullet, \blacksquare$. However, $\bullet, \blacksquare$ and $A \uparrow B_\downarrow$ include the same relation name in the composition, and they therefore are correlated, $\rho \sim 0.25$. What is the difference between binding and chunking?

The AMU learns mappings between hierarchical structures according to the circuit shown in Figure 3, and the output $y_k$ of the AMU is calculated as illustrated in Figure 4. The output is compared with alternative mapping results, both for retrieval and generalization, in terms of the correlation coefficient. The number of training examples is $N_e$. To estimate the average performance of the AMU we repeat each simulation 5000 times.

![Figure 5: The analogical mapping unit (AMU), including the learning and mapping circuits introduced above. This unit learns mappings of the type $x_k \rightarrow y_k$ and uses that experience to calculate analogical mappings $y'_k$. If an $x_k$ is transmitted to the AMU, but no $y_k$, the analogical mapping $y'_k$ is calculated (mapping mode). If both an $x_k$ and a $y_k$ is transmitted to the AMU it will learn that mapping (learning mode). The AMU has three exogenous parameters: $S, \chi$ and $D$, see Table 2.]
with independent AMUs. This is sufficient to achieve ~1% accuracy in the mean and standard deviation of calculated correlations and error rates.

To compare our model with Kanerva (2000), we set the parameters of the AMU to $S = 1, D = 1000$ and $\chi = 1$. The result of this simulation is presented in Figure 7. The panel on the (left-) right-hand side shows the generalization (retrieval) correlations of the AMU. These results are consistent with Kanerva (2000) and demonstrate that the number of training examples affects the generalization ability of the AMU. One difference between our result and Kanerva (2000) is the constant correlation with the correct mapping alternative in the in the right-hand side panel of Figure 7. In our model the correlation of the correct generalization alternative does not increase with $N_e$, but the correlations with incorrect alternatives decrease with increasing $N_e$. This difference is caused by the use of binary spatter codes within the AMU, instead of integer mapping vectors. If we remove that constraint and use integer vectors we reproduce the results in Kanerva (2000). It is possible to use integer mapping vectors in our model by modifying the normalization condition (8). This improves the performance of the model, but we find that it is sufficient to use ordinary binary spatter codes. The latter approach appears to be more aesthetic and simple, and we therefore develop that here. The idea that integer mapping vectors are needed (Kanerva, 2000, Neumann, 2001) seems to emerge from an analytical underestimate of the mapping performance, caused by an approximation based on normally distributed correlation coefficients. By explicitly calculating the rate of mistakes numerically we conclude that binary mapping vectors perform well. We return to that below.

Next, we increase the memory size to $S = 100$. This is a novel feature of the AMU model that was not possible in former studies (Neumann, 2002, Plate, 1995, 2003, Kanerva, 2000, 2009). We investigate the effect of different values of $\chi$ and present results for $\chi = 0.05$ and $\chi = 0.25$. That is, when the AMU activates 5% and 25% of the SDM memory to learn a particular mapping vector, which encodes a particular mapping. Fig-

<table>
<thead>
<tr>
<th>Relation</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A is above B</td>
<td>$A_1B_1 = \langle a + a_1 \times A + a_2 \times B \rangle$</td>
</tr>
<tr>
<td>B is below A</td>
<td>$B_1A_2 = \langle b + b_1 \times B + b_2 \times A \rangle$</td>
</tr>
</tbody>
</table>

Table 3: Representation of “above-below” relations between novel structures.
Figure 7: Correlation, \( \rho \), between the output of the analogical mapping unit (AMU) and alternative mapping results versus the number of training examples, \( N_e \). The panel on the left-hand side shows the average correlation between the output, \( y'_{i,c} \), and four alternative mappings of the input, \( x_{i,c} = \bigcirc \square \), which is in the training set. The panel on the right-hand side shows the average correlation between four alternative mappings of a novel input, \( x_{i,c} = A \uparrow B \downarrow \). The alternative with the highest correlation is selected as the correct answer. Error bars denote standard deviations. With more training examples the ability of the AMU to generalize increases, because the correlations with wrong alternatives decrease with increasing \( N_e \). The correlations between one particular output \( y'_{i,c} \) and the four alternative mappings are also correlated. For example, a relatively low correlation with the correct alternative is accompanied by low correlations with the wrong alternatives. Therefore, fluctuations of correlations are less problematic than suggested by the error bars. The alternative with the highest correlation always corresponds to the correct result in this simulation, even in the case of generalization from three training examples only. The parameters of the AMU are \( S = 1 \), \( D = 1000 \) and \( \chi = 1 \). Averages and standard deviations for \( N_e \) mappings are calculated with 5000 independent AMUs.
Figure 8: Correlations of mapping results calculated with an AMU having a larger memory. This figure is analogous to Figure 7, with the difference that here the AMU parameters are $S = 100$ and $\chi = 0.05$ or $\chi = 0.25$. The panel on the (left-) right-hand side shows the generalization (retrieval) correlations of the AMU for $\chi = 0.05$ and $\chi = 0.25$. When retrieving learned examples for $\chi = 0.05$ and $\chi = 0.25$ the alternative with the highest correlation always corresponds to the correct result in this simulation. See Figure 7 for details of how averages and standard deviations are calculated.

The error rate is the number of errors divided by the total number of mappings, $N_c N_e N_s$, where $N_e$ is the number of training examples for each particular type of mapping, $N_c$ is the number of classes of different mapping types and $N_s = 5000$ is the number of independent AMU simulations performed to estimate the average error rate. In some cases where the error rate is low we have verified the estimate of the average with higher values of $N_s$. In the simulations presented in Figure 7 the AMU always provides the correct result. In Figure 9 we therefore present only the generalization error rate of the AMU versus the number of training examples. Figure 9 shows that for $\chi = 0.25$ the AMU provides less than 0.01% correct mappings in generalization with less than 30 training examples. For $\chi = 0.25$ the AMU generalizes well with fewer training examples compared to the case with a lower value of $\chi = 0.05$. This is consistent with Figure 8 which suggest that the AMU with $\chi = 0.25$ generalizes with less training examples. This is because a higher $\chi$ results in more overlap between different mapping memories in the SDM. The effect is visible also in Figure 7, where the AMU always provides the correct mapping, even after learning three examples only. Since mapping memories are successively taking shape in different parts of the distributed memory it is reasonable to expect that we can
4. Results

Figure 9: The generalization error rate of the AMU versus the number of training examples, \( N_e \), for \( \chi = 0.05 \) and \( \chi = 0.25 \). The overlap parameter of memory representations, \( \chi \), affects the generalization performance of the AMU. For \( \chi = 0.25 \) and \( N_e = 30 \), the AMU provides less than 0.01% correct mappings and therefore this point is excluded. In Figure 7 all mapping vectors are bundled to one vector, but here each mapping vector is stored in a fraction of the memory. This figure complements the results shown in Figure 8 since the parameters of the AMU are the same.

learn several different mapping relations, which may something else than “above-below” relations.

Given that mappings are successively taking shape in specific parts of the memory, it is reasonable to expect that we can learn other mappings, which do not hold the “above-below” relation and in principle could be any reasonably complex type of mapping. The AMU can learn new mapping vectors and organizes the experience in a simple and effective way. To test this idea we introduce a number of different classes of training examples, \( N_c \). The “above-below” relation considered above is one example of such a class.

An interesting effect appears when varying the number of mapping classes and the size of the memory, see Figure 10. With a sufficiently large memory there is a minimum in the generalization error rate for a certain number of learned mapping classes. This indicates that optimal performance is achieved when the size of the memory used is matched to the number of mapping classes that is learned. When the memory is too small the error rate increases with an increasing number of learned mapping classes. This is to be expected because different mapping memories are superimposed on the few memory locations that are available. Memories representing different mapping classes therefore mix, resulting in an increased noise level in the mapping results. The ability of the memory to generalize depends on the correlation between the output and the correct/incorrect mapping results. The minimum error rate represents the optimal balance between suppressing incorrect mapping generalizations with noise while minimizing destructive interference between memories of different mapping classes. A similar effect appears when varying the overlap between mapping memories, see Figure 11. In that figure we illustrate what happens with
Figure 10: The generalization error rate of the AMU versus the number of classes of different mapping types examples, $N_c$, for $S = 100$ and $S = 1000$. The memory size, $S$, affects the performance of the AMU. Another interesting aspect of this result is that there exists a minimum when $N_c = 3 - 4$ for $S = 1000$. The parameters of the AMU are $D = 1000$, $\chi = 0.05$ and $N_e = 10$. See Figure 7 for details of how averages are calculated.

Figure 11: The error rate of the AMU versus the overlap of memory representations, $\chi$, for retrieval and generalization of mappings. For this choice of parameters, the optimal coding density is $\chi = 0.1 - 0.2$, which indicates that the sparseness of memory representations is important. A similar effect occurs in Figure 9, where sparseness $\chi = 0.25$ gives a lower error rate than $\chi = 0.05$. The parameters of the AMU are $S = 100$, $D = 1000$, $N_e = 10$ and $N_c = 10$.

the worst-case scenario of Figure 10, $S = 100$ and $N_c = 10$, when the overlap parameter, $\chi$, varies. A minimum appears in the error rate for $\chi$ around 0.2, which shows that the “sparseness” of memories is important to achieve a low generalization error rate.

In these simulations we have chosen a dimensionality of the structures $D = 1000$ that is lower than the typical value used for binary representations of hierarchical structures. A higher dimensionality lowers the probability of mistakes and therefore makes it difficult
5. Related work

Research on computational modelling of analogy-making traces back to the classical works by Evans (1964) and Reitman (1966) and since then many approaches have been developed. In this section we introduce some models that either have been influential or are closely related to the work presented here. For more comprehensive survey see French (2002), or the more recent survey by Gentner and Forbus (2011). French (2002), Gentner and Forbus (2011) categorize computational models of analogy as symbolic, connectionist, or symbolic-connectionist hybrids.

The most well-known symbolic model is Structure Mapping Engine (SME) (Falkenhainer et al., 1989) which tends to implement one of the best theories of analogy, called Structure Mapping Theory (SMT) (Gentner, 1983). SMT shifted the emphasis in analogy-making from attributes to structural similarity between the source and target domains. The AMU, like most present models, incorporates the two major principles underlying SMT: relation-matching and systematicity, see Section 4. However, in contrast to SMT and other symbolic models the AMU uses semantic representations, thereby allowing the model to handle the problem of similar but not identical hierarchical struc-

---

Figure 12: The generalization error rate of an AMU with saturated memory versus the dimensionality of the representations, $D$. The analytical result by Kanerva (2000) suggesting that $D = 10,000$ is sufficient for reliable operation remains true for this model. A slightly higher dimensionality may be motivated, but $10^5$ is too much. The parameters of the AMU are $S = 100$, $\chi = 0.05$, $N_e = 10$ and $N_c = 10$.

to estimate the error rate with numerical simulation. The effect of dimensionality is illustrated in Figure 12 with an overpopulated memory that suffers from severe interference noise. By choosing a dimensionality around $10^4$ the probabilities of mistakes reported in this work can be lowered significantly, but that would also make it difficult to simulate the dependence of the model performance and dependence on the three parameters $S$, $\chi$ and $D$.
Satisfying semantic constraints reduces the number of potential correspondence mappings significantly, to a level that is psychologically more plausible (Eliasmith and Thagard, 2001, Gentner and Markman, 2006) and computationally feasible (Gentner and Forbus, 2011).

Connectionist models of analogy-making includes Analogical Constraint Mapping Engine (ACME) (Holyoak and Thagard, 1989) and Learning and Inference with Schemas and Analogies (LISA) (Hummel and Holyoak, 1997). Distributed Representation Analogy Mapper (Drama) (Eliasmith and Thagard, 2001) is a more recent connectionist model based on holographic reduced representations (HRR) and is therefore more similar to the AMU. A detailed performance comparison and assessment in terms of neurological and psychological plausibility between ACME, LISA and Drama is made by Eliasmith and Thagard (2001). Here, we briefly outline and add some new points that were not mentioned by Eliasmith and Thagard (2001), and then we comment on the difference between the AMU and Drama. ACME is sometimes referred to as a connectionist model. However, it is similar to symbolic models. Like the SME, it uses localist representations, but the search mechanism for mappings uses a connectionist approach. ACME includes semantics after structural constraints have been satisfied. In contrast to that the AMU and Drama implements semantics and syntax in parallel allowing both aspects to influence the mapping process. Eliasmith and Thagard (2001) suggest that in comparison to the previous models Drama integrates structure and semantics to a degree that is more in accordance with human cognition. Semantics in ACME are not decomposable. For example, it doesn’t know why “Dollar of Sweden” (Krona) is similar to the “Dollar of Mexico” (Peso). This is possible with the AMU and Drama, since distributed representations allow complex concepts to be explicitly encoded and decomposed (Eliasmith and Thagard, 2001, Neumann, 2002, Plate, 1995, 2003, Kanerva, 2000, 2009). LISA has also been put forward as a connectionist model with distributed representations that uses dynamic binding to associate relevant structures. Like the AMU and Drama, LISA is semantically driven, stochastic, and designed with connectionist principles. However, LISA has a complex architecture that represents propositions in working memory by dynamically binding roles to their fillers, and encoding those bindings in long-term memory. It is unclear if this model scales well and is able to handle complex analogies (Eliasmith and Thagard, 2001, Gentner and Forbus, 2011, Stewart and Eliasmith, 2012).

The third category reported by French (2002) and Gentner and Forbus (2011) is the hybrid approach, incorporating symbolic and connectionist parts. Renowned examples include Copycat (Mitchell, 1993), Tabletop (French, 1995), Metacat (Marshall and Hofstadter, 1997) and AMBR (Kokinov and Petrov, 2001).

In contrast to Drama the AMU it is not based on HRR but simple binary representations, called BSC (Kanerva, 1994). Moreover, instead of operating with individual mapping vectors it incorporates an associative memory, called SDM (Kanerva, 1988) that stores multiple mappings in a simple and coherent way. That is the major contribution presented here, because former work demonstrating analogical mapping with BSC (Kanerva, 2000, 2009) and HRR (Plate, 1995, 2003, Neumann, 2002) are limited to isolated mapping vectors, which makes online learning less transparent.
6 Conclusions

Former work on analogical mapping of hierarchical structures deals with isolated mappings represented in vector form. The aim of this work is to investigate whether that approach to analogical mapping can be extended with an associative memory so that multiple mappings can be learnt and applied in a unified way. We show that this is possible in the case of binary mappings, and we demonstrate the solution on hierarchical structures similar to those considered by others. The suggested model combines existing ideas into a new computational unit, which we call an analogical mapping unit (AMU). The AMU automatically combines mapping vectors of similar training examples so that different types of mappings can be learnt and stored in the same memory. It is based on a sparse distributed memory (SDM) and it operates on high-dimensional binary spatter codes (BSC). The AMU has a one-shot learning process and it is able to recall analogical mappings. After learning many similar mapping examples no specific mapping is recalled but it anyway provides the correct mapping by analogy.

The ability of the AMU to retrieve specific mappings increases with the size of the associative memory. When too many mappings are stored in a memory the different mapping memories interfere and the results are essentially noise. Interference is negligible if the memory is large compared to the number of mapping classes learnt and the AMU recalls accurately in that limit. The ability of the AMU to generalize does not increase monotonously with the size of the memory, but it is optimal when the size of the memory is matched to the number of different mappings learnt. This can be understood qualitatively by a thought experiment; If the memory is too small there is much interference between mapping memories, and the output of the AMU is essentially noise. If the memory is infinitely large there is no interference between mapping memories, meaning that each mapping is stored in unique memory locations not shared with other memories. Without interference between mapping memories the AMU makes perfect recall, but is unable to generalize. Good generalization performance requires a certain level of interference between similar mapping memories. Mapping memories of non-related mappings should ideally not interfere. The AMU is designed to separate memories of different mappings, while the interference between similar mapping memories is specified by two of the three model parameters.

The sparseness of mapping memories is important for the overall performance of the AMU. Technically, sparseness is related to the overlap parameter, $\chi$ (a possible qualitative relation between this parameter and sparseness of neural coding is discussed in Kanerva (1988)). When mapping memories are too sparse the AMU makes perfect recall of known mappings, but it is unable to generalize. A less sparse code results in more interference between mapping memories. This has a negative effect on recall performance, but the effect on the ability of the AMU to generalize is positive. The generalization performance is optimal for a specific choice of the sparseness. The existence of an optimal coding sparseness and effective size of allocated memory per mapping appears to be a general feature of the AMU. However, the location of the optimum in parameter space depends on the structure of mappings that the AMU is learning. The principles of the optimal
analogue mapping performance in this model is thereby linked to the important problem of encoding the hierarchical structures. This is a general problem that needs further research. It is a key enabling step for practical application of this and other models of cognitive function based on hierarchical structures.

A final technical note concerns the representation of mapping vectors in former work versus our mapping memories. Kanerva (2000) used integer vectors for BSC mappings to improve performance. A similar approach is possible with the AMU if the normalization of SDM memories is omitted in (8), and this improves the performance. However, we find that it is sufficient to use binary mapping memories. This is appealing because it enables us to represent mappings in the same form as any other hierarchical structure. Kanerva (2000) estimate that the optimal dimensionality for the hierarchical structures is of order $10^4$ remains true for the AMU. The error rate of the AMU decreases with increasing dimensionality up to that order, and remains practically constant at higher dimensionality.

References


PAPER B

N-way random indexing of arrays

Authors:
Fredrik Sandin, Blerim Emruli and Magnus Sahlgren

Submitted for publication in a journal.
N-way random indexing of arrays: 
a simple method for identifying common relationships in 
large datasets and information streams

Fredrik Sandin, Blerim Emruli and Magnus Sahlgren

Abstract

Random Indexing (RI) is a dimension reduction method that was originally developed for natural language processing to encode vector-semantic relationships from the statistical analysis of word usage in text. We introduce a generalized mathematical formulation of RI that enables N-way RI of multidimensional arrays. This method is an approximate, incremental, scalable, and lightweight dimension reduction method for large non-sparse arrays, and can be used for identifying array elements that have relatively high accumulated values. Therefore, it is suggested that this method is useful for identifying common relationships between features in large data sets and complex information streams. In addition, the presented method is based on approximate and randomly distributed reduced representations of the array elements, and the range of array indices can be extended without modification of the array representation. This is beneficial when the feature space is large and the number of features is unknown. We present simulation results of two-way and ordinary one-way RI that illustrate when the approach is feasible. C++ and Matlab software is provided that supports arrays of dimensionality \( \geq 1 \).

1 Introduction

During the past decade there has been an enormous increase in the annual amount of data produced in fields such as biology, economics, information retrieval, marketing, medicine, physics, political science, sensor networks, and surveillance. At a global scale we have passed the critical point where more data is generated than we can physically store, and this gap widens rapidly. Choosing which data to archive and process, and which to discard is a necessary and common procedure in data-intensive fields and applications. The growing challenge to store and analyze data motivates development of new methods and approaches for data representation and analysis (Science Staff, 2011, Baraniuk, 2011, Hastie et al., 2009).

One basic approach to analyze large datasets is to search for frequently occurring relationships between features. It is a challenging problem because the number of features can be high, and the number of potential pairwise relationships scales like the square of the number of features. If you do not know what relationships to search for, how can you efficiently identify the important ones in a large non-sparse dataset? How do you handle ternary or higher-order relationships, which is a problem that requires substantially more
storage space? Problems of that type naturally appears in applications in the form of context- or time-dependent relationships, but the associated storage and processing requirements are prohibitive.

In this paper we outline an incremental dimension reduction method for non-sparse arrays named N-way Random Indexing (NRI). It enables identification of array elements with high accumulated values, which for example can represent common relationships. This method does not require that the original data is stored and no computationally expensive analysis is needed to insert new data. Array elements are coded in a randomly distributed reduced representation, so that elements with high accumulated values can be approximately decoded and distinguished from elements with low values. It is not necessary that the array is sparse, and it may be fully populated. Traditional tools like principal component analysis (PCA), singular value decomposition (SVD) and similar methods for higher-order arrays such as the Tucker decomposition (Kolda and Bader, 2009) are more precise than the approximate method outlined here. Such methods are, however, computationally complex and therefore of limited use when dealing with large datasets. NRI is incremental, scalable and lightweight. It can therefore be used for online processing of data streams and analysis of large datasets.

The method introduced here is a generalization of Random Indexing (RI), which is a dimension reduction method that originally was developed for semantic analysis in natural language processing (Kanerva et al., 2000). See Sahlgren (2005, 2006) for an introduction. In that context it is used to compress a large word-document or word-context co-occurrence matrix (Turney and Pantel, 2010). Each document or context is associated with a sparse random ternary vector of high dimensionality, a so-called index vector. The high dimensionality of index vectors makes them approximately orthogonal by chance (we return to that point in more detail in the next section). Each word is represented by a high-dimensional vector of integers, a so-called context vector. Context vectors are initially set to zero and for each appearance of a particular word in a document (or context) the index vector of that document is added to the context vector of the word. Words with similar context vectors therefore tend to appear in the same context and have similar semantic meaning. This approach to encode co-occurrence matrices for semantic analysis works surprisingly well (Cohen and Widdows, 2009, Turney and Pantel, 2010, Kanerva, 2009, Sahlgren, 2006). RI has recently been adopted in other applications, such as indexing of literature databases (Vasuki and Cohen, 2010), event detection in blogs (Jurgens and Stevens, 2009) and graph searching for the semantic web (Damjanovic et al., 2010). In general there is an increasing interest for randomization in information processing because it enables simple algorithms that can simplify the interpretation of problems and results, and such algorithms can often be organized to exploit parallel computational architectures in an efficient way (Boyd, 2010, Halko et al., 2011).

In principle the problem solved by RI is the identification of frequent relationships, for example between words and the contexts in which they appear. This task is non-trivial due to the large size of the pair frequency (co-occurrence) matrix in real-world problems. In this paper we generalize RI of vectors to NRI of matrices and higher order arrays, and we present simulation results of two-way and one-way RI to demonstrate that the
Indifference property of high-dimensional ternary vectors

High-dimensional spaces are different from the two- and three-dimensional spaces that we are naturally trained to imagine. In two and three dimensions, randomly generated vectors of equal norm are often similar, e.g., when compared with the dot product. This is not so at high dimensionality where nearly all vectors are unsimilar. Another counterintuitive property is that the volume of the unit hypersphere relative to that of the hypercube with corresponding width rapidly approaches zero at increasing dimensionality. This has profound consequences for the kind of information that can be represented with high-dimensional representations compared to the ordinary “number” concept used in mathematics and computer science. For example, high-dimensional vectors have semantic properties and can represent hierarchical structures (Kanerva, 2009). Such vectors can also be used to create simple mechanisms for associative memory (Kanerva, 1988) and analogical mapping memories (Emruli and Sandin, 2012). The tendency of vectors in high-dimensional spaces to be indifferent, meaning that by chance they are unrelated and nearly orthogonal, is well known in the context of binary vectors (Kanerva, 1988).

We introduce the key points here and then show how to generalize that to ternary vectors \([-1, 0, 1]^n\), which are needed for RI.

Consider the binary space \(\{0, 1\}^n\) of vectors with length \(n\), which have equal probability for the states 0 and 1 in each element. The distance, \(d\), between two binary vectors can be defined as the number of non-zero bits in the bit-wise exclusive or (XOR) of the vectors. This is equivalent to the square of the Euclidean distance and it corresponds to how many bits that are different in two vectors, which is known as the Hamming distance. The number of vectors in the space that are at a distance \(d\) from a specific vector is given by the binomial coefficient

\[
C(n, d) = \binom{n}{d},
\]

because this is the number of different ways to choose (flip) \(d\) bits out of \(n\). The number
of vectors at a certain distance from a reference vector therefore follows the binomial distribution with probability \( p = \frac{1}{2} \), which has mean \( n/2 \) and variance \( n/4 \).

At high values of \( n \) the binomial distribution can be approximated with a normal distribution. If a distribution is approximately normal the proportion within \( z \) standard deviations of the mean is \( \text{erf}(z/\sqrt{2}) \). This implies that the distance distribution is highly concentrated around the mean, because the error function quickly approaches unity for increasing \( z \). For example, 99.7% of the distances are within three standard deviations from the mean distance. Only one billionth \( (10^{-9}) \) of the distances deviate more than six standard deviations from the mean. The mean distance is \( n/2 \) and the standard deviation of distance is \( \sqrt{n}/2 \). This implies that the mean distance is \( \sqrt{n} \) standard deviations, \( e.g., \) 31.6 standard deviations for \( n = 1000 \). A striking consequence of this distribution of distances is that practically all vectors in a high-dimensional binary space are located at a distance \( \sim n/2 \) from any specific vector. For 1000-bit vectors the standard deviation is \( \sqrt{1000}/2 \approx 15.8 \) bits. Six standard deviations correspond to 95 bits. All but one billionth of 1000-bit vectors are located at 500 ± 95 bits from any specific vector in that space. The concentration of distances around the mean increases with \( n \) and it implies that randomly generated high-dimensional vectors are indifferent with high probability. This indifference property is a key idea that underpins several approaches to model cognitive functions (Kanerva, 2009), such as learning, representation of semantics and associative memories, and analogical mapping. Concepts and features represented with arbitrary random high-dimensional patterns are indifferent by chance, but novel associations and mappings can be constructed from existing representations with a learning mechanism.

In this work we are interested in ternary vectors. Instead of bits with two possible states, \( \{0, 1\} \), we consider balanced trits with states \( \{-1, 0, 1\} \). The introduction of a third state with negative sign is crucial because it enables sparse distributed coding of array elements, we will return to that in the next section. This discussion concerns sparse ternary vectors of length \( n \) with \( k \) positive (1) and \( k \) negative (-1) trits, where \( k \ll n/2 \). The ternary space can be visualized as a subset of an inner product space where orthogonality is defined by a vanishing dot product between two vectors. With this definition of orthogonality it follows that an \( n \)-dimensional ternary space has at most \( n \) mutually orthogonal vectors. However, in a high-dimensional space there are many more vectors that are indifferent. This is analogous to the high probability of indifference between vectors in high-dimensional binary space. The total number, \( N \), of ternary vectors of length \( n \) that has \( k \) positive and \( k \) negative elements is

\[
N = \binom{n}{2k} \binom{2k}{k} = \binom{n}{k} \binom{n-k}{k}, \tag{2}
\]

because there are \( C(n, 2k) \) different ways to choose \( 2k \) non-zero trits and \( C(2k, k) \) different ways to distribute the signs to the non-zero trits. The alternative (second) definition above can be interpreted in a similar way: There are \( C(n, k) \) different ways to choose the positive trits and \( C(n-k, k) \) ways to choose the negative trits, or vice versa. How many of these \( N \) vectors are indifferent? The number of vectors with an absolute value of the
dot product, $d = ||\langle \cdot, \cdot \rangle||$, with respect to any reference vector is (see Appendix 6)

$$N(n, k, d) \approx \binom{n - 2k}{2k - d} \binom{2k - d}{k} \binom{k}{k - d}$$

$$\times \, _3F_2(-d, -k, -k; 1 + k - d, 1 + k - d; -1), \quad d \leq k, \ n \gg k, \quad (3)$$

where $\, _3F_2$ is a generalized hypergeometric function (Olver et al., 2010). In the analysis leading to this expression we assume that $0 \leq d \leq k$, because we are interested in approximately orthogonal vectors only, and we assume that the vectors are sparse so that $n \gg k$. If we divide the number of vectors, $N(n, k, d)$, which has a specific value of $d$ with respect to any reference vector, with the total number of vectors in the space, $N$, the result is the relative size of the space as a function of $d$. The relative size of the space is equivalent to the probability of randomly choosing a vector from the space that has a dot product of $\pm d$ with respect to a reference vector,

$$P(n, k; \langle \cdot, \cdot \rangle = \pm d) \approx N^{-1}(n, k, d), \quad d \leq k, \ n \gg k. \quad (4)$$

This function is the result that we need, because it describes the probability that randomly chosen vectors from the space are indifferent. The numbers $N$ and $N(n, k, d)$ are enormous ($n$ is a high number). For practical purposes we therefore make a series expansion of factors involving $n$ in the limit $n \to \infty$. The result is,

$$P(n, k; \langle \cdot, \cdot \rangle = \pm d) \approx \frac{T}{\binom{n}{d}} \sum_{i=0}^{d} \frac{(k!)^4}{[(k - d + i)!][((k - i)!)^2](d - i)!} \cdot \frac{1}{d!} \frac{i^4}{n^i}$$

$$T = 1 - \frac{8k^2 + d^2 + d - 8kd}{2n}$$

$$+ \frac{1}{n^2} \left[ 2(1 - 2k)^2k^2 + \frac{d^4}{8} + \left( \frac{5}{12} - 2k \right) d^3 + \left( 10k^2 - 4k + \frac{3}{8} \right) d^2 \right. \right.$$\left. + \left. \left( -16k^3 + 10k^2 - 2k + \frac{1}{12} \right) d \right] + O(n^{-3}). \quad (6)$$

where the terms in $T$ originate from the series expansion. The assumptions $d \leq k$ and $n \gg k$ are to be respected in applications of this result, which has not been presented elsewhere as far as we know.

The following example is helpful to understand the essential point developed here. Let $n = 10^4$ and $k = 10$, which are typical parameters used (Kanerva et al., 2000, Kanerva, 2009). It then follows from (4) that 96% of the space is orthogonal with respect to any reference vector, and less than 4% of the space has a dot product of $+1$ or $-1$ (see Table 3 in Appendix 6). Only $7 \times 10^{-9}$ of the space has a dot product with a magnitude higher than or equal to four, which corresponds to $\sim 20\%$ non-zero trits in common. With $25\%$ common trits ($d = 5$ and $k = 10$) the relative size of the space is $2 \times 10^{-11}$. Most of the space is approximately orthogonal to any particular vector in the space. Analogously,
the dot products of pairs of vectors that are randomly chosen from the space are given by the probability (4). The probabilities for $n = 10^4$ and some different values of $k$ are illustrated in Figure 1. Note that for a given subset of $N_s$ vectors the distribution of

![Figure 1: Indifference property of high-dimensional ternary vectors $\{−1,0,1\}^n$. The panel on the left-hand side shows the probability (4) for inner products of sparse ternary vectors of length $n = 10^4$ and different numbers of non-zero trits, $2k$. The panel on the right-hand side shows the probability (4) for $k = 10$ and different lengths of the ternary vectors, $n$. In both cases the horizontal scale is normalized to the maximum value of the inner product, which is $2k$. Probabilities for absolute values of $\langle \cdot, \cdot \rangle$ higher than 50% of the maximum are excluded, because (5) is valid for $d \leq k$ only. For $n = 10^4$ and $k = 4$, i.e., ternary vectors of length ten thousand with four positive and four negative trits, the probability that a randomly generated vector has an inner product of four with respect to a reference vector is about $10^{-12}$. The probability of an inner product of minus four is about $10^{-12}$ also. Similarly, for $n = 10^4$ and $k = 12$ the probability of 50% overlap ($\langle \cdot, \cdot \rangle = \pm 12$) is about $10^{-30}$.]

dot products is different, because there are more pairs of vectors than individual vectors in a set. This is similar to birthday type problems, where the probability that at least two people in a room has the same birthday depends on the number of pairs rather than the number of individuals in the room. The number of pairs in a subset of $N_s$ vectors is $C(N_s, 2) = N_s(N_s - 1)/2$. This implies that the number of pairs increases roughly as the square of the number of vectors in the set, and that the number of vectors that can be chosen randomly with a low probability for significant correlation should be reduced with a square root compared to what is suggested by equation (4) and Figure 1. Next we present how sparse ternary vectors can be used for RI of multidimensional arrays and we present simulation results for vectors and matrices that demonstrate the indifference property at work.
3 N-way Random Indexing

In the following we generalize RI to arrays of arbitrary order and present the algorithm for NRI. Array elements are denoted with $a_{ijk...}$ and indices $\{i, j, k,...\}$ are used in element space. Array states are denoted with $s_{\alpha\beta\gamma...}$ and indices $\{\alpha, \beta, \gamma,...\}$ are used in state space. States have physical representations that are stored in memory, but they are accessed by encoder and decoder functions only. Array elements, $a_{ijk...}$, are related to the states and constitute the input to (output from) the encoder (decoder) function. The order of the state array is equivalent to that of $a_{ijk...}$, but the state array may be of significantly smaller size. The basic idea of NRI is that a large array $a_{ijk...}$ can be represented in a small state array $s_{\alpha\beta\gamma...}$ by sparse distributed coding, so that array elements with high accumulated values can be approximately decoded and distinguished from elements with low values.

For each index (dimension) of these arrays there is an associated random-index array, $r_D,i$, where $D$ is a dimension index. For vectors $D = 1$, for matrices $D \in \{1, 2\}$ and so on. If $D$ and $i$ are fixed the state-space elements of $r_D,i$ form a sparse high-dimensional ternary vector, a so-called index vector

$$r_D,i = [\ldots 0001000 \ldots 000-1000\ldots]_{D,i}. \quad (7)$$

Index vectors have a few non-zero elements at random positions $\alpha$, thereby the name “random index”. The non-zero elements of an index vector have an absolute value of one and half of them are negative. In other words, index vectors are sparse ternary vectors, see Section 2 (note that the symbol $k$ in that section is different from the index $k$ defined here). The number of non-zero trits in the index vectors, $\chi_D$, is a model parameter that typically has a value of order ten. We denote the ranges of state indices, $\{\alpha, \beta, \gamma,...\}$, with $[1,n_D]$ so that, e.g., $\alpha \in [1,n_1]$ and $\beta \in [1,n_2]$. Similarly, the ranges of element indices, $\{i, j, k,...\}$, are $[1,N_D]$. The length of an index vector is equivalent to the maximum value of the state index, $n_D$, in each dimension. For example, if the state array of a matrix is of size 1000x2000 the index vectors would be of length 1000 (2000) for $D = 1$ ($D = 2$). Index vectors can be represented in compact form because most elements are zero. Here the indices of non-zero trits are used to represent index vectors, and signs are encoded implicitly with the position of the indices so that the first half are positive. The number of non-zero trits in an index vector is denoted with $\chi_D$, which is an even number. For each dimension, $D$, there are $N_D$ index vectors of length $n_D$ and each index vector has $\chi_D$ non-zero trits. In practical applications an index vector is represented in compact form by at most a few dozen integers, so the storage space required for an NRI representation is essentially determined by the size of the state array. A summary of parameters and definitions is presented in Table 1.

3.1 Encoding algorithm

State elements, $s_{\alpha\beta\gamma...}$, are initially set to zero. This implies that the array elements $a_{ijk...}$ are zero also (see decoding). Array elements are updated with addition and subtraction
Table 1: Summary of parameters.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{ijk...}$</td>
<td>Array elements</td>
</tr>
<tr>
<td>$s_{\alpha\beta\gamma...}$</td>
<td>State array, accessed by encoder/decoder functions only</td>
</tr>
<tr>
<td>$D$</td>
<td>Dimension index ($1 \leq D \leq$ dimensionality of array)</td>
</tr>
<tr>
<td>$N_D$</td>
<td>Number of index vectors in dimension $D$, ${i, j, k,...} \in [1, N_D]$</td>
</tr>
<tr>
<td>$n_D$</td>
<td>Length of index vectors in dimension $D$, ${\alpha, \beta, \gamma,...} \in [1, n_D]$</td>
</tr>
<tr>
<td>$\chi_D$</td>
<td>Number of non-zero trits in index vectors for dimension $D$</td>
</tr>
<tr>
<td>$\mathcal{N} = \prod D \chi_D$</td>
<td>Number of states that encode one array element</td>
</tr>
<tr>
<td>$\propto \prod_D n_D \chi_D$</td>
<td>Disk/memory space required to store the state array</td>
</tr>
<tr>
<td>$\propto \sum_D N_D \chi_D$</td>
<td>Disk/memory space required to store index vectors</td>
</tr>
</tbody>
</table>

operations only, not by assignment. An array element $a_{ijk...}$ is encoded in the state array $s_{\alpha\beta\gamma...}$ using the index vectors. The addition of a scalar weight $w$ to an array element $a_{ijk...}$ is defined as

$$s'_{\alpha\beta\gamma...} = s_{\alpha\beta\gamma...} + w(r_1,i_1 r_2,j_2 r_3,k_3 \ldots),$$  

where indices $\{i, j, k,\ldots\}$ are determined by the choice of array element and $s'_{\alpha\beta\gamma...}$ denotes the resulting state array. This means that the indices of the array element are used to select a particular set of index vectors, forming an outer product of indifferent index vectors in state space. The product of index vectors is an array with $\mathcal{N}$ non-zero elements with values $+1$ and $-1$. It has the same order and size as the state array. Subtraction of $w$ is defined by the replacement $w \rightarrow -w$ in (8). Assignment of array elements is not defined because the representation is distributed and states usually represent multiple array elements. It is the accumulated values of the states corresponding to each particular array element with associated index vectors that hold useful information. The encoding process is illustrated in Figure 2.

3.2 Decoding algorithm

The decoding operation is the projection of the state array on the index vectors that correspond to the array element $a_{ijk...}$

$$a_{ijk...} = \mathcal{N}^{-1} \sum_{\alpha,\beta,\gamma...} r_1,i_1 r_2,j_2 r_3,k_3 \ldots \ s_{\alpha\beta\gamma...},$$  

where $\mathcal{N}$ is a normalization factor that is defined below. The encoding procedure (8) is based on a sequence of outer products of index vectors, and the decoding procedure is the corresponding sequence of inner products. It follows from (8) and (9) that the decoded value would be an exact reconstruction of the accumulated encoded weight if all index vectors are orthogonal. However, that would be quite useless since no dimension reduction can be achieved that way. For index vectors of length $n_D$, at most $n_D$ linearly independent vectors can be constructed (a set of basis vectors). Equation (4) and Figure 1 illustrates
Figure 2: Encoding of one array element with two-way Random Indexing. In this example a value of 5 is added to the array element $a_{ij}$ of a two-dimensional array of size $N_1 \times N_2$. The highlighted two-dimensional array is accessible to the user via encoder and decoder functions, while the state array and the index vectors have physical representations that are stored in memory / on disk. The three-step encoding procedure is illustrated in the shaded area. First the indices $i$ and $j$ of the element are used to select the appropriate index vectors for each dimension of the array. Index vectors are sparse high-dimensional ternary vectors, which are represented in compact form as the indices of non-zero elements. Signs are represented by the index position, so that the first (last) half correspond to positive (negative) trits. The number of non-zero elements in an index vector is typically of order ten, but are set to four and two here to make the illustration comprehensible. The second step is to activate the subset of elements in the state array that represents the value of $a_{ij}$. Note that the activated rows and columns correspond to the indices of non-zero trits in the index vectors. The third and last step is to add (subtract) 5 to all selected states with an associated positive (negative) product of non-zero trits. This three-step encoding process corresponds to the addition of the state array with a weighted outer product of index vectors (8).

that for large $np$ there are many more vectors that are approximately orthogonal. This makes it possible to approximately encode and decode array elements in a distributed
representation in the smaller state array. The result is that array elements $a_{ijk...}$ with high accumulated weights can be identified, while array elements with low accumulated weights disappear in noise. A practical implementation of the decoding function may therefore return a top-list of elements $a_{ijk...}$ with high values, for given values of some of the indices \(\{i, j, k, \ldots\}\). These high-value elements can for example represent frequently occurring relationships. We return to the details of this approach in the next section, which deals with simulation results.

The normalization factor in (9) compensates for the sum over states. An index vector in dimension $D$ has $\chi_D$ non-zero trits. The normalization factor therefore is

$$\mathcal{N} = \prod_D \chi_D.$$ (10)

For example, the two-dimensional array in Figure 2 has $\mathcal{N} = 4 \times 2 = 8$, because the index vectors have, respectively, four and two non-zero trits. The quantity $\mathcal{N}$ is a measure of the computational complexity of the encoding and decoding processes, because $\mathcal{N}$ states are accessed when encoding or decoding the value of one array element.

### 3.3 Random indexing of vectors

The RI approach used in natural language processing is based on vectors (Kanerva et al., 2000, Sahlgren, 2005). We return to this special case in Section 4. In the traditional RI algorithm, each term that appears in a text corpus is associated with a context vector, and each context or document is associated with a ternary index vector. A context vector therefore corresponds to the states of a one-dimensional NRI array, and the index vectors are the ternary index vectors of that array. It would be impractical to construct one state array with associated index vectors for each term / item that is to be represented. One-way RI can therefore be implemented as a set of one-dimensional arrays that share the same set of index vectors. For arrays of order $\geq 1$ the relative size of the index vectors, $(\Sigma N_D \chi_D)/(\Pi n_D)$, is low and a special implementation of that type is not necessary.

### 3.4 Comments

NRI is an approximate linear dimension reduction method that make use of the indifference property of randomly generated high-dimensional vectors (4). It is therefore to be used with large-size arrays only ($n_D > 10^3$), see Section 5 for information about methods that are suitable for smaller arrays. Quantitative details are lost in the encoding process, but qualitative features can be represented and identified. This point is demonstrated with simulation results in the next section.

We mentioned in Section 2 that the co-existence of positive and negative states is essential, and that index vectors should be ternary and not binary. The reason is that, on average, states are about equally likely to be incremented and decremented when random or otherwise non-systematic data is encoded. The equal numbers of positive and negative non-zero trits in the index vectors effectively prevents saturation of the state
Figure 3: Decoding of an element of a two-dimensional array with the random-index method. The process illustrated here is the inverse of that in Figure 2. Refer to that figure for further details. The first step is to select the appropriate index vectors using the indices \( i \) and \( j \) of the array element that is to be decoded. The index vectors are then used to activate the subset of states that encode the value of \( a_{ij} \). Note that the signs of the activated rows and columns correspond to the signs of the indices in the index vectors. The third and last step is to sum the activated states taking the overall signs into account, and to divide the sum with the normalization constant (10). Note that the decoded value is similar to that encoded in Figure 2. The difference (5.25 vs 5.00) is a consequence of the non-zero initial states, which are caused by preceding updates of array elements. This three-step process is mathematically equivalent to (9).
The order of element updates does not matter, because repeated use of the encoding operation (8) corresponds to a commutative sum. To break this symmetry, for example to represent time evolution or sequential order, one has to encode that degree of freedom explicitly. Temporal Random Indexing (Jurgens and Stevens, 2009) is an example that is used to analyze the time evolution of word semantics to detect novel events in online texts.

From a historical point of view RI is a cognitive computation method, which was inspired by the structure of circuits in the cerebellum and cortex. The state array is analogous to the counters in Sparse Distributed Memory, which represents the synapse weights of an associative memory, see Chapter 4 in (Kanerva, 1988). RI and NRI is based on sparse distributed coding of information, which is another biologically inspired principle. Next, we present simulation results for one- and two-way RI.

4 Simulation results
Randomly generated index vectors are indifferent by chance, but they are not orthogonal. Different array elements are encoded in partially overlapping subsets of states and therefore interfere with each-other to some degree. The magnitude of this effect depends on several aspects, such as the: Order of NRI; Array dimensionality and length of the index vectors, \( n_D \); Number of non-zero trits in the index vectors, \( \chi_D \); Dimension reduction, \( \Pi_D N_D \); \( \Pi_D n_D \); And the characteristics of the data that is encoded. In general we find that arrays with relatively few high-value elements of comparable magnitude are well represented, while arrays with many or significantly varying high-value elements are poorly represented with NRI. The relative magnitude of array elements therefore plays an important role for the usefulness of the representation. It is, however, not necessary that the array is sparse. A full array can be well represented if it has relatively few high-value elements that encode relevant information. This is useful if data is accumulated over time and the subset of significant elements needs to be identified.

In this section we illustrate these aspects with simulation results and present a simple application of NRI to natural language processing. The software used to perform the simulations is available online (Sandin, 2011) and it includes a C++ template that implements the N-way RI algorithm. The template has user-defined data types for the state array and index vectors, and a mechanism to monitor state saturation. It includes also a Matlab interface to the C++ template, which is convenient for small-scale experiments.

4.1 Finding relationships with random indexing
We consider two generic examples where a non-sparse matrix is represented with one-way and two-way RI. Matrices can be represented with one-way RI if each column or row is treated as a vector, which is the traditional RI approach. Each column of the matrix represents a class, and each row represents a feature that the classes may have. In principle the array elements could represent other types of relationships, but we use this terminology for simplicity. Each element of the matrix is initialized with a random integer.
drawn from the flat distribution \([0,10]\). This represents a background of insignificant features of the classes (noise) that make the matrix non-sparse. A relatively low number of significant features are then added to each class, which are represented by high-value matrix elements.

The difference between the significant elements and the background can be translated into a signal-to-noise ratio (SNR). It can be expressed in the root-mean-square (RMS) amplitudes of signal and noise, \(\text{SNR} = (A_{\text{signal}}/A_{\text{noise}})^2\). We take the number of features to be proportional to the size of the matrix, \(N_D\), and define the constant of proportionality as \(\rho\). The number of features in each class then is \(\rho N_D\), and each feature is represented by an element with value \(w\). The RMS amplitude of the signal is \(\sqrt{\rho N_D w^2 / N_D} = \sqrt{\rho w}\). The RMS amplitude of a uniform random distribution on the interval \([0,M]\) is \(\sqrt{M(2M+1)/6}\). It then follows that the SNR can be expressed in \(\rho, w\) and \(M\) in the following way

\[
\text{SNR} = 10 \log \left( \frac{A_{\text{signal}}}{A_{\text{noise}}} \right)^2 \text{dB} = 10 \log \left( \frac{6\rho w^2}{M(2M+1)} \right) \text{dB}. \tag{11}
\]

For example, the SNR would be 5 dB if there are one percent features, \(\rho = 0.01\), with element values \(w = 100\). The SNR of decoded features is lower, because the RI representation is approximate. Note that a high SNR does not automatically imply that the data can be accurately represented with NRI. A high value of \(\rho\) does for example make the representation inaccurate. That point is discussed above and demonstrated below.

In the first numerical example we consider a matrix of size \(10,000 \times 10,000\) that is encoded with two-way RI in \(5,000 \times 5,000\) states. This implies that the dimension reduction is 4:1 and the index vectors have length \(n_D = 5,000\). Unless stated otherwise we will use index vectors with four positive and four negative trits so that \(\chi_D = 8\). The motivation for this choice is given below. For each of the 10,000 classes we add high values, \(w\), to 50 randomly selected elements that represent features of the classes. These features are given a weight of \(w = 100\) in one case, and \(w = 1,000\) in a second case. How many of these fifty features of each class can be identified?

To answer this question we extract the fifty matrix elements of each class that have the highest decoded values and compare these with the features that were encoded in the matrix. The result of that simulation is presented in Figure 4, which includes the decoded values of the one hundred largest elements. The answer to the question posed above is that \(39 \pm 6.4\) (46 \pm 1.6) of the fifty features are represented among the top-fifty matrix elements of each class when the weight of the features is 100 (1,000). These numbers are somewhat higher if the matrix is encoded with one-way RI and the standard deviation of the decoded values is lower in that case also, we return to that below.

Some things can be learned from the simple exercise presented in Figure 4: For a fixed number of features with identical weights, a higher weight tends to increase the probability that the features are correctly identified; The standard deviations of the number of correctly identified features and the decoded weights decrease when the weight of the features increases; The decoded weights of the top-fifty elements is distributed around the encoded weight and can deviate from it by at least \(\pm 50\) percent; The transition
Figure 4: Decoded element values of a matrix encoded with two-way RI. In this example a 10,000 × 10,000 matrix is encoded in 5,000 × 5,000 states, which means that the dimension reduction is 4:1. Each column of the matrix represents a class and each row represents a feature that the classes may have. The non-sparse matrix is initialized with random numbers in the range [0, 10]. Fifty elements are randomly selected in each class to represent features of that class. In the figure on the (left-) right-hand side these features have element values of (100) 1,000, which corresponds to a signal to noise ratio of about (1.5 dB) 21 dB. Solid curves illustrate the average decoded value, which has been normalized with the value of encoded features (100 and 1,000, respectively). The horizontal axis is a counter of the first one hundred high-value elements returned by the decoding function. The shaded area represents the standard deviation of the decoded values for the ten thousand different classes. The data points with horizontal error bars represent the average and standard deviation of the number of correctly identified features within the first fifty elements. With a weight of 100 (1,000) of the encoded features the number of correct features out of 50 is 39 ± 6.4 (46 ± 1.6).

from features to noise at an index of 50 is more distinct with the higher feature weight.

The relative number of features also has a significant effect on the accuracy of the representation and the transition from features to noise in the top-list of decoded matrix elements. This point is illustrated in Figure 5, which corresponds to the panel on the left-hand side in Figure 4 with the modification that $p = 0.001$ so that there are only 0.1 percent features in each class. A formal analysis of the fidelity of RI-encoded array elements has so far not been carried out. The interested reader is referred to (Kanerva, 1988) for a similar discussion in the context of Sparse Distributed Memory. This is an interesting issue for further investigation, because it is related to the question of how to select an optimal length of the list of decoded high-value elements when the number of features is unknown. Our results provide two guiding principles. First, the reliability of the high-value elements tend to decrease with their index in the descending list. Second, when the relative number of features is low and the relative weight of features is high the decoded weights of features is separated from the lower weights of noise in a distinct transition.
4. Simulation results

Figure 5: Example of a distinct transition between features and noise represented by matrix elements encoded with two-way RI. The case illustrated here is identical to the panel on the left-hand side of Figure 4, with the difference that the number of features in each class is ten instead of fifty. In this case $9.2 \pm 1.6$ features out of the ten in each class are correctly identified. This is better than the result presented in Figure 4 and it illustrates that the relative number of features in each class is an important quantity that affects the accuracy of the representation. The matrix can be non-sparse, but the high-value elements that encode relevant features must be sparse.

4.2 Comparison of one-way and two-way random indexing

Next we turn to a comparison of one-way and two-way RI, and the effect of varying the dimensionality of index vectors. The approach here is identical to that above, meaning that we consider matrices that encode features of classes. We vary the relative number of features, $\rho$, from 0.1 to 10 percent of the size of the matrix, $N_D$. If $\rho N_D$ features are encoded in a class, then a top-list of equally many decoded high-value elements is used. The number of features in that top-list determines the number of correct features retrieved. This is analogous to the analysis leading to the data points with error bars in Figure 4. The dimension reduction, $\Pi D N_D : \Pi D n_D$, is 4:1 in this example also, but the size of the matrix is varied so that the dimensionality of the index vectors varies.

The analysis of indifference in Section 2 intuitively suggests that the relative number of correctly decoded features could increase with dimensionality. We find that the mean relative number of correctly decoded features is practically independent of dimensionality. It is the standard deviation of the number of correctly decoded features that decreases with increasing dimensionality. Provided that the dimension reduction is kept fixed and that the number of encoded features is proportional to the size of the matrix, the effect of increasing the size of the matrix and thereby the dimensionality of index vectors is a reduction in the uncertainty of the number of correctly decoded features. This point is illustrated numerically in Figure 6, and it is valid as long as the dimensionality is sufficiently high. It is not valid for low-dimensional index vectors, $n_D \lesssim 1000$, because
Figure 6: The number of correctly decoded features represented by a matrix encoded with one-way and two-way RI, see Figure 4 and the text for further information. The horizontal axis represents the relative number of encoded features, $\rho$, and the vertical axis of the panel on the (right-) left-hand side represents the average (standard deviation) of the relative number of correctly decoded features. The average is practically independent of the size of the matrix and the dimensionality of the index vectors provided that the dimension reduction is kept constant, which is the case here. The standard deviation of the number of correctly decoded features decreases with increasing dimensionality of the index vectors. The shaded areas in the panel on the left-hand side illustrate the standard deviations for two different dimensionalities of the index vectors. In the case of one-way (two-way) RI the higher standard deviation corresponds to a $5,000 \times 5,000$ matrix encoded in $1,250 \times 5,000$ (2,500 $\times$ 2,500) states, while the lower standard deviation corresponds to a $10,000 \times 10,000$ matrix encoded in $2,500 \times 10,000$ (5,000 $\times$ 5,000) states. The results presented in the panel on the right-hand side corresponds to these dimensionalities also, and it includes an additional result for a $20,000 \times 20,000$ matrix that is encoded with the same dimension reduction of 4:1. The higher dimensionality of the index vectors in the $20,000 \times 20,000$ case gives a lower standard deviation compared to the other two cases.

in that domain the average performance decreases. Note that the relative number of correctly decoded features first decreases with an increasing number of encoded features, as expected, but then starts to increase somewhat for $\geq 8$ percent features in the case of two-way RI. This effect is caused by the increasing probability of obtaining features in the top-list by chance when the relative number of features and the relative length of the top-list increases. In the case of one-way RI the standard deviation has a maximum around 0.7–0.9 percent features, this feature is robust and may be related to a similar tradeoff.

4.3 Dependence on dimension reduction ratio

Up to this point the dimension reduction is kept fixed at 4:1. How does the dimension reduction ratio affect the performance of NRI? We address this question with an example
that is analogous to those considered above with the modification that the dimension reduction, \( \Pi_D N_D : \Pi_D n_D \), is varied from 4:1 to 64:1. The size of the matrix, \( N_D \), is kept constant when varying the dimension reduction ratio, which implies that the number of features encoded in the classes is constant also. The result of this simulation is presented in Figure 7. A surprising result of this simulation is that the performance of one-way and two-way RI is comparable at a dimension reduction of 64:1. The nature of this trend is related to an interesting scaling phenomena that we sketch here. Assume that a square matrix of size \( N \times N \) is encoded with one-way (two-way) RI in a state of size \( n \times N \) \((n \times n)\). The dimension reduction ratio, \( \xi \), then is \( \xi = N/n \) \((\xi = N^2/n^2)\) for one-way (two-way) RI. Solving for the dimensionality of the state matrix and the index vectors we get \( n = N/\xi \) for one-way RI and \( n = N/\sqrt{\xi} \) for two-way RI. For any order, \( r \), of NRI the dimensionality of the index vectors scales as \( n = N^{1/r} \). The implication of this scaling relation for dimension reduction of high-order arrays with NRI will be investigated in a future publication.

Figure 7: Effect of the dimension reduction, \( \Pi_D N_D : \Pi_D n_D \), on the relative number of correctly decoded features. The panel on the left-hand side shows the average relative number of correctly decoded features, which is practically independent of matrix size as long as the dimensionality of the index vectors is sufficiently high, see the text for further information. The panel on the right-hand side shows the signal to noise ratio, defined as the average relative number of correctly decoded features, \( \mu \), divided by the corresponding standard deviation, \( \sigma \). Note that this signal to noise ratio refers to the decoded information, while (11) refers to the features encoded with NRI. The size of the matrix, \( N_D \), is taken to be \( 64,000 \times 64,000 \) for both one-way and two-way RI. At the maximum dimension reduction of 64:1 this matrix is encoded in \( 1,000 \times 64,000 \) \((8,000 \times 8,000)\) states with one-way (two-way) RI. The large size of the matrix is necessary to maintain high dimensionality of the index vectors for one-way RI at high dimension reduction. The effect of an increasing dimension reduction ratio on the dimensionality of index vectors is lower for two-way RI than for one-way RI, and is increasingly weaker at higher order.
4.4 Importance of sparse index vectors

Next we investigate how these results depend on the number of non-zero trits, $\chi_D$, in the index vectors. Up to this point we keep this parameter fixed at $\chi_D = 8$, which means that index vectors have four positive and four negative trits. This is a good compromise, because the performance in the examples considered here increases insignificantly at higher values of $\chi_D$. More importantly, the computational cost increases with increasing $\chi_D$, because the number of states that needs to be accessed each time an array element is encoded or decoded is $\Pi_D \chi_D$. In Figure 8 we illustrate how the average relative number of correctly decoded features varies for different values of $\chi_D$ and the relative number of encoded features, $\rho$. Illustrated in that figure is also the corresponding standard dev-

![Figure 8: The average number of correctly decoded features and the corresponding standard deviation for different numbers of non-zero trits in the index vectors, $\chi_D$, and different relative number of encoded features, $\rho \in \{0.1, 0.5, 1, 2, 4, 6, 8, 10\}$ percent. The matrix considered here has size $5000 \times 5000$ and it is encoded with one-way and two-way RI such that the dimension reduction is 4:1. Results for $\chi_D = 8$ are identical to those presented in Figure 6. Refer to that figure and the related text for further information about the $\rho$-dependence of the quantities plotted here. The optimal choice for the sparseness of index vectors is $\chi_D \sim 8$ because the performance increases insignificantly with higher values, but the computational cost increases significantly.](image-url)
4. Simulation results

viation. This result is the motivation for our choice of $\chi_D = 8$ as the best compromise between performance and computational cost. In two-way RI the number of states associated with one matrix element is $\chi_1 \times \chi_2$, so there is practically a quadratic dependence of the computational cost on the number of non-zero trits in the index vectors, $\chi_D$. This is why we use $\chi_D = 8$ in most simulations presented in this section.

4.5 Analysis of semantic similarity – a natural language processing example

Next we apply this methodology to a basic task in natural language processing to illustrate how the RI approach can be generalized to NRI. In statistical models of natural language it is common to construct a large word–document or word–context matrix, a so-called co-occurrence matrix Turney and Pantel (2010). This is for example the case in the Hyperspace Analogue to Language (HAL) (Lund et al., 1995, Lund and Burgess, 1996) and in Latent Semantic Analysis (LSA) (Landauer and Dutnais, 1997), which are two pioneering models in the field. In practical applications the number of words can be hundreds of thousands. The number of documents or contexts is necessarily high also, otherwise the statistical basis is insufficient for analysis. Word co-occurrence matrices therefore tend to be large objects. The relatively simple example considered below uses more than 5 billion matrix elements representing word–context relationships. Fortunately, co-occurrence matrices can be compressed to make the semantic analysis more efficient. It was demonstrated by Kanerva et al. (2000) that one-way RI can be used to effectively encode co-occurrence matrices for semantic analysis, see also Sahlgren (2005, 2006), Kanerva (2009).

The definition of “context” is model specific, but it typically involves a set of words or a document. In HAL the context is defined by a number of words that immediately surround a given word, while in LSA the context is defined as the document where the word exists. Linguistically, the former relation can be described as a paradigmatic (semantic) relation, while the latter can be characterized as an associative (topical) relation. In the traditional RI algorithm, each word type that appears in the data is associated with a context vector, and each context is associated with a ternary index vector. If the context is defined in terms of the neighboring words of a given word, which is the strategy that we adopt here, context vectors are created by addition of index vectors (that can be weighted differently) of the nearest preceding and succeeding words every time a word occurs in the data (Karlgren and Magnus, 2001). If the context is defined as the document where the word exists, context vectors are created by adding the index vectors of all documents where a word occurs, weighted by the frequency of the word in each document. In either case a context vector is the sum of weighted index vectors of all contexts where that word occurs. The RI algorithm has traditionally been evaluated using various kinds of vocabulary tests, such as the synonymy part of the “Test of English as a Foreign Language” (TOEFL) (Kanerva et al., 2000, Sahlgren, 2006).

In the following we briefly reconsider the synonym identification task presented by Kanerva et al. (2000) with three changes. First, we want to demonstrate the use of
one-way and two-way RI, so we encode the co-occurrence matrix with both methods. Second, while Kanerva et al. (2000) used the LSA definition of context, we use a strategy similar to that in HAL and define the context as a window that spans ±2 words away from the word itself. This means that for each occurrence of a word there will be four additional word-word relationships encoded in the co-occurrence matrix. This strategy avoids the potential difficulty of defining document boundaries in online text, and it captures semantic relations between words rather than topical relations. The length of the context window is a parameter that affects the quantitative results presented here, but it is not essential in our qualitative discussion. The third and last difference compared to the study by Kanerva et al. (2000) is that we do not introduce cutoffs on term frequencies to further improve the result. Frequently occurring words like “the”, “at” and “be” have high frequencies that render the occurrences of more interesting combinations less significant. This effect is stronger for two-way RI than for one-way RI, because in two-way RI the interference between array elements is higher. The significance of this effect should increase with array order and we therefore illustrate it here. We include the complete word-context spectrum, including the high-frequency relationships, and we present results for two different transformations of the spectrum. In one case we encode the unaltered frequencies directly, and in the other case we encode the square root of the frequencies. The square root decreases the significance of high frequencies and improves the result. The results can be further improved with a more careful choice of preprocessing method, for example by introducing weighted context windows and cutoffs on the encoded relationship frequencies (Karlgren and Magnus, 2001). Another example is the preprocessing method used in LSA, where the frequencies are transformed with a logarithm and then divided with the conditional entropy of the context given that the word has occurred (Landauer and Dumais, 1997). Our aim here is not to achieve an optimal result, but rather to demonstrate the generalization of one-way RI to two-way NRI with a practical example.

We construct the co-occurrence matrix from 37,620 short high-school level articles in the TASA (Touchstone Applied Science Associates, Inc.) corpus. The text has been morphologically normalized so that each word appears in base form (Karlgren and Magnus, 2001). It contains about 74,200 word types that are encoded in an co-occurrence matrix with the NRI software (Sandin, 2011). For one-way RI we use index vectors of length 1,000, so that the dimension reduction is \( \sim 74,200 \times 74,200 : 1,000 \times 74,200 \rightarrow 74 : 1 \). In the case of two-way RI we use a state array of size \( 1,000 \times 74,200 \), thereby maintaining the same dimension reduction ratio. We have repeated the two-way RI calculations with a square state of size \( 8,600 \times 8,600 \) with similar results. There are plenty of misspellings (low-frequency words) in the corpus and the most frequent word is “the”, which occurs nearly 740,000 times. At second place is “be” with just over 420,000 occurrences. The task consists of eighty TOEFL synonym tests, which contains five words each. One example of a synonym test considered here is presented in Table 2. One out of the five words in each test is given and the task is to identify the synonym of that word among the other four words. There is only one correct synonym in each case, and consequently three incorrect alternatives.
4. Simulation results

Table 2: Example of a TOEFL synonym test. The first word is given and the task is to determine which of the four remaining words that is a synonym of that word. Illustrated are also the number of occurrences of each word in the TASA (Touchstone Applied Science Associates, Inc.) corpus.

<table>
<thead>
<tr>
<th>Word</th>
<th>Number of occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>essential (given)</td>
<td>855</td>
</tr>
<tr>
<td>basic</td>
<td>1920</td>
</tr>
<tr>
<td>ordinary</td>
<td>837</td>
</tr>
<tr>
<td>eager</td>
<td>480</td>
</tr>
<tr>
<td>possible</td>
<td>3348</td>
</tr>
</tbody>
</table>

The task to identify the correct synonym is addressed with the NRI-encoded co-occurrence matrix in the following way. First, the word–word frequencies / weights of the five word pairs in each synonym test are decoded and sorted in descending order (using the function “find” in Sandin (2011)). This means that the first (last) word has the relationship with the highest (lowest) frequency and therefore is most (least) significant. In practice the interference between array elements makes this top-list an approximate one, roughly with decreasing accuracy further down the list. We therefore select a subset of the high-frequency word relationships and base the similarity tests on this subset, omitting other word–word relationships with lower frequencies. The set of high-frequency words that relate to the given (query) word of a synonym test is denoted with $W_0$, and the sets of high-frequency words that are related to the four alternative answers are denoted with $W_1$, $W_2$, $W_3$ and $W_4$. The similarity of two words is measured with the Jaccard index,

$$J(W_0, W_i) = \frac{|W_0 \cap W_i|}{|W_0 \cup W_i|}, \quad i \in [1, 4].$$

This means that two words that share many similar word–word relationships are considered to be similar. The synonym is identified as the pair of words with the highest value of the Jaccard index out of the four calculated values. We repeat each synonym test ten times with different index vectors and calculate the average success rate and the standard deviation. The result is presented in Figure 9.

The difference between one-way and two-way RI demonstrates that the feature space in this particular problem is non-sparse, see Figure 6. This difference is partially to be expected, because the interference between array elements is higher in two-way RI and we include high-frequency words and relationships in this analysis. The performance of two-way RI can be increased by introducing a more discriminating definition of semantic similarity, so that the population of high-weight relationships in the co-occurrence matrix becomes more sparse. These results can also be further improved by introducing cutoffs on the relationship frequencies (Karlgren and Magnus, 2001). One benefit of two-way RI is that words can be defined incrementally with a minimum impact on the storage space needed. For comparison we calculate also the success rate using the traditional
Figure 9: Performance of different NRI techniques on TOEFL (Test Of English as a Foreign Language) synonym tests. Illustrated here is the number of correctly identified synonyms vs. the size of the word–word relationship sets used to calculate the Jaccard index (12). The co-occurrence matrix is encoded with NRI and includes 37,620 short high-school level articles in the TASA (Touchstone Applied Science Associates, Inc.) corpus. Each TOEFL synonym test consists of five words, out of which two are synonyms. The task is to identify the correct synonym of a given word in a set of four alternative answers. These results are based on 80 synonym tests, each comprising five words. Each test were repeated ten times and the symbols represent the average results of these ten simulations. Error bars represent standard deviations. When performing the synonym similarity test in the traditional way using one-way RI and the cosine of angles between full-length context vectors the result is 47 ± 2.4 percent correctly identified synonyms, which increases to 51 ± 3.7 percent when the square root of word frequencies is used. This is slightly lower than the best results obtained with the Jaccard index test invented here, which are 49 ± 1.7 and 54 ± 3.9 percent, respectively. Note that an average score of 25 percent can be achieved by random guesses, but that approach would imply that about 75 percent of the answers are incorrect. The NRI approach typically provides a correct answer or an inconclusive result, but rarely an incorrect answer when the word–word relationship list is a few hundred elements long or more.

one-way RI approach where word similarity is determined with the cosine of angle between full-length context vectors (Kanerva et al., 2000, Sahlgren, 2005). The best result obtained with the Jaccard index comparison is 54 ± 3.9 percent correct synonyms, and the corresponding cosine result is 51 ± 3.7 percent. This is an expected result, because the full-length context vectors include noise that to some extent is excluded in the high-frequency sets used in the Jaccard index test introduced here. The Jaccard index test of semantic similarity can also be generalized to higher-order NRI, which is not the case with the cosine of angle test. It is, however, not clear how to find the optimal length of
the top-list and we expect that it varies between problems. One possibility is to develop a statistical model of the top-list, for example by explaining the numerical result illustrated in Figure 5.

## 5 Related work

This work was stimulated by Pentti Kanerva's work on sparse distributed memory (Kanerva, 1988) and a number of papers related to neuro-symbolic hyperdimensional computing, see Kanerva (2009) for an introduction. In particular it is a generalization of the RI method developed for natural language processing (Kanerva et al., 2000, Sahlgren, 2005), see also p. 153 in Kanerva (2009) where the possibility to extend RI to two dimensions is mentioned. Latent Semantic Analysis (LSA) (Deerwester et al., 1990) and the Hyperspace Analogue to Language (HAL) (Lund and Burgess, 1996) are two pioneering and successful vector space models (Turney and Pantel, 2010) used for semantic analysis of text. In these methods a co-occurrence matrix is explicitly constructed and then Singular Value Decomposition (SVD) is used to identify semantic relationships between terms, see Bullinaria and Levy (2012) for recent examples. This process requires a large storage space for the full co-occurrence matrix and it is computationally costly. The SVD can be calculated with parallel and iterative methods optimized for sparse matrices (Berry et al., 2003), but the computational cost still prevents researchers to process large datasets with these methods (Cohen and Widdows, 2009). Approximate and efficient methods for representation and analysis of (semantic) relationships are therefore needed.

RI is a recent and promising alternative that is computationally lightweight. It avoids the construction of a full co-occurrence matrix by direct coding of vectors with reduced dimensionality. RI requires a fraction of the memory and processing power of LSA and HAL (Cohen and Widdows, 2009), but is comparable with VSM's based on SVD in terms of accuracy. Another approximate method is Locality Sensitive Hashing (LSH) (Broder, 1997). Gorman and Curran (2006) presents a detailed comparison of RI and LSH on a distributional similarity task. They find that RI outperforms LSH both in efficiency and accuracy when the size of the dataset increases. The performance of RI is comparable to SVD-based methods in a TOEFL synonym identification test (Kanerva et al., 2000), and this result can be further improved with RI (Sahlgren et al., 2008). The improvement presented in Sahlgren et al. (2008) is important in the context of semantic analysis of text, because it enables representation of linguistic structure that is not captured by a traditional “bag-of-words” analysis (Mitchell and Lapata, 2010). Dominic Widdows and Trevor Cohen are recent proponents of the RI method who suggest that none of the present methods perform as well as RI in terms of: Scaling, RI scales comfortably to large corpora such as the MEDLINE collection of around 9 million abstracts (Cohen, 2008); Computational complexity, the entire MEDLINE corpus was for example processed in around 30 minutes (Cohen and Widdows, 2009); Incremental processing of new data (Widdows and Ferraro, 2008). A more detailed presentation of the computational complexity of RI can be found in Turney and Pantel (2010). The useful properties of RI are also demonstrated by implementations in common VSM packages such as the S-
Space Package (Jurgens and Stevens, 2010) and the Semantic Vectors Package (Widdows and Ferraro, 2008), and extensions of the basic method to new domains and problems (Jurgens and Stevens, 2009, Vasuki and Cohen, 2010). In this paper we generalize the RI approach to matrices and higher-order arrays in a unified framework. The proposed method enables dimension reduction of high-dimensional arrays, which in principle can be of arbitrary order.

There are a few well-known algorithms for dimension reduction of arrays, see Kolda and Bader (2009) for a recent review (note that arrays are commonly referred to as tensors in the literature cited here, but we use the term array to avoid confusion with the concept of tensor fields in generic metric spaces). In particular the Tucker decomposition and the parallel factor model are commonly used methods, and there are some extensions of these two algorithms that are commonly used also. Tucker decomposition (known also as N-mode PCA, N-mode SVD etc.) is a form of higher-order principal component analysis, which decomposes an array, $a_{ijk...}$, according to the scheme

$$a_{ijk...} = \sum_{\alpha} \sum_{\beta} \sum_{\gamma} \ldots g_{\alpha\beta\gamma...} a_{i\alpha} b_{j\beta} c_{k\gamma} \ldots,$$

where $g_{\alpha\beta\gamma...}$ is the core array and $\{a_{i\alpha}, b_{j\beta}, c_{k\gamma} \ldots\}$ are factor matrices. The factor matrices are usually taken to be orthogonal and can be thought of as the principal components in each dimension of the array. The core array represents interactions between the principal components. Several methods to compute the Tucker decomposition have been developed, see the review by Kolda and Bader (2009). The parallel factor decomposition is a special case of the Tucker decomposition, which results if the core array in (13) is enforced to be superdiagonal. These methods are computationally expensive, because the goal to find an “optimal” decomposition with side constraints on the factor matrices is non-trivial. Other examples include two-dimensional singular value decomposition (Ding and Ye, 2005), low-rank approximation of matrices (Ye, 2005), tensor subspace analysis (He et al., 2005), and a generalized form of singular value decomposition for tensors (Drineas and Mahoney, 2007). These methods are either limited to 2nd-order arrays or implicitly assume that the data is static.

The NRI method presented here is incremental and suitable for online processing of information streams (tensor streams), and it complements other methods for incremental tensor analysis (Sun et al., 2008) that are computationally more costly. NRI is somewhat similar to the Tucker decomposition, because it uses a mathematically equivalent expression for decoding of the representation, which is evident from a comparison of equations (13) and (9). Our method is essentially different from Tucker decomposition in the sense that it is based entirely on random coding, and it thereby avoids the computationally costly process of calculating the factor matrices. In our model the factor matrices are randomly generated objects called random indices. Random projection (Papadimitriou et al., 2000) and random mapping (Kaski, 1998) are other methods that use high-dimensional random vectors for dimension reduction. Random projection is used to reduce the dimensionality of a set of points in Euclidean space, while approximately preserving pairwise distances. This possibility follows from the Johnson–Lindenstrauss
lemma (Johnson and Lindenstauss, 1984), which essentially states that a small set of points in high-dimensional space can be mapped into a space of lower dimension such that the distances between the points are approximately preserved. Random projection is used for combinatorial optimization, machine learning and problems related to information retrieval, see Vempala (2005) for a review.

6 Conclusions

The usefulness of RI is demonstrated by numerous applications in natural language processing and generalizations to other problem domains in the last decade. A formal analysis of the approach does not yet exist and more remains to learn about the mathematical principles on which this sparse random coding model is based. The primary objective of this paper is to generalize the mathematical formulation of RI of vectors to arrays of arbitrary order, and to present first results that describe why the RI approach works well for some types of problems.

NRI could be the most simple approximate method for dimension reduction of arrays when no prior assumption can be made about the structure of the data that is encoded, other than the natural assumption that elements with high absolute values are more significant than elements with low values. It is an incremental method in the sense that data can be stored and retrieved sequentially, and the size of the array can be extended in a simple operation. NRI is useful for online analysis of relationships in large datasets and streaming data when a small fraction of the actual feature space is informative and useful. In particular there are a number of evident ways to formulate and use high-order input data for natural language processing applications, for example by augmenting standard co-occurrence matrices with temporal information (Jurgens and Stevens, 2009) or linguistic relations (Baroni and Lenci, 2010, Van de Cruys, 2009), or by incorporating structural information in distributed representations (Clark and Pulman, 2007, Yeung and Tsang, 2004). There have been few attempts at extending traditional matrix-based natural language processing methods to higher-order arrays due to the high computational costs involved. This is something that NRI is likely to facilitate.

The explicit mathematical formulation of NRI that is presented here serves as a starting point for further theoretical developments. In particular the analytical result for the approximate orthogonality of ternary vectors (4) is useful and has not been clarified in this context before. Representations of relationships with NRI, either in the traditional way as vectors or in the form of higher-order arrays, is accurate when the significant relationships (elements) are sparse. Non-sparse random features in the data are averaged out and do not prevent identification of significant relationships with high accumulated weights. There is presently no methodological way to determine a threshold that distinguishes useful relationships from noise, but we show with numerical simulations that the decoded weight is discontinuous and essentially separates true relationships from noise when the feature space is sufficiently sparse. This is interesting since it can allow for an accurate discrimination between useful information and noise in some problems. We present an improved method to test the similarity of RI encoded relationships that
Paper B

is based on the Jaccard index, which is compatible with two-way and higher-order RI.

The possibility to represent non-sparse arrays in a compact representation that can be quickly updated online is one major advantage of NRI over other methods. Other useful properties are the possibility to extend the size of the array at the insignificant cost of generating new sparse random vectors, and that decoding is a simple operation. The NRI method depends on the high dimensionality of index vectors and it is therefore suitable for high-dimensional problems only. In this work we present numerical simulations of two-way and one-way RI, but the software provided (Sandin, 2011) supports NRI of arrays of arbitrary order. We find that the performance of NRI depends significantly on the sparseness of relationships and their relative weight, the dimension reduction ratio and the dimensionality of the index vectors. The comparison between one-way RI of a set of vectors and two-way RI of matrices shows that the signal to noise ratio of decoded features is higher for one-way RI, when all comparable parameters are equivalent. This is to be expected, because the two-way algorithm introduces additional interference between the array elements. The benefit of two-way RI is that the size of the matrix can be extended in both directions and that it scales better at high dimension reduction ratios. There is an interesting scaling relation between the dimension reduction ratio, $\xi$, the order, $r$, and size, $N$, of the array, and the dimensionality of index vectors, $n$: $n = N \xi^{-1/r}$. This suggests that NRI with high dimension reduction ratios should perform better for increasing order of the array, mainly because the performance of NRI is poor at low dimensionality of the index vectors. This is an interesting issue for further investigation, because dimension reduction and analysis of high-order and high-dimensional arrays is a challenging problem in general.

An unexpected numerical result is that the average performance of NRI in terms of the number of correctly identified relationships does not depend significantly on the dimensionality of the index vectors, provided that the dimensionality exceeds a critical threshold of about $n = 10^3$. Below that threshold the average number of correctly decoded relationships decreases notably with decreasing dimensionality, but above the threshold the average is practically independent of dimensionality. We find, however, that the standard deviation of the number of correctly decoded features depends significantly on the dimensionality. The standard deviation decreases when the dimensionality of index vectors increases. We find that the number of non-zero trits in the index vectors, $\chi_D$ ($= 2k$), has an effect on the performance but that the effect is smaller than expected from the analysis of approximate orthogonality of index vectors. The performance increases notably when increasing $\chi_D$ from two to four, but there is no practical reason to go beyond $\chi_D = 8$ since the computational cost increases too much. This demonstrates that there is a tradeoff between the indifference of index vectors and the magnitude of interference between different array elements.

Acknowledgments

The TASA and TOEFL items have been kindly provided by Professor Thomas Landauer, University of Colorado. We thank Pentti Kanerva for valuable comments on the
manuscript of this paper.

Appendix: Indifference of high-dimensional ternary vectors

This section presents the derivation of a result used in Section 2. The total number, \( N \), of ternary vectors of length \( n \) that has \( k \) positive and \( k \) negative trits is

\[
N = \binom{n}{2k} \binom{2k}{k},
\]

(14)
because there are \( C(n, 2k) \) different ways to choose \( 2k \) non-zero trits and \( C(2k, k) \) different ways to distribute the signs to the non-zero trits. How many of these \( N \) vectors have a dot product that is nearly zero, i.e., how many of them are approximately orthogonal? Let \( d = |\langle \cdot, \cdot \rangle| \) be the absolute value of the dot product between two vectors. For simplicity we restrict the analysis to \( 0 \leq d \leq k \), because we are interested in approximately orthogonal vectors only. This restriction does not affect the accuracy of the result. We assume also that the vectors are sparse so that \( n \gg k \). Imagine a fixed reference vector that is picked at random from the space of \( N \) vectors. This reference vector has \( k \) positive trits, \( k \) negative trits and \( n - 2k \) trits that are zero. The large majority of vectors with \( \langle \cdot, \cdot \rangle = \pm d \) with respect to this reference vector will have \( d \) trits that coincides with the \( 2k \) non-zero trits of the reference vector, and the remaining \( 2k - d \) non-zero trits will be distributed among the \( n - 2k \) trits that are zero in the reference vector. There are additional vectors with the same value of \( d \), because cancellations of type \( 1 + 1 - 1 = 1 \) result from higher-order coincidences. The relative number of such vectors is insignificant and we therefore neglect them here. This simplification is justified with a numerical calculation that is presented below. The selection of \( 2k - d \) non-zero trits out of \( n - 2k \) gives a factor of \( C(n - 2k, 2k - d) \). Then remains the question how many possibilities there are to select those \( 2k - d \) non-zero trits from the \( 2k \) non-zero trits in the reference vector, and how many combinations that arise because of signs. These questions are not independent, because the number of ways to choose \( 2k - d \) trits from \( 2k \) trits depends on the number of \( +1 \) trits that are chosen, and the relative number of \( +1 \) trits that are chosen will affect also the number of possible permutations. Accounting for these constraints the number of vectors is

\[
N(n, k, d) \approx \binom{n - 2k}{2k - d} \sum_{n_+ = k - d}^{k} \binom{k}{n_+} \binom{k}{2k - d - n_+} \binom{2k - d}{n_+}, \quad d \leq k, \ n \gg k,
\]

(15)
where \( n_+ \) denotes the number of positive trits that are chosen from the \( 2k \) non-zero trits in the reference vector. The number of negative trits chosen is \( n_- = 2k - d - n_+ \). The sum in (15) arises because there are multiple choices for the number of positive trits to choose from the reference vector. At most \( k \) positive trits can be chosen, i.e., all positive trits. The lower limit of \( n_+ = k - d \) corresponds to the maximum value for the number
of negative trits chosen, \( n_− = k \). The first factor in the sum, \( C(k, n_+) \), accounts for
the number of ways to choose \( n_+ \) positive trits from the \( k \) positive trits in the reference
vector. Similarly, the second factor accounts for the number of ways to choose \( n_− \) negative
trits from the \( k \) negative trits in the reference vector. The last factor accounts for sign
permutations when distributing the chosen trits to the \( 2k - d \) non-zero trits that are
selected by the prefactor. If we divide the number of vectors, \( N(n, k, d) \), which has a
specific value of \( d \) with respect to any reference vector, with the total number of vectors
in the space, \( N \), the result is the relative size of the space as a function of \( d \). The relative
size of the space is equivalent to the probability of randomly choosing a vector from the
space that has a dot product of \( \pm d \) with respect to a reference vector. Since the number
of positive and negative signs are fixed, the combinatorial problem solved here should
have a hypergeometric character. This is indeed the case, because the sum in (15) can
be replaced with a generalized hypergeometric function. The result is presented in (4).

Numerical results for the dot product between a reference vector and \( 10^{12} \) randomly
chosen ternary vectors are presented in Table 3. These numerical results confirm the
analytical result. Observe, however, that the accuracy of the analytical result is poor for
low values of \( n \) and high values of \( k \), as indicated in the table. This is connected to the
assumption that \( n \gg k \) in the analysis above.

References

(6018), 717–719.

Baroni, M. and A. Lenci (2010). Distributional memory: a general framework for corpus-

Berry, M., D. Mezher, B. Philippe, and A. Sameh (2003). Parallel computation of the
singular value decomposition. In E. J. Kontoghiorghes (Ed.), Handbook on Parallel


Bullinaria, J. and J. Levy (2012). Extracting semantic representations from word co-

Clark, S. and S. Pulman (2007). Combining symbolic and distributional models of mean-

Table 3: Approximate orthogonality of the high-dimensional space \((-1, 0, 1)^n\). Tabulated here is the probability, \(P\), in (5) for different values of the vector length, \(n\), and number of non-zero elements, \(2k\). These probabilities are to be compared with the corresponding probabilities obtained from explicit numerical simulations, \(P_{\text{sim}}\). Entries marked with an asterisk demonstrate the effect of neglecting contributions to the inner product arising from higher-order trit combinations (like \(\langle \cdot, \cdot \rangle = \ldots + 1 \times 1 \ldots - 1 \times 1 \ldots + 1 \times 1 \ldots = 1\)) in the analysis leading to (5). The series expansion is marginally applicable in the case \(n = 10^2\) for low values of \(k\), and \(n \gg k\) is violated for high \(k\).

<table>
<thead>
<tr>
<th>(2k)</th>
<th>(\langle \cdot, \cdot \rangle)</th>
<th>(n = 10^2)</th>
<th>(n = 10^3)</th>
<th>(n = 10^4)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(P_{\text{sim}})</td>
<td>(P)</td>
<td>(P_{\text{sim}})</td>
<td>(P)</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>8.5e-1</td>
<td>8.47e-1</td>
<td>9.8e-1</td>
</tr>
<tr>
<td></td>
<td>±1</td>
<td>7.3e-2</td>
<td>7.29e-2</td>
<td>7.93e-3</td>
</tr>
<tr>
<td></td>
<td>±2</td>
<td>2.0e-3</td>
<td>1.99e-3</td>
<td>2.0e-5</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>5.5e-1</td>
<td>*5.17e-1</td>
<td>4.9e-1</td>
</tr>
<tr>
<td></td>
<td>±1</td>
<td>1.9e-1</td>
<td>1.90e-1</td>
<td>3.0e-2</td>
</tr>
<tr>
<td></td>
<td>±2</td>
<td>2.8e-2</td>
<td>2.74e-2</td>
<td>3.9e-4</td>
</tr>
<tr>
<td></td>
<td>±3</td>
<td>2.0e-3</td>
<td>1.96e-3</td>
<td>2.4e-6</td>
</tr>
<tr>
<td></td>
<td>±4</td>
<td>7.4e-5</td>
<td>7.42e-5</td>
<td>8.2e-9</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>3.5e-1</td>
<td>–</td>
<td>8.7e-1</td>
</tr>
<tr>
<td></td>
<td>±1</td>
<td>2.3e-1</td>
<td>–</td>
<td>6.4e-2</td>
</tr>
<tr>
<td></td>
<td>±2</td>
<td>7.9e-2</td>
<td>–</td>
<td>2.0e-3</td>
</tr>
<tr>
<td></td>
<td>±3</td>
<td>1.6e-2</td>
<td>–</td>
<td>3.4e-5</td>
</tr>
<tr>
<td></td>
<td>±4</td>
<td>2.0e-3</td>
<td>–</td>
<td>3.6e-7</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>2.5e-1</td>
<td>–</td>
<td>7.8e-1 *7.73e-1</td>
</tr>
<tr>
<td></td>
<td>±1</td>
<td>2.0e-1</td>
<td>–</td>
<td>1.0e-1</td>
</tr>
<tr>
<td></td>
<td>±2</td>
<td>1.1e-1</td>
<td>–</td>
<td>5.9e-3</td>
</tr>
<tr>
<td></td>
<td>±3</td>
<td>4.3e-2</td>
<td>–</td>
<td>2.0e-4</td>
</tr>
<tr>
<td></td>
<td>±4</td>
<td>1.1e-2</td>
<td>–</td>
<td>4.4e-6</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>2.0e-1</td>
<td>–</td>
<td>6.9e-1 *6.72e-1</td>
</tr>
<tr>
<td></td>
<td>±1</td>
<td>1.8e-1</td>
<td>–</td>
<td>1.4e-1</td>
</tr>
<tr>
<td></td>
<td>±2</td>
<td>1.2e-1</td>
<td>–</td>
<td>1.3e-2</td>
</tr>
<tr>
<td></td>
<td>±3</td>
<td>6.5e-2</td>
<td>–</td>
<td>7.4e-4</td>
</tr>
<tr>
<td></td>
<td>±4</td>
<td>2.7e-2</td>
<td>–</td>
<td>2.8e-5</td>
</tr>
</tbody>
</table>


