ISSN 1870-4069

Sequence Prediction with Hyperdimensional Computing

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

Instituto Politécnico Nacional (IPN), Centro de Investigación en Computación,
Mexico City, Mexico

jobquiroz@hotmail.com, barron2131@gmail.com, marco.a.ramirez.s@gmail.com

Abstract. Hyperdimensional Computing is an emergent model of computation where all objects are represented in high-dimensional vectors. This model includes a well-defined set of arithmetic operations that produce new high-dimensional vectors, which, in addition to represent basic entities, can also represent more complex data structures such as sets, relations and sequences. This paper presents a method for sequence prediction using Hyperdimensional Computing and the Sparse Distributed Memory model. The proposed method is based on the encoding, storage and retrieval of *sequence vectors,* which store the *k* consecutive vectors of a sequence. The next element of a sequence is selected by taking into account the current, as well as the *k* immediate preceding elements of the sequence. Each vector is associated to a sequence vector that is stored in memory; the way in which each vector is associated to its sequence vectors is the main contribution of this paper. We present experimental results for the encoding and prediction of randomly generated sequences and the results indicate that the method performs correct predictions.

Keywords: hyperdimensional computing, prediction methods, sequence representation, sparse distributed memory.

1 Introduction

Sequence learning is essential to human intelligence, sequences are everywhere, going from low-level sensory-motor behavior up to high-level problem solving and reasoning. Sequential processes can also be found in fields such as robotics, finances, language processing, etc.

Hyperdimensional Computing is an emergent model of computation. It is based on the manipulation of high-dimensional vectors which are used not only to represent variables and values, but also to represent relations, sets and sequences [2].

In this work, we use a Hyperdimensional Computing approach for sequence prediction. We present the main idea behind the use of high-dimensional vectors as a representation scheme, describe the operations used to manipulate these vectors and propose a method for storage and prediction of sequences.

The rest of the paper is organized as follows: Section 2 summarizes several works related to the use of HD Computing for sequence representation. Section 3 explains the main properties and operations used in HD Computing. In Section 4 we describe a new method for storage and prediction of sequences, Section 5 presents the experimental results and finally, Section 6 draws the conclusions and future work.

2 Related Work

2.1 Hyperdimensional Computing

The main difference between hyperdimensional computing (HD computing) and traditional computing relies on the type of elements used for computation. In HD computing information is encoded in high-dimensional vectors (typically between 1,000 and 10,000 bits long). These vectors have no dedicated fields, information is distributed along all vector.

Kanerva [6] summarizes the main properties of HD Computing and explains three applications where it can be used: language processing, learning from example and analogy-making.

Hyperdimensional spaces can be binary, real or complex, in [9] Rahimi et al. give a brief review of several frameworks for HD Computing, each framework having its own set of symbols and operations. They also describe a hardware architecture for operating high dimensional vectors.

HD Computing has been applied in different domains, such as visual character recognition [3], cognitive software agents [14], robotics [5], biosignal processing [10], and sequence prediction [1, 13].

2.2 Sequence Prediction with Hyperdimensional Computing

One of the first applications conceived for the Sparse Distributed Memory [7] was to store and retrieve sequences, in order to learn a sequence the memory can be operated in heteroassociative mode, where each location stores the next value of the sequence, forming a chain of pointers. Even though this approach has several flaws, it was the first idea to develop a prediction system based on HD Computing.

Kanerva proposed the use of several Sparse Distributed Memories for storing sequences, where each memory could store higher order sequential structures [7]. The main limitation of this approach was the need of a very large memory system. This type of learning has been applied in the modeling of service robot movement [22]. Bose et al. [1] proposed a framework for online sequence learning based on an associative memory model. In their approach, the prediction value is obtained as a result of the sum of a set of history vectors.

In [13] Snaider and Franklin proposed a modification to the SDM model that allows a more efficient way to store and retrieve sequences. These sequences are encoded using hyperdimensional operations (sums and permutations). Räsänen and Saarinen propose a predictor based on HD Computing where, unlike the two previous approaches, there is no need of a special memory, "the idea is to represent the previously observed history of each possible sequence state using a single vector of very large dimensionality" [12].

In this work we propose a prediction model based on HD Computing operators and the original SDM model, without the need of increasing the size of the memory.

3 Hyperdimensional Computing Background

One of the most relevant properties of the high-dimensional spaces is that most of the space is nearly orthogonal to any given point. This means that if two random vectors are generated, it is highly probable (more than 99.999%) that they are mutually orthogonal. These properties where exploited in the SDM model developed by Kanerva in 1988 [7]. However, the properties of high-dimensional spaces can also be used to perform other type computations, for which is necessary to define a set of HD computing operators.

3.1 HD Computing Operations

HD Computing is based on three operations: addition, multiplication and permutation. In this paper we use binary vectors, but these three operations can be defined for other types of vectors [9]. A more detailed explanation of HD Computing operations can be found in [6].

Addition
Addition is an element-wise binary average. The sum vector $X_s = \sum_{k=1}^{n} x_k$ is computed by adding each component i of each vector x_k and then using the threshold function Θ to maintain the values as 0 or 1:

$$X_{s,i} = \Theta\left(\frac{1}{n}\sum_{k=1}^{n} x_{k,i}\right), \tag{1}$$

where

$$\Theta(u) = \{1\ for\ u > 0.5, 0\ for\ u < 0.5, random\ otherwise.$$

The components of the sum vector can be computed independently, which means that it can be easily parallelized. The result vector X_s is the concatenation of all the corresponding components.

The sum vector has the property of being similar to the vectors added together; therefore the addition operation is used to encode sets of vectors. The elements of the sum are 'visible' in the representation of the set.

Multiplication
For binary vectors, multiplication is realized with an element-wise exclusive-or

(XOR); the resulting vector is dissimilar to the original ones. Multiplication is a reversible operation; in fact, the XOR operator is its own inverse.

Multiplication is a mapping of points in space. Given two vectors A and B, it is possible to generate C, and later retrieve either A or B.

$$C = A * B \quad \Rightarrow A = C * B \quad \Rightarrow B = C * A. \tag{2}$$

Another interesting property of multiplication is that preserves distance, $d(A,B) = d(A * X, B * X)$, "it is like moving a constellation of points bodily into a different part of the space while maintaining the relations (distances) between them" [6].

Permutation

Permutation randomly reorders the components of the original vector, and just as multiplication, the resulting vector is dissimilar to the original one. Permutation is invertible and it also maintains distance: $d(A,B) = d(\Pi\mathrm{A}, \Pi\mathrm{B})$.

$$B = \Pi A \quad \Rightarrow \quad A = \Pi^{-1} B. \tag{3}$$

Unlike the two previous operations, permutation is not explicitly defined; it is possible to permute a vector by changing all its components to a different place, or only some of them. An easy way to permute a vector is by performing a circular shift operation, which will change all the components and can easily be inverted. In this work we implemented permutation as a shift right logical operation and the inverse permutation as a shift left logical operation.

3.2 Clean-Up Memory

All previous operators allow us to encode, map and retrieve hyperdimensional patters, but in most cases the retrieval is not going to be exact. For example, $X = X_1 * A + X_2 * B$ is storing the association of A with X_1 and B with X_2. In order to retrieve A we can multiply X by X_1:

$$X * X_1 = (X_1 * A + X_2 * B) * X_1 \tag{4}$$

$$\Rightarrow \; X * X_1 = A + X_1 * X_2 * B.$$

The resulting vector contains the sum of the desire value (A) and an unknown vector $(X_1 * X_2 * B)$. In order to discriminate this last vector we can use an auto-associative memory that approximates $X * X_1$ to A.

This auto-associative memory is called clean-up memory [8], its function is store all the items that the system should recognize. When a noisy version of an item is given as input, the memory must either output the most similar item or indicate that the input is not close enough to any of the store items.

In the previous example is possible to obtain A by performing a read operation from memory.

$$A \cong Read(X * X_1) = Read(A + noise). \tag{5}$$

Every time a pattern becomes meaningful it can be recorded in memory, which becomes in “a catalog of meaningful patterns”.

In this work we use the Sparse Distributed Memory [7] as clean-up memory. The SDM is an architecture that can store high dimensional binary vectors and retrieve them based on partial matches. The storing of patters is distributed, meaning that a single pattern is stored in many physical locations (called Hard Addresses).

3.3 Sequence Representation

A sequence is a complex structure that represents events that occur in certain order. The ability to store and recall sequences is very important in a system that pretends to have an intelligent behavior. There are several ways to represent sequences with HD Computing [6], in this work we focus in the representation of sequences by permuting sums.

The addition operation allow us to represent a set of vectors without a particular order, this may not be useful to store sequences, unless the order of each vector is somewhat codified. Permutations can be used for this purpose.

Since permutation generates a vector dissimilar from the original one, the order of the vector can be encoded as the number of permutations performed to the vector before the sum. For example, the sequence *ABC*, can be encoded as:

$$S = A + \Pi B + \Pi\Pi C. \tag{6}$$

In order to retrieve each element we have to perform inverse permutations and use the clean-up memory. Since all the permuted vectors are not stored in memory, they are considered noise:

$$\begin{aligned} A &\cong Read(S) = Read(A + noise), \\ B &\cong Read(\Pi^{-1} S) = Read(B + noise), \\ C &\cong Read(\Pi^{-1}\Pi^{-1} S) = Read(C + noise). \end{aligned} \tag{7}$$

The amount of vectors that can be encoded with this method manly is limited by its dimensionality. For dimensions near 1,000 the limits is close to 4 to 5 vectors.

4 Prediction Model

The sequence prediction problem consists in that given $x_{n-k+1}, \dots, x_{n-1}, x_n$, we want to predict x_{n+1}. In this section we describe a prediction model based on HD Computing. The model is divided in two: a storage phase, where a set of sequences is stored in memory, and a prediction phase, where the model provide the next value for a given sequence.

4.1 Encoding and Storage of Sequences

The first step is to associate each value of a sequence x_i to a random binary vector X_i. Due to the properties of high dimensional spaces, we can assume that all vectors X_i are not only different, but also mutually orthogonal. All vectors X are then stored in memory in auto-associative mode.

Each vector X_i must be associated with a sequence vector S_i, which encodes the k future vectors $X_{i+1}, \dots, X_{i+k}$.

$$S_i = X_{i+1} + \Pi X_{i+1} + \cdots + \Pi^k X_{i+k}. \tag{8}$$

Since X_i has already been used as an address to memory, it is not possible to associate the value X_i to S_i in a direct way. But we can map the address X_i to another part of the space where the association of X_i and S_i could take place. One way to implement such mapping is by multiplicate X_i by a known random binary vector M. The resulting vector $X_i * M$ is used as an address to store S_i:

$$Write(M * X_i) = S_i. \tag{9}$$

In this prediction model the memory is not only used as a clean-up system (auto-associative mode), but also as a way to associate symbols (hetero-associative mode). Once that all X_i vectors have been stored and mapped to its corresponding sequence vectors, the storage phase is finished.

4.2 Prediction of Sequences

In order to predict the next value of a sequence, the systems needs to have as an input at least k consecutive elements of the sequence: $x_{j-k+1}, \dots, x_j$. The system approximates each value x_j to a known value x_i and then associates a vector X_j to each x_j.

Each vector $X_{j-k+1}, \dots, X_j$ is associated with a sequence vector $S_{j-k+1}, \dots, S_j$ which can be read from memory as follows:

$$\begin{gathered} S_{j-k+1} = Read\left(X_{j-k+1} * M\right), \\ \dots \\ S_j = Read\left(X_j * M\right). \end{gathered} \tag{10}$$

Since each sequence vector has encoded a vector X_{j+1}^k, the last step is to extract each of these values in order to generate the final prediction vector X_{j+1} from which is possible to obtain the prediction value x_{j+1}, Fig 1.

$$\begin{gathered} X_{j+1}^1 = Read\left(\Pi^{-k} S_{j-k+1}\right), \\ \dots \\ X_{j+1}^k = Read\left(S_{j-k+1}\right), \end{gathered} \tag{11}$$

$$X_{j+1} = \sum_{r=1}^{k} X_{j+1}^{r} \;\rightarrow\; x_{j+1}.$$

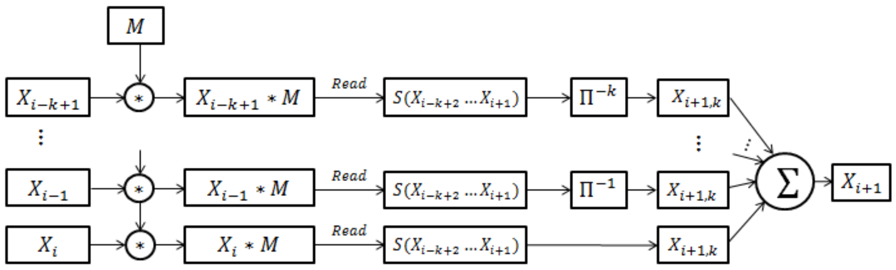

Fig. 1. Prediction model with Hyperdimensional Computing.

5 Experimental Results

We performed several simulations to test how the proposed method is able to store and predict sequences. As stated before, we use binary vectors with length of 1,000 dimensions, these vectors have 50% ones and 50% zeros randomly distributed. We also use the original Sparse Distributed Memory model.

For each simulation we vary the number of Hard Addresses of the memory to see how the capacity of the memory increases. We also increment the value of *k* which represents the number of vectors taken into account to form the sequence vector and to make a prediction (see section 4.2).

Table 1. Prediction Rate for different values of *k* and N = 50 sequences

N = 50 sequences				
Hard Addresses	**Successful Prediction**			
	k = 1	k = 2	k = 3	k = 4
100,000	41.28%	26.40%	58.89%	33.25%
250,000	60.17%	51.36%	70.12%	51.12%
500,000	71.50%	59.85%	79.85%	70.78%
1,000,000	74.80%	64.98%	91.24%	74.98%

At the beginning of the experiment we generate random sequences, which are then encoded and stored into memory. Then we take *k* consecutive points of each sequence and try to predict the entire sequence. Each sequence consists of 20 points and we consider a prediction to be successful if it has at most 5% of error. The number of points and the criteria for a successful prediction was taken as in [13], so we can compare the results. Table 1 summarizes the results from the experiments.

In the SDM model all the data is distributed along several physical locations, if the capacity of the memory increases then the number of sequences that can be successfully predicted also increases. Unlike the system in [13] we do not modify the architecture of the memory, which results in a smaller size of memory, however, in this method each element of each sequence takes two addresses, one for storing itself (auto-associative mode) and one for storing its sequence vector (hetero-associative mode).

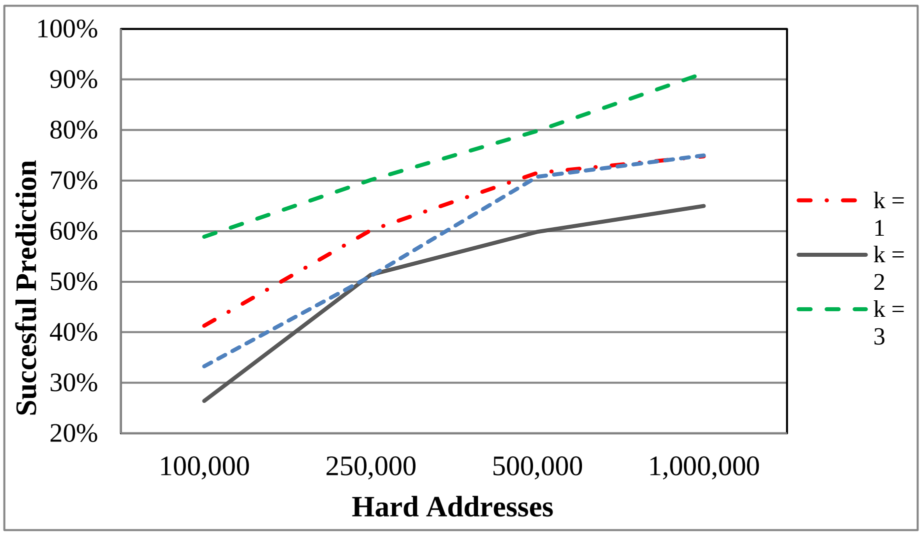

Fig. 2. Percentage of successfully predicted sequences for different values of *k*.

The rate of successful predictions do not increase linearly with respect to the value of *k* (Fig. 2), which might seem counterintuitive, because as the number of preceding points increases the prediction is expected to get better. But one of the downsides of the sum operation is that when the number of added vectors increases, the interference between them increases as well. This interference mainly depends on the dimensionality of the vectors, for vectors with length 1,000 starts to be noticeable at approximately 4 to 5 vectors.

The results presented in [13] show that their system was capable of successfully retrieve up to 49 sequences out of 50, but when the number of sequences increased to 100 none of the sequences could be restored. In this case the capacity of the memory system was the limiting factor, since they only implemented 200,000 Hard Addresses.

Another interesting behavior from Fig. 2 is that the prediction rate for k = 1 is better than for k = 2. As seen in Section 3.1, the sum operation returns a random value when the sum of the components is equal to 0.5 (which prevents the sum vector to be constantly filled with ones or zeros) and when k = 2 the probabilities for this to happen are much greater that when k = 1 or k = 3. This behavior was also observed in [13].

6 Conclusions and Future Work

This works presented some highlights of Hyperdimensional Computing, which is an emergent model of computation based on the storage and manipulation of high-dimensional vectors. This type of computation allows us to encode from single concepts up to more complex data structures such as sequences.

We proposed a method for the encoding and prediction of sequences, based on arithmetical operations and a Sparse Distributed Memory system. The main feature of this method is the use of the multiplication operation as a way of mapping vectors in space, in this particular case: mapping a vector to a *sequence vector* which allows us to make predictions.

We presented experimental results for the prediction method and explain some of its limitations, such as the restriction in the number of vectors to be added into a single vector. One of the goals for future implementations is to increase the dimensionality of the vectors in order to reduce the interference, not only in sums, but in all three operations.

Another future modification is to develop a hierarchical model which allows us, not only to predict single points, but also to predict sequences and other complex structures, such as relations between concepts and sets.

References

1. Bose, J., Furber S., Shapiro, J.: An associative memory for the on-line recognition and prediction of temporal sequences. In: Proc. IEEE International Joint Conference on Neural Networks, Montreal, Canada, 1223–1228 (Jul./Aug. 2005)
2. Gallant, S., Okaywe T.: Representing Objects, Relations and Sequences. Neural Computation 25(8), 2038–2078 (2013)
3. Hong, Y., Chen S.: Character recognition in a Sparse Distributed Memory. IEEE Transactions on Systems, Man and Cybernetics, 21(3), 674–678 (1991)
4. Jockel, S.: Crossmodal learning and prediction of autobiographical episodic experiences using a Sparse Distributed Memory. Doctoral Thesis, University of Hamburg, Department of Informatics (2010)
5. Jockel, S., Mendes, M., Zhang, J., Coimbra, P., Crisóstomo, M.: Robot navigation and manipulation based on a predictive associative memory. In: Proceedings IEEE 8th International Conference on Development and Learning, Shanghai, China (June 2009)
6. Kanerva, P.: Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High Dimensional Random Vectors. Cognitive Computation 1(2), 139–159 (2009)
7. Kanerva, P.: Sparse Distributed Memory. Cambridge, MA: Bradford/MIT Press (1988)
8. Plate, T.: Holographic reduced representation: distributed representation of cognitive structure. Stanfor: CSLI (2003)
9. Rahimi, A., Datta, S., Kleyko, D., Paxon, E., Olshausen, B., Kanerva, P., Rabaey, J.: High-Dimensional Computing as a Nanoscalable Paradigm. IEEE Transactions on Circuits and Systems: Regular Papers PP(99), 1–14 (2017)
10. Rahimi, A., Benatti, S., Kanerva, P., Benini, L., Rabaey, J.: Hyperdimensional Biosignal Processing: A Case Study for EMG-based Hand Gesture Recognition. In: Proceedings of

the International Conference on Rebooting Computing, San Diego, CA, USE, pp. 1–8 (Oct. 2016)
11. Rao, R., Fuentes, O.: Hierarchical Learning of Navigational Behaviors in an Autonomous Robot using a Predictive Sparse Distributed memory. Autonomous Robots 5(3–4), 297–316 (1998)
12. Räsänen, O., Saarinen, J.: Sequence Prediction with Sparse Distributed Hyperdimensional Coding Applied to the Analysis of Mobile Phone Use Patterns. IEEE Transactions on Neural Networks and Learning Systems 27(9), 1878–1889 (2016)
13. Snaider, J., Franklin, S.: Extended Sparse Distributed Memory and Sequence Storage. Cognitive Computation 4(2), 172–180 (2012)
14. Snaider, J., Franklin, S.: Vector LIDA. Procedia Computer Science 41, 188–203 (2004)