HDCluster: An Accurate Clustering Using Brain-Inspired High-Dimensional Computing

Mohsen Imani, Yeseong Kim, Thomas Worley, Saransh Gupta, and Tajana Rosing
Computer Science and Engineering Department, UC San Diego, La Jolla, CA 92093, USA
{moimani, yek048, tworley, sgupta, tajana}@ucsd.edu

Abstract—Internet of things has increased the rate of data generation. Clustering is one of the most important tasks in this domain to find the latent correlation between data. However, performing today’s clustering tasks is often inefficient due to the data movement cost between cores and memory. We propose HDCluster, a brain-inspired unsupervised learning algorithm which clusters input data in a high-dimensional space by fully mapping and processing in memory. Instead of clustering input data in either fixed-point or floating-point representation, HDCluster maps data to vectors with dimension in thousands, called hypervectors, to cluster them. Our evaluation shows that HDCluster provides better clustering quality for the tasks that involve a large amount of data while providing a potential for accelerating in a memory-centric architecture.

Index Terms—Hyperdimension computing, Clustering, Brain-inspired computing

I. INTRODUCTION

Internet of things (IoT) significantly increases the number of devices around the world. Recent studies report that more than 25 billion connected smart devices exist in 2015. [1] This number is expected to be doubled by 2020. [2] As the large network of connected devices generates a huge amount of data, machine learning gains popularity as an autonomous solution that extracts the useful information and learns patterns from the collected data [3]–[5]. However, the large amount of data dominates the processing capability of the current computing systems [6]–[8]. This inefficiency is mainly due to significant data movement costs between processing cores and memory. For example, to perform clustering tasks which are one of the most important unsupervised learning, [9], [10] the processors expensively compute the similarity between data points by fetching every point from the memory. A traditional solution is to migrate this issue by running the tasks on a cloud, but transferring large amount of data incurs significant congestion on the network. In addition, this may inherently lead to security and privacy issues in many applications. Thus, going toward IoT, it is crucial to have a light-weight clustering technique with adequate architectural supports which can efficiently run even on end-node devices.

Brain-inspired Hyperdimensional (HD) computing is based on understanding the fact that brains compute with patterns of neural activity which are not readily associated with numbers [11]. However, due to the very large size of the brain’s circuits, such neural activity patterns can only be modeled with points of high-dimensional space (e.g., $D=10,000$). Operations on hypervectors can be combined into interesting computational behavior with unique features that make them robust and efficient. HD computing builds upon a well-defined set of operations with random HD vectors, is extremely robust in the presence of failures, and offers a complete computational paradigm that is easily applied to learning problems [11], [12]. Its main differentiation from other paradigms is that data are represented as approximate patterns, which can favorably scale for many learning applications.

In this paper, we present a new clustering algorithm which maps a large amount of original data to a hardware-friendly high-dimension space and performs the clustering tasks by using Hyperdimensional (HD) computing. HD computing is an alternative computational model which emulates cognition tasks by computing with vectors in high-dimensional space, called hypervectors. A hypervector has the dimensionality in thousands (e.g., $D=10,000$) to mimic neural activity [11]. Since the elements of hypervectors are independent in processing, we can design a fully-parallelized hardware architecture that handles the hypervectors. In this work, we show how the proposed HDCluster encodes the original data to the hypervectors without losing the necessary information and performs the clustering tasks with well-defined linear algebra for the data types in the high-dimension space. To the best of our knowledge, HDCluster is the first design which processes the clustering tasks with the hypervectors. Our evaluation shows that HDCluster provides better clustering quality for the tasks that involve a large amount of data while providing a potential for accelerating in a memory-centric architecture.

II. HDCLUSTER ALGORITHM

A. HDCluster Overview

We propose HDCluster, a brain-inspired clustering algorithm which clusters input data into a high-dimensional space. Instead of clustering input data in either fixed-point or floating-point representations, HDCluster maps data to vectors with thousands of dimensions, called hypervectors, and then clusters them using concrete linear algebra. The well-defined set of HD operations is known to be extremely robust in the presence of failures, and offers a complete computational paradigm that is easily applied to learning problems such as analogy-based reasoning, sequence memory, language recognition, biosignal processing.

### TABLE I

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>Feature vector in original domain</td>
<td>$I$</td>
<td>Local hypervector $\in [0,1]^D$</td>
</tr>
<tr>
<td>$N$</td>
<td># of training in original domain</td>
<td>$E$</td>
<td># of executed iterations</td>
</tr>
<tr>
<td>$D$</td>
<td>Dimension of encoded data</td>
<td>$ID$</td>
<td>An ID hypervector $\in [0,1]^D$</td>
</tr>
<tr>
<td>$K$</td>
<td># of clusters</td>
<td>$h$</td>
<td>A cluster center hypervector $\in [0,1]^D$</td>
</tr>
<tr>
<td>$D^p$</td>
<td>Encoded data points of a cluster</td>
<td>$N^p$</td>
<td># of data points</td>
</tr>
</tbody>
</table>

978-3-9819263-2-3/DATAE19©2019 EDAA 1591
speech recognition, and prediction from multimodal sensor fusion [13]–[20].

Figure 1 illustrates an overview of HDCluster. As the first step, HDCluster encodes data into high-dimensional space and then applies clustering tasks to the encoded data (1). The clustering procedure starts with the initial centers of each cluster. For each iteration, HDCluster identifies which center is the most similar to each data point. The identified centers are stored as tags (2). Then, HDCluster updates the centers by calculating the average of the data points whose tags are the same (3). The iterations are repeated to obtain the converged cluster centers (4). In the next section, we show the details of the clustering procedure.

B. Encoding into HD Space

The first step of HDCluster is to encode input data into the hypervectors, where an original data point has \(n\) features, i.e., \(v = \langle v_1, \ldots, v_n \rangle\) [21]. Table I summarizes all notations used in this paper. An encoded hypervector that corresponds to one data point has \(D\) dimensions (e.g., \(D = 10,000\)). We need to keep all information of a data point in the original space, i.e., features and their indexes. As Figure 2a shows, we use two sets of pre-computed hypervectors: level and ID hypervectors. To create level hypervectors, we compute the minimum and maximum feature values among all data points, \(v_{min}\) and \(v_{max}\), and then quantize the range of \([v_{min}, v_{max}]\) linearly into \(Q\) levels, \(L = \{L_1, \ldots, L_Q\}\). Each level hypervector, \(L_i\), is unique and has \(D\) binary dimensions, i.e., \(L_i \in \{0,1\}^D\). The level hypervectors need to have the spectrum of similarities, such that the neighbor levels get more similar hypervectors. We create the first level hypervector, \(L_1\), by randomly selecting each element of a hypervector to be either 0 or 1 value. The second level hypervector, \(L_2\), is created by flipping \(D/Q\) random dimensions of the \(L_1\). This continues until creating the \(L_Q\) hypervector by flipping \(D/Q\) random dimensions of \(L_{Q-1}\). Since we select and flip the dimensions randomly, there is a high probability that the \(L_1\) and \(L_Q\) will have \(D/2\) dimension difference. As a result, the level hypervectors have similar values if the corresponding original data are closer, while \(L_1\) and \(L_Q\) will be nearly orthogonal.

The hypervector also needs to contain all the information that the original features have. To differentiate the impact of each feature index, we devise ID hypervectors, \(\{ID_1, \ldots, ID_n\}\). An ID hypervector has the binarized dimensions, i.e., \(ID_i \in \{0,1\}^D\). We create IDs with random binary values so that the ID hypervectors of different feature indexes are nearly orthogonal:

\[
\delta(ID_i, ID_j) \simeq D/2 \quad (i \neq j \& 0 < i, j \leq n)
\]

where the similarity metric, \(\delta(ID_i, ID_j)\), is the Hamming distance between the two ID hypervectors. The orthogonality of ID hypervectors is ensured as long as the hypervector dimension, \(D\), is large enough compared to the number of features \((D >> n)\) in the original data point.

Figure 2b shows how we map each data point, \(v\), to the high-dimensional space using the precomputed hypervectors. For each feature, we perform an element-wise \(\oplus\) operation for the ID and level hypervector corresponding to the feature value. The different features are combined by adding each element. For example, when a feature value of an original data point, \(v_i\), is quantized to \(L_i \in \mathbb{L}\), the following equation represents the calculated hypervector, \(h\):

\[
h = ID_1 \oplus L_1 + ID_2 \oplus L_2 + \ldots + ID_n \oplus L_n.
\]

Note that the element-wise addition can make a hypervector that has integer elements, i.e., \(H \in \mathbb{N}^D\). To perform all the other clustering procedures with binarized hypervectors, we apply a majority function for the calculated hypervector. For a given hypervector, \(h = \langle h_1, \ldots, h_D \rangle\), the majority function is defined as follows:

\[
MAJ(h, \tau) = \langle h'_1, \ldots, h'_D \rangle \text{ where } h'_i = \begin{cases} 0, & \text{if } h_i < \tau \\ 1, & \text{otherwise.} \end{cases}
\]

Using the majority function, the final hypervector for each data point is encoded by \(e = MAJ(h, n/2)\), and \(e \in \{0,1\}^D\).

C. Clustering in HD Space

HDCluster procedure identifies the cluster indexes (tags) through an iterative process as shown in Figure 2c and d, by using the encoded hypervectors. The proposed HD clustering algorithm is inspired by k-means. [22] In a similar way to the standard k-means algorithm, we initially choose \(K\) random hypervectors as cluster centers. We denote each hypervector for the centers as \(\{C_1, C_2, \ldots, C_K\}\), where \(C_i \in \{0,1\}^D\), as
shown in Figure 2d. Randomness and high dimensionality of hypervectors ensure that the centers of clusters are orthogonal at the initial iteration. With the initial cluster centers, we perform an iterative process to find the best clusters. There are two steps in each iteration, computing distance similarity and updating cluster centers.

**Computing distance similarity:** In this step, HDCluster assigns a cluster center for each encoded hypervector among all the center candidates. HDCluster measures the Hamming distances between the hypervector of a data point and each k center and identifies the cluster center which has the highest similarity. For an encoded hypervector, e, the index of the cluster center, called \( k \), is chosen as follows:

\[
\arg\min_k \delta(C_k, e)
\]

**Updating cluster centers:** After identifying tags of all data points, HDCluster updates the cluster centers using the data points which belong to each cluster. For a set of the data points in the \( k^{th} \) cluster: \( \mathcal{E}_k = \{e_i\}_k \), we perform element-wise additions to produce the hypervector sum, \( s_k = \sum e_i \). Then, HDCluster binarizes the hypervector sum so that it is mapped into the \( \{0,1\}^P \) space. The following equation illustrates this procedure:

\[
C_k = MAJ(s_k, |\mathcal{E}_k|/2)
\]

**Termination of HDCluster:** The algorithm converges when there is no significant change in the cluster centers. This iterative procedure continues until (i) the hypervectors representing the center of clusters has minor change during two consecutive iterations or (ii) the number of iterations exceeds a pre-defined parameter.

## III. EXPERIMENTAL RESULTS

### A. Experimental Setup

We implemented full HDCluster functionality using C++ implementation. We test the clustering quality of the proposed HDCluster, with diverse datasets including: MNIST handwritten digit [23], DIM dataset [24], Glass Identification [25], Iris [26], Voice dataset (IsoLET) [27] and medical-related datasets such as Gene Expression Cancer RNA Sequence (RNA-Seq) [28], Breast Cancer dataset [29], Primary Tumor dataset [30], and Parkinsons dataset [31]. We evaluate the clustering quality for each dataset by using the provided ground truth.

### B. Clustering Quality

Table II show the quality of clustering for HDCluster and \( k \)-means algorithms over different applications. The quality of clustering has been measured by comparing the result of clustering with the provided ground truth. For all the reported results, we use \( Q = 16 \) and \( D = 10K \). To better understand when HDCluster provides higher quality and efficiency, we perform the evaluation for \( \text{DIM} \) which has multiple datasets with different feature sizes. Table III summarizes the evaluation results. As compared to the \( k \)-means++, HDCluster shows higher robustness with the increase in the dimension of the original data. For example, \( k \)-means++ and HDCluster exhibit the similar quality for \( \text{DIM} \) 32, while for \( \text{DIM} \) 1024, the HDCluster provides 10% better clustering quality. In addition, since the computation cost of HDCluster is independent of dimensions of a dataset after encoding the dataset, we achieve higher efficiency for larger dimension than \( k \)-means++ which does not scale.

### C. Parameter Exploration

The precision of the encoding procedure is defined by how accurate it can map original data points to the high-dimensional space. For example, quantizing each feature to a small level \( Q \) may not keep all the required information of the original data points. With the relatively large number of clusters \( K \), the encoded hypervector may need to maintain more fine-grained information. Figure 3 shows the sufficient quantization.
D. Scalability with Hypervisor Dimension

Since the hypervisors map the original information into a high-dimensional space, the quality of clustering is directly related to the vector size. In the accelerator design, it also determines the system efficiency. Figure 4a shows the quality loss of clustering when HDCluster dimensionality changes from 1k to 10k. The results show that, for most of the applications, HDCluster can perform the clustering tasks with a relatively small dimensionality, while providing similar quality of clustering. When reducing the dimensionality of hypervectors from 4,000 and 2,000, HDCluster on average has 1.7% and 3.1% loss of clustering when HDCluster dimensionality changes from 1k to 10k. Figure 4b also shows the normalized energy and execution time of HDCluster using different dimensions.

IV. CONCLUSION

We propose a novel clustering algorithm, HDCluster, that maps data points to the high-dimensional space and clusters them using concrete linear algebra for hypervectors. HDCluster provides high clustering quality for diverse and practical applications that involve a large number of samples and high complexity in feature domains. Our future work is to design an accelerator which processes the entire HDCluster operations in a memory-centric architecture.

ACKNOWLEDGEMENTS

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, and also NSF grants #1730158 and #1527034.

REFERENCES