Running Head: ENCODING STRUCTURE

Encoding Structure in Holographic Reduced Representations

Matthew A. Kelly

Carleton University

Dorothea Blostein

Queen's University

D. J. K. Mewhort

Queen's University

Send correspondence to:

Matthew A. Kelly

Institute of Cognitive Science

Carleton University

Ottawa, ON, Canada

Email: matthew_kelly2@carleton.ca

Abstract

Vector Symbolic Architectures (VSAs) such as *Holographic Reduced Representations* (HRRs) are computational associative memories used by cognitive psychologists to model behavioural and neurological aspects of human memory. We present a novel analysis of the mathematics of VSAs and a novel technique for representing data in HRRs. Encoding and decoding in VSAs can be characterized by Latin squares. Successful encoding requires the structure of the data to be orthogonal to the structure of the Latin squares. However, HRRs can successfully encode vectors of locally structured data if vectors are shuffled. Shuffling results are illustrated using images, but are applicable to any non-random data. The ability to use locally structured vectors provides a technique for detailed modelling of stimuli in HRR models.

Encoding Structure in Holographic Reduced Representations

First proposed by Longuet-Higgins (1968) and Gabor (1969), a holographic associative memory is a computational memory based on the mathematics of holography. Holographic associative memory has been of interest to cognitive psychologists because:
(i) associative memories are *content-addressable*, allowing items to be retrieved without search, in a manner similar to the fast, parallel retrieval of memories in the human mind.
(ii) just as human memory can store complicated and recursive relations between ideas, *holographic* associative memories can compactly store associations between associations.
(iii) holographic associative memories have what is called "lossy" storage, which is useful for modelling human forgetting.

The mathematics of holography has long been suggested as the principle underlying the neural basis of memory (Pribram, 1969). Cognitive models based on holographic associative memory, such as TODAM (Murdock, 1982), TODAM2 (Murdock, 1993) and CHARM (Metcalfe-Eich, 1982) can explain and predict a variety of human memory phenomena.

Holographic Reduced Representations (HRRs; Plate, 1994) are a refinement of Gabor's holographic associative memory. HRRs have also been used to model how humans understand analogies (Plate, 2000b; Eliasmith & Thagard, 2001) and the meaning of words (BEAGLE; Jones & Mewhort, 2007), how humans encode strings of characters (Hannagan, Dupoux, & Christophe, 2011), and to model how humans perform simple memory and problem-solving tasks such as playing *rocks, paper, scissors*, (Rutledge-Taylor, 2010) and solving Raven's progressive matrices (Rasmussen & Eliasmith, 2011).

Research into HRRs and HRR-based models has been motivated by limitations in the ability of traditional connectionist models (i.e., non-recurrent models with one or two layers of connections) to represent knowledge with complicated structure (Plate, 1995). In traditional connectionist models, an item is represented by a pattern of activation across a group of neurons. Mathematically, the pattern of activation is represented by a vector of numbers that stand for the activations of the neurons. Relationships between pairs of items are defined by the connection weights between groups of neurons. The connection weights between two groups of neurons can be represented as a matrix of numbers. Relationships between more than two items can be defined using more connections, represented as tensors of numbers (i.e., multi-dimensional arrays; for psychological theory see Humphreys, Bain, & Pike, 1989; for computational theory see Smolensky, 1990). Smolensky's tensor memories provide a powerful approach to representing and manipulating relationships between items that differs substantively from traditional connectionist models. In particular, tensor memories (and, in fact, HRRs) do not need to be trained. However, as the number of items bound together into an association grows, the size of the tensor needed to represent the relationship grows exponentially.

HRRs provide a means by which relationships between items can be represented using vectors of dimensionality equal to that of the vectors that represent the individual items. In other words, the size of the vector required to represent a relationship does not grow with the number of items bound together into an association. HRRs are not in themselves a model of human memory; rather HRRs are best understood as a tool for creating and manipulating compact representations of associations that, when used in a model of human memory, will be situated within a larger framework of structures and processes (e.g., Jamieson & Mewhort, 2011).

The majority of models using HRRs are purely cognitive and, as such, do not concern themselves with the question of how an HRR could be implemented in the brain. However, the *Neural Engineering Framework* implements HRRs in detailed, realistic neural models (e.g., Rasmussen & Eliamsith, 2011). Thagard and Stewart (2011) suggest that HRRs are the neural underpinning of creativity: HRRs present a plausible means by which the brain could recursively combine patterns of neural activations representing simple concepts to generate new patterns of activation that represent novel, complicated concepts.

The properties of HRRs dictate that successful encoding and decoding of associations can only happen between vectors with certain statistical properties. According to Plate (1995), a sufficient condition for successful decoding is that the values of the elements within each vector be independently and identically distributed with a mean of zero. To ensure that the vectors have these desirable properties, all applications of holographic memories to date make use of vectors generated with random values independently sampled from a probability distribution, typically a normal distribution. If one attempts to store a vector of data that deviates strongly from the noise-like properties of a vector of values independently sampled from a probability distribution, the data retrieved from the HRR will not much resemble the original. For illustrative purposes, we use photographic images to demonstrate the systematic distortions introduced.

If an image, such as the photo of a church in Figure 1 (left), in stored in an HRR by concatenating the columns of the image to form a vector, then trying to retrieve the image from the HRR will yield an unrecognizable smudge (Figure 1, centre). However, we find that if the elements of the image are randomly permuted before storage, such that the vector representation of the image has a noise-like character, and if the permutation is reversed after retrieval, we successfully retrieve a grainy approximation of the original image (Figure 1, right). We describe this in detail later in the paper, in *Shuffling to Store and Retrieve Structured Data*.

[FIGURE 1 ABOUT HERE]

To our knowledge, only one attempt to use HRRs without using random vectors is documented in the literature. Using vectors generated to represent the meanings of words, Mitchell and Lapata (2010) explore possible means of combining the vectors to represent the meanings of pairs of words. Of the mathematical operations for combining vectors considered by Mitchell and Lapata as models of semantic composition, *circular convolution*, the operation used to create associations in HRRs, provided the poorest fit to human data. Mitchell and Lapata note their use of non-random vectors is one possible cause of circular convolution's poor performance.

In the rest of this paper, we introduce shuffling as a technique for using HRRs with vectors that are not generated by sampling from a probability distribution, such as vectors storing pixels from an image. As background, we discuss the limitations imposed by the requirement that vectors stored in HRRs be composed of random values. We also present a novel analysis in terms of Latin squares of the properties of HRRs and the properties of the family of related representations, *Vector Symbolic Architectures* (VSAs; Gayler, 2003).

Holographic Reduced Representations

The theory of holography was developed by Dennis Gabor in the late 1940s. Psychologists working in the 1960s first suggested holography as a theory of the operation of the brain (see

Willshaw, 1981, for a brief review). Karl Pribram (1969), for instance, suggested that the firing patterns of different neurons may interfere with each other in a manner analogous to the way beams of light interfere, producing hologram-like patterns of interference in the brain. These hologram-like patterns could form the basis of human memory.

Gabor (1969) noted that a system that uses the mathematical operations of convolution and cross-correlation can mimic a hologram and could be used as an associative memory. Holography, thus, can be used to define a computational memory system. Borsellino and Poggio (1972) note a formal identity between Gabor's (1969) holographic associative memory and a functional characterization of the neurophysiology of the optomotor response of insects, suggesting that convolution and correlation may indeed be fundamental to neural processes.

Holographic reduced representations, and vector symbolic architectures in general, are defined by three mathematical operations. The names for these operations vary in the literature, but we will borrow the terminology used by Plate (1995): *encoding, decoding,* and *trace composition.* In HRRs, an association between two vectors is *encoded* as the convolution of the two vectors. In the analogy between the mathematics of HRRs and holography, the convolution of the two vectors corresponds to the hologram, but for our purposes we will refer to it as a memory trace or simply *trace* for short. Correlation defines a decoding operation between a cue (one of the two associated vectors) and a memory trace to produce an approximation of the other vector in the association. Addition defines a trace composition operation that combines traces to produce a new trace, a vector that retains the associations stored in the vectors that were added together. Henceforth, we use the term *trace* to refer to a vector produced by encoding or trace composition, and *approximation* to refer to a vector produced by decoding.

These operations restrict the kinds of vectors that can be used. The use of addition as a trace composition operation requires traces to be linearly independent, or nearly so, or else there will be interference between the traces (Smolensky, 1990). Further restrictions on the kind of vectors that can be used are imposed by the use of correlation as the decoding operation. Correlation is an approximate rather than exact inverse of convolution. The reason correlation is used to decode is because of its tolerance for noise. The exact inverse of convolution allows us to recover from a trace a perfect copy of one of the original vectors, but only if there is no noise in the trace. When noise is added to the trace, the exact inverse will decode a vector that little resembles the original (Plate, 1995, p. 633). Because adding traces together adds noise to the convolution of interest, the exact inverse is rarely useful. Correlation allows us to recover an approximation of one of the original vectors in spite of added noise, though only if certain conditions are met.

As Plate (1995, p. 626) notes, a sufficient condition for correlation to decode convolution is that the elements of the vectors be values independently sampled from a probability distribution with a mean of zero and a variance of 1 / *n*, where *n* is the dimensionality of the vectors. Correlation decodes convolution probabilistically, in the sense that it produces a large number of irrelevant terms that cancel only if the terms are equally likely to be positive and negative, thus summing roughly to zero (see Figure 2). The closer the irrelevant terms come to cancelling out, the less noise there will be in the approximation produced by decoding.

[FIGURE 2 ABOUT HERE]

Shuffling

In addition to encoding and decoding, *shuffling* has historically played an important, if sometimes overlooked, role in holographic reduced representations and other vector symbolic architectures. We use the term *shuffling* to mean applying a randomly selected permutation to a vector, randomly re-ordering the elements of the vector like cards in a card deck. Gayler (2003) describes shuffling as an operation used "to quote or protect the vectors from the other operations" (p. 134). Permutation of the numbers 1 ... n defines a unary function that can transform a vector. A randomly chosen permutation of a vector is unlikely to be similar to the original vector, but the permutation is also reversible. Given any permutation p, there is a permutation p^{-1} such that, $p^{-1}(p(\mathbf{a})) = \mathbf{a}$, for any vector **a**. When shuffled, the information within a vector is essentially hidden and protected from being affected by other vector operations.

For example, circular convolution (denoted by *) is commutative, that is, for any vectors **a** and **b**, $\mathbf{a}*\mathbf{b} = \mathbf{b}*\mathbf{a}$. This symmetry of circular convolution has been a major strength for the purposes of modelling memory for word pairs (Murdock, 1982; for review see Kahana, 2002), but it can be a hindrance in situations where the order of items matter. For example, "dog feed" and "feed dog" are phrases which carry different meanings by virtue of differences in word order.

A non-commutative variant of circular convolution can be defined using a randomly selected permutation p and its inverse p^{-1} (Plate 2003, p. 121). By always shuffling one of the vectors before convolution, one defines an encoding operation that is non-commutative, i.e., whereas $\mathbf{a}*\mathbf{b} = \mathbf{b}*\mathbf{a}$, $p(\mathbf{a})*\mathbf{b} \neq p(\mathbf{b})*\mathbf{a}$. Decoding using circular correlation (denoted by #) then uses the inverse permutation p^{-1}, e.g.,

$$p^{-1}(\mathbf{a} \,\#\, (\mathbf{a} * p(\mathbf{b}))) \approx \mathbf{b},$$

$$p^{-1}(\mathbf{b} \,\#\, (\mathbf{a} * p(\mathbf{b}))) \not\approx \mathbf{a}.$$

This non-commutative form of encoding is used by the BEAGLE model (Jones & Mewhort, 2007) to bind vectors that stand for words in sentences in order to construct representations of the semantics of each of those words.

Permutation is a powerful tool that can itself be used to encode information. For example, Sahlgren, Holst, and Kanerva (2008) propose a model that encodes information about word-order using permutation. For example, given the sentence "*All cows eat grass*" and vectors that stand for **all**, **cows**, **eat**, and **grass**, one can construct a vector that represents the positional relationships of the word *eat* to the other words using a randomly selected permutation *p* and addition for superposition:

$p^2(\mathbf{all}) + p(\mathbf{cows}) + p^{-1}(\mathbf{grass})$,

where p^2 denotes applying the permutation *p* to the vector **all** twice, meaning that the word *all* occurs two words to the left of *eat*. Likewise, applying the inverse permutation once, p^{-1} means that *grass* occurs one word to the right of *eat.*

As we demonstrate in what follows shuffling provides a useful technique for working with vectors of non-random values. Though the circumstances of application are different, like Gayler (2003), we use shuffling as a means to protect the data within the vector.

Representation in HRRs

Suppose we wish to represent an association between two stimuli: a blue circle and a red triangle. In an HRR, this association could be represented by convolving a vector that represents the blue circle with a vector that represents the red triangle. But how are the vectors that represent the two individual shapes formed? We explore this example not with the intent of answering the question of how neurons in the brain in fact represent shape and colour, but with the intent of illustrating some of the challenges involved with creating representations in HRRs and thereby motivating work that follows in the paper.

In designing a cognitive model, one needs to settle on a representation scheme. One possible representation scheme is a model perceptual system that takes data of the sort that might be produced by sensory organs and interacts with the memory system to arrive at an interpretation of the data. In practice, however, existent models that use VSAs do not model perception, using instead an ad hoc representation scheme suited for the particular cognitive task being modelled.

The simplest representation scheme assigns each to-be-represented item a random vector. The *blue circle* is represented by one random vector and the *red triangle* is represented by another random vector. If each vector is a vector of *n* values randomly sampled from a normal distribution with a mean of zero and variance of 1/*n,* then the vectors will have the recommended statistical properties for encoding with convolution (Plate, 1995). When decoding, the approximation is matched against a library of canonical vectors, termed a *clean-up memory,* in order to determine which item has been retrieved. For example, if *blue circle* is represented by the vector **b** and *red triangle* by the vector **r**, then their association is **b** * **r**, and **r** # **b** * **r** is the approximation of **b** created by decoding with **r**. The vector **r** # **b** * **r** is identified as an approximation of **b** using the clean-up memory. The clean-up memory measures the similarity of **r** # **b** * **r** to each vector that stands for a unique item (at least **b** and **r** in this example), and then the vector to which **r # b** * **r** is most similar is the vector to which **r # b** * **r** is said to be a noisy approximation.

However, representing each item as a randomly generated vector is arguably too simple a representation scheme. To make this point clear, we need to introduce some terminology. Gayler (2003) makes a distinction between *atomic* and *complex* vectors. An *atomic* vector is randomly generated and stands for a primitive feature of representations in the cognitive system. Atomic vectors are approximately orthogonal to each other, and, thus, the features that the atomic vectors stand for ought to be likewise completely dissimilar to each other. A *complex* vector is generated by the addition and convolution of atomic vectors and stands for a concept or percept that can be decomposed into the features that each constituent atomic vector stands for. Because a complex vector is composite in nature, a complex vector can be very similar to other complex vectors with similar composition and structure.

The items that cognitive models tend to deal with are complex rather than atomic in nature. For example, while it makes intuitive sense for the representations of a *blue circle* and a *red triangle* to be highly dissimilar, a *red circle* ought to be similar to both *blue circle* and *red triangle*. To represent this similarity, we could represent each coloured shape as a complex vector created by combining an atomic vector that stands for a colour with an atomic vector that stands for shape (e.g., *red * circle = red circle*). However, a *maroon oval* ought to be similar to a *red circle*, even though it has both a different colour and a different shape. This in turn suggests that even colour categories and shape categories ought not to be treated as primitive features represented by atomic vectors.

How should we represent *blue circle* or *red triangle* or other coloured shapes? Intuitively, we could represent them using digital images, which capture in detail the shape and colour. In adopting such a representation scheme, we would not be making the dubious claim that the mind uses digital images as representations; rather, we would be claiming that the mind is sensitive to features of the coloured shapes that are captured by the digital image, and, thus, the digital image may serve as a convenient proxy for the actual representation used by the mind.

Of course a digital image is not a close proxy to the representation used by the mind: there are features of a real object that the mind is sensitive to that are altogether absent from a digital image, such as depth and motion, and likewise, there are features of the image that the mind is fairly insensitive to, such as the quantity of white-space surrounding an object or the exact location of an object relative to the boundaries of the image. But the image does capture in detail the two features of objects that we are concerned with in this example: colour and shape.

The simplest means of transforming a greyscale digital image into a vector is to concatenate the columns (or rows) of pixels into one long vector. For a colour image, concatenate columns of pixels to produce three vectors of intensity values, one for each colour (red, green, blue), then concatenate those three vectors to form a single vector. Unfortunately, the vector produced by concatenating columns of pixels does not have the statistical properties necessary for accurate encoding and decoding with circular convolution.

To produce a vector that encodes all of the information in a greyscale image but has the necessary statistical properties, one could assign each pixel location in the image a random, atomic vector. The image can then be represented as a complex vector that is a sum of atomic vectors weighted by the intensity of the pixel that each vector represents. For a colour image, one would need to assign three atomic vectors to each pixel. To give an example, to represent any 100 x 100 pixel colour image, a small image the size of a typical internet avatar, one would need a library of 30 000 atomic vectors, each of a sufficiently large dimensionality that each atomic vector is approximately orthogonal to each other atomic vector. These atomic vectors would need to be stored in the clean-up memory. Given the billions of neurons in the brain, the kind of storage required for this does not seem theoretically implausible, but working with 30 000 or more atomic vectors, each with thousands of dimensions, is computationally expensive with conventional computer hardware.

By using convolution, we can represent a digital image with fewer atomic vectors. Suppose we assign to each row and each column of the image an atomic vector, and assign each of the three primary colours a vector. Each pixel in the image can then be represented as $\mathbf{x}_i * \mathbf{y}_j * (w_r\, \mathbf{r} + w_g\, \mathbf{g} + w_b\, \mathbf{b})$ where $\mathbf{x}_i$ is the atomic vector for *i*th column, $\mathbf{y}_j$ likewise represents the *j*th row, **r**, **g**, and **b** are vectors representing the three colours, and w_r, w_g, and w_b are the colour intensities. The image as a whole would be represented by a complex vector that is the sum of the representations for each pixel. Using this scheme, representing any 100 x 100 pixel image requires a library of only 203 atomic vectors.

However, if we use the scheme just described to construct a representation of *blue circle* and a representation of *red triangle*, we cannot use circular convolution to create a representation that unambiguously represents an association between the two shapes. Convolving the complex vector for *blue circle* with the complex vector for *red triangle* convolves each of the terms in *blue circle* with each of the terms in *red triangle*, such that the resultant vector would contain terms in which it is ambiguous which colour belongs with which row and which column (e.g., $w\, \mathbf{x}_i * \mathbf{x}_j * \mathbf{y}_k * \mathbf{y}_l * \mathbf{r} * \mathbf{b}$). To protect the internal structure of the shapes from the external structure of the relationship between the shapes, we can shuffle (Gayler, 2003). A shuffled complex representation of each shape can be used as if it were an atomic vector, allowing us to then use circular convolution and addition to describe the relationships between the shapes.

But as it happens, we can use a simpler approach based on the original idea of representing the digital image by concatenating the columns of pixels into a single vector. We can normalize the vector to a mean of zero and a Euclidean length of one, and shuffle the elements. The resulting vector of image data can be used as easy-to-generate representation that captures many of the features of the stimulus. Why exactly this technique of shuffling works is not obvious, so in what follows, we delve first into the theory of HRRs and VSAs, presenting a novel analysis, before returning to this technique and demonstrating its effectiveness.

Our approach is an alternative to the random generation of vectors for use as representations of items in HRRs. Shuffling allows the traditional, atomic symbol to be replaced with a (shuffled) complex representation that could be perceptual or conceptual in nature. This could help in bridging the gap between perceptual and conceptual processing.

Encoding and Decoding with Latin Squares

For HRRs and the family of related representations, *Vector Symbolic Architectures* (VSAs), we present a novel framework for understanding and analyzing encoding and decoding in terms of Latin squares.

Circular convolution, used to encode associations in HRRs, is a compression of the outer-product of the two vectors being convolved (see Figure 3). Each element of the convolution is a sum of elements of the outer-product. Likewise, circular correlation is as a compression of the outer-product of the two vectors being correlated. More generally, in VSAs, encoding and decoding with real valued representations can be understood as a compression of the outer-product of those representations. Plate (2003, p. 228) observes that any encoding or decoding operation that can be understood as a compression of the outer-product matrix will introduce minimal noise under the following conditions:

Given $\mathbf{x}$ and $\mathbf{y}$, vectors of independently distributed real values with a mean of zero and a Euclidean length of one, and their outer-product $\mathbf{x}\,\mathbf{y}^T$, the trace $\mathbf{z}$ is defined as such:

(1) Each element z_i of the trace is a sum of elements of $\mathbf{x}\,\mathbf{y}^T$ such that each element $x_j\,y_k$ of $\mathbf{x}\,\mathbf{y}^T$ occurs in the sum of one and only one z_i.

(2) Each z_i is a sum chosen such that each element x_j and y_k of the items occur in the sum for z_i exactly once.

Requirements (1) and (2) together define a Latin square. A Latin square is an n x n matrix filled with the numbers 1 ... n, where each number occurs exactly once in each row and each column. Thus any compression of the outer-product defined by a Latin square is an optimal encoding operation for a VSA.

As shown in Figure 3, the result of an encoding operation (right) is determined by a Latin square (centre). For example, the first element of the result is the sum of all elements of $\mathbf{a}\,\mathbf{b}^T$ at positions marked by **1** in the Latin square.

[FIGURE 3 ABOUT HERE]

While only the compressions defined by Latin squares are optimal, in high dimensional spaces, any randomly chosen compression of the outer-product will almost always still be close enough to being a Latin square to be an effective encoding operation (Plate, 2000a). This observation lends plausibility to VSAs as a model of how representations are constructed in the brain by showing that the necessary neural structures for encoding can easily arise by chance arrangements of connections (Plate, 2000a).

An important property of Latin squares is that any permutation of the rows and columns of a Latin square is also a Latin square. A Latin square permuted so that the elements in the first row and first column are in rank order is said to be in reduced form (e.g., the Latin square in Figure 3, centre, is already in reduced form).

In what follows, we characterize encoding and decoding in terms of Latin squares and characterize relationships between the properties of encoding and decoding operations and the properties of the Latin squares that define them.

Let us use the symbol ⊠ to denote encoding an association. Let us define *compressed outer-product* as a function that produces a vector $\mathbf{a}\boxtimes\mathbf{b}$ of dimensionality n that encodes the association of $\mathbf{a}$ and $\mathbf{b}$ for any given pair of vectors $\mathbf{a}$ and $\mathbf{b}$ of dimensionality n, and any n x n Latin square $\mathbf{E}$:

$\mathbf{a}\boxtimes\mathbf{b}$ = compressed outer-product($\mathbf{a}$, $\mathbf{b}$, $\mathbf{E}$)

where the vector $\mathbf{a}\boxtimes\mathbf{b}$ is defined such that for all indices $k = 1 \ldots n$ of $\mathbf{a}\boxtimes\mathbf{b}$:

$(\mathbf{a}\boxtimes\mathbf{b})_k = \sum a_i b_j$, for all i and j where $E_{ij} = k$

The *compressed outer-product* function can be used to define both encoding and decoding.

Because most encoding operations defined by a Latin square are non-commutative, i.e. $\mathbf{a}\boxtimes\mathbf{b} \neq \mathbf{b}\boxtimes\mathbf{a}$, two decoding operations are necessary. One of the two decoding operations retrieves the first item of an associated pair, whereas the other retrieves the second item of the pair. Like encoding, both decoding operations are defined by Latin squares. Decoding can be understood as compressing the outer-product of the cue (i.e., the item being used to decode) and the trace. In the outer-product of the cue and trace, there are *n* noisy estimates of each element of the item being decoded. For each element, the decoding Latin squares sum those *n* estimates together to arrive at a less noisy estimate, essentially averaging out the noise.

Let us use the symbol $\triangleright$ to denote decoding from an association. Given a Latin square **E**, and vectors **a** and **b**, and their association $\mathbf{a}\boxtimes\mathbf{b}$, which is a compression of the outer-product defined by **E**, there are two decoding Latin squares, $\mathbf{D_1}$ and $\mathbf{D_2}$, such that:

$\mathbf{a}\boxtimes\mathbf{b}$ = compressed outer-product(**a**, **b**, **E**)

$\mathbf{b} \approx \mathbf{a}\triangleright\mathbf{a}\boxtimes\mathbf{b}$ = compressed outer-product(**a**, $\mathbf{a}\boxtimes\mathbf{b}$, $\mathbf{D_1}$)

$\mathbf{a} \approx \mathbf{b}\triangleright\mathbf{a}\boxtimes\mathbf{b}$ = compressed outer-product(**b**, $\mathbf{a}\boxtimes\mathbf{b}$, $\mathbf{D_2}$)

We observe that $\mathbf{D_1}$ and $\mathbf{D_2}$ can be obtained from **E**. For all indices *i* and *j* from 1 ... *n*:

$D_{1\,ik} = j$, where $k = E_{ij}$

$D_{2\,ik} = j$, where $k = E_{ji}$

If **E** is symmetrical, that is, if $E_{ij} = E_{ji}$, then $\mathbf{D_1} = \mathbf{D_2}$, and the encoding operation defined by **E** will be commutative, i.e., $\mathbf{a}\boxtimes\mathbf{b} = \mathbf{b}\boxtimes\mathbf{a}$ for any **a** and **b** where $\boxtimes$ is defined by a symmetrical **E**. If **E** is approximately symmetrical, that is, if $E_{ij} = E_{ji}$ for most *i* and *j*, the encoding defined by **E** will be approximately commutative, i.e., $\mathbf{a}\boxtimes\mathbf{b} \approx \mathbf{b}\boxtimes\mathbf{a}$, such that *compressed outer-product*(**b**, $\mathbf{a}\boxtimes\mathbf{b}$, $\mathbf{D_1}$) will be a less good approximation of **a** than *compressed outer-product*(**b**, $\mathbf{a}\boxtimes\mathbf{b}$, $\mathbf{D_2}$).

Thus, a real-valued VSA can be characterized in terms of the Latin squares **E**, $\mathbf{D_1}$, and $\mathbf{D_2}$, which respectively describe encoding a pair of items (**E**), cueing with the first item to decode for the second item ($\mathbf{D_1}$), and cueing with the second item to decode for the first item ($\mathbf{D_2}$).

The properties of these Latin squares provide a novel framework for analyzing the properties of the VSAs they define. For example, encoding and decoding in some real-valued VSAs are defined by *partial* Latin squares, i.e., some elements of the outer-product are not included in the compression. For real-valued vectors, if the encoding and decoding are defined by partial Latin squares, decoding is less accurate than if defined by full Latin squares, as is discussed in *Encoding and Decoding with Partial Latin Squares*.

Analysis of Properties of VSAs

The various kinds of VSA differ primarily in two respects: how the items are represented, and how encoding (and decoding) are defined (see Table 1 for comparison). HRRs (Plate, 1995) and *Multiply-Add-Permute* (MAP) coding (first introduced in 1998, but see Gayler, 2003) both operate on vectors of real numbers, but HRRs use circular convolution to encode associations, whereas MAP coding uses element-wise multiplication. Square matrix representations operate on square matrices of real numbers and use matrix multiplication to encode (Kelly, 2010). Binary spatter codes operate on vectors of binary values and use element-wise XOR to encode (Kanerva, 1996). Frequency-domain HRRs operate on vectors of complex numbers, with values on the unit circle of the complex plane, and use element-wise multiplication to encode associations between vectors (Plate, 2003, chap. 4). Plate (1997) observes that binary spatter codes are equivalent to frequency-domain HRRs restricted to vectors of only two values on the unit circle of the complex plane: +1 and -1.

The decoding accuracy of a VSA is a measure of the amount of noise added to an item by encoding and decoding. We use, as is standard (Goebel & Lewandowsky, 1991), the cosine of the angle between the vectors as a measure of their similarity. Table 1 presents a comparison of

VSAs. For each VSA, we calculated the decoding accuracy as the mean of $\mathit{cosine}(\mathbf{b}, \mathbf{a} \triangleright \mathbf{a} \boxtimes \mathbf{b})$ across 1000 different randomly generated **a** and **b**. The items **a** and **b** are vectors of 2401 dimensions, except in the case of square matrix representations, where **a** and **b** are 49 x 49 matrices. Real-valued **a** and **b** were generated by sampling from a normal distribution with mean zero and variance $1/n$, then normalizing the vectors to have exactly a mean of zero and exactly a Euclidean length of one. For binary spatter codes, binary **a** and **b** were generated by setting each value to either 1 or 0 with equal probability. For MAP coding, binary **a** and **b** were generated by setting each value to either +1 or -1 with equal probability.

[TABLE 1 ABOUT HERE]

To compare the VSAs that use vectors of normally distributed values, HRRs have a slightly greater decoding accuracy (cosine of 0.71) than CHARM (0.70) and square matrix representations (0.70), which omit some of the elements from the outer-product. However, the difference is slight enough to be of little concern to modellers.

MAP coding is something of a special case. Vector elements are randomly initialized to +1 or -1 and element-wise multiplication is used to encode and decode. For vectors of complex values on the unit circle (which includes binary-valued vectors), element-wise multiplication is mathematically equivalent to a full Latin square compression of the outer-product matrix (see the section *Circular Convolution and Correlation* later in the paper) and allows for noiseless encoding and decoding. MAP coding, binary spatter codes, and frequency-domain HRRs all make use of this property to encode and decode.

However, unlike frequency-domain HRRs and binary spatter codes, MAP vectors cease to be binary valued after trace composition. MAP coding uses addition for trace composition. With each summation, a MAP vector will increasingly deviate from being binary valued to being normally distributed. For vectors of real values normally distributed around zero, MAP coding achieves a decoding accuracy of 0.58, reflecting the fact that element-wise multiplication is a very sparse partial Latin square, as it uses only the main diagonal of the outer-product (see Figure 4). The low decoding-accuracy of MAP coding for normally distributed values can be addressed by periodically recasting the MAP vectors to binary values or by using a clean-up memory. If vectors are always recast to binary values after addition, MAP vectors are mathematically equivalent to binary spatter codes and binary-valued frequency-domain HRRs.

[FIGURE 4 ABOUT HERE]

Encoding and Decoding with Partial Latin Squares

In general, using encoding and decoding operations described by partial Latin squares results in a greater amount of noise in the approximations decoded. The reason for this is simple. As noted by Plate (1995; see Figure 2), the approximation decoded from the association of **a** and **b** by the cue **b** can be decomposed as:

$$\mathbf{b} \triangleright (\mathbf{a} \boxtimes \mathbf{b}) = (\mathbf{b} \bullet \mathbf{b})\, \mathbf{a} + \mathbf{w}$$

where $\triangleright$ is decoding, $\boxtimes$ is encoding, • is the dot product, and **w** is noise. If **b** is a vector of Euclidean length 1, (**b** • **b**) will be 1, and thus the approximation will be **a** + **w**, i.e., the signal plus some noise. In order to factor (**b** • **b**) out of the signal portion of the approximation, each element i of the approximation must be a sum that contains the terms $b_j^2\, a_i$ for all $j = 1 \ldots n$. Just as the approximation can be understood as a compression of the outer-product of the trace **a**⊠**b** and the cue **b**, the trace **a**⊠**b** can be understood as a compression of the outer-product of **a** and **b**. Thus, in order for each element i of the approximation to contain the terms $b_j^2 a_i$ for all j, the terms $b_j a_i$ for all i and j must be present in **a**⊠**b**. A partial Latin square compression of the outer-product, i.e., a compression of the outer-product $\mathbf{a}\,\mathbf{b}^{\mathrm{T}}$ that leaves out elements $b_j a_i$ for some i and j, cannot produce an approximation that allows $(\mathbf{b} \bullet \mathbf{b})$ to be factored out of the signal. As a result, when using partial Latin squares, different parts of the signal component of the approximation are weighted by different scalars, increasing the noise by distorting the signal.

The difference in decoding accuracy between partial Latin squares and complete Latin squares depends on the structure of the partial Latin square. The decoding accuracy of CHARM and square matrix representations is only slightly less (0.70 cosine) than the decoding accuracy of HRRs (0.71 cosine). In square matrix representations, each column of the decoded approximation matrix is weighted by a different scalar, though each scalar is of approximately the same magnitude, slightly distorting the signal. However, when using element-wise multiplication to encode and decode with normally distributed vectors, each individual element of the vector is weighted by a different scalar, severely distorting the signal (cosine 0.58).

Arguably, the use of a clean-up memory obviates the need for a high degree of decoding accuracy for most applications of VSAs, as vectors in high-dimensional spaces can be correctly identified even with the addition of a large amount of noise (Kanerva, 2009). However, for applications where a vast amount of information is being stored in each vector (e.g., Jones & Mewhort, 2007; Rutledge-Taylor, Vellino, & West, 2008) minimizing the amount of information lost in each encoding may be desirable. For such applications, noise may thus be a reason to preferentially use encoding and decoding operations that are described by a full Latin square compression of the outer-product.

Circular Convolution and Circular Correlation

Properties of HRRs can be understood by examining the properties of the Latin squares that define encoding with circular convolution and decoding with circular correlation.

The Latin square that defines circular convolution for vectors of any given dimensionality is symmetrical along the main diagonal (see Figure 5), and as a result, circular convolution is commutative. Commutativity is a useful trait, but not always desirable, because for some applications the associations **a⊠b** and **b⊠a** need to be distinguishable, such as in language processing (e.g., Jones & Mewhort, 2007). However, by randomly permuting one of the operands, circular convolution can be made non-commutative (Plate 2003, p. 121), as will be discussed further in the next section.

[FIGURE 5 ABOUT HERE]

In addition to commutativity, circular convolution has other interesting properties. Circular convolution and circular correlation are both highly ordered compressions of the outer-product matrix (see Figure 5; Plate, 1995). When using circular convolution to convolve highly ordered data, the order in the data becomes confounded with the ordered structure of the compression, badly distorting the data on retrieval (see Figure 6).

While we have described circular convolution as a sum of elements of the outer-product, it has long been known that circular convolution can be computed more quickly as element-wise multiplication in the frequency-domain. To formalize the notion of "more quickly", we can use big O notation. Given vectors of dimensionality n, element-wise multiplication takes O(n) time to compute, that is, the time to compute grows linearly with n. The outer-product takes longer, O(n^2) time, to compute, that is, the time to compute grows with the square of n. In order to compute circular convolution quickly as element-wise multiplication it is necessary to convert to and from the frequency-domain, which can be accomplished using the fast Fourier transform (FFT) and the inverse fast Fourier transform (IFFT). Both the FFT and IFFT can be computed in O(n log n), that is, log-linear time, so circular convolution can be computed as follows in overall O(n log n) time without having to compute the outer-product (Plate, 2003, p. 116):

$$\mathbf{a} * \mathbf{b} = \mathrm{IFFT}(\mathrm{FFT}(\mathbf{a}) \circ \mathrm{FFT}(\mathbf{b}))$$

where $\circ$ is the element-wise product and * is circular convolution.

Circular correlation can be computed in O(n log n) time by reversing all the elements of the cue, **a**, save for first (e.g., for $n = 4$, $\mathbf{a} = [a_1, a_2, a_3, a_4]$ becomes $\mathbf{a}'' = [a_1, a_4, a_3, a_2]$), then calculating the circular convolution of the trace, **t**, and the modified cue, **a''** (Plate, 2003, p. 97):

$$\mathbf{a} \# \mathbf{t} = \mathbf{a}'' * \mathbf{t} = \mathrm{IFFT}(\mathrm{FFT}(\mathbf{a}'') \circ \mathrm{FFT}(\mathbf{t}))$$

The seemingly strange equivalence between circular correlation and circular convolution can easily be understood with reference to the Latin squares that describe these operations. As can be seen in Figure 5, reversing the order of all the rows but the first transforms one Latin square into the other. By permuting the elements of the cue, one is effectively permuting the Latin square that describes circular convolution to match the Latin square that describes circular correlation, such that computing the convolution of the modified cue is equivalent to correlating with the original cue. This rearrangement can be equivalently performed in the frequency domain simply by taking the complex conjugate (denoted by *conj*) of the Fourier transform of the cue, **a**:

$$\mathbf{a} \# \mathbf{t} = \mathbf{a}'' * \mathbf{t} = \text{IFFT}(\, conj(\text{FFT}(\mathbf{a})) \circ \text{FFT}(\mathbf{t})\,)$$

Random Latin Squares

A compression of the outer-product defined by a randomly generated, highly unstructured Latin square will not be commutative (Plate 2003, p. 121), which is useful in any application where the order of the items is important, such as in language processing (e.g., Jones & Mewhort, 2007). We have found that an unstructured compression is also able to store and retrieve structured data, such as images.

However, computing the outer-product is slow, O(n^2). To be practical, we need a way to quickly compute a compression of the outer-product defined by an unstructured Latin square. As discussed in the previous section, circular convolution can be computed in O(n log n) time. Any Latin square that can be reduced to the Latin square for circular convolution, by permuting the rows and columns, can be computed rapidly in O(n log n) time.

Suppose we have a Latin square that we wish to use for encoding and a pair of vectors we wish to associate. Suppose that this Latin square is not the Latin square for circular convolution, but can be reduced to the Latin square for circular convolution by permuting rows and columns. We can compute the outer-product of those vectors and then permute the rows and columns of the outer-product matrix in the same way as the Latin square. Compressing the permuted outer-product matrix in accordance with the Latin square that describes circular convolution will give the same result as compressing the original outer-product matrix according to the original Latin square. The permuted outer-product matrix is the outer-product of permuted variants of the original pair of vectors. Thus, the unstructured compression can be computed quickly, without having to calculate the outer-product, by first permuting the vectors, then computing the convolution of those vectors using FFTs. We permute the first operand by rearranging the elements in accordance with the row rearrangements necessary to transform the Latin square into the square for circular convolution. We permute the second operand in accordance with the necessary column rearrangements.

To encode using a random, unstructured Latin square, we randomly permute the vectors, then convolve. If we permute the first and second operand differently, the random Latin square is asymmetric, and encoding will be non-commutative. If we permute the first and second operand identically, the random Latin square is symmetric, and encoding will be commutative. To decode, we can use circular correlation, but apply the inverse of the permutation to the approximation. Using randomly selected permutations, which take O(n) time to compute, it is thus possible to encode or decode structured data in overall O(n log (n)) time.

These time complexity considerations assume a serial computer architecture. In neural models (or neural tissue) permutation can be trivially implemented by a very disordered cross-wiring between neural groups. Indeed, such randomly disordered cross-wiring may be more neurally plausible than the orderly alternative (Plate, 2000a). Using a spiking neuron architecture (such as the *Neural Engineering Framework* used in Rasmussen & Eliasmith, 2011) one can compute a fixed permutation and circular convolution using two layers of connections: the first layer to permute and perform a Fourier transform on the data and the second layer to compute element-wise multiplication and the inverse Fourier transform. If each neuron is run in parallel (as in a real brain) permutation and convolution can be computed rapidly.

Plate (2003, p. 121) first proposed using permutation with HRRs as a means of making circular convolution non-commutative, and in this role, permutation is already in use in the literature (e.g., Jones & Mewhort, 2007). As we detail in the next section, we present a novel application of permutation to the storing of structured data in HRRs.

Shuffling to Store and Retrieve Structured Data

By normalizing a vector to have a mean of zero, and then shuffling (i.e., randomly permuting) the elements, it is possible to store and retrieve vectors of structured data in an HRR.

To illustrate the use of vectors of structured data, we concatenate the columns of a grey scale image to form a vector, then we normalize the vector to a Euclidean length of one and a mean of zero. Figure 6 illustrates that convolving the image with itself, and then correlating with the image, produces an unrecognizable smudge.

[FIGURE 6 ABOUT HERE]

The image is smudged on retrieval because the structure of the image is confounded with the structure of the outer-product compression defined by circular convolution. Convolution is more or less computed as a sum of the diagonals of the outer-product matrix. To work, convolution memories need the elements of the diagonals to have values that are evenly and independently distributed around zero in order for the irrelevant terms to cancel (see Figure 2). In this example, the image data is arranged in a vector such that vertically adjacent locations in the image are represented by adjacent elements of the image vector, so values along the diagonals of the outer-product of the image vector with itself will be highly correlated. The compression thus introduces systematic distortions in the data recovered from memory.

[FIGURE 7 ABOUT HERE]

We can resolve this problem by shuffling. Shuffling is merely a re-ordering of the values in the vector, and, thus, shuffling does not change the mean or variance of the vector. Shuffling does change how the values are distributed within the vector, destroying the autocorrelation structure of the image. A shuffled vector looks like noise to the eye and, as illustrated by Figures 6 to 10, works as well as a random vector when using a structured compression of the outer-product to encode or decode. Shuffling thus provides a tool for making the structure of the data orthogonal to the structure of the Latin square used for encoding, which useful in VSAs such as HRRs and square matrix representations.

Shuffling is reversible. Keeping track of how we shuffled the elements enables us to unshuffle them afterward. Thus the approximation produced by correlation can be unshuffled to produce an image recognizable to the human eye (see Figure 7). This technique works both when an image is associated with itself, or when two different images are associated with each other (see Figure 8). If the two images are shuffled in the same way (as in Figure 8) the shuffled versions of the images will be as similar to each other as the unshuffled versions of the images (as measured by cosine). Conversely, if two images are shuffled differently, the shuffled versions of the images will have only incidental similarity.

Circular convolution is element-wise multiplication in the frequency-domain. Ideally, the frequency components of a vector have a magnitude of one, such that the components neither grow nor shrink in magnitude when multiplied. Frequency-domain HRRs (Plate, 2003, chap. 4) use circular vectors (vectors of values sampled from the unit circle; see Figure 9a) and can achieve loss-less encoding (see Table 1). HRRs use Gaussian vectors, which, when Fourier transformed, fall in and around the unit circle (Figure 9b). Thus, when multiplying with Gaussian vectors in the frequency-domain, there will be small increases and decreases in the magnitudes of the frequency-components, resulting in small distortions (cosine = 0.71, see Table 1).

The Fourier transform of the grave image vector reveals that the image has frequency components with large magnitudes (Figure 9c). When the grave image is convolved with itself, these large magnitudes are squared, becoming even larger, introducing significant distortions (cosine = 0.50, see Figure 6). Even if the image is convolved with a vector of random values, these large magnitude frequency components introduce serious distortions. The Fourier transform of the shuffled grave image, however, strongly resembles the Fourier transform of the Gaussian vector (Figure 9d) and works as well (cosine 0.82, Figure 7).

Figures 10 and 11 illustrate the same operations as Figures 6 and 7 but with a simpler image. For simpler images, shuffling does not improve the cosine similarity of the approximation to the original. However, shuffling changes the character of the distortions in the approximation. Without shuffling, the approximation of the circle is systematically distorted (i.e., smudged), whereas with shuffling, the approximation of the circle is randomly distorted (i.e., noisy).

[FIGURE 8 ABOUT HERE]

[FIGURE 9 ABOUT HERE]

[FIGURE 10 ABOUT HERE]

[FIGURE 11 ABOUT HERE]

Figure 12 illustrates concomitant variation between the granularity of the permutation and the decoding accuracy or amount of systematic distortion. We can treat sequences of values within a vector as discrete units and shuffle the position of those sequences within the vector. In the vector for the image of the grave, these sequences correspond to vertical strips of pixels. We associate the shuffled image with itself, then decode from the association using the shuffled image to retrieve an approximation. The approximations are shown in Figure 12. Shuffling individual pixels yields an image that is easily recognizable to the eye (Figure 12, top-left). This is followed by clear degradation, both visually and according to vector cosine, for every increase in sequence length (Figure 12).

[FIGURE 12 ABOUT HERE]

Shuffling is not the only operation that could be used to serve this purpose, but shuffling is both simple and effective. For instance, the discrete cosine transform, which serves as the basis for JPEG compression, can substitute for shuffling. Figure 13, centre, is an approximation decoded from an association of the image with itself, but rather than shuffling the image before encoding the association (as in Figures 7 and 10), we apply a discrete cosine transform to the image. Though, as Figure 13 illustrates, more information is lost with the discrete cosine transform than when using shuffling.

[FIGURE 13 ABOUT HERE]

Compression is another alternative to shuffling. Highly compressing data such that each bit in the compressed vector is statistically independent from the other bits gives the vector the noise-like character it needs for successful encoding in VSAs. However, in a vector of compressed data, redundancies have been eliminated, such that the vector will be especially susceptible to the lossy nature of VSAs.

Discussion

We have presented an analysis of encoding and decoding in VSAs in terms of Latin squares. Building on the work of Plate (1994), we note that encoding and decoding in a VSA can be defined by one Latin square for encoding and two Latin squares for decoding. We define how the decoding Latin squares can be derived from the encoding Latin square, and, thus, how the two decoding operations can be derived from the encoding operation. We find that certain properties of VSAs are determined by properties of the Latin squares that define their encoding and decoding operations. For example, symmetric Latin squares define a commutative encoding operation and asymmetric Latin squares are non-commutative. Encoding defined by partial Latin squares is more lossy than encoding defined by complete Latin squares.

Of particular note are the properties of VSAs where encoding is defined by a highly-structured Latin square, such as in HRRs (Plate, 1995) and square matrix representations (Kelly, 2010, see also Table 1 and Figure 4), which encode using circular convolution and matrix multiplication respectively. In such VSAs, we find that for encoding and decoding to be successful, the structure of the encoded data must be orthogonal to the structure of the Latin square used for encoding, or else the data's structure becomes confounded with the structure of the Latin square. If a random, that is, a highly unstructured, Latin square defines encoding, data with local structure can be encoded and decoded successfully. Shuffling (i.e., randomly permuting) data before encoding with a structured Latin square is mathematically equivalent to encoding using an unstructured Latin square. Shuffling thus provides a means to use circular convolution efficiently and successfully to encode relations between vectors of locally structured data. We illustrate in detail the success of this technique.

Shuffling allows for encoding and decoding with minimal noise in both (1) vectors with local structure, such as images, and (2) vectors with non-local structure, such as the structures produced by repeated application of an encoding operation to random vectors. Thus, regardless of the nature of the structure, shuffling provides a tool for hiding or protecting that structure from the structure of vector operations.

There are two areas for future research based on this work. One is the mathematical/ theoretical exploration of the space of possible encoding and decoding operations, investigating how different Latin squares with different properties affect the VSA that they define.

The other avenue for research is cognitive psychological: exploring the use of non-random representations in VSA cognitive models. Chalmers, French, and Hofstadter (1992) argue that conceptual processes cannot be successfully modelled separately from perceptual processes, and the use of atomic symbols is holding back cognitive modelling by avoiding the important questions of representation formation. To take a first step towards addressing both the theoretical concerns (e.g., Chalmers et al., 1992; Johns & Jones, 2010) and the technical limitations associated with using randomly generated representations to stand for concepts in HRR-based cognitive models, we explored the alternative: using non-random representations. The work we have outlined here is intended as a stepping stone towards building vector-symbolic cognitive models that better address questions of representation.

References

Borsellino A., & Poggio, T. (1972). Holographic aspects of temporal memory and optomotor responses. *Kybernetik 10*, 58-60. doi: 10.1007/BF00288785

Chalmers, D. J., French, R. M., Hofstadter, D. R. (1992). High-level perception, representation, and analogy: A critique of artificial intelligence methodology. *Journal of Experimental and Theoretical Artificial Intelligence, 4*, 185-211. doi: 10.1080/09528139208953747

Eliasmith, C., & Thagard, P. (2001). Integrating structure and meaning: a distributed model of analogical mapping. *Cognitive Science, 25*, 245-286. doi: 10.1016/S0364-0213(01)00036-2

Gabor, D. (1969). Associative holographic memories. *IBM Journal of Research and Development, 13*, 156-159. doi: 10.1147/rd.132.0156

Gayler, R. W. (2003) Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. In Slezak, P. (Ed.), *Proceedings of the Joint International Conference on Cognitive Science* (pp. 133-138). Sydney, Australia: University of New South Wales.

Goebel, R. P. & Lewandowsky, S. (1991). Retrieval measures in distributed memory models. In Hockley, W. E. & Lewandowsky, S. (Eds.), *Relating Theory and Data: Essays on Human Memory in Honor of Bennet B. Murdock* (pp. 509-527). Lawrence Erlbaum Associates.

Hannagan, T., Dupoux, E., & Christophe, A. (2011). Holographic string encoding. *Cognitive Science, 35*, 79-118. doi: 10.1111/j.1551-6709.2010.01149.x

Humphreys, Bain & Pike (1989) Different ways to cue a coherent memory system: A theory for episodic, semantic, and procedural tasks. *Psychological Review, 96*, 208-233. doi: 10.1037/0033-295X.96.2.208

Jamieson, R. K., & Mewhort, D. J. K. (2011). Grammaticality is inferred from global similarity: A reply to Kinder (2010). *The Quarterly Journal of Experimental Psychology, 64*, 209-216. doi: 10.1080/17470218.2010.537932

Johns, B. T., & Jones, M. N. (2010). Evaluating the random representation assumption of lexical semantics in cognitive models. *Psychonomic Bulletin & Review*, *17*, 662-672. doi: 10.3758/pbr.17.5.662

Jones, M. N., & Mewhort, D. J. K. (2007). Representing word meaning and order information in a composite holographic lexicon. *Psychological Review, 114*, 1-37. doi: 10.1037/0033-295X.114.1.1

Kahana (2002) Associative symmetry and memory theory. *Memory & Cognition, 30*, 823-840. doi: 10.3758/BF03195769

Kanerva, P. (1996). Binary spatter-coding of ordered k-tuples. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, B. Sendhoff (Eds.), *Artificial Neural Networks – International Conference on Artifical Neural Networks 96 Proceedings* (pp. 869-873). Bochum, Germany.

Kanerva, P. (2009). Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation 1*, 139-159. doi: 10.1007/s12559-009-9009-8

Kelly, M. A. (2010). Advancing the theory and utility of holographic reduced representations (Master's thesis, Queen's University, Canada). *ProQuest Dissertations and Theses*, retrieved from http://search.proquest.com/docview/853293422

Longuet-Higgins, H. C. (1968). Holographic Model of Temporal Recall. *Nature, 217*, 104. doi: 10.1038/217104a0

Metcalfe-Eich, J. (1982). A composite holographic associative recall model. *Psychological Review, 89*, 627–661. doi: 10.1037/0096-3445.119.2.145

Mitchell, J., & Lapata, M. (2010). Composition in distributional models of semantics. *Cognitive Science, 34,* 1388-1429. doi: 10.1111/j.1551-6709.2010.01106.x

Murdock, B. B. (1982). A theory for the storage and retrieval of item and associative information. *Psychological Review, 89*, 609–626. doi: 10.1037/0033-295X.89.6.609

Murdock, B. B. (1993). TODAM2: a model for the storage and retrieval of item, associative and serial-order information. *Psychological Review, 100*, 183–203. doi: 10.1037/0033-295X. 100.2.183

Plate, T. A. (1994). Distributed representations and nested compositional structure (Doctoral dissertation, University of Toronto, Canada). *ProQuest Dissertations and Theses*, retrieved from http://search.proquest.com/docview/304158858

Plate, T. A. (1995). Holographic reduced representations. *IEEE Transactions on Neural Networks, 6*, 623–641. doi: 10.1109/72.377968

Plate, T. A. (1997). A common framework for distributed representation schemes for compositional structure. In Maire, F., Hayward, R., & Diederich, J., (Eds.), *Connectionist Systems for Knowledge Representation and Deduction* (pp. 15-34). Queensland University of Technology.

Plate, T. A. (2000a) Randomly connected sigma-pi neurons can form associative memories. *Network: Computation in Neural Systems*, *11*, 321-332. doi:10.1088/0954-898X_11_4_305

Plate, T. A. (2000b). Analogy retrieval and processing with distributed vector representations. *Expert Systems: The International Journal of Knowledge Engineering and Neural Networks, 17*, 29-40. doi: 10.1111/1468-0394.00125

Plate, T. A. (2003). *Holographic Reduced Representations*. CSLI Lecture Notes Number 150, CSLI Publications, Stanford, CA.

Pribram, K. H. (1969). The neurophysiology of remembering. *Scientific American*, *220* (1), 73-86.

Rasmussen, D., & Eliasmith, C. (2011). A neural model of rule generation in inductive reasoning. *Topics in Cognitive Science, 3*, 140–153. doi: 10.1111/j.1756-8765.2010.01127.x

Rutledge-Taylor, M. F., Vellino, A., & West, R. L. (2008). A holographic associative memory recommender system. In *Proceedings of the Third International Conference on Digital Information Management* (pp. 87-92). London, UK.

Rutledge-Taylor, M. (2010). Dynamically structured holographic memory: A hybrid of discrete and distributed representation (Doctoral dissertation, Carleton University, Canada). *ProQuest Dissertations and Theses*, retrieved from http://search.proquest.com/docview/882379209

Sahlgren, M., Holst, A. & Kanerva, P. (2008). Permutations as a means to encode order in word space. In B. C. Love, K. McRae, & V. M. Sloutsky (Eds.), *Proceedings of the 30th Annual Conference of the Cognitive Science Society* (pp. 64-70). Austin, TX: Cognitive Science Society.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence, 46*, 159-216. doi: 10.1016/0004-3702(90)90007-M

Thagard, P., & Stewart, T. C. (2011). The Aha! experience: Creativity through emergent binding in neural networks. *Cognitive Science, 35*, 1-33. doi: 10.1111/j.1551-6709.2010.01142.x

Willshaw, D. J. (1981). Holography, associative memory, and inductive generalization. In Hinton, G. E., & Anderson, J.A., (Eds.), *Parallel Models of Associative Memory* (pp. 83-104). Lawrence Erlbaum Associates.

Author Notes

Matthew A. Kelly, Institute of Cognitive Science, Carleton University; Dorothea Blostein, School of Computing Science, Queen’s University; D. J. K. Mewhort, Department of Psychology, Queen’s University.

The research was supported by the R. S. MacLaughlin Fellowship and Ontario Graduate Scholarship awarded to the first author and by grants to the second and third authors from the Natural Sciences and Engineering Research Council of Canada. The paper is based on a master’s thesis by Matthew Kelly presented to the School of Computing Queen’s University (August, 2010) supervised by the junior authors. The authors thank the reviewers for their extensive comments on earlier drafts of this paper. Their knowledge and insight greatly contributed to the final version.

Correspondence concerning this article should be addressed to Matthew A. Kelly, who is now at the Institute for Cognitive Science, Carleton University, 1125 Colonel By Drive, Ottawa, ON, Canada K1S 5B6.   Electronic Mail: matthew_kelly2@carleton.ca.

Table 1. Comparison of VSAs and tensor representations.

Vector Symbolic Architecture	Representation	Encoding ⊠	Encoding Time Complexity	Trace Composition	Decoding Accuracy
Tensor representations (Smolensky, 1990)	rank k tensor of n^k real values	tensor product	$O(n^k)$	addition	1
HRRs (Plate, 2003, chap. 3)	vector of n real values	*, circular convolution	$O(kn \log n)$	addition	0.71
Frequency-domain HRRs (Plate, 2003, chap. 4)	vector of n complex values on the unit circle	$\circ$, element-wise multiplication	$O(kn)$	normalized addition	1
Binary spatter codes (Kanerva, 1996)	vector of n binary values	bit-wise XOR	$O(kn)$	majority	1
MAP coding (Gayler, 2003)	vector of n real values, initially +1s and -1s	$\circ$, element-wise multiplication	$O(kn)$	addition	1 or 0.58 (see text)
Square matrix representations (Kelly, 2010)	$n^{1/2}$ x $n^{1/2}$ square matrix of n real values	matrix multiplication	$O(kn^{3/2})$	addition	0.70
Used in CHARM (Metcalfe-Eich, 1982)	vector of n real values	⧆, truncated aperiodic convolution	$O(kn \log n)$	addition	0.70

Note. Complexity expressed in terms of vectors (or square matrices) of n values and the number of vectors associated together, k.

Figure Captions

Figure 1. Left: a 512 x 683 pixel image. Centre: failed attempt to retrieve image from HRR when the image is stored as concatenated columns. Right: grainy approximation successfully retrieved when image pixels are shuffled before storage.

Figure 2. Figure adapted from Plate (1995). The circular convolution (*) of two three-dimensional vectors $\mathbf{a} = [a_1, a_2, a_3]$ and $\mathbf{b} = [b_1, b_2, b_3]$ is decoded by circular correlation (#) to recover an approximation of **b**. When the conditions named by Plate are met, ξ and **w** will be zero mean noise, such that $\mathbf{a} \# (\mathbf{a} * \mathbf{b}) \approx \mathbf{b}$.

Figure 3. Left: the outer-product $\mathbf{a}\,\mathbf{b}^T$ of the items **a** ($\mathbf{a} = [a_1, a_2, a_3]$) and **b** ($\mathbf{b} = [b_1, b_2, b_3]$). Right: the circular convolution $\mathbf{a}*\mathbf{b}$ of **a** and **b**. Centre: A Latin square which conveys how to calculate the circular convolution of a pair of three-dimensional vectors using their outer-product. The first, second, and third elements of the convolution are respectively the sum of the elements of the outer-product located in a position that is marked as a 1, 2 or 3 in the Latin square. A different Latin square would define a different encoding operation.

Figure 4. The association of $\mathbf{a} = [a_1, a_2, a_3, a_4]$ and $\mathbf{b} = [b_1, b_2, b_3, b_4]$ in real-valued VSAs (shown on left) is a compression of $\mathbf{a}\,\mathbf{b}^T$ (right) defined by a Latin square (centre). Black shading indicates omitted matrix elements. Circular convolution (*) is characterized by a full Latin square, whereas truncated aperiodic convolution ($\boxed{*}$) and element-wise multiplication ($\circ$)

are characterized by partial Latin squares. Square matrix multiplication can be understood as a compression of **a b**^{T} defined by a partial Latin square, where **a** and **b** are vectors formed by concatenating the columns of matrices **A** and **B**.

Figure 5. Latin squares that describe circular convolution (on left) and circular correlation (right) as sums of elements of an outer-product of vectors of dimensionality $n = 4$. The Latin square for circular convolution is symmetrical along the axis defined by the main diagonal (from top-left to bottom-right) and thus describes a commutative encoding operation. The Latin square for circular correlation is asymmetric, and thus describes a non-commutative operation.

Figure 6. When using circular convolution to convolve highly ordered data, the order in the data becomes confounded with the ordered structure of the compression, badly distorting the data on retrieval. A 512 x 683 pixel digital image is transformed into a 349 696 dimensional vector and associated with itself using circular convolution (denoted by *) to produce a trace. The image is used again to decode from the trace using circular correlation (#) to produce an approximation that should resemble the original image, but in this case is a smudge, unrecognizable to the eye. The cosine between the approximation and the original is 0.50. Performance is improved by shuffling (see Figure 7).

Figure 7. The elements of the image are shuffled. The shuffled image is convolved with itself (*) to produce a trace, then used again to decode from the trace (#) to produce an approximation

of itself. Shuffling is reversed to reveal a good apporximation (cosine = 0.82; compare Figure 6).

Figure 8. Shuffling to store and retrieve a hetero-association. Two 512 x 683 pixel digital images (denoted by **c** and **g**) are each transformed by concatenating columns into 349 696 dimensional vectors and permuted with a randomly selected permutation *p.* The permuted images are associated using circular convolution (*). Approximations of each image are retrieved with circular correlation (#) by using the other image as a cue and then taking the inverse permutation p^{-1} of the result.

Figure 9. Plots of vector elements on complex plane, *n* = 349 696. Shown clockwise are: (a) a vector of values sampled from the unit circle, as used in frequency-domain HRRs, representing the ideal for encoding and decoding with convolution; (b) the Fourier transform of a vector of values selected from a normal distribution with a mean of zero and variance of 1/*n*, as used in HRRs; (c) the Fourier transform of the image of the grave used in Figures 6, 7 and 11, **note the order of magnitude difference in scale**; (d) the Fourier transform of the shuffled grave image.

Figure 10. A 380 x 228 pixel digital image of a circle is transformed into a 86 640 dimensional vector and convolved with itself (*) to produce a trace. Decoding from the trace (#) retrieves a recognizable but systematically distorted (i.e., "smudged") approximation (cosine = 0.81).

Figure 11. Image from Figure 10 is shuffled before encoding and decoding. The unshuffled approximation is a recognizable but randomly distorted (i.e., "noisy") approximation of the original. The cosine between the approximation and the original is 0.81.

Figure 12. Shown clockwise are approximations produced by shuffling within the image the location of 1-pixel, 2-pixel, 9-pixel, 20-pixel, 100-pixel, and 500-pixel sequences, i.e., vertical strips of varying length. The cosine values indicate similarity to the original image.

***Figure 13*.** Left, Original image. Centre, Streaked approximation retrieved when image is discrete cosine transformed before encoding (cosine = 0.72). Right, Grainy approximation retrieved when image is shuffled before encoding (cosine = 0.82).

Figure 1

$$\mathbf{a} * \mathbf{b} = \begin{array}{|c|}\hline a_1b_1 + a_2b_3 + a_3b_2 \\ \hline a_1b_2 + a_2b_1 + a_3b_3 \\ \hline a_1b_3 + a_2b_2 + a_3b_1 \\ \hline \end{array}$$

$$\mathbf{a} \# (\mathbf{a} * \mathbf{b}) = \begin{array}{|r|}\hline b_1 (a_1^2 + a_2^2 + a_3^2) + b_2a_1a_3 + b_3a_1a_2 + b_2a_1a_2 \\ + b_3a_2a_3 + b_2a_2a_3 + b_3a_1a_3 \\ \hline b_2 (a_1^2 + a_2^2 + a_3^2) + b_1a_1a_2 + b_3a_1a_3 + b_1a_1a_3 \\ + b_3a_2a_3 + b_1a_2a_3 + b_3a_1a_2 \\ \hline b_3 (a_1^2 + a_2^2 + a_3^2) + b_1a_1a_2 + b_2a_2a_3 + b_1a_2a_3 \\ + b_2a_1a_3 + b_1a_1a_3 + b_2a_1a_2 \\ \hline \end{array}$$

$$= \begin{array}{|c|}\hline b_1 (1 + \xi) + w_1 \\ \hline b_2 (1 + \xi) + w_2 \\ \hline b_3 (1 + \xi) + w_3 \\ \hline \end{array} = (1 + \xi)\,\mathbf{b} + \mathbf{w}$$

Figure 2

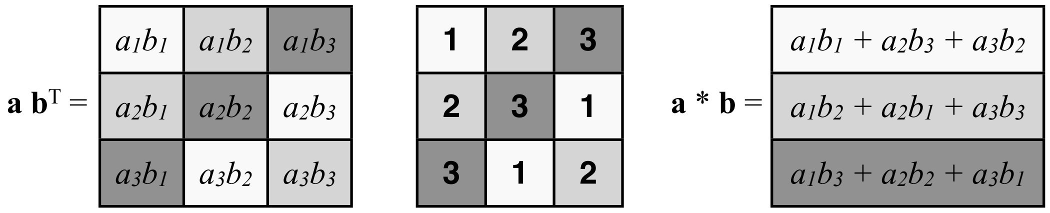

$\mathbf{a}\,\mathbf{b}^T =$
a_1b_1 a_1b_2 a_1b_3
a_2b_1 a_2b_2 a_2b_3
a_3b_1 a_3b_2 a_3b_3
1 2 3
2 3 1
3 1 2
$\mathbf{a} * \mathbf{b} =$
$a_1b_1 + a_2b_3 + a_3b_2$
$a_1b_2 + a_2b_1 + a_3b_3$
$a_1b_3 + a_2b_2 + a_3b_1$

Figure 3

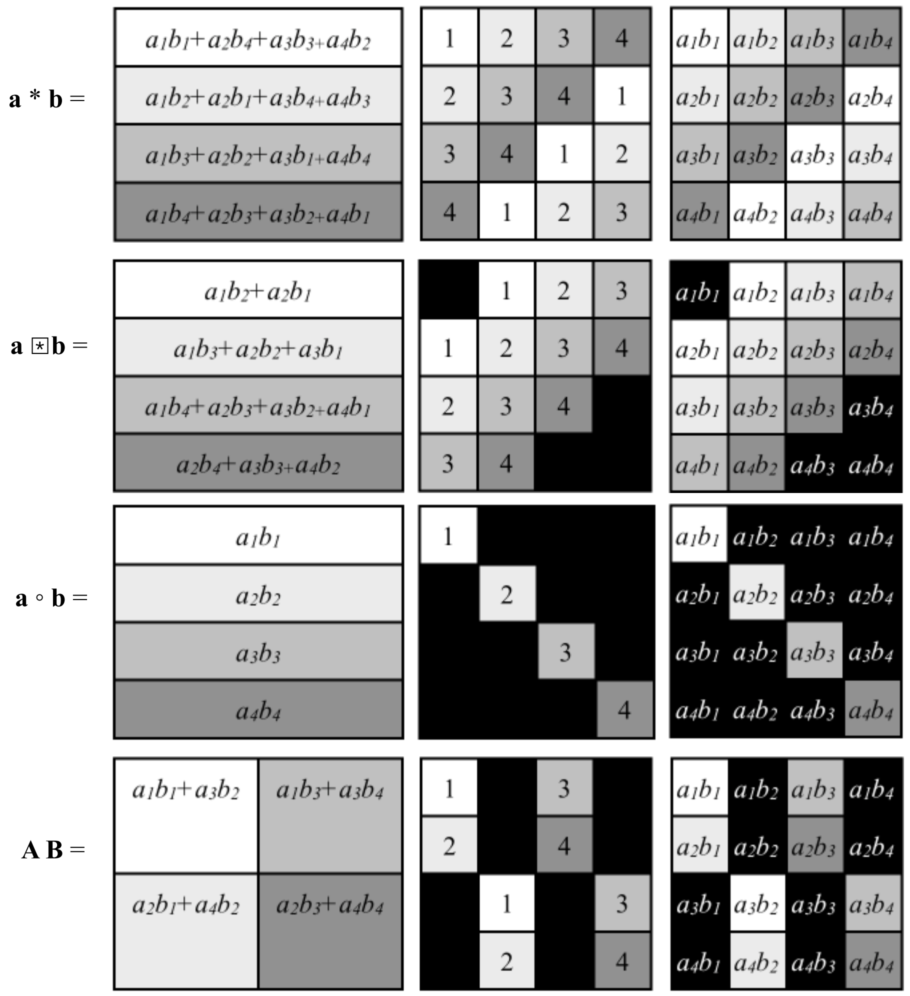

a * b =
a1b1+a2b4+a3b3+a4b2
a1b2+a2b1+a3b4+a4b3
a1b3+a2b2+a3b1+a4b4
a1b4+a2b3+a3b2+a4b1
a ⊡ b =
a1b2+a2b1
a1b3+a2b2+a3b1
a1b4+a2b3+a3b2+a4b1
a2b4+a3b3+a4b2
a ∘ b =
a1b1
a2b2
a3b3
a4b4
A B =
a1b1+a3b2
a1b3+a3b4
a2b1+a4b2
a2b3+a4b4

Figure 4

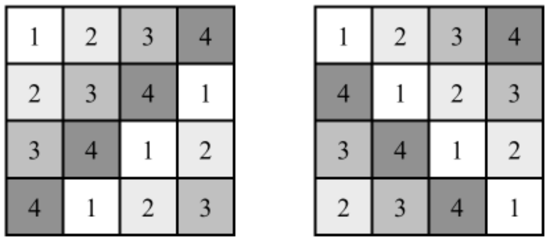


Figure 5

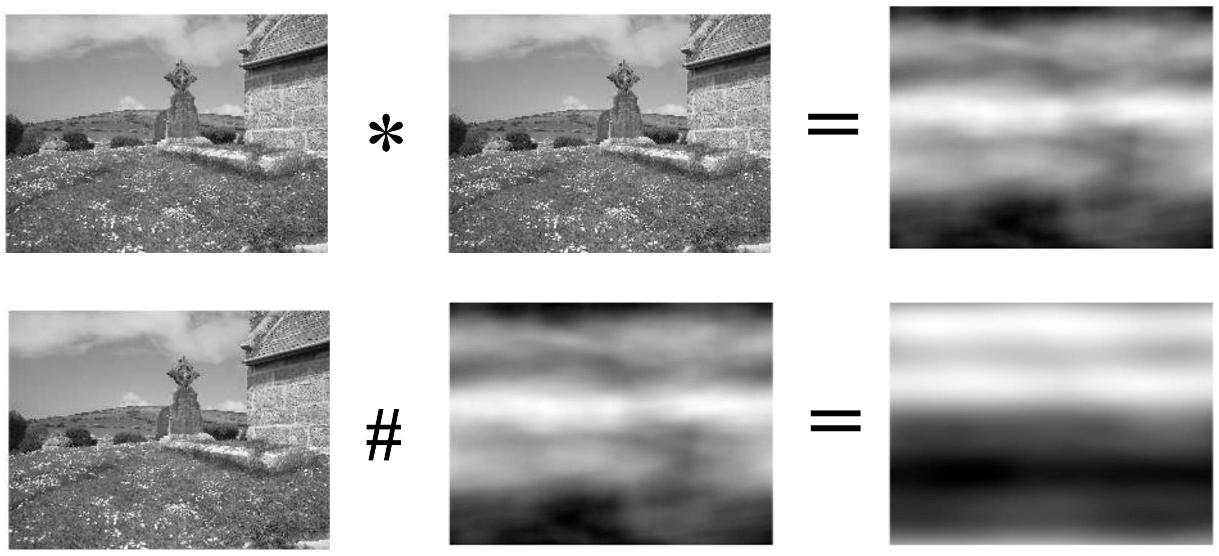


Figure 6

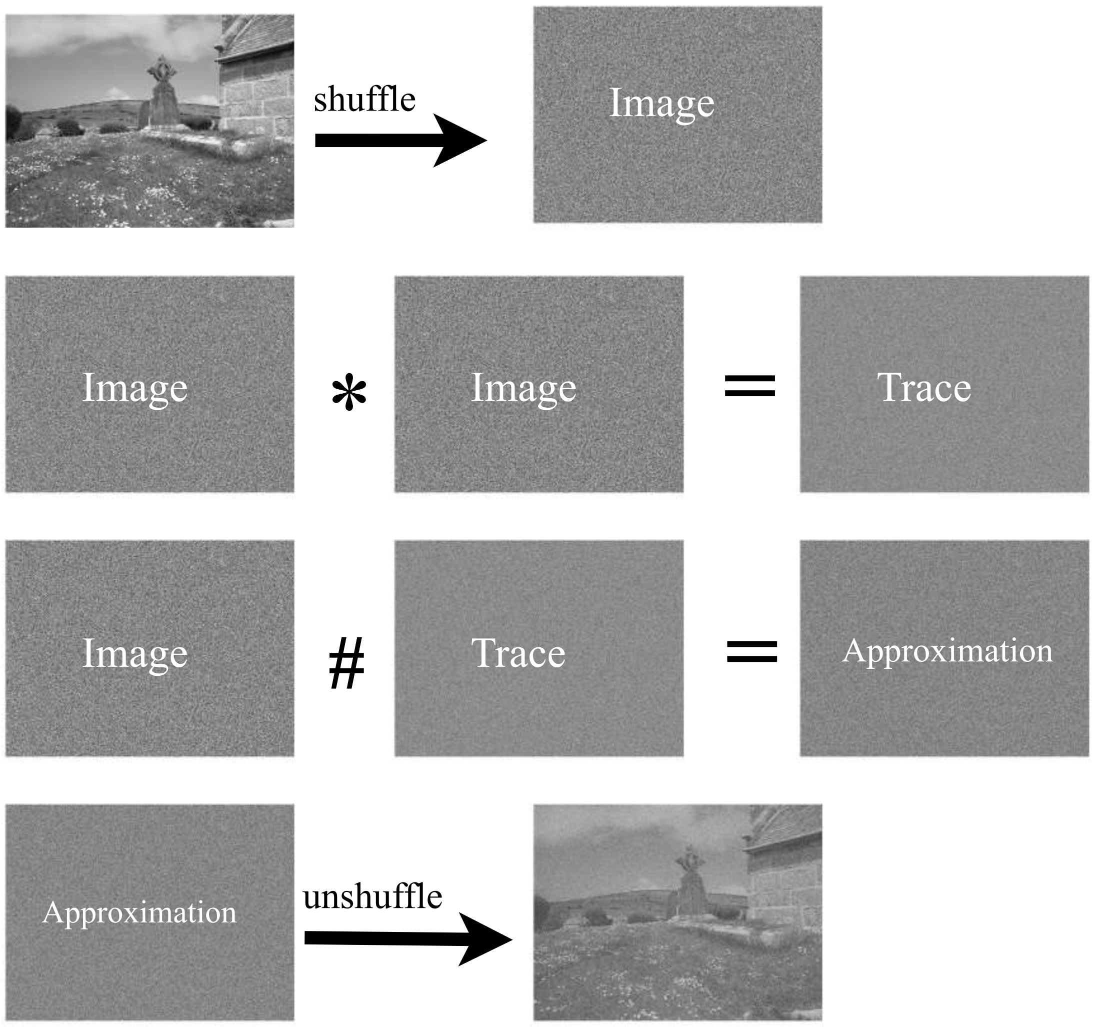

shuffle
Image
Image
*
Image
=
Trace
Image
#
Trace
=
Approximation
Approximation
unshuffle

Figure 7

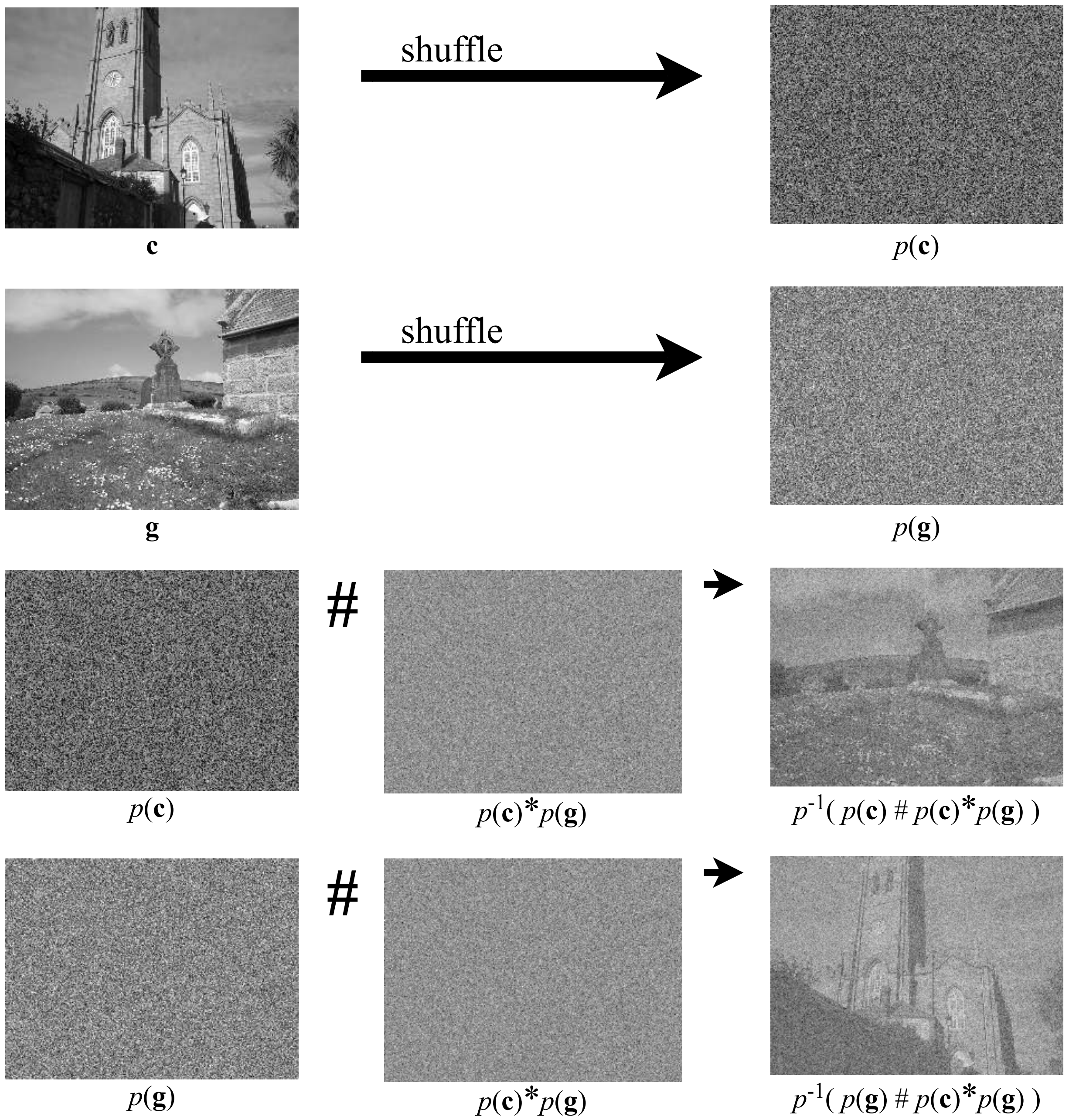
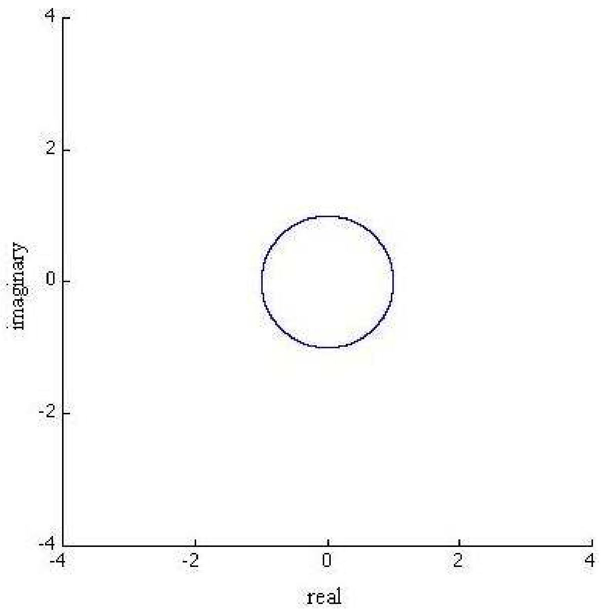
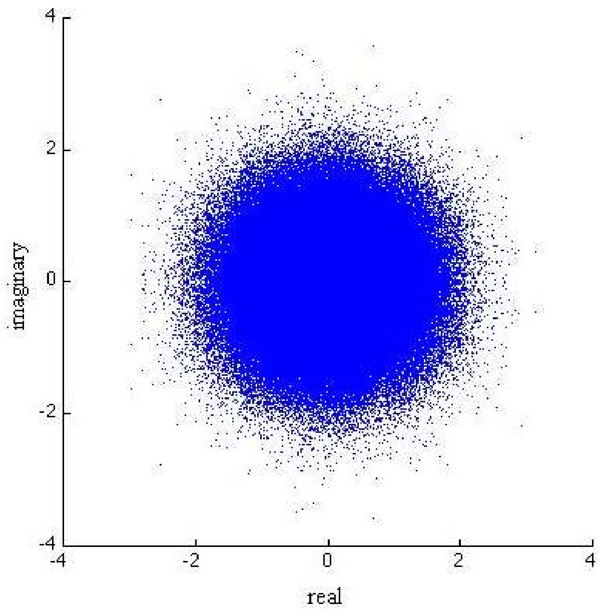
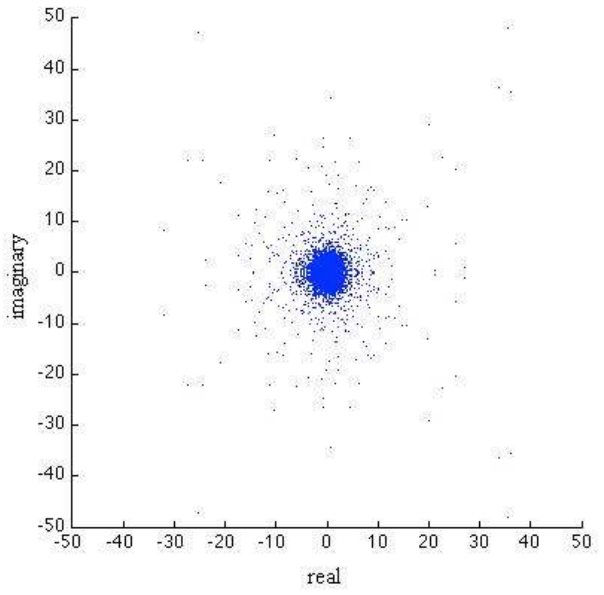

shuffle
shuffle
$\mathbf{c}$
$p(\mathbf{c})$
$\mathbf{g}$
$p(\mathbf{g})$
#
#
$p(\mathbf{c})$
$p(\mathbf{c})*p(\mathbf{g})$
$p^{-1}(\ p(\mathbf{c})\ \#\ p(\mathbf{c})*p(\mathbf{g})\ )$
$p(\mathbf{g})$
$p(\mathbf{c})*p(\mathbf{g})$
$p^{-1}(\ p(\mathbf{g})\ \#\ p(\mathbf{c})*p(\mathbf{g})\ )$

Figure 8


(a) Circular vector on complex plane


(b) Gaussian vector on complex plane


(c) Grave image on complex plane

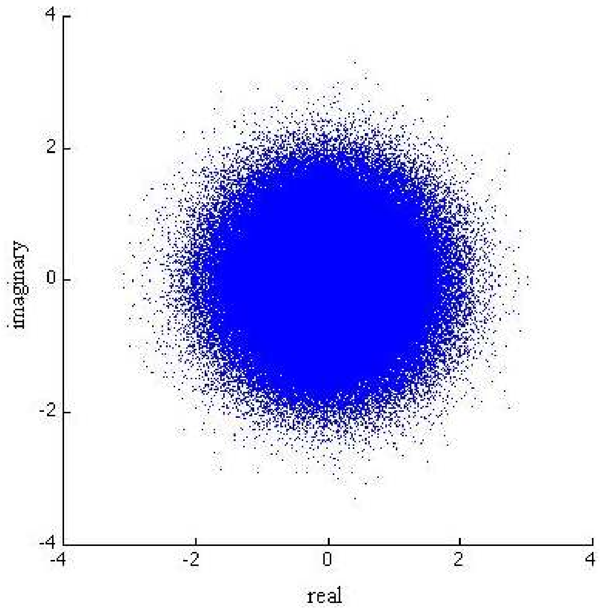

(d) Shuffled grave image on complex plane

Figure 9

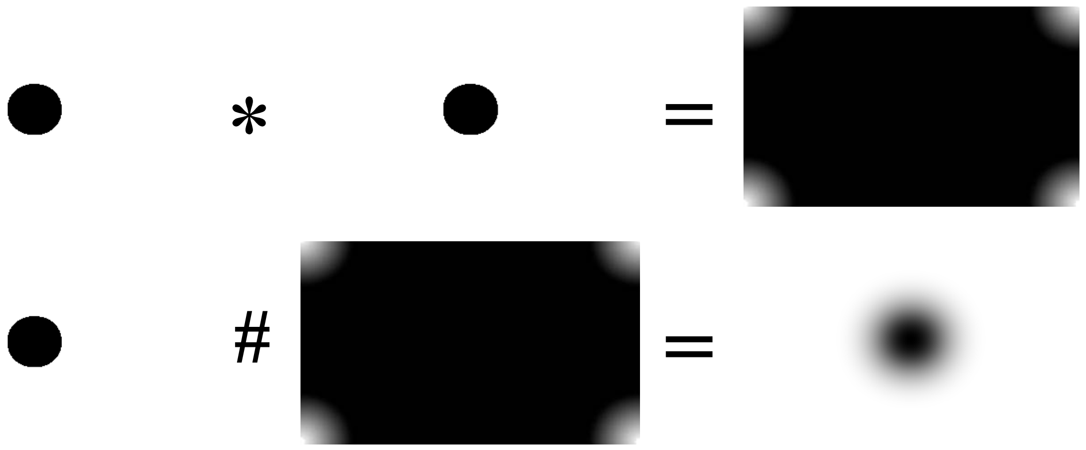


Figure 10

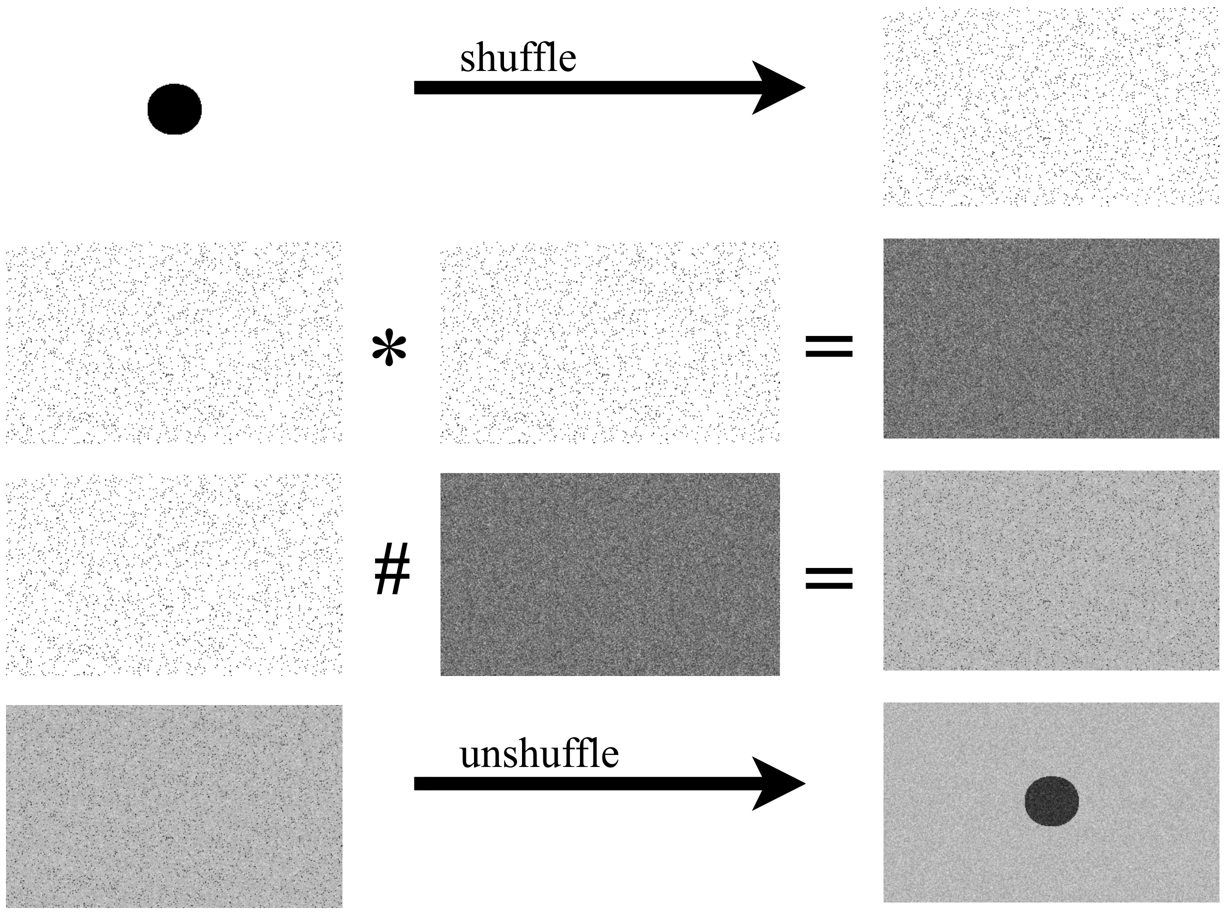
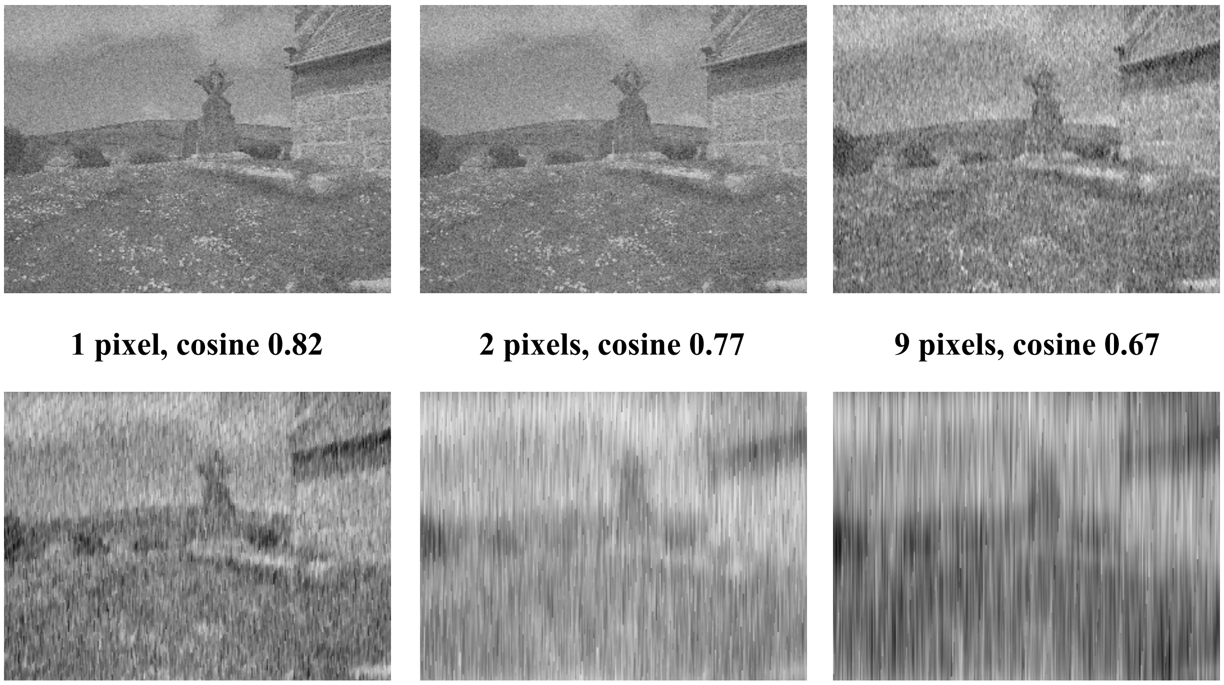


Figure 11



1 pixel, cosine 0.82 **2 pixels, cosine 0.77** **9 pixels, cosine 0.67**

20 pixels, cosine 0.64 **100 pixels, cosine 0.58** **500 pixels, cosine 0.52**

Figure 12

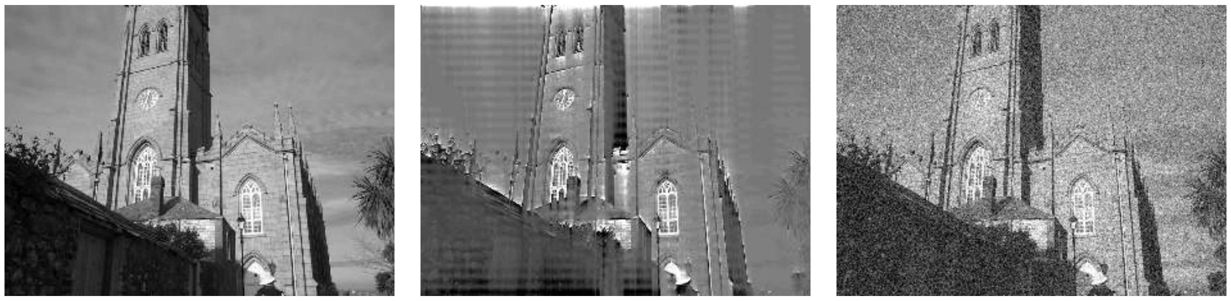


Figure 13