ISSN 2007-9737

Measuring the Storing Capacity of Hyperdimensional Binary Vectors

Job Isaías Quiroz Mercado, Ricardo Barrón Fernández, Marco Antonio Ramírez Salinas

Instituto Politécnico Nacional,
Centro de Investigación en Computación,
Mexico

jobquiroz@hotmail.com, barron2131@gmail.com, mars@cic.ipn.mx

Abstract. Hyperdimensional computing is a model of computation based on the properties of high-dimensional vectors. It combines characteristics from artificial neural networks and symbolic computing. The area where hyperdimensional computing can be applied is natural language processing, where vector representations are already present in the form of word embedding models. However, hyperdimensional computing encodes information differently, its representations can include the distributional information of a word in a given context and it can also account for its semantic features. In this work, we investigate the storing capacity of hyperdimensional binary vectors. We present two different configurations in which semantic features can be encoded and measure how many can be stored, and later retrieved, within a single vector. The results presented in this work lay the foundation to develop a concept representation model with hyperdimensional computation.

Keywords. Hyperdimensional computing, vector symbolic architectures, reduced representations.

1 Introduction

Hyperdimensional computing is a new model of computation based on the properties of high-dimensional vectors [1]. Hyperdimensional computing takes ideas from artificial neural networks, because it performs distributed processing, and it is also inspired on symbolic computing because complex structures, such as hierarchical trees or sequences, can be formed by manipulating symbols (vectors) representing simpler objects.

The set of architectures working under the hyperdimensional computing principles is known as Vector Symbolic Architectures (VSAs) [2]. In artificial neural networks, the use of vectors comes implicitly with the architecture description, in contrast, high-dimensional vectors in VSAs are not only part of the architecture but are the basic computing entities itself [1]. VSAs have been increasing in popularity during the last years; they have been applied in cognitive architectures [3], in analogy-based reasoning [4], to represent sequences and hierarchical structures [5, 6], and in pattern recognition [7].

Hyperdimensional computing has also been used in natural language processing (NLP). The Random Indexing technique [8], for example, uses high-dimensional vectors to create vector representations for texts.

This technique has a similar approach to vector semantic models, the current state-of-the-art models in most NLP applications. These models, commonly known as *word embeddings*, provide vector representation for words, paragraphs and entire documents, and they are based on the distributional hypothesis of meaning [9], which states that words with similar meaning tend to occur in similar contexts.

We propose a method for representing concepts using hyperdimensional computing principles, which are based on knowledge rather than in the distributional information of a word. These representations are created from a list of semantic features [10] encoded within a single high-dimensional vector. This work focuses on measuring the limit number of semantic features

Table 1. Properties of arithmetic operations

Operation	Symbol	Properties
Addition (bundling)	+	- n-ary function, - Combines a set of vectors, - Elementwise majority with ties broken at random, - Resultant vector is similar to argument vectors.
Multiplication (binding)	⊗	- 2-ary function, - Combination of a pair of vectors, - Elementwise exclusive-or, - Invertible operation, - Distributes over addition, - Resultant vector is dissimilar to vectors being multiplied.

that a high-dimensional binary vector can successfully store. Our experimental results will be used for selecting the appropriate dimensionality of vectors within a hyperdimensional computing system still in development.

The rest of the paper is organized as follows. Section 2 explains the general properties of hyperdimensional computing and describes how to encode semantic features into high-dimensional vectors. In Section 3 we present the experimental results that are later discussed in Section 4. Finally, Section 5 draws the conclusions of the work.

2 Methods

The most distinctive property of high-dimensional spaces (i.e. N > 1,000) is the *tendency to orthogonality*. This means that most of the space is *nearly* orthogonal to any given point [11]. For instance, if two random binary vectors are generated, it is highly probable that the Hamming distance between them is approximately N/2. As a consequence of this, arithmetic operations between this type of vectors yields to a new way for encoding information. In this section, we describe the basic arithmetic operators for high-dimensional vectors: addition and multiplication, and how they can be used to encode a list of semantic features within a single high-dimensional vector to represent a concept.

2.1 Arithmetic Operations

In general, there are two basic operations in hyperdimensional computing: binding (or multiplication) and bundling (or addition). These operations are used to encode, map and retrieve information. In this work we implement a framework called Binary Spatter Codes (BSC) [12], which uses binary vectors, see Table 1.

To represent higher level objects through arithmetic operations, first a set of *primitive* objects have to be defined and be associated with randomly generated vectors. The simplest method to combine a set of *primitive* objects is by bundling them together through addition. For instance, to create the vector class *Animals* the vector from each animal specie in the system can be added together, (1):

$$Animals = Dog + Cat + Birds + \ldots \quad . \tag{1}$$

While this method might be useful for some small systems, it does not allow to encode more complex relations as other methods. Gallant & Okaywe [5] presented another method for encoding a single sentence where, rather than bundling all vectors together, the subject, verb and object are multiplied by a *label vector*, after which all the vectors are added. For instance, the sentence “Mary loves pizza” will be encoded as in (2):

$$S = subj \otimes Mary + vrb \otimes loves + obj \otimes pizza\,. \tag{2}$$

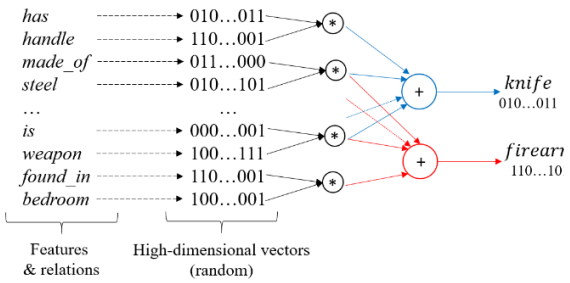

Fig. 1. Encoding concept's definition based on its semantic features. Two concept vectors are geometrically closer if they share the same semantic features

Table 2. List of semantic features for the concept *knife*

Feature	Classification
has a handle	Visual-form and surface
made of steel	Visual-form and surface
is shiny	Visual-form and surface
used for cutting	Function
used for killing	Function
is sharp	Tactile
is dangerous	Encyclopedic
found in kitchens	Encyclopedic
is a weapon	Taxonomic
is a utensil	Taxonomic

Each vector added can be a randomly generated vector, or be another encoded sentence itself. Unlike the previous method, the order is encoded within the final vector, and therefore, the vectors produced for "Mary loves pizza" and "Pizza loves Mary" will be different.

2.2 Encoding Semantic Features

Semantic features represent the basic conceptual components of meaning for a word [10]; they try to establish the meaning of a word in terms of its relationships with other words. Semantic features must be obtained from humans, either directly, in studies where humans are tasked with enumerating properties from a given concept, or indirectly, by taking information from knowledge bases. They incorporate different type of information, including both perceptual (e.g., shape and color), and non-perceptual attributes (e.g. taxonomic and functional). Table 2 gives an example of the semantic features in the McRae dataset [10] for the concept *knife*.

Within a semantic feature it is possible to identify two different parts: a relation (e.g. *has*, *is*, *used_for*) and a feature value (e.g. *handle, weapon*, *cutting*). From this observation, we propose to construct a vector representation as shown in equation (3):

$$Concept = \sum_{i=1}^{n} Relation_i \otimes Feature_i . \tag{3}$$

Vectors representing relations are called *relation* vectors. They represent the main semantic relations between two different words. Vectors representing features are called *feature* vectors. Both feature and relation vectors can be selected at random, or be an encoded vector itself. For instance, the *knife* concept will be encoded as (4):

$$knife = has \otimes handle + made_{of} \otimes steel + is \otimes utensil. \tag{4}$$

The hypothesis behind this encoding method is that another vector with a similar set of semantic features will be close to the original vector, Fig.1. In the case of the BSC the metric use to measure distance between two vectors is Hamming distance.

An interesting property of the multiplication operation is its invertibility. This means that it is possible to extract back a previously multiplied vector. For example, in (5) the vector Z is the multiplication between X and Y, however, since the XOR operation is its own inverse, it is possible to obtain X back by multiplying Z by Y (6):

$$Z = X \otimes Y, \tag{5}$$

$$Z \otimes Y = (X \otimes Y) \otimes Y,$$

$$X \otimes Y \otimes Y, \tag{6}$$

$$X \otimes 0,$$

$$X.$$

As a consequence of the invertibility property, the representations created by the proposed

Table 3. Precision for retrieving features

n_{pairs}	N = 2,000	N = 5,000	N = 7,000	N = 10,000
20	1.0	1.0	1.0	1.0
30	1.0	1.0	1.0	1.0
40	0.88	0.97	0.98	1.0
50	0.68	0.84	0.84	0.90
60	0.58	0.51	0.53	0.75
70	0.35	0.31	0.31	0.32
80	0.32	0.20	0.13	0.12

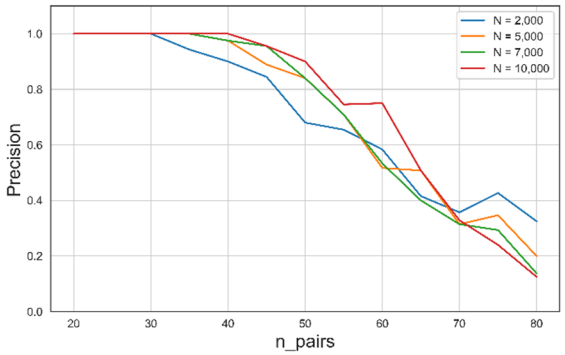

Fig. 2. Precision for the retrieving of features from high-dimensional vectors

method are *interpretable*, that is to say, once the final representation of a concept is created it is possible to analyze what features were included within the representation. For example, when the *knife* vector produced in (4) is multiplied by the relation vector *has*, the resultant vector would include the vector *handle* plus an additional noise vector:

$$knife \otimes has = has \otimes handle \otimes has + \cdots + is \otimes utensil \otimes has \; handle + \cdots + is \otimes utensil \otimes has \; handle + noise. \quad (7)$$

This *noise* vector is the addition of all the other relation-feature pairs that do not have *has* as a relation vector. Due to the properties of the multiplication operation, this noise vector will be nearly orthogonal to *handle* and can be eliminated through an associative memory, an operation called *clean-up*. For more details and examples of the hyperdimensional computing operations and the use of autoassociative memories in VSAs refer to [1, 4, 5].

3 Experimental Results

In this section, we present the results of two experiments performed to test the maximum storing capacity of high-dimensional binary vectors. Each experiment was focused in a specific semantic features configuration. In each case, we quantified the storing capacity of hyperdimensional binary vectors for different vector sizes. The code for all the experiments is publicly available repository[1].

3.1 One Relation – One Feature Configuration

In this first experiment, we took a simple semantic feature configuration where each relation is associated with a single feature.

While this configuration is not common to find in knowledge bases, mainly because there are always more features than relations, it is the configuration storing the largest number of orthogonal vectors.

The parameters for this experiment were: **N**, the dimensionality of the vectors, and $\mathbf{n_{pairs}}$, the number of relation-feature pairs to encode. The encoded vectors have the form:

$$C = R_1 \otimes f_1 + R_2 \otimes f_2 + \cdots + R_{n_{pairs}} \otimes f_{n_{pairs}}. \quad (8)$$

Each relation and feature vectors (R_i and f_i, respectively) was randomly generated and paired up with another vector to create the concept vector C. After this, multiplications $(C \otimes R_i)$ and *clean-up* operations were performed to extract back each feature (f_i).

Table 2 shows the precision, the relation between the number of encoded and retrieved features, for different **N** and $\mathbf{n_{pairs}}$ values. Fig. 2. illustrates these results.

The results presented indicate that the storing capacity of high-dimensional binary vectors do not increase in a linear fashion. As the dimensionality *N* increases the storing capacity increases but not in the same proportion.

For instance, the maximum number of relation-feature pairs that a 2,000-size vector can store is 30 pairs, by increasing the size to 10,000 the capacity increases to 40 pairs.

[1] https://github.com/jobquiroz/StoringCapacity_HDC

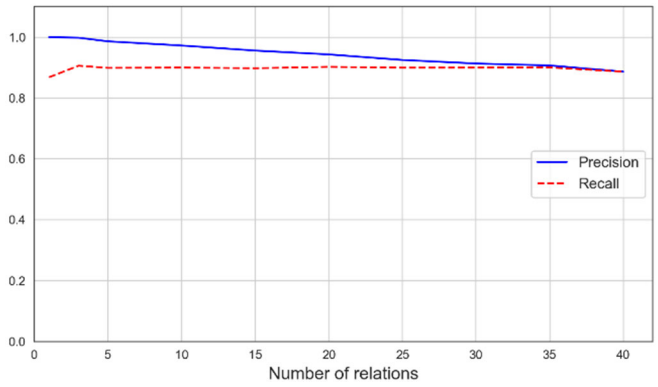

Fig. 3. Precision and recall for N = 2,000 and storing 40 features in *one relation – multiple features* configuration

Table 4. Precision and recall for feature retrieving using *one relation-multiple features* configuration ($\mathbf{n_{feat}}$ = 40)

n_rels	N = 2,000		N = 5,000		N = 7,000		N = 10,000	
	P	R	P	R	P	R	P	R
1	1.0	0.87	1.0	0.97	1.0	0.95	1.0	0.99
5	0.99	0.89	0.99	0.98	1.0	0.99	1.0	0.99
10	0.97	0.90	0.99	0.98	0.99	0.99	0.99	0.99
20	0.94	0.91	0.98	0.97	0.99	0.99	0.99	0.99
40	0.88	0.88	0.97	0.97	0.98	0.98	1.0	1.0

Given that the intended application for this method is to represent concept's definitions, 40 pairs is enough to describe the most important semantic features for a concept.

3.1 One Relation – Multiple Features Configuration

In this experiment, we focus on a more general configuration, where each relation could be associated with more than one feature. This configuration is more common to find in knowledge bases. For instance, in the *knife* concept from Table 1 the relation *is* was associated with five different features.

The parameters for this experiment were: **N**, the dimensionality of the vectors, $\mathbf{n_{rel}}$, the number of relations, and $\mathbf{n_{feat}}$ the total number of features to encode.

Unlike in the previous configuration, there are several ways to combine the set of relations with the set of features. This seems to lead to performing a full combinatorial analysis. However, since this experiment was focused on measuring the storing capacity of vectors, we care about the *number* of features associated to each relation rather than *which* features are with each relation. This assumption reduces the number of possibilities to analyze. Equation (8) express the possible ways to encode a concept with $\mathbf{n_{rel}}$ relations and $\mathbf{n_{feat}}$ features:

$$C = \sum_{i=1}^{n_{rel}} R_i \otimes \left[\sum_{k=1}^{m_i} f_k^i\right], \quad where \sum_i^{n_{rel}} m_i = n_{feat}\ with\ m_i > 0. \tag{9}$$

A configuration example for $\mathbf{n_{rel}}$ = 3 and $\mathbf{n_{feat}}$ = 6 is shown in (9).

$$C = R_1 \otimes [f_1^1] + R_2 \otimes [f_1^2 + f_2^2 + f_3^2] + R_3 \otimes [f_1^3 + f_2^3]. \tag{10}$$

To simplify the analysis, in this experiment we set a fixed number of features, $\mathbf{n_{feat}}$ = 40, and iterate over different $\mathbf{n_{feat}}$ and dimensionality values, Table 4. The reason for this is that, according to the previous experiment, at $\mathbf{n_{feat}}$ = 40 the precision for the feature retrievals do not

Table 4. Precision and recall for feature retrieving using *one relation-multiple features* configuration ($\mathbf{n_{feat}}$ = 40)

	One relation – one feature	**One relation – multiple features**
Advantage	Straightforward retrieving process.	Less space needed within the definition vector
Disadvantage	More space needed within the definition vector	Retrieving can be ambiguous
Example	$Apple = is \otimes fruit + shape \otimes round + flavor \otimes tasty + has \otimes skin$ $is \otimes Apple = fruit$ (8 vectors encoded)	$Apple = is \otimes fruit + is \otimes round + is \otimes tasty + has \otimes skin$ $is \otimes Apple = [fruit, round, tasty]$ (6 vectors encoded)

reach 1 for most of the vector sizes tested. By leaving $\mathbf{n_{feat}}$ fixed we can observe how the rearrangement between features and relations change the precision of the retrievals.

In this experiment, we include the recall value, which measures the total amount of encoded features f_k^i that were actually retrieved. The recall value for all the measures in the previous experiment was the same than the precision value, meaning that when a feature was retrieved it was always the encoded feature; in this experiment this is not the case as Fig. 3 shows.

The *one relation – multiple features* configuration reorganizes the information by distributing the features among a lower number of relations. This configuration resembles more how concepts are commonly described in knowledge bases like ConceptNet [13].

Unlike in the previous experiment, it was necessary to measure the recall value of the retrieval operations because in some cases the list of retrieved features did not match the list of encoded features. This was especially problematic in lowest-dimension tested (N = 2,000).

4 Discussion

The goal of the present article was to measure the storing capacity of high-dimensional binary vectors following the Binary Spatter Codes framework.

Our experimental results showed that the relation between the increase in the size of the vectors do not maintain a linear relation with the total amount of items encoded.

Unlike other vector representations where each component stores specific information, the representations described in this article distribute the information across all components (holistic processing [1]).

The presented results also indicate that the determining factor in the overall storing capacity of the vectors is not the configuration used for encoding, but the total number of orthogonal vectors stored.

However, the configuration used dictates how the vectors are going to be retrieved. In the first configuration, after the inverse multiplication and the *clean-up* operations are performed, only one feature vector is obtained, while in the second configuration the final output is a list of features.

Table 4 summarizes the advantage and disadvantage of each configuration according to our experimental results.

As the example in Table 4 shows there are general relations (‘is’) that can be substituted by more specific relations (‘shape’, ‘flavor’) to make the retrieving less ambiguous.

However, adding more relations implies using more space. In the case of the goal application for this method, it should be noted that in knowledge bases like ConceptNet [13] and the McRae dataset [10], most concepts are characterized by less than 40 semantic features.

Based on our findings, we propose N = 10,000 as an appropriate vector size for representing concepts based on its semantic features. Vector sizes of 5,000 and 7,000 can also be worth considering if the processing speed is a

constraint. In that case, the maximum number of semantic features to encode must be reduced accordingly.

5 Conclusions

This work presented an empirical exploration of the storing capacity of binary vectors using a VSA framework. We presented some aspects of hyperdimensional computing, a model of computation based on the manipulation of high dimensional vectors, and proposed a method for representing vectors based on a list of its semantic features.

We presented experimental results for encoding and then retrieving semantic features under two types of configurations: *one relation – one feature*, and *one relation – many features*. We identify the main advantage and disadvantage of each configuration and selected the 10,000-size vectors as most appropriate for representing concepts. This result will be later used to further develop this encoding method.

This works lays the foundation from a representation model intending to encode larger knowledge bases, like ConceptNet, for modeling language using hyperdimensional computing.

Acknowledgments

This work has been funded by SIP-IPN under grant SIP-20201415 and by CONACYT scholarship number 666415.

References

1. **Kanerva, P. (2009).** Hyperdimensional computing: An introduction to computing in distributed representation with high dimensional random vectors. Cognitive Computation, Vol. 1, No. 2, pp. 139–159.
2. **Gayler, R. (2003).** Vector Symbolic Architectures answer Jackendoff's challenge for cognitive neuroscience. ICCS/ASCS International Conference on Cognitive Science. Sydney Australia, pp. 133–138.
3. **Snaider, J., Franklin, S. (2014).** Vector LIDA. Procedia Computer Science, Vol. 41, pp. 188–203.
4. **Emruli, B., Sandin, F. (2013).** Analogical Mapping with Sparse Distributed Memory: A simple model that learns to generalize from examples. Cognitive Computation, Vol. 6, pp. 74–88.
5. **Gallant, S., Okaywe, T. (2013)**. Representing objects, relations and sequences. Neural Computation, Vol. 25, No. 8, pp. 2038–2078.
6. **Quiroz, J.I., Barrón, R., Ramírez, M.A. (2017).** Sequence prediction with Hyperdimensional Computing. Research in Computer Science, Vol. 138, pp. 117–126.
7. **Rahimi, A., Datta, S., Kleyko, D., Paxon, E., Olshausen, B., Kanerva, P., Rabaey, J. (2017).** High-Dimensional computing as a nanoscalable paradigm. IEEE Transactions on Circuits and Systems: Regular Papers, Vol. 99, pp. 1–14.
8. **Kanerva, P., Kristofersson, J., Holst, A. (2000).** Random Indexing of text samples for Latent Semantic Analysis. Proceedings of the 22nd Annual Conference of the Cognitive Science Society. New Jersey, USA.
9. **Harris, Z. S. (1954).** Distributional Structure. Word, Vol. 10, No. 2-3, pp. 146–162.
10. **McRae, K., Cree, G., Seidenberg, M., McNorgan, C. (2005).** Semantic feature production norms for a large set of living and nonliving things. Behavior Research Methods, Instruments & Computers, Vol. 37, No. 4, pp. 547–559.
11. **Kanerva, P. (1988).** Sparse Distributed Memory. Cambridge, MA: Bradford/MIT Press.
12. **Kanerva, P. (1996).** Binary spatter-coding of ordered *K*-tuples. Artificial Neural Networks – ICANN 96, pp. 896–873.
13. **Speer, R., Chin, J., Havasi, C. (2017).** ConceptNet 5.5: An open multilingual graph of general knowledge. Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, pp. 4444–4451.

Article received on 05/03/2020; accepted on 19/02/2021.
Corresponding author is Ricardo Barrón Fernández.