ODHD: One-Class Brain-Inspired Hyperdimensional Computing for Outlier Detection

Ruixuan Wang
Villanova University
rwang8@villanova.edu

Xun Jiao
Villanova University
xun.jiao@villanova.edu

X. Sharon Hu
University of Notre Dame
shu@nd.edu

ABSTRACT

Outlier detection is a classical and important technique that has been used in different application domains such as medical diagnosis and Internet-of-Things. Recently, machine learning-based outlier detection algorithms, such as one-class support vector machine (OCSVM), isolation forest and autoencoder, have demonstrated promising results in outlier detection. In this paper, we take a radical departure from these classical learning methods and propose **ODHD**, an outlier detection method based on hyperdimensional computing (HDC). In **ODHD**, the outlier detection process is based on a P-U learning structure, in which we train a one-class HV based on inlier samples. This HV represents the abstraction information of all inlier samples; hence, any (testing) sample whose corresponding HV is dissimilar from this HV will be considered as an outlier. We perform an extensive evaluation using six datasets across different application domains and compare **ODHD** with multiple baseline methods including OCSVM, isolation forest, and autoencoder using three metrics including accuracy, F1 score and ROC-AUC. Experimental results show that **ODHD** outperforms all the baseline methods on every dataset for every metric. Moreover, we perform a design space exploration for **ODHD** to illustrate the tradeoff between performance and efficiency. The promising results presented in this paper provide a viable option and alternative to traditional learning algorithms for outlier detection.

1 INTRODUCTION

Outlier detection, also known as anomaly detection, remains to be an essential and important technique which has been used in different application domains such as medical diagnosis [1], Internet-of-Things (IoTs) [2], and financial fraud detection [3]. Outliers in a dataset are typically defined as extreme values that deviate from other observations on data, or an observation that diverges from an overall pattern in a sample set. In general, outliers are an indication of variability in a measurement, experimental errors or a novelty. Recently, cyber-attackers also fabricate outliers by intentional manipulations, which threaten the security of cyber-physical systems such as those in autonomous vehicles [4].

Over the years, researchers continue to design robust solutions to detect outliers efficiently where statistical methods and machine learning methods are two most popular solutions. Statistical methods include parametric methods such as Gaussian mixture model (GMM) methods [5, 6] and regression methods [7, 8], and non-parametric methods such as kernel density estimation methods [9]. While statistical methods are mathematically well explainable, fast to evaluate, and easier to implement, their results could be unreliable for practical applications due to their dependency on and assumption of a specific distribution model.

Recently, machine learning-based approaches are increasingly applied in outlier detection. Some of the most successful methods are one-class support vector machine (OCSVM), isolation forest, and neural network-based autoencoder. OCSVM is an alteration of traditional SVM that separates outliers from the inliers with the maximum margin [10]. Isolation forest is an ensemble model of isolation trees where outliers are more sensitive to isolation and have a relatively short traversal path length [11]. Autoencoder is an emerging unsupervised learning-based outlier detection method using a neural network to reconstruct the data samples, where outliers are detected based on the reconstruction errors [12].

We take a radical departure from these learning methods by developing an hyperdimensional computing (HDC)-based method referred to as **ODHD**. As an emerging computing paradigm, HDC is inspired by the attributes of human brain circuits including high-dimensionality and fully distributed holographic representation [13, 14]. Similarly, HDC represents symbols with high-dimensional vectors called hypervectors that usually have a dimension $D = 10,000$. HDC postulates the generation, manipulation, and comparison of such HVs to perform learning tasks. Compared with DNNs, the advantages of HDC include smaller model size, less computation cost, and one/few-shot learning, making it a promising alternative, especially in low-cost computing platforms [14]. Recently, HDC has shown promising results in different applications such as computer vision [15], bio-signal processing [16], drug discovery [17], robotics [18], and natural language processing [19]. In this paper we introduce a novel outlier detection method by developing a one-class HDC based on a P-U learning structure. This is inspired by a simple yet reasonable assumption that a single HV can represent the abstract information of all inlier samples, which will be sufficiently different than the HV representing outlier sample for detection. Specifically, this paper makes the following contributions:

(1) We introduce **ODHD**, a novel one-class HDC-based outlier detection employing a P-U learning structure, which attempts to form a high-dimensional representation of inliers samples. The promising results present in this paper provide a viable alternative to existing approaches in outlier detection field.
(2) We develop a complete pipeline for HDC-based outlier detection. We map all inliers samples in high-dimensional space

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530395

and form a one-class hypervector to represent the abstraction information of inliers. We propose a confidence-based method to automatically compute a threshold which will be used for outlier detection. During testing time, we compute the similarity between the unseen testing sample and the one-class HV, and compare such similarity with the precomputed threshold, based on which **ODHD** can detect the outliers.

(3) We evaluate the performance of **ODHD** using six datasets from the Outlier Detection Datasets (ODDS) Library [20] and compare **ODHD** with emerging baseline methods including OCSVM, isolation forest, autoencoder, and HDAD. The comprehensive evaluation results show that **ODHD** is able to outperform all the baseline methods on all six datasets in all metrics including accuracy, F1 score and ROC-AUC.

2 RELATED WORK

Outlier detection has been intensively studied in the past. Yang et al. developed an globally optimal Exemplar-Based Gaussian Mixture Model (GMM)-based unsupervised outlier detection method [5]. They used global optimal expectation maximization (EM) algorithm to fit the GMM to a given data set. Tang et al. further enhanced such a method by combining GMM with locality preserving projections [6]. Regression-based methods are also proposed such as [7] which detects outliers with linear regression. The key idea is based on a non-interactive covariance matrix and concentration steps applied in the least trimmed square estimation. While statistical methods are mathematically well acceptable, fast to evaluate, and easier to implement, their assumption and dependency on a specific distribution model could hinder their practical use.

Among the most popular machine learning-based outlier detection methods are OCSVM [10], isolation forest [11], and autoencoder [12]. OCSVM separates outliers from the inliers with the maximum margin, where the inference results within the estimated region are detected as normal samples where the samples outside the region are predicted as outliers [10]. In the isolation forest, outliers are more sensitive to isolation and have a relatively short traversal path length thus can be detected by examining the path length [11]. Neural network-based autoencoder contains an encoding network and a decoding network. The encoder maps the original input sample into a low-dimensional feature space, while the decoder attempts to reconstruct this sample from this encoded feature. The autoencoder is trained to minimize the overall reconstruction error, and the preserved information of the autoencoder is required to be as much relevant as possible to the normal instances. Outlier samples that diverge from the majority of the training samples are hardly reconstructed, which leads to a high reconstruction error. Thus, the outliers can be detected by examining the reconstruction error [12].

Recently, HDAD [21] applied HDC to anomaly detection by following the principles of autoencoder; it attempts to "recontruct" the input samples and detect anomalies based on reconstruction error. However, the detection process of HDAD is tedious where it needs to go through both encoding and decoding process (similar to autoencoder) to "reconstruct" the sample. **ODHD** is drastically different from HDAD by proposing a one-class HDC approach to detect outliers. We comprehensively compare the performance of **ODHD** with all the four baseline methods mentioned above — OCSVM, isolation forest, autoencoder, and HDAD.

3 HDC PRELIMINARIES

In this section, we briefly describe the basic background and mathematical foundations of HDC.

Basic HDC Component Hypervectors (HVs) are the fundamental components in HDC. An HV is a high-dimensional holographic vector with independent and identically distributed (i.i.d.) elements. The HV with $D = d$ dimensions is denoted as $\vec{H} = \langle h_1, h_2, \ldots, h_d \rangle$, where h_i is an element in HV and can be a binary, bipolar or integer number [14]. HVs are used to embed and represent information in different scales and levels, such as representing new information or aggregating existing information.

In this paper, we employ bipolar HVs, which means each element in an HV are either -1 or 1. We use cosine distance to measure the similarity of information embedded between two HVs, as shown in Eq. 1. Moreover, when the dimensionality is sufficiently high (e.g., $D = 10,000$), HVs are quasi-orthogonal where any two random bipolar HVs are nearly orthogonal [13].

$$\delta(\overrightarrow{H_x}, \overrightarrow{H_y}) = \frac{\overrightarrow{H_x} \cdot \overrightarrow{H_y}}{||\overrightarrow{H_x}|| \times ||\overrightarrow{H_y}||} = \frac{\sum_{i=1}^{d} h_{xi} \cdot h_{yi}}{\sqrt{\sum_{i=1}^{d} {h_{xi}}^2} \cdot \sqrt{\sum_{i=1}^{d} {h_{yi}}^2}} \tag{1}$$

Basic HDC Operations HDC supports three basic arithmetic operations including addition (+), multiplication ($*$) and permutation (ρ), as illustrated in Eq. 2. Additions and multiplications both take two input HVs as operands and perform **element-wise** add or multiply operations. Permutation takes one HV as the input operand and perform **cyclic rotation**. All the three operations preserve the dimensionality of the input HVs, i.e., the input HVs and the output HVs have the same dimension.

$$\begin{aligned} \overrightarrow{H_x} + \overrightarrow{H_y} &= \langle h_{x1} + h_{y1}, h_{x2} + h_{y2}, \ldots, h_{xd} + h_{yd} \rangle \\ \overrightarrow{H_x} * \overrightarrow{H_y} &= \langle h_{x1} * h_{y1}, h_{x2} * h_{y2}, \ldots, h_{xd} * h_{yd} \rangle \\ \rho^1(\vec{H}) &= \langle h_d, h_1, h_2, \ldots, h_{d-1} \rangle \end{aligned} \tag{2}$$

All the three HD operations have their corresponding physical meanings. Addition is used to bundle the same-type of information, while multiplication is used to bind different types of information together and to generate new information. Permutation is used to reflect spatial or temporal changes in the information, such as time series or spatial coordinates [13].

4 ODHD FRAMEWORK

In this section, we leverage the mathematical properties of HDC and develop a novel one-class HDC-based outlier detection framework called **ODHD**, which essentially learns abstract representation of inlier samples and then performs one-class classification-based outlier detection. In **ODHD**, the outlier detection process is based on a P-U learning structure [22], which means we use only inlier samples for training, and test on an testing set (may contain both inliers and outliers) without the information of labels. The one-class HV we trained contains the information from all the patterns of inlier (training) samples. For testing, we detect whether a query HV conforms to the one-class HV by using cosine similarity. We develop a confidence-based algorithm to calculate a threshold based

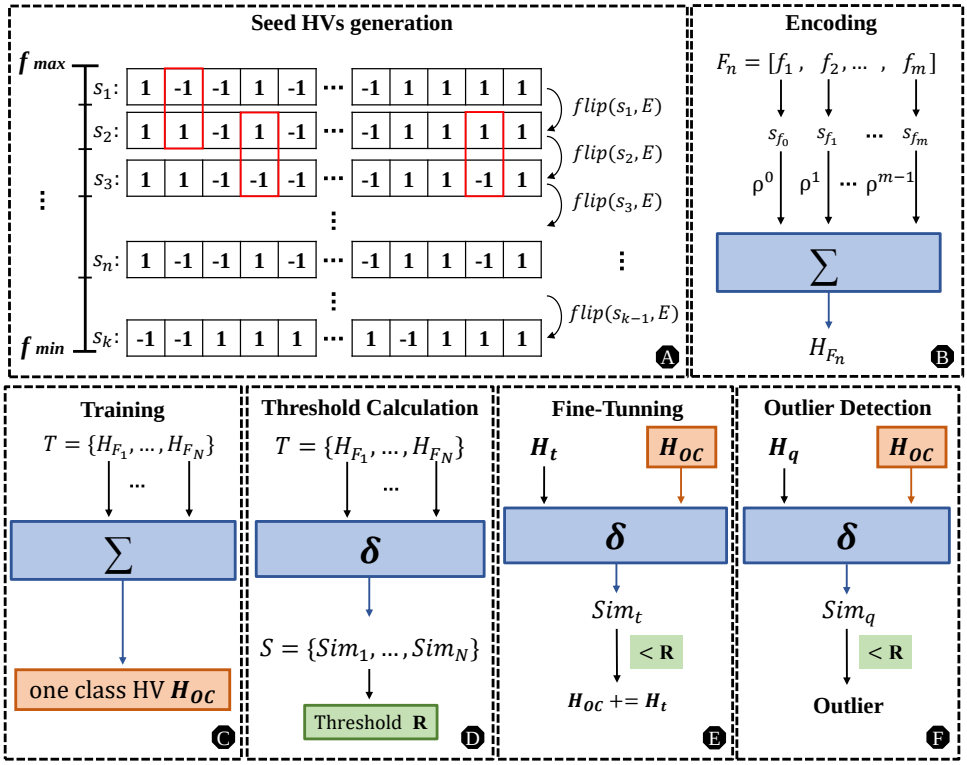


Figure 1: ODHD framework with six key phases.

on training samples. If the cosine similarity between a query HV and one-class HV is lower than the threshold, the query HV will be detected as an outlier. Fig. 1 illustrates the whole process of **ODHD**, which is divided into six key phases: Seed HV Generation, Encoding, Training, Threshold Calculation, Fine-Tuning, and Outlier Detection. We describe each phase of **ODHD** in detail in the following sections.

4.1 Seed HVs Generation

As the first step, we need to generate seed HVs so that we can encode the raw sample features to HVs. As noted previously, each HV is a high-dimensional vector with i.i.d elements [13]. We choose to use a simple HV generation method consistent with the one in [23] to create k seed HVs that can support later encoding. Specifically, we initiate a random bipolar D-dimension HV $\overrightarrow{s_1}$, and then generate other HVs by randomly flip $E = D/2k$ elements. This method is more computationally efficient compared to randomly generating k random HVs directly, while preserving the orthogonality of HVs [23].

k is a configurable parameter depending on how we discretize the input data. Assume for a specific dataset, we have each feature vector with m features $\overrightarrow{F_n} = \langle f_1, f_2, \ldots, f_m \rangle$. According to the training set we can capture the minimum and the maximum values of each feature value f_{min} and f_{max}. Then we discretize the input feature space (f_{min}, f_{max}) into k uniform intervals. Thus, each feature value corresponds to a specific interval, and we can convert the feature vector into an integer vector, which will be used for encoding. A larger k means that we have more quantization levels, and hence we can more precisely represent the original data but also increase the memory overhead (will discuss in Section.5.4).

4.2 Encoding

The encoding module projects the original feature vector into an HV. The encoding process of feature vector $\overrightarrow{F_n} = \langle f_1, f_2, \ldots, f_m \rangle$ is shown in the Part B of Fig. 1. We first index the seed HV corresponding to each feature value. For example, if a feature value f_4 falls into the 5^{th} interval among the k intervals, the corresponding seed HV is the 5^{th} of the k seed HVs.

Then, we employ the permutation operation to embed the information of the feature position into the seed HV. As the permutation operation reflects the spatial change of information, we bundle the information of feature position by deploying a cyclic rotation on each seed HV as shown in Eq. 2. Particularly, we keep the first seed HV un-permuted ($\rho^0(\overrightarrow{s_{f_1}})$), and for seed HV $\overrightarrow{s_2}$ to $\overrightarrow{s_k}$, we circularly rotate the i^{th} seed HV by $i-1$ elements, i.e., $\rho^{i-1}(\overrightarrow{s_{f_i}})$.

At the end of the encoding process, we aggregate all permuted seed HVs corresponding to all feature values into one HV $\overrightarrow{H_{F_n}}$ representing the entire feature vector $\overrightarrow{F_n}$. Note that if we have 100 inlier samples (i.e., 100 feature vectors) in the training dataset, we would have 100 corresponding encoded HVs. The overall encoding process is denoted as Eq. 3.

$$\overrightarrow{H_{F_n}} = \rho^0(\overrightarrow{s_{f_1}}) + \rho^1(\overrightarrow{s_{f_2}}) + \cdots + \rho^{m-1}(\overrightarrow{s_{f_m}}) \tag{3}$$

4.3 Training

After encoding all feature vectors in the training set, the training phase generates the one-class HV (H_{OC}) of the entire training set, i.e., all inlier samples. Eq. 4 illustrates the process of HDC training by adding every HV representing each inlier feature vector. For example, if there are 100 inlier samples, then the 100 corresponding encoded HVs generated by the encoding process are added together to generate a single one-class HV H_{OC} representing inlier samples or patterns.

$$\overrightarrow{H_{OC}} = \sum_{i=1}^{N} \overrightarrow{H_{F_i}} \tag{4}$$

4.4 Threshold Calculation

In **ODHD**, we propose a confidence-based threshold calculation approach. In order to calculate a threshold to separate inliers and outliers, we measure the cosine similarity between $\overrightarrow{H_{OC}}$ and all training HVs to obtain a similarity array S. As the Part D in Fig. 1 shows, each similarity Sim_i in array S can be considered as the confidence of the training HV to be an inlier sample.

We calculate the mean value S_{mean} and the standard deviation S_{std} of all the similarity values in array S. And we deploy a threshold estimation strategy according to [12] shown in Eq.5.

$$R = S_{mean} + 2 * S_{std} \tag{5}$$

Ultimately, we compute the threshold R based on the confidence of all training HVs. In the outlier detection domain, only the samples with cosine similarity higher than the threshold are determined as an inlier, while all the samples with cosine similarity lower than the threshold are identified as outliers.

4.5 Fine-Tuning

We perform extra fine-tuning for **ODHD**. Note that we still only use the given training dataset for the fine-tuning process. The fine-tuning process of **ODHD** is shown in Part E of Fig. 1.

After the training phase, we expect that all training HVs should be properly determined as inliers (but this may not be the case). In each fine-tuning epoch, we feed all the training samples to the **ODHD**. For each training sample, we estimate the cosine similarity Sim_t between the training HV $\overrightarrow{H_t}$ and one-class HV $\overrightarrow{H_{OC}}$. When Sim_t is higher than the threshold R, which means the estimation is correct, we do not make any changes on the $\overrightarrow{H_{OC}}$. However, if Sim_t is lower than R, which means **ODHD** mistakenly considers the inlier sample t as an outlier, we will update the one-class HV: we add the mis-detected training HV $\overrightarrow{H_t}$ into the one-class HV to update the corresponding information in $\overrightarrow{H_{OC}}$. For consistency, we perform 10 fine-tuning epoches for all cases.

$$\overrightarrow{H_{OC}} = \overrightarrow{H_{OC}} + \overrightarrow{H_t} \tag{6}$$

4.6 Outlier Detection

After we train the $\overrightarrow{H_{OC}}$ and obtain the threshold R based on confidences of the training HVs, we deploy the outlier detection on an unseen sample without the knowledge of labels based on **ODHD**. The outlier detection process is shown as the Part F in Fig. 1.

During the outlier detection phase, we encode testing sample q into an HV called query HV $\overrightarrow{H_q}$ following the same encoding process in Eq. 3 based on the same seed HVs. Then we compute the cosine similarity Sim_q between the query HV $\overrightarrow{H_q}$ and the one-class HV $\overrightarrow{H_{OC}}$. In the event that Sim_q is lower than the pre-computed threshold, the sample q will be determined as an outlier.

$$\overrightarrow{H_q} = \begin{cases} Inlier & Sim_q \geq R \\ Outlier & Sim_q < R \end{cases} \tag{7}$$

5 EXPERIMENT RESULTS

In this section, we evaluate the performance of **ODHD** on six datasets and compare **ODHD** with four baseline methods. We also present a design space exploration for **ODHD** to illustrate the tradeoff between **ODHD** performance and memory efficiency.

5.1 Experiment Setup

We evaluate the performance of **ODHD** on six datasets selected from the Outlier Detection Datasets (ODDS) Library [20] spanning multiple application domains such as medical diagnosis and wireless communication. These datasets are Wisconsin-Breast Cancer (Diagnostics) dataset (WBC), Mammography (MAMMO), MNIST, Cardiotocography (CARDIO), lymphography (LYMPHO), and Landsat Satellite (SATI2). These datasets are widely used as benchmarks in existing outlier detection studies [24–26]. Each dataset contains a certain number of outliers specified by ODDS library, e.g., WBC dataset has 21 outliers. The testing dataset is a mixed inliers and outliers, e.g., 25% − 75% outlier - inlier mix. More details of these datasets can be found on the (ODDS) Library website [20]. We repeat the experiments independently for 10 times and report the average performance. We also present error bars as shown in Fig. 2 to illustrate the performance variations due to the randomness in different learning methods.

We compare **ODHD** with the following four baseline outlier detection methods:

- **Autoencoder**: Autoencoder is an emerging unsupervised learning outlier detection approach based on neural network. In this paper, we use the same autoencoder architecture as [12].
- **Isolation Forest**: Isolation Forest is an ensemble model of isolation trees, which uses the path length of each sample to detect outliers. In this paper, we establish an isolation forest model using the same configuration as [11].
- **OCSVM**: OCSVM attempts to separate outliers from the inliers with the maximum margin. We have a grid search for an appropriate set of hyper-parameters such as kernel functions and the value of gamma to fine-tune the OCSVM model following [27].
- **HDAD**: HDAD follows the similar principles of autoencoder; it first "recontruct" the input samples and then it detects anomalies based on reconstruction error. We use the same architecture of [21].

We implement **ODHD** and the four baseline methods in Python and perform our experiments on a desktop with i7-7700 CPU and 12 GB RAM. To comprehensively quantify the performance, we use three metrics to evaluate the performance of **ODHD**: accuracy (ACC), F1 score (F1) and ROC-AUC (AUC). Note that while accuracy is widely used and easy to understand, an outlier detection dataset

may be significantly imbalanced. Hence, accuracy may not precisely reveal the performance of outlier detectors. Therefore, we also use ROC-AUC, which is widely used for outlier detection as it can accurately represent the tradeoff between true positive and false positive [28]. Meanwhile, F1 score is also a widely-used metric in binary classification which can comprehensively indicate the tradeoff between precision and recall [29].

5.2 ODHD Performance

As shown in Fig. 2, we compare the performance of **ODHD** with the baseline methods on all six datasets for the three metrics *with error bar*. We can observe several important facts.

First, we can observe that **ODHD** is able to consistently outperform the four baseline methods **on every dataset for every metric**. For ACC, the average ACC of **ODHD** is 90.4% on all datasets, representing an improvement of 17.1% over OCSVM (73.3%), 11.1% over isolation forest (79.3%), 10.5% over HDAD (81.8%) and 15.7% over autoencoder (74.7%). For F1, the average F1 of **ODHD** is 82.3% on all datasets, representing an improvement of 18.5% over OCSVM (63.8%), 12.8% over isolation forest (69.5%), 15.8% over HDAD (71.1%) and 19.8% over autoencoder (62.5%). For AUC, the average AUC of **ODHD** is 89.4%, representing an improvement of 10.9% over OCSVM (78.5%), 6.5% over isolation forest (82.9%), 6.8% over HDAD (83.7%) and 12.7% over autoencoder (76.7%).

Second, while **ODHD** has a certain level of fluctuation (error bars) in different runs (just like all the other models), we can observe that even the low end of **ODHD** is higher than the high end of any baseline method, representing the robustness of the performance of **ODHD**.

Third, while the performance of different methods all vary with the change of different datasets, **ODHD** shows a better stability compared to other methods. For example, for ACC, the lowest ACC of **ODHD** is over 80%, while the lowest ACC are about 60%, 70%, 60% and 70% for OCSVM, Isolation Forest, Autoencoder and HDAD, respectively. Similar phenomenon can also be seen in F1 and AUC.

Last but not least, in certain datasets, e.g., LYMPHO, all baseline methods significantly under-perform while **ODHD** maintains a high accuracy close to 100%. The reason is possibly related to the fact that the lymphography data are relatively small so that the baseline methods cannot converge to a proper point; however, **ODHD** is able to learn useful information even from a small amount of data. Similar advantages of HDC have been observed in various supervised classification studies for biomedical datasets that are often small [30].

5.3 ODHD Execution Time

In this experiment we compare the execution time (both training and testing) of different methods. For a fair comparison, since neural network-based Autoencoder requires GPU support, we compare the **ODHD** with the other three baseline models, OCSVM, isolation forest and HDAD. The comparison result is indicated in Table 2. In general, HDC-based approaches (HDAD and **ODHD**) take longer execution time than conventional approaches such as OCSVM and isolation forest. It is worth mentioning that the main time consumption comes from the encoding process in **ODHD**, which takes 72.3% of the training time and 82.5% of the testing time on average. While it is out of scope of this paper, there are existing methods and a growing research effort in accelerating HDC encoding process both from software and hardware [31, 32]. Our future work will consider the efficient hardware implementation of **ODHD**.

Table 1: Average performance of different models over six datasets

	OCSVM	Isolation Forest	HDAD	Autoencoder	**ODHD**
ACC(%)	73.3±2.3	79.3±2.5	81.8±1.8	74.7±1.5	**90.4±0.8**
F1(%)	63.8±2	69.5±2.4	71.1±2.8	62.5±1.3	**82.3±1.1**
AUC(%)	78.5±1.4	82.9±1.7	83.7±2.3	76.7±1.1	**89.4±0.7**

Table 2: Executime time(s) comparison of different models over six datasets (training time/testing time)

	OCSVM	Isolation Forest	HDAD	**ODHD**
WBC	0.003/0.002	0.112/0.03	0.412/0.198	0.399/0.112
MNIST	0.925/0.782	0.355/0.198	20.773/25.662	18.024/9.631
CARDIO	0.031/0.045	0.115/0.042	1.134/1.248	1.212/0.615
LYMPHO	0.001/0.001	0.111/0.028	0.169/0.053	0.119/0.02
SATI2	0.965/0.084	0.211/0.036	7.205/0.704	9.109/0.549
MAMMO	1.942/0.478	0.151/0.044	7.474/1.129	5.644/0.555

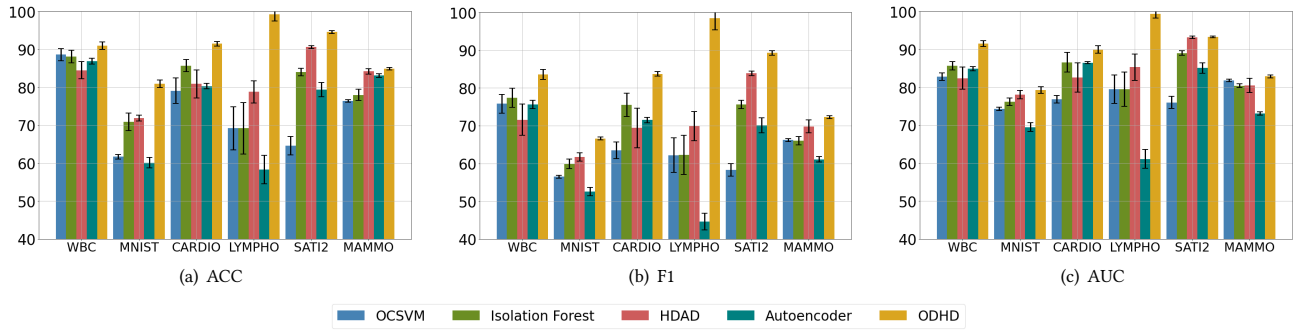
5.4 Design Space Exploration

Recall that in Section 4.1, we describe the configurable parameter k which represents the quantization level of input data, and directly determines the number of seed HVs. In general, the more levels we have for input data (i.e., the more seed HVs), the more distinct information we can represent using seed HVs, hence the higher performance we can obtain. However, more seed HVs lead to a higher memory requirement. Note that the total memory storage required by **ODHD** is the storage size of k seed HVs and the single one-class HV. In this section, without loss of generality, we explore the tradeoff between the model size and the model performance.

We perform an extensive design space exploration by experimenting six different k values and repeat the experiments on all datasets and metrics for every k value. The results are summarised in Table.3. Interestingly, k value reducing to a certain extent of does not affect the performance of **ODHD**. For example, when reducing k from 200 to 100, the performance is almost unchanged for all metrics while the model size is almost half of what it was before. This indicates that by carefully tuning k value, we can improve the memory efficiency of **ODHD** without affecting the performance. However, when further reducing the k value to 50, some datasets can see a slightly larger drop for certain metrics. For example, we can see a 4% drop for the F1 score of the LYMPHO dataset. Such exploration can lead to follow-up studies in how to efficiently balance the performance-efficiency tradeoff.

6 CONCLUSION

Outlier detection aims to detect extreme values that deviate from other observations in a dataset, or an observation that diverges from an overall pattern on a sample. In this paper we propose **ODHD**, a novel outlier detection algorithm based on one-class brain-inspired HDC. The aim of **ODHD** is to generate (train) a one-class HV that can represent the abstract information of inliers in a high-dimensional space. To this end, **ODHD** employs a P-U



(a) ACC (b) F1 (c) AUC

Figure 2: Comparison between ODHD and four baseline methods based on three metrics, ACC, F1 and AUC

Table 3: Design Space Exploration of ODHD on six datasets

Level (k)	Memory (KB)	WBC ACC F1 AUC	MNIST ACC F1 AUC	CARDIO ACC F1 AUC	LYMPHO ACC F1 AUC	SATI2 ACC F1 AUC	MAMMO ACC F1 AUC
200	7897	0.912 0.84 0.916	0.788 0.653 0.791	0.883 0.775 0.857	0.975 0.956 0.983	0.938 0.882 0.934	0.836 0.702 0.815
100	3988	0.903 0.829 0.916	0.784 0.653 0.793	0.867 0.758 0.855	0.967 0.941 0.978	0.934 0.875 0.931	0.822 0.696 0.819
50	2033	0.9 0.824 0.911	0.785 0.65 0.79	0.867 0.741 0.831	0.95 0.914 0.966	0.93 0.868 0.929	0.82 0.695 0.819
20	860	0.895 0.814 0.902	0.779 0.651 0.794	0.861 0.684 0.78	0.942 0.901 0.961	0.927 0.864 0.929	0.819 0.679 0.808
10	470	0.876 0.794 0.902	0.781 0.635 0.775	0.844 0.629 0.739	0.925 0.877 0.95	0.92 0.854 0.925	0.798 0.669 0.808
5	274	0.874 0.779 0.881	0.779 0.627 0.767	0.788 0.603 0.739	0.858 0.664 0.761	0.908 0.838 0.921	0.797 0.369 0.612

learning structure and leverages the HDC arithmetics to generate one-class HV. We use a confidence-based approach to determine the threshold. We evaluate **ODHD** using six datasets and compare **ODHD** with four baseline methods. Experimental results show that **ODHD** is able to outperform all the baseline methods on all six datasets in three performance metrics including accuracy, F1 score and ROC-AUC. The promising results presented in this paper open the door for a new viable alternative to traditional learning algorithms for outlier detection.

Acknowledgments. This work is supported in part by NSF CCF-2028879, CCF-2028889, CNS-1822099 and CCF-1640081.

REFERENCES

[1] Q. Wei *et al.*, "Anomaly detection for medical images based on a one-class classification," in *Medical Imaging 2018: Computer-Aided Diagnosis*, 2018.
[2] T. Yu *et al.*, "Recursive principal component analysis-based data outlier detection and sensor data aggregation in iot systems," *IEEE Internet of Things Journal*, 2017.
[3] A. Anandakrishnan *et al.*, "Anomaly detection in finance: editors' introduction," in *KDD 2017 Workshop on Anomaly Detection in Finance*. PMLR, 2018.
[4] J. Petit *et al.*, "Remote attacks on automated vehicles sensors: Experiments on camera and lidar," *Black Hat Europe*, 2015.
[5] X. Yang *et al.*, "Outlier detection with globally optimal exemplar-based gmm," in *Proceedings of the 2009 SIAM International Conference on Data Mining*, 2009.
[6] X.-m. Tang *et al.*, "Outlier detection in energy disaggregation using subspace learning and gaussian mixture model," *Int. J. Control Autom*, 2015.
[7] M. H. Satman, "A new algorithm for detecting outliers in linear regression," *International Journal of statistics and Probability*, vol. 2, no. 3, p. 101, 2013.
[8] C. M. Park and J. Jeon, "Regression-based outlier detection of sensor measurements using independent variable synthesis," in *International Conference on Data Science*. Springer, 2015, pp. 78–86.
[9] L. J. Latecki *et al.*, "Outlier detection with kernel density functions," in *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer, 2007.
[10] Y. Li *et al.*, "Anomaly detection of user behavior for database security audit based on ocsvm," in *International Conference on Information Science and Control Engineering*. IEEE, 2016.
[11] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *International Conference on Data Mining*. IEEE, 2008.
[12] T. He *et al.*, "Exploring inherent sensor redundancy for automotive anomaly detection," in *DAC*. IEEE, 2020.
[13] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive computation*, vol. 1, no. 2, pp. 139–159, 2009.
[14] L. Ge *et al.*, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, 2020.
[15] M. Hersche *et al.*, "Integrating event-based dynamic vision sensors with sparse hyperdimensional computing: a low-power accelerator with online learning capability," in *ISLPED*, 2020.
[16] A. Rahimi *et al.*, "Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals," *Proceedings of the IEEE*, 2018.
[17] D. Ma *et al.*, "Molehd: Automated drug discovery using brain-inspired hyperdimensional computing," 2021. [Online]. Available: https://arxiv.org/abs/2106.02894
[18] A. Mitrokhin *et al.*, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Science Robotics*, 2019.
[19] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *ISLPED*, 2016.
[20] S. Rayana, "Outlier detection datasets (odds) library," 2016. [Online]. Available: http://odds.cs.stonybrook.edu
[21] R. Wang *et al.*, "Brief industry paper: Hdad: Hyperdimensional computing-based anomaly detection for automotive sensor attacks," in *RTAS*, 2021.
[22] B. Liu *et al.*, "Building text classifiers using positive and unlabeled examples," in *IEEE International Conference on Data Mining*, 2003, pp. 179–186.
[23] Y. Kim *et al.*, "Efficient human activity recognition using hyperdimensional computing," in *International Conference on the Internet of Things*, 2018.
[24] C. C. Aggarwal and S. Sathe, "Theoretical foundations and algorithms for outlier ensembles," *Acm sigkdd explorations newsletter*, vol. 17, no. 1, pp. 24–47, 2015.
[25] A. Zimek *et al.*, "Subsampling for efficient and effective unsupervised outlier detection ensembles," in *KDD*, 2013.
[26] S. Sathe and C. Aggarwal, "Lodes: Local density meets spectral outlier detection," in *SIAM international conference on data mining*, 2016.
[27] S. Wang *et al.*, "Hyperparameter selection of one-class support vector machine by self-adaptive data shifting," *Pattern Recognition*, 2018.
[28] Z. Wang *et al.*, "Further analysis of outlier detection with deep generative models," *Advances in Neural Information Processing Systems*, 2020.
[29] M. Everingham *et al.*, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, 2010.
[30] A. Burrello *et al.*, "One-shot learning for ieeg seizure detection using end-to-end binary operations: Local binary patterns with hyperdimensional computing," in *IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2018.
[31] S. Salamat *et al.*, "Accelerating hyperdimensional computing on fpgas by exploiting computational reuse," *IEEE Transactions on Computers*, 2020.
[32] Y. Kim *et al.*, "Geniehd: efficient dna pattern matching accelerator using hyperdimensional computing," in *DATE*, 2020.