

Training a HyperDimensional Computing Classifier using a Threshold on its Confidence

Laura Smets¹, Werner Van Leekwijck¹, Ing Jyh Tsang¹, Steven Latré¹

¹University of Antwerp - imec, IDLab - Department of Computer Science, Sint-Pietersvliet 7, 2000 Antwerp, Belgium, E-mail: {Laura.Smets,Werner.Vanleekwijck}@uantwerpen.be; {Inton.Tsang,Steven.Latre}@imec.be.

Keywords: Hyperdimensional Computing, Vector Symbolic Architectures

Abstract

Hyperdimensional computing (HDC) has become popular for light-weight and energy-efficient machine learning, suitable for wearable Internet-of-Things (IoT) devices and near-sensor or on-device processing. HDC is computationally less complex than traditional deep learning algorithms and achieves moderate to good classification performance. This article proposes to extend the training procedure in HDC by taking into account not only wrongly classified samples, but also samples that are correctly classified by the HDC model but with low confidence. As such, a confidence threshold is introduced that can be tuned for each dataset to achieve the best classification accuracy. The proposed training procedure is tested on UCIHAR, CTG, ISOLET and HAND dataset for which the performance consistently improves compared to the baseline across a range of confidence threshold values. The extended training procedure also results in a shift towards higher confidence values of the correctly classified samples making the classifier not only more accurate but also more confident about its predictions.

1 Introduction

Hyperdimensional computing (HDC) has gained a lot of interest in the field of low-power, brain-inspired artificial intelligence (AI). It tries to mimic the human brain by distributing the information across thousands of vector elements in analogy to the large number of neurons present in our brains. HDC is a light-weight and energy-efficient algorithm that has already been used in several applications which can be divided in three categories according to Ge and Parhi (2020): (i) text classification (Rachkovskij, 2007; Rahimi, Kanerva, and Rabaey, 2016), (ii) signals such as speech recognition (Imani

et al., 2017), human activity recognition (Kim et al., 2018), handgesture recognition (Moin et al., 2021; Rahimi, Benatti, et al., 2016; Zhou et al., 2021) and time series classification (Schlegel et al., 2022), and (iii) images such as classification of medical images (Kleyko et al., 2017; Watkinson et al., 2021), character recognition (Manabat et al., 2019) and robotics (Neubert et al., 2019). It has been shown that HDC is suitable for wearable Internet-of-Things (IoT) devices, near-sensor AI applications and on-device processing due to few data requirement (Rahimi et al., 2019), robustness to noise (Kanerva, 2009; Rahimi et al., 2019; Widdows and Cohen, 2015), low latency (Rahimi et al., 2019) and fast processing (Rahimi et al., 2019). This avoids the limitations of IoT architectures in which data is sent to the cloud and consequently processed causing high latencies, large communication energy and privacy concerns (Basaklar et al., 2021).

Although HDC has the advantage of being computationally less complex than traditional deep learning algorithms, it is only able to achieve moderate to good performance in classification tasks. Hence, research is ongoing to adjust and improve HDC to boost its performance. This article aims to contribute to this research by proposing a simple, yet effective extended training procedure in the binary HDC framework to improve its performance on signal applications, suitable for wearable IoT devices. The next section gives a detailed introduction to HDC after which HDC adjustments that are already proposed in literature are discussed in Section 3. Thereafter, the proposed extended training procedure is introduced in Section 4. In the fifth section, an overview of the performed experiments is given of which the results are presented and discussed in the sixth section. Finally, the conclusions of the article are presented.

2 Hyperdimensional Computing

HDC is a mathematical framework using hyperdimensional (HD) vectors (i.e., vectors with very high dimension typically up to ten thousands, also called hypervectors (HVs)) and simple HD arithmetic operations to represent data. The focus of this article is on dense binary HVs (i.e., the elements are 0 or 1 with an equal probability of occurrence of both values). The analysis of data relies on the similarity between HVs which is calculated using the normalized Hamming distance between two binary HVs \mathbf{v}_1 and \mathbf{v}_2 ¹:

$$s(\mathbf{v}_1, \mathbf{v}_2) = 1 - \frac{h(\mathbf{v}_1, \mathbf{v}_2)}{D} \quad (1)$$

with s the similarity between \mathbf{v}_1 and \mathbf{v}_2 , D the dimensionality (e.g., $D = 10,000$) and h the Hamming distance between \mathbf{v}_1 and \mathbf{v}_2 which counts the number of elements for which the coordinates differ (i.e., the sum of elements of the exclusive disjunction (XOR)):

$$h(\mathbf{v}_1, \mathbf{v}_2) = \sum_{d=1}^D (\mathbf{v}_1[d] \text{ XOR } \mathbf{v}_2[d]). \quad (2)$$

The HD arithmetic operations include:

(a) bundling $\oplus: \mathcal{B} \times \mathcal{H} \rightarrow \mathcal{B}: (\mathbf{B}, \mathbf{v}) \rightarrow \mathbf{B} + \mathbf{v}$ where $\mathcal{B} = \mathbb{N}^D$ and $\mathcal{H} = \{0, 1\}^D$

¹A list of used symbols can be found in the appendix (Table A1).

(i.e., element-wise addition) after which the bundle \mathbf{B} is binarized into the HV \mathbf{v} with the majority rule $[\cdot] : \mathcal{B} \rightarrow \mathcal{H} : \mathbf{B} \rightarrow \mathbf{v}$ according to:

$$\mathbf{v}[d] = [\mathbf{B}[d]] = \begin{cases} 1 & \text{if } \mathbf{B}[d] > \frac{n}{2} \\ 0 & \text{if } \mathbf{B}[d] < \frac{n}{2} \\ \text{rand}(0, 1) & \text{if } \mathbf{B}[d] = \frac{n}{2} \end{cases} \quad (3)$$

with n the number of HVs bundled in \mathbf{B} and $\text{rand}(0, 1)$ means that the component $\mathbf{v}[d]$ is randomly assigned to 0 or 1 in the presence of ties (which can only occur when the number of bundled vectors is even);

- (b) **binding** $\otimes: \mathcal{H} \times \mathcal{H} \rightarrow \mathcal{H} : (\mathbf{v}_1, \mathbf{v}_2) \rightarrow \mathbf{v}_1 \text{ XOR } \mathbf{v}_2$ (i.e., XOR in binary HDC); and
- (c) **permutation** ρ (i.e., cyclic shift in binary HDC).

Figure 1 gives a schematic overview of the framework of HDC in which five main steps are distinguished: (1) mapping, (2) encoding, (3) initial prototype construction, (4) training and (5) inference, which will be explained in more detail in the following section.

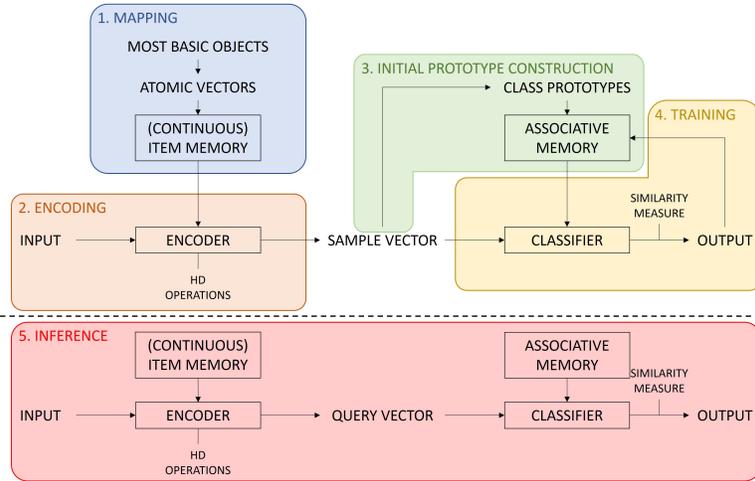


Figure 1: Schematic overview of the HDC framework in which five main steps are distinguished: (1) mapping, (2) encoding, (3) initial prototype construction, (4) training and (5) inference.

(1) Mapping. The way of mapping depends on the type of data:

(a) For nominal data, each possible category is mapped to a randomly chosen atomic HV and stored in an Item Memory (IM). These random HVs are pseudo-orthogonal in high dimensional spaces which converges to exact orthogonality with increasing dimensionality (Kleyko et al., 2022).

(b) In the case of ordinal or discrete data, there is a natural ordering of levels of categories or integer values such that closer levels should be mapped to more similar HVs than levels further apart. This is typically achieved by a Continuous Item Memory (CIM) applying *linear mapping* of levels to atomic HVs (Kleyko et al., 2018; Rahimi, Benatti, et al., 2016). Figure 2 illustrates the similarity of values to the lowest level

(feature value = -100) that decreases linearly up until orthogonality (similarity = 0.5) and the similarity of values to the feature value equal to -30 that decreases linearly for smaller and larger feature values.

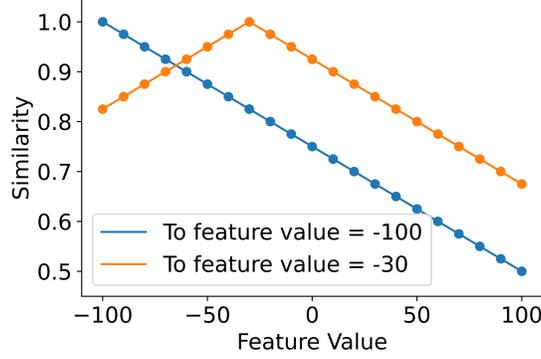


Figure 2: Example of *linear mapping* for a feature with discrete values ranging from -100 to 100 with steps of 10 . The similarity of each feature value's level hypervector to the lowest level hypervector (feature value = -100) and to the hypervector for the feature value of -30 is shown.

(c) Continuous data is quantized with a quantization step into a predefined number of discrete levels to which *linear mapping* can be applied.

(2) **Encoding.** Input data is encoded in HVs using the atomic vectors made in the previous step, and HD arithmetic operations. An input sample x having n features is encoded as (Figure 3):

For each feature ($j = 1 \dots n$), a CIM translates the feature value to an HV $\mathbf{v}_{x[j]}$. Next, all $\mathbf{v}_{x[j]}$'s are bundled together to form the sample bundle \mathbf{S} by initializing

$$\mathbf{B}_0 = \{0\}^D \quad (4)$$

and bundling each $\mathbf{v}_{x[j]}$ one at a time:

$$\mathbf{B}_j = \mathbf{B}_{j-1} \oplus \mathbf{v}_{x[j]}. \quad (5)$$

The sample bundle \mathbf{S} is then simply:

$$\mathbf{S} = \mathbf{B}_n. \quad (6)$$

For notation purposes, this iterative bundling (Equation 4-6) will be written in short as:

$$\mathbf{S} = \bigoplus_{j=1}^n \mathbf{v}_{x[j]} \quad (7)$$

which is binarized into the HV $\mathbf{s} = [\mathbf{S}]$ with the majority rule (Equation 3).

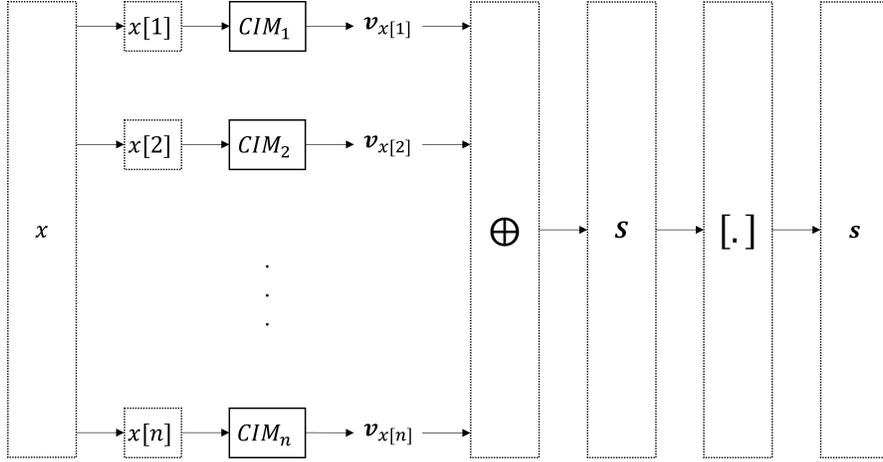


Figure 3: Schematic overview of the encoding with a separate CIM for the values of each feature.

Additionally, when the input data is encoded as n -grams, permutations can be used for the encoding (Rahimi, Benatti, et al., 2016):

$$\begin{aligned} \mathbf{B}_0 &= \{0\}^D \\ \mathbf{B}_j &= \rho(\mathbf{B}_{j-1}) \oplus \mathbf{s}_j \\ \mathbf{s}_{n\text{-gram}} &= [\mathbf{S}_{n\text{-gram}}] = [\mathbf{B}_n] \end{aligned} \quad (8)$$

where $j = 1 \dots n$, $\mathbf{s}_{n\text{-gram}}$ is the encoded, binarized HV of the n -gram of samples, and \mathbf{s}_j is a sample HV calculated with Equation 7 followed by binarization (Equation 3).

(3) Initial prototype construction. Sample HVs \mathbf{s}_i belonging to the same class l are bundled to form a class bundle \mathbf{C}_l representing the considered class:

$$\mathbf{C}_l = \bigoplus_{i=1}^m \{\mathbf{s}_i | y_i = l\} \quad (9)$$

with y_i the i th sample's class. After binarization of the class bundle \mathbf{C}_l , the l th class prototype $\mathbf{c}_l = [\mathbf{C}_l]$ is obtained and stored in the Associative Memory (AM). This is repeated for each class present in the dataset.

(4) Training. The HDC classifier predicts the class for all training samples by calculating the similarity between the training sample's HV \mathbf{s}_i and each class prototype \mathbf{c}_k stored in the AM. The predicted class \hat{y}_i of the input sample is the class with the highest similarity to the input sample's HV:

$$\hat{y}_i = \arg \max_k s(\mathbf{s}_i, \mathbf{c}_k) \quad (10)$$

If the predicted class is correct (i.e., $\hat{y}_i = y_i = l$), nothing happens. However, if the

sample HV \mathbf{s}_i is wrongly classified (i.e., $\hat{y}_i \neq y_i$), it is bundled again in the class bundle of the correct class \mathbf{C}_l and bundled out of the class bundle of the wrong class $\mathbf{C}_{\hat{l}}$:

$$\mathbf{C}_l = \mathbf{C}_l \oplus \mathbf{s}_i \quad (11)$$

$$\mathbf{C}_{\hat{l}} = \mathbf{C}_{\hat{l}} \ominus \mathbf{s}_i \quad (12)$$

with $\ominus: \mathcal{B} \times \mathcal{H} \rightarrow \mathcal{B}: (\mathbf{B}, \mathbf{v}) \rightarrow \mathbf{B} - \mathbf{v}$ the bundling out operation which is element-wise subtraction. As such, the class prototypes are adjusted to better classify wrongly classified samples.

The training procedure is performed iteratively until either a predefined accuracy on the training set is reached or either a predefined number of iterations is performed. After each iteration, the updated class bundles are binarized into updated class prototypes to be used in the next iteration or finally, in the inference phase.

(5) Inference. The i th test sample is encoded in a query HV \mathbf{q}_i following the same encoding procedure as for the training samples. The predicted class is obtained similarly as during the training procedure with Equation 10 where $\mathbf{s}_i = \mathbf{q}_i$, i.e., the predicted class label of the test sample is the class with the highest similarity to the test samples’s HV \mathbf{q}_i .

3 Previously proposed adjustments to HDC

Already several suggestions have been made to improve the initial prototype construction and the training procedure in the HDC framework. For instance, Rahimi, Benatti, et al. (2016) only add a sample vector to the class bundle if the similarity between the sample vector and class vector is smaller than 0.9 such that a sample vector is not added to the class bundle if the class vector is already highly similar to the sample vector. Consequently, no redundant information is added and the initial prototypes are assumed to be better, reducing the training time afterwards. Imani et al. (2019) propose to perform the training procedure with an adaptive learning rate that depends (1) on the average error rate over the last few training iterations (= iteration-dependent learning), (2) on the difference in similarity between query and wrong class on the one hand and query and right class on the other hand (= data-dependent learning) or (3) a combination of both (= hybrid learning). During the training process explained by Hernández-Cano et al. (2021), sample vectors are added to or subtracted from class bundles with a weight depending on the similarity of that sample vector to the considered class vector. As such, a sample with high similarity will be added with smaller weight than a sample with low similarity, since the sample is already highly represented by the class vector in case of high similarity and thus redundant information in the class prototype is limited.

In addition, more complex methods to improve HDC have been introduced in literature such as applying manifold learning for unsupervised non-linear dimensionality reduction to project the original data to a smaller dimension before applying HD encoding (Zou et al., 2021). Hsiao et al. (2021) use a learnable projection to train the HDC in a similar way as a binary neural network. The trained binary weights are then transformed

into learned IM and CIM. A different way of *linear mapping* is proposed by Basaklar et al. (2021) where a variable number of bits are flipped between levels to emphasize the distinct levels, instead of flipping a uniform number of bits. Chuang et al. (2020) use a confidence metric to decide whether binary HDC is suitable for a specific sample, i.e., samples predicted with low confidence in binary HDC will be predicted with non-binary HDC to improve the classification performance. On the other hand, Duan et al. (2022) map the HDC framework to an equivalent binary neural network (BNN) which is trained to minimize the training loss to increase the confidence in predictions. These methods in general achieve higher accuracies, but as they are more complex, they also have a large computational overhead.

In contrast, the focus of this article is on a simple, yet effective extension of the binary HDC training procedure that consistently improves the baseline performance, but without additional complexity. Its basic idea is to not only take into account wrong predictions when updating the class bundles, but also samples that are correctly classified but for which the class with the second highest similarity is only slightly less similar to the sample than the correct class. This simple extension of the HDC training procedure is explained in the next section.

4 Our proposal: confidence-based training procedure

As indicated by Duan et al. (2022), for a correctly classified sample the similarity to the class with the second highest similarity could only be slightly lower than the similarity to the class with the highest similarity. In such cases, the HDC classifier is less confident about its prediction for that specific sample. Knowing this, a confidence metric c_i is introduced by Chuang et al. (2020) as the difference between similarity of the sample vector \mathbf{s}_i to the class vector with the highest similarity and the similarity to the class vector with the second highest similarity:

$$c_i = \max_k s(\mathbf{s}_i, \mathbf{c}_k) - \max_{k \neq l} s(\mathbf{s}_i, \mathbf{c}_k). \quad (13)$$

The confidence metric reflects with how much certainty the HDC model classifies a specific sample, i.e., if the confidence is low, the HDC model is less certain about its prediction of a specific sample.

While Chuang et al. (2020) use the confidence metric only to measure certainty of classification on the test set, our proposal is to use this confidence metric in a simple way to extend the training procedure in HDC resulting in a more accurate binary HDC model. In all the basic HDC training frameworks, class bundles are only updated in case of a wrong prediction. However, in our proposal a training sample's HV \mathbf{s}_i that is correctly classified (i.e., $\hat{y}_i = y_i = l$) but with a confidence smaller than a threshold α (i.e., $c_i < \alpha$), is also added again to the class bundle of the correct class \mathbf{C}_l and bundled out of the class bundle of the class with second highest similarity $\mathbf{C}_{\hat{l}'}$ with $\hat{l}' = \arg \max_{k \neq l} s(\mathbf{s}_i, \mathbf{c}_k)$:

$$\mathbf{C}_l = \mathbf{C}_l \oplus \mathbf{s}_i \quad (14)$$

$$\mathbf{C}_{\hat{r}} = \mathbf{C}_{\hat{r}} \ominus \mathbf{s}_i \quad (15)$$

Note that if $\alpha = 0$, no correctly classified samples are used in the training procedure since the confidence is non-negative. As a consequence, this threshold setting corresponds to the baseline HDC classifier. Updating the class bundles with samples correctly classified with low confidence could be seen as pulling the wrong class further away from the considered sample and pushing the right class closer to the sample. As such, the main idea of this updating procedure has some analogies to prototype learning (Chang et al., 2006; Ji et al., 2021), prototype alignment (Hersche et al., 2022), distance metric learning (Kulis, 2012; Weinberger and Saul, 2009), linear discriminant analysis (Weinberger and Saul, 2009) and support vector machines (Weinberger and Saul, 2009) where the goal is to minimize the distance between samples from the same class while maximizing the distance between samples from different classes.

With this proposal of a simple extended training procedure, an improved classification performance is expected since the class prototypes are not only adjusted to better classify wrongly classified samples (Equation 11-12), but also samples that are correctly classified with low confidence (Equation 14-15).

5 Experiments

Four datasets, that are publicly available and commonly or previously used in other HDC-related research, are selected to test the proposed extended training procedure (a more detailed summary of the datasets can be found in Table 1):

(1) **UCIHAR dataset** (Anguita et al., 2013; Dua and Graff, 2019). To obtain this dataset, 30 subjects performed six activities (walking, walking upstairs, walking downstairs, sitting, standing and laying) during which the acceleration and velocity were recorded with the accelerometer and gyroscope of a smartphone attached to the chest of the subjects.

(2) **Cardiotocography (CTG) dataset** (Dua and Graff, 2019). This dataset consists of features for 2,126 fetal cardiotocograms that are classified with respect to one of three fetal states (normal, suspect or pathologic).

(3) **ISOLET dataset** (Dua and Graff, 2019). This dataset includes features extracted from speech signals that were collected for 150 subjects speaking each letter of the alphabet twice.

(4) **Hand gesture (HAND) dataset** (Rahimi, Benatti, et al., 2016). EMG signals with four channels of five subjects are included in this dataset. During the recording, the subjects performed four hand gestures (closed hand, open hand, 2-finger pinch and point index) ten times for three seconds each. Between each hand gesture contraction, a period of three seconds in the rest position is included which serves as the fifth class.

To quantize the feature / signal values of all datasets, a step size is chosen such that they are quantized into 21 quantization levels. As such, a step size of 0.1 is chosen for UCIHAR because values are between -1 and 1, a step size of 5 for CTG's values ranging from 0 to 100, a step size of 10 for ISOLET since the values range from -100 to 100 and a step size of 1 for HAND because the signals can take amplitudes from 0

to 20. The samples of UCIHAR, CTG and ISOLET are encoded following Equation 7 followed by binarization (Equation 3), whereas 4 – *grams* (Equation 8) are created for the encoding of HAND samples that are also encoded according to Equation 7 with binarization (Equation 3) where the four channels are treated as four features.

Table 1: Summary of the four datasets used to test the proposed extended training procedure. The table includes the number of training samples, the number of test samples, the number of classes, the number of features or channels, the range of values of the features, the step size that is used to quantize the feature values and the number of vectors (i.e., quantization levels) that are stored in the CIM.

	UCIHAR	CTG	ISOLET	HAND
# training samples	7,352	1,701	6,238	526,396
# test samples	2,947	425	1,559	131,608
# classes	6	3	26	5
# features / channels	561	21	617	4
range of feature values	[-1,1]	[0,100]	[-100,100]	[0,20]
quantization step size	0.1	5	10	1
# vectors in CIM	21	21	21	21

The training procedure is performed iteratively for a maximum of 2500 iterations while saving the classifier with the best accuracy. After every 100 iterations, it is evaluated whether this best training accuracy exceeds 99%. If this is the case, the training procedure is terminated and the classifier with the best accuracy is used in the inference phase. The HDC classifier is performed for 50 independent runs for UCIHAR, CTG and ISOLET, and for 10 independent runs for each subject of HAND (thus, also 50 independent runs in total) since it starts from random vectors to form the atomic vectors.

For each dataset, the distribution of confidence values of all correctly classified training samples after initial prototype construction is investigated to decide the range of confidence thresholds to be tested (Figure 4). This figure illustrates that the confidence values are rather low for all datasets (i.e., < 8%). Hence, the choice for the thresholds to be tested are $\alpha = \{0.00; 0.25; 0.50; 0.75; 1.00; 1.25; 1.50\}$ for UCIHAR, $\alpha = \{0.00; 1.00; 2.00; 3.00; 4.00; 5.00; 6.00\}$ for CTG, $\alpha = \{0.00; 0.25; 0.50; 0.75; 1.00; 1.25; 1.50\}$ for ISOLET and $\alpha = \{0.00; 1.00; 2.00; 3.00; 4.00; 5.00\}$ for HAND, since these values include most correctly classified samples for the considered datasets. The performance of the HDC classifier for each confidence threshold setting is documented as the classification accuracy and error rate on the test set averaged over all performed independent runs.

6 Results and discussion

The accuracies on the train and test set and the error rate on the test set averaged over all independent runs for each of the chosen confidence threshold settings and each of the four datasets are given in Table 2. The introduction of the confidence metric in the

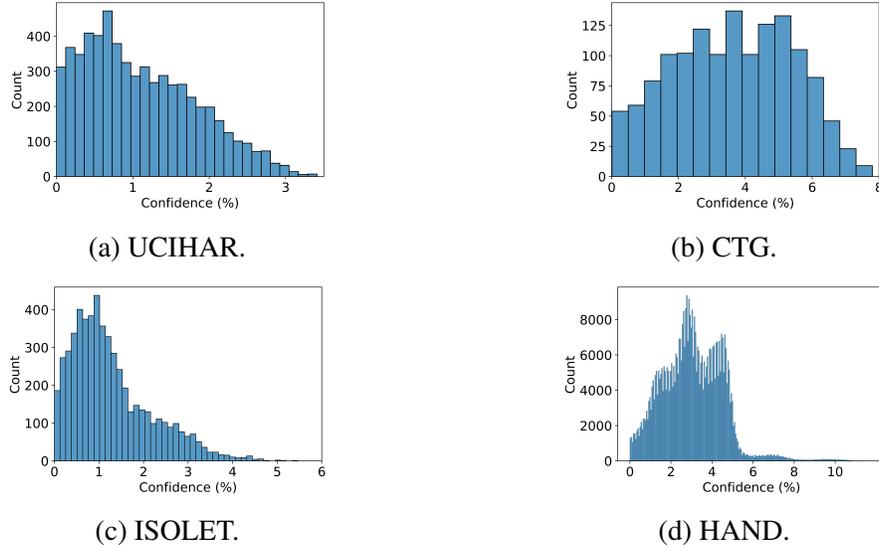


Figure 4: Distribution of the confidence values of all correctly classified training samples of (a) UCIHAR, (b) CTG, (c) ISOLET and (d) HAND after initial prototype construction.

training procedure improves the baseline classification accuracy ($\alpha = 0.00$) for each dataset for all tested confidence thresholds. The mean test accuracy for all datasets increases with increasing α up to the point where maximal performance is reached after which the mean test accuracy decreases for larger α .

(1) UCIHAR. The obtained baseline accuracy of 92.75% is improved by 1.58% with $\alpha = 0.75$ reaching an accuracy of 94.33%. (A 21.79% relative decrease in error rate from 7.25% for the baseline to 5.67%.)

(2) CTG. The obtained baseline accuracy of 73.65% is improved by 13.24% with $\alpha = 4.00$ reaching an accuracy of 86.89%. (A 50.25% relative decrease in error rate from 26.35% for the baseline to 13.11%.)

(3) ISOLET. The obtained baseline accuracy of 92.12% is improved by 2.18% with $\alpha = 1.00$ reaching an accuracy of 94.30%. (A 27.66% relative decrease in error rate from 7.88% for the baseline to 5.70%.)

(4) HAND. The obtained baseline accuracy of 95.38% is improved by 0.93% with $\alpha = 5.00$ reaching an accuracy of 96.31%. (A 20.13% relative decrease in error rate from 4.62% for the baseline to 3.69%.)

The effect of introducing the confidence threshold in the training procedure of HDC is visualized in Figure 5. This figure gives for each of the four datasets (one column for each) the distribution of the confidence values of all correctly classified training samples after training with $\alpha = 0.00$ (baseline, top row) and after training with the setting of α resulting in the best performance for the considered dataset (bottom row). When comparing with Figure 4, already larger confidence values are seen after training with $\alpha = 0.00$ for all four datasets. However, there is an even more clear shift in the distribution towards higher confidence values for the best settings of α for each dataset illustrating nicely the effect of the proposed simple extended training procedure in the

Table 2: Averaged accuracy (%) on the train and test set and averaged error rate (%) on test set over 50 independent runs (for UCIHAR, CTG and ISOLET) and 10 independent runs (for each subject of HAND, thus 50 in total) of the four datasets for the tested settings of the confidence threshold α . Data are *mean* (\pm *standard deviation*), in **bold** are the test accuracies (and error rates) that are higher (and lower) than the baseline ($\alpha = 0.00$) and underlined is the best test accuracy and error rate.

(a) UCIHAR.

α	Train accuracy	Test accuracy	Test error rate
0.00	99.43 (\pm 0.24)	92.75 (\pm 0.51)	7.25
0.25	99.11 (\pm 0.20)	93.94 (\pm 0.50)	6.06
0.50	98.96 (\pm 0.51)	94.03 (\pm 0.53)	5.97
0.75	98.62 (\pm 0.61)	<u>94.33</u> (\pm 0.49)	<u>5.67</u>
1.00	98.32 (\pm 0.99)	94.25 (\pm 0.70)	5.75
1.25	97.13 (\pm 1.15)	94.16 (\pm 0.62)	5.84
1.50	95.99 (\pm 1.19)	93.45 (\pm 0.90)	6.55

(b) CTG.

α	Train accuracy	Test accuracy	Test error rate
0.00	99.57 (\pm 0.21)	73.65 (\pm 2.48)	26.35
1.00	98.91 (\pm 0.17)	77.52 (\pm 2.53)	22.48
2.00	98.05 (\pm 0.13)	80.44 (\pm 1.79)	19.56
3.00	96.39 (\pm 0.17)	85.14 (\pm 1.28)	14.86
4.00	94.47 (\pm 0.29)	<u>86.89</u> (\pm 1.04)	<u>13.11</u>
5.00	93.19 (\pm 0.27)	86.19 (\pm 0.94)	13.81
6.00	92.47 (\pm 0.14)	84.97 (\pm 1.01)	15.03

(c) ISOLET.

α	Train accuracy	Test accuracy	Test error rate
0.00	100.00 (\pm 0.00)	92.12 (\pm 0.47)	7.88
0.25	100.00 (\pm 0.00)	93.19 (\pm 0.42)	6.81
0.50	99.99 (\pm 0.03)	93.80 (\pm 0.45)	6.20
0.75	99.83 (\pm 0.27)	94.13 (\pm 0.55)	5.87
1.00	99.81 (\pm 0.22)	<u>94.30</u> (\pm 0.73)	<u>5.70</u>
1.25	99.70 (\pm 0.28)	93.98 (\pm 0.75)	6.02
1.50	99.47 (\pm 0.25)	93.68 (\pm 0.76)	6.32

(d) HAND.

α	Train accuracy	Test accuracy	Test error rate
0.00	97.05 (\pm 1.84)	95.38 (\pm 1.82)	4.62
1.00	96.94 (\pm 1.92)	95.77 (\pm 1.83)	4.23
2.00	96.73 (\pm 1.85)	95.60 (\pm 2.25)	4.40
3.00	96.45 (\pm 2.01)	95.81 (\pm 2.03)	4.19
4.00	96.07 (\pm 2.20)	96.11 (\pm 1.65)	3.89
5.00	95.83 (\pm 2.26)	<u>96.31</u> (\pm 1.34)	<u>3.69</u>

HDC framework.

The abovementioned results show the impact of the parameter α on the test set. To determine the optimal α for a particular dataset, the standard procedure of using a validation set should be used. Table A2 in the appendix illustrates the procedure for the ISOLET dataset when performing 10-fold cross validation, resulting in an optimal α of 1.25.

The obtained highest accuracy of 94.30% for ISOLET is an improvement compared to the accuracy of 92.4% obtained by Imani et al. (2019) using an adaptive learning rate during training procedure and by Hsiao et al. (2021) who project original HDC to learnable HDC that is trained similarly as a BNN. Also for the CTG dataset, the highest accuracy of 86.89% is higher compared to the accuracy of 82% obtained by Basaklar et al. (2021) who flip a variable number of bits between levels when applying *linear mapping*.

While our method consistently improves the accuracy compared to baseline, some other studies in literature report slightly better results on the considered datasets but with often more complex methods. For example, Imani et al. (2019) report an accuracy of 96% on the UCIHAR dataset with their adaptive learning rate, Hernández-Cano et al. (2021) use weighted bundling in and out of class bundles and obtain an accuracy of 96.5% for UCIHAR and 94.6% for ISOLET, learnable HDC of Hsiao et al. (2021) results in an accuracy of 95.54% for UCIHAR, Zou et al. (2021) obtain an accuracy of 98% for UCIHAR and 95% for ISOLET applying manifold learning and Duan et al. (2022) map HDC into a BNN achieving 95.23% and 94.89% accuracy for UCIHAR and ISOLET, respectively.

As future work, the proposed training procedure could possibly be extended with an adaptive confidence threshold. Namely, the distribution of confidence values shift towards higher confidence while correcting for low-confident correctly classified training samples (Figure 5) such that the threshold could increase along with this shift in distribution. Moreover, it might be interesting to combine the proposed extended training procedure with previously proposed adjustments to the training procedure (Hernández-Cano et al., 2021; Imani et al., 2019) and investigate whether this would further improve the HDC performance.

Conclusion

A simple, yet effective extension to the training procedure in binary HDC is introduced which takes into account not only wrongly classified samples, but also samples that are correctly classified by HDC but with a low confidence. A threshold on the confidence value is introduced and tested on four datasets for which the performance consistently improves compared to the baseline across a range of confidence threshold values. The extended training procedure also results in a shift towards higher confidence values of the correctly classified samples making the classifier not only more accurate but also

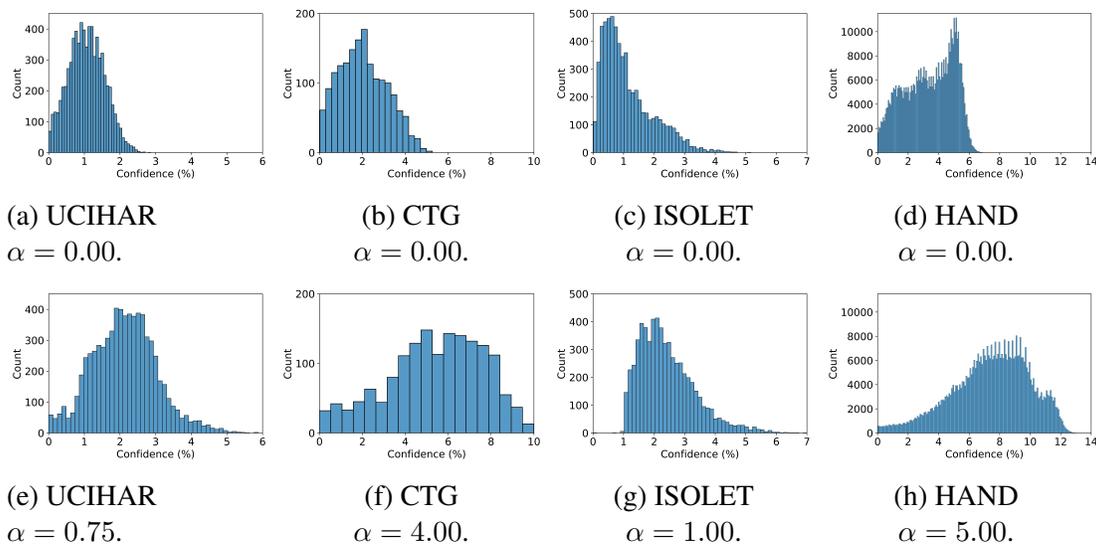


Figure 5: Distribution of the confidence values of all correctly classified training samples of UCIHAR (first column), CTG (second column), ISOLET (third column) and HAND (last column) after training with $\alpha = 0.00$ (first row) and with α yielding the best accuracy for the considered dataset (second row).

more confident about its predictions.

Acknowledgments

This research received funding from the Flemish Government under the "Onderzoek-sprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

Appendix

A.1 Notation

A summary of notation can be found in Table A1.

A.2 Cross-Validation

Table A2 contains the results of 10-fold cross validation (CV) on the training set of ISOLET for the different settings of α . The table includes the training accuracy, validation accuracy and validation error rate, averaged over the ten folds of 10-fold CV. This shows that the optimal α value with 10-fold CV is $\alpha = 1.25$, resulting in a test accuracy of 93.98% (Table 2c) which is very near to the best test accuracy achieved in Table 2c, i.e., 94.30% for $\alpha = 1.00$.

Table A1: List of symbols. (HD = hyperdimensional, CIM = Continuous Item Memory)

Symbol	Definition	Symbol	Definition
x	vector in input space	i	$1 \dots m$
m	number of samples	j	$1 \dots n$
n	number of features	k	$1 \dots K$
K	number of classes	d	$1 \dots D$
D	HD vector dimension	y_i	true class of i th sample
s	similarity	\hat{y}_i	predicted class of i th sample
h	Hamming distance	l	true class label
c	confidence value	\hat{l}	predicted class label
α	confidence threshold	\hat{l}'	class label with second highest similarity
\mathbf{v}	vector in HD space \mathcal{H}	\mathbf{B}	bundle in HD space \mathcal{B}
\mathbf{s}	sample vector	\mathbf{S}	sample bundle
\mathbf{c}	class vector/prototype	\mathbf{C}	class bundle
\mathbf{q}	query vector		
\mathcal{H}	vector HD space, $\{0, 1\}^D$	\mathcal{B}	bundle HD space, \mathbb{N}^D
CIM_j	CIM of j th feature	\oplus	bundling operator
\otimes	binding operator	\ominus	bundling out operator
ρ	permutation operator	$[\cdot]$	majority rule

Table A2: Averaged accuracy (%) on the train and validation folds and averaged error rate (%) on the validation folds of 10-fold cross validation for ISOLET for the tested settings of the confidence threshold α . Data are *mean* (\pm *standard deviation*), in **bold** are the validation accuracies (and error rates) that are higher (and lower) than the baseline ($\alpha = 0.00$) and underlined is the best validation accuracy and error rate.

α	Train accuracy	Validation accuracy	Validation error rate
0.00	100.00 (± 0.00)	90.56 (± 1.97)	9.44
0.25	100.00 (± 0.00)	92.43 (± 1.96)	7.57
0.50	99.99 (± 0.01)	93.14 (± 1.85)	6.86
0.75	99.92 (± 0.13)	93.64 (± 1.30)	6.36
1.00	99.62 (± 0.29)	93.41 (± 2.06)	6.59
1.25	99.78 (± 0.21)	<u>93.83</u> (± 1.93)	<u>6.17</u>
1.50	99.61 (± 0.24)	93.68 (± 1.55)	6.32

References

- Anguita, D., Ghio, A., Oneto, L., Parra, X., & Reyes-Ortiz, J. L. (2013). A public domain dataset for human activity recognition using smartphones. *21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2013*.
- Basaklar, T., Tuncel, Y., Narayana, S. Y., Gumussoy, S., & Ogras, U. Y. (2021). Hypervector design for efficient hyperdimensional computing on edge devices. *TinyML Research Symposium*. <http://arxiv.org/abs/2103.06709>
- Chang, F., Lin, C.-C., & Lu, C.-J. (2006). Adaptive prototype learning algorithms: Theoretical and experimental studies. *Journal of Machine Learning Research*, 7, 2125–2148.
- Chuang, Y. C., Chang, C. Y., & Wu, A. Y. A. (2020). Dynamic hyperdimensional computing for improving accuracy-energy efficiency trade-offs. *IEEE Workshop on Signal Processing Systems, SiPS: Design and Implementation, 2020-October*. <https://doi.org/10.1109/SiPS50750.2020.9195216>
- Dua, D., & Graff, C. (2019). *Uci machine learning repository*. <http://archive.ics.uci.edu/ml>
- Duan, S., Liu, Y., Ren, S., & Xu, X. (2022). Lehdc: Learning-based hyperdimensional computing classifier. *Design Automation Conference*. <http://arxiv.org/abs/2203.09680>
- Ge, L., & Parhi, K. K. (2020). Classification using hyperdimensional computing: A review. *IEEE Circuits and Systems Magazine*, 20, 30–47.
- Hernández-Cano, A., Matsumoto, N., Ping, E., & Imani, M. (2021). Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system. *Design, Automation and Test in Europe Conference and Exhibition (DATE)*. <https://gitlab.com/biaslab/onlinehd>
- Hersche, M., Karunaratne, G., Cherubini, G., Benini, L., Sebastian, A., & Rahimi, A. (2022). Constrained few-shot class-incremental learning. *Conference on Computer Vision and Pattern Recognition*. <https://github.com/>
- Hsiao, Y.-R., Chuang, Y.-C., Chang, C.-Y., & Wu, A.-Y. (2021). Hyperdimensional computing with learnable projection for user adaptation framework. *IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI)*, 436–447. https://doi.org/10.1007/978-3-030-79150-6_62
- Imani, M., Kong, D., Rahimi, A., & Rosing, T. (2017). Voicehd: Hyperdimensional computing for efficient speech recognition. *IEEE International Conference on Rebooting Computing (ICRC)*, 1–8.
- Imani, M., Morris, J., Bosch, S., Shu, H., Micheli, G. D., & Rosing, T. (2019). Adapthd: Adaptive efficient training for brain-inspired hyperdimensional computing. *IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 1–4.
- Ji, Z., Cui, B., Yu, Y., Pang, Y., & Zhang, Z. (2021). Zero-shot classification with unseen prototype learning. *Neural Computing and Applications*. <https://doi.org/10.1007/s00521-021-05746-9>
- Kanerva, P. (2009). Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1, 139–159. <https://doi.org/10.1007/s12559-009-9009-8>

- Kim, Y., Imani, M., & Rosing, T. S. (2018). Efficient human activity recognition using hyperdimensional computing. *Proceedings of the 8th International Conference on the Internet of Things*. <https://doi.org/10.1145/3277593.3277617>
- Kleyko, D., Khan, S., Osipov, E., & Yong, S. P. (2017). Modality classification of medical images with distributed representations based on cellular automata reservoir computing. *Proceedings - International Symposium on Biomedical Imaging*, 1053–1056. <https://doi.org/10.1109/ISBI.2017.7950697>
- Kleyko, D., Rachkovskij, D. A., Osipov, E., & Rahimi, A. (2022). A survey on hyperdimensional computing aka vector symbolic architectures, part i: Models and data transformations. *ACM Computing Surveys*. <http://arxiv.org/abs/2111.06077>
- Kleyko, D., Rahimi, A., Rachkovskij, D. A., Osipov, E., & Rabaey, J. M. (2018). Classification and recall with binary hyperdimensional computing: Tradeoffs in choice of density and mapping characteristics. *IEEE Transactions on Neural Networks and Learning Systems*, 29, 5880–5898. <https://doi.org/10.1109/TNNLS.2018.2814400>
- Kulis, B. (2012). *Metric learning: A survey*. <https://doi.org/10.1561/22000000019>
- Manabat, A. X., Marcelo, C. R., Quinquito, A. L., & Alvarez, A. (2019). Performance analysis of hyperdimensional computing for character recognition. *International Symposium on Multimedia and Communication Technology (ISMATC)*.
- Moin, A., Zhou, A., Rahimi, A., Menon, A., Benatti, S., Alexandrov, G., Tamakloe, S., Ting, J., Yamamoto, N., Khan, Y., Burghardt, F., Benini, L., Arias, A. C., & Rabaey, J. M. (2021). A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition. *Nature Electronics*, 4, 54–63. <https://doi.org/10.1038/s41928-020-00510-8>
- Neubert, P., Schubert, S., & Protzel, P. (2019). An introduction to hyperdimensional computing for robotics. *KI - Kunstliche Intelligenz*, 33, 319–330. <https://doi.org/10.1007/s13218-019-00623-z>
- Rachkovskij, D. A. (2007). Linear classifiers based on binary distributed representations. *Information Theories and Applications*, 14, 270–274.
- Rahimi, A., Benatti, S., Kanerva, P., Benini, L., & Rabaey, J. M. (2016). Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition. *IEEE International Conference of Rebooting Computing (ICRC)*.
- Rahimi, A., Kanerva, P., Benini, L., & Rabaey, J. M. (2019). Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals. *Proceedings of the IEEE*, 107, 123–143. <https://doi.org/10.1109/JPROC.2018.2871163>
- Rahimi, A., Kanerva, P., & Rabaey, J. M. (2016). A robust and energy-efficient classifier using brain-inspired hyperdimensional computing. *Proceedings of the International Symposium on Low Power Electronics and Design*, 64–69. <https://doi.org/10.1145/2934583.2934624>
- Schlegel, K., Neubert, P., & Protzel, P. (2022). Hdc-minirocket: Explicit time encoding in time series classification with hyperdimensional computing. *International Joint Conference on Neural Network*. <http://arxiv.org/abs/2202.08055>
- Watkinson, N., Givargis, T., Joe, V., Nicolau, A., & Veidenbaum, A. (2021). Detecting covid-19 related pneumonia on ct scans using hyperdimensional computing. *Proceedings of the Annual International Conference of the IEEE Engineering*

- in Medicine and Biology Society, EMBS*, 3970–3973. <https://doi.org/10.1109/EMBC46164.2021.9630898>
- Weinberger, K. Q., & Saul, L. K. (2009). Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10, 207–244.
- Widdows, D., & Cohen, T. (2015). Reasoning with vectors: A continuous model for fast robust inference. *Logic Journal of the IGPL*, 23, 141–173. <https://doi.org/10.1093/jigpal/jzu028>
- Zhou, A., Muller, R., & Rabaey, J. (2021). Memory-efficient, limb position-aware hand gesture recognition using hyperdimensional computing. *TinyML Research Symposium*. <http://arxiv.org/abs/2103.05267>
- Zou, Z., Kim, Y., Najafi, M. H., & Imani, M. (2021). Manihd: Efficient hyper-dimensional learning using manifold trainable encoder. *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 850–855.