Hyperdimensional Computing for Efficient Distributed Classification with Randomized Neural Networks

Antonello Rosato
DIET Department
University of Rome
"La Sapienza"
Rome, Italy
antonello.rosato@uniroma1.it

Massimo Panella
DIET Department
University of Rome
"La Sapienza"
Rome, Italy
massimo.panella@uniroma1.it

Denis Kleyko
UC Berkeley
Berkeley, USA
Research Institutes of Sweden
Kista, Sweden
denkle@berkeley.edu

2021 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-3900-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IJCNN52387.2021.9533805

***Abstract*—In the supervised learning domain, considering the recent prevalence of algorithms with high computational cost, the attention is steering towards simpler, lighter, and less computationally extensive training and inference approaches. In particular, randomized algorithms are currently having a resurgence, given their generalized elementary approach. By using randomized neural networks, we study distributed classification, which can be employed in situations were data cannot be stored at a central location nor shared. We propose a more efficient solution for distributed classification by making use of a lossy compression approach applied when sharing the local classifiers with other agents. This approach originates from the framework of hyperdimensional computing, and is adapted herein. The results of experiments on a collection of datasets demonstrate that the proposed approach has usually higher accuracy than local classifiers and getting close to the benchmark – the centralized classifier. This work can be considered as the first step towards analyzing the variegated horizon of distributed randomized neural networks.**

***Index Terms*—randomized neural networks, hyperdimensional computing, random vector functional link networks**

I. Introduction

In recent times, randomization techniques in neural networks have been the target of advanced studies, broadening the design prospects and generalization capabilities of different models [1]. A practical advantage of these techniques is that they can simplify the training process, which limits required computational costs, and, hence, is an appealing feature in applications where resources are constrained.

Theoretically speaking, the study of randomized neural networks can enhance the analysis of the inner workings of neural models [2], deepening the understanding of certain properties such as interpreting representation inside networks, mapping specific activities to a certain portion of the network or even gain insights on the internal structure and patterns of the weight set. While some of the most recent literature [3], [4] is steering towards deep randomized neural networks, it is still of great interest to study shallow models for a variety of reasons. The trade-off between the accuracy and the efficiency of such models is so delicate that it is far from being considered a closed problem. Moreover, there are specific application areas (e.g., low-power, edge, and green computing) in which properties of randomized neural networks make them the sole candidate for solving supervised and unsupervised problems.

Most of the studies of randomized neural networks are done in the centralized scenario where a single agent have access to the full dataset. In fact, the common case study is to analyse the performance of a model by feeding it with a complete dataset, to satisfactory prove its generalization capability. There are, however, many cases scenarios in which it is intriguing to engage in a more complex discussion with respect to how a model can be assessed when data is not found at a single point in space and time. Actually, decentralized and distributed learning techniques have being recently gaining traction in fields where data cannot be shared or moved [5], [6].

To our knowledge, the field of distributed learning as a whole has still a vast room for improvement since its compelling implementation means have not been fully explored yet, given the rise of ubiquitous data presence. As a matter of fact, owing to their peculiar design boundaries, we can consider randomized neural networks as an ideal test bench for exploiting the appealing possibilities of decentralized techniques. In fact, the prevalent advantage of randomized neural networks lies in a much simpler training process while retaining a satisfactory accuracy. For this reason, the possibility of extracting information where training is restricted to a portion of the network, enables the study of distributed neural networks implementations. While shallow networks could have some deficit in large problems, the simplicity of their characteristics can be taken advantage of in the distributed context. They could be a factor in strengthening the generalization capabilities with respect to local elementary solutions. In

The work of DK was supported by the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie Individual Fellowship Grant Agreement 839179 and in part by the DARPA's VIP (Super-HD Project) and AIE (HyDDENN Project) programs.

particular, in an interconnected network of local agents, where each agent is a randomized neural network with access to its own subset of training data, the randomized weights can be shared amongst the agents. Training data available to the agent can be used to train the rest of the local model. The agent could use its local mode for inference as is but it is expected that it will benefit from getting the additional information about the local models of other agents. This scenario stems from the following consideration: data is ubiquitous and computational power is scattered all around us. In other words, it is hard to assume that training data can always be gathered in a single place and it is unlikely to rely on a single centralized agent for handling and analyzing it. Therefore, our main focus is to set up a framework in which different agents (i.e., nodes in the network) make use of local training samples to train their own local model, and then sharing some information about their trainable connections (but no training samples) with each other to enhance local predictions. Our approach hinges on principles and premises which are also studied in the Federated Learning (FL) framework. Namely, the availability of raw data only at local agents (i.e., "siloed data") and the impossibility of sharing such raw data are common aspects of both FL and distributed learning. In fact, depending on the definition used, there are certain aspects which might differ; in particular, in our work, there is an absence of a "master" agent orchestrating the training, which is often present in FL. This fundamental distinction makes our work challenging in several aspects (computation, combination) which are specific to the fully distributed framework.

In this setting, the main contribution of this paper is a lossy compression approach based on hyperdimensional computing, which allows us decrease the amount of information being exchanged between the agents. As demonstrated by the empirical experiments, the proposed approach achieves a trade-off between the improvement in performance when compared to the local models and communication overheads necessary for information exchange between the agents.

The paper is structured as follows. Section II describes methods used for the proposed approach. The experimental setup and materials are presented in Section III. Section IV is devoted to experimental results on different variations of the proposed approach. The results are discussed in Section V. Section VI presents the concluding remarks.

II. Methods

A. Hyperdimensional Computing

Hyperdimensional computing, also known as Vector Symbolic Architectures (HDC/VSA) [7]–[12], is a family of computational frameworks based on random distributed representations, which are capable of exhibiting the behavior of both a symbolic and a neural nature in a high-dimensional space. Vectors of high (but fixed) dimensionality (denoted as D) are the basis for representing information in HDC/VSA. We refer to them as hypervectors. The information is distributed across hypervectors components, therefore, hypervectors use distributed representations [13], which are contrary to the localist representations [14] since any subset of the components can be interpreted. It is worth noting that an important property of high-dimensional spaces is that, with an extremely high probability, all random hypervectors are approximately orthogonal to each other.

HDC/VSA also defines a set of operations to manipulate hypervectors. In this paper, we use two key operations: binding and superposition. The binding operation is used to associate two hypervectors together. The result of binding is another hypervector. Here, we will use two implementations of the binding operation. First, in the Multiply-Add-Permute framework [9], the result of binding (denoted as $\mathbf{z}$) two hypervectors $\mathbf{x}$ and $\mathbf{y}$ is calculated as follows: $\mathbf{z} = \mathbf{x} \odot \mathbf{y}$, where $\odot$ denotes the binding operation, since it is implemented by component-wise multiplication. Another implementation via the circular convolution is presented in Section II-E. An important property of the binding operation is that the resultant hypervector $\mathbf{z}$ is approximately orthogonal to the hypervectors being bound.

The superposition operation combines several hypervectors into a single hypervector. In contrast to the binding operation, the hypervector resulting from the superposition operation is similar to all component hypervectors, which allows storing information in hypervectors [15], [16]. Its simplest realization is a component-wise addition. We will use this realization for the compression procedure in Section II-E. Other realizations of the superposition operation would usually involve the component-wise addition the first step. The disadvantage of the component-wise addition is that the vector space becomes unlimited so it is often practical to limit the range of values in the resultant hypervector. This, for example, can be done with a clipping function denoted as $f_\kappa(*)$:

$$f_\kappa(x) = \begin{cases} -\kappa & x \leq -\kappa \\ x & -\kappa < x < \kappa, \\ \kappa & x \geq \kappa \end{cases} \tag{1}$$

where κ is a configurable threshold. The implementation via the clipping function is in particular useful for resource-efficient variants of neural networks such as Self-Organizing Maps [17] and Echo State Networks [18], [19]. This implementation is also used for a randomized neural network presented in the next section.

The above operations applied to distributed representations allow using HDC/VSA to produce vector associations that represent, e.g., compositional structures such as sequences [10], sets [20], state automata [21], [22], hierarchies, or predicate relations [7], [8], [23]. Please consult [24] for a general overview. What is more important here is that HDC/VSA can solve a variety of learning tasks with comparable performance to conventional machine learning algorithms [25]–[27] or alternatively hypervectors can be used as an input to conventional machine learning algorithms [28]–[32].

In the next subsection, we present a particular example of how HDC/VSA were applied to modify a known machine learning algorithm.

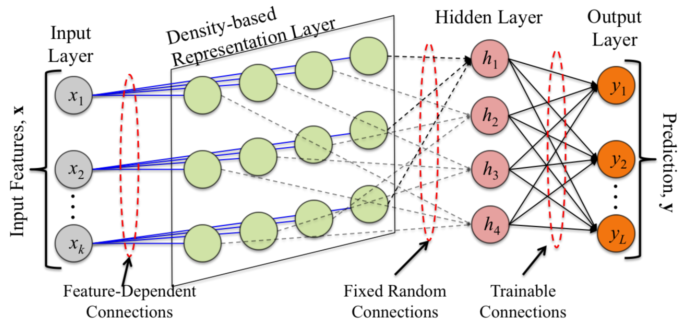

Fig. 1: Overview of the RVFL network used in this work.

B. Random Vector Functional Link networks

As a model for randomized neural networks, here we use Random Vector Functional Link (RVFL) networks [33] also known as Extreme Learning Machines [34]. In these feedforward neural networks, connections between the input and hidden layers are set at random and fixed during network's lifetime [35]. In the experiments, we used a recent modification of RVFL networks from [36], which is based on the ideas from HDC/VSA to compute the activation values of the hidden layer from input features. Fig. 1 illustrates the network's architecture.

The network begins by quantizing the values $\mathbf{x}_i$ of individual input features from the input sample $\mathbf{x} \in [K \times 1]$. Each feature is then represented in the form of D-dimensional bipolar hypervector using the thermometer code [37]. These hypervectors are collected in matrix $\mathbf{F} \in [D \times K]$. This step is called density-base representation layer in Fig. 1. The thermometer codes are going to be bound with random (but fixed) bipolar hypervectors assigned to each feature at the initialization step of the RVFL network. These K D-dimensional hypervectors are stored in a matrix $\mathbf{W}^{\text{in}} \in [D \times K]$. In Fig. 1, $\mathbf{W}^{\text{in}}$ is indicated as fixed random connections between the density-base representation and hidden layers. For the jth feature the result of the binding operation is simply $\mathbf{W}^{\text{in}}_j \odot \mathbf{F}_j$. This results gets fed into the hidden layer. The full activation of the hidden layer (denoted as $\mathbf{h} \in [D \times 1]$) is computed as:

$$\mathbf{h} = f_\kappa \left(\sum_{j=1}^{K} \mathbf{W}^{\text{in}}_j \odot \mathbf{F}_j \right). \tag{2}$$

Note that the clipping function here acts as an activation function and introduces the nonlinearity. Since the connections prior to the hidden layer are fixed, RVFL networks need to train only the classifier part of the network (trainable connections in Fig. 1) located between the hidden and output layers. The next subsection describes this step.

C. Classifiers

In this paper, we study two ways of forming the classifier (denoted as $\mathbf{W}^{\text{out}}$) of the representations produced by the hidden layer. The first way is standard in RVFL networks [33]: it uses the result of the regularized least squares applied to hidden layer activations. The second way is common in HDC/VSA and relies on class centroids, which are formed by simply superimposing all hidden layer activations for the corresponding class. In our opinion, it is interesting to analyse the generalization capabilities of these two different approaches because they reach the solution to the classification problem by carrying out radically different operations. In particular, the centroid classification exploits the VSA high dimensionality to obtain $\mathbf{W}^{\text{out}}$ using a simpler, more elementary method. This might be especially desirable in resource-constrained devices.

Once $\mathbf{W}^{\text{out}}$ is formed, to classify a given input sample $\mathbf{x}$, the RVFL network first produces the activations of the hidden layer $\mathbf{h}$. Then it computes the possible classification labels $\hat{\mathbf{y}}$, corresponding to the activations of the output layer, as the dot product between $\hat{\mathbf{y}}$ and $\mathbf{W}^{\text{out}}$: $\hat{\mathbf{y}} = \mathbf{W}^{\text{out}}\mathbf{h}$. The predicted class corresponds to a component in $\hat{\mathbf{y}}$ whose activation is the largest; this mechanism is known as winner-takes-all. We now describe the two classifiers studied herein, which will be at the core of the experiments reported in Sec. III and IV.

1) Regularized least squares classifier: The output layer of an RVFL network corresponds to a classifier. The standard way to formulate the problem of obtaining the optimal values in $\mathbf{W}^{\text{out}}$ is by minimizing the mean squared error between the ground truth and the output of the RVFL network. When solving classification problems, the ground truth is represented as one-hot vectors of the corresponding class labels (denoted as $\mathbf{y} \in [L \times 1]$), where L denotes the number of classes in the problem. Activations of the hidden layer ($\mathbf{h}$) for all M training samples are collected as rows in an activation matrix $\mathbf{H} \in [M \times D]$. While the corresponding ohe-hot encodings $\mathbf{y}$ are collected in another matrix $\mathbf{Y} \in [M \times L]$.

Following this reasoning, the regularized least squares (RLS) classifier can be obtained in one analytic step using $\mathbf{H}$ and $\mathbf{Y}$ as:

$$\mathbf{W}^{\text{out}} = (\mathbf{H}^\top \mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{H}^\top \mathbf{Y}, \tag{3}$$

where λ is a regularization parameter; $\mathbf{I} \in [D \times D]$ is the identity matrix. $D \times D$ matrix inverse dominates the computational complexity of (3), which might be rather intense especially when D is getting large.

2) Centroids classifier: In HDC/VSA, the superposition operation is one of the key operations, so it is widely used for classifiers using centroids. In this case, the classifier $\mathbf{W}^{\text{out}}$ consists of individual centroids where $\mathbf{W}^{\text{out}}_i$ denotes the centroid for class i. The main idea with centroids is that they might provide high between-class variability (i.e., centroids for different classes are dissimilar to each other) while they would also have low within-class variability [38]. In other words, it is expected that hidden layer activations of class i are going to be very similar to their class centroid $\mathbf{W}^{\text{out}}_i$. A centroid of a class is computed simply from the hidden layer activations of the training samples, which belong to that class, as follows:

$$\mathbf{W}^{\text{out}}_i = \frac{\sum_{\mathbf{h}^{(t)} \in i} \mathbf{h}^{(t)}}{\left\| \sum_{\mathbf{h}^{(t)} \in i} \mathbf{h}^{(t)} \right\|_2}, \tag{4}$$

where $\mathbf{h}^{(t)} \in i$ denotes that tth training sample is used to compute the centroid for class i only if this sample belongs to class i. The normalization is used to account for the fact that different classes might be represented by different number of training samples. In [39], the centroids were used as an initial step for a more general classifier known as Generalized Learning Vector Quantization [40] but here we limit ourselves to centroids only.

D. Distributed classification

In Section I, we already touched the reasons why a distributed approach is worth exploring. In this work, we limit our scope to the case where a set of agents is scattered in the space, forming a network. Each agent has its own computation and communication capabilities. Every agent gets its own local subset from a full dataset spread over the network. In our case, we consider these subsets being sampled independently without replacement from the full dataset[1]. Moreover, the agents cannot share their training samples as it would hinder the pure distributed definition of the problem. They, however, are allowed to share information about their classifiers with each other.

Formalizing, we consider the training data $\mathcal{T}$ being distributed over a network of N interconnected agents. The network of agents can be modeled as a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \ldots, N\}$ is the set of the agents, and $\mathcal{E}$ is the set of the edges. The connectivity of the graph is fixed and known a priori and can be described as a $N \times N$ adjacency matrix denoted as $\mathbf{\Omega}$; where $\mathbf{\Omega}_{p,q} \neq 0$ if and only if agents p and q are connected. While there are several strategies to choose the weights of the connections, in the following, for the sake of simplicity, we will consider the undirected, fully connected graph ($\mathbf{\Omega}_{p,q} = 1, \forall p, q \in \mathcal{V}$). Also, for the sake of working in a fully-distributed framework, we impose three additional constraints in the design of our approach:

- no agent is allowed to coordinate the training process;
- agents are not allowed sharing data samples;
- communication is allowed only between connected agents.

These principles do not restrict the proficiency of our proposal in its suitability to be employed in different distributed applications.

Next, we detail the means by which the distributed training procedure is carried out. First, each agent p obtains its own classifier $\mathbf{W}^{\text{out}(p)}$ using agent's training samples. Second, each agent p has a set of neighbor agents $\mathcal{S}$ that are all the agents s for which $\mathbf{\Omega}_{p,s} \neq 0$. Given the limitations placed for the purely distributed case, the crucial step is now for agent p to collect the information about the classifiers of its neighbor agents $\mathbf{W}^{\text{out}(s|s\in\mathcal{S})}$. Once these classifiers are collected, the agent can combine them together into an aggregated classifier, $\mathbf{W}^{\text{dist}(p)}$. The aggregation is done by simply summing up all the classifiers (including agent's own one) received from the neighbors:

$$\mathbf{W}^{\text{dist}(p)} = \sum_{s\in\mathcal{S}} \mathbf{W}^{\text{out}(s)}. \tag{5}$$

We do not apply any iterative approach, avoiding to analyse the convergence by using something similar to a one shot average. In the proposed approach we consider two different ways of exchanging $\mathbf{W}^{\text{out}(p)}$ between the agent. In the most straightforward case, they are shared as is. A more subtle case is when $\mathbf{W}^{\text{out}(p)}$ is sent in the compressed form to save communication resources. The next subsection elaborates on the compression procedure.

[1]With this regard, other mechanisms can be used. We have chosen this as being the most straightforward one, giving us the ability to distill the consequence of the use of centroids classifier in the distributed scenario without introducing variations that would impede the direct comparison with the RLS classifier

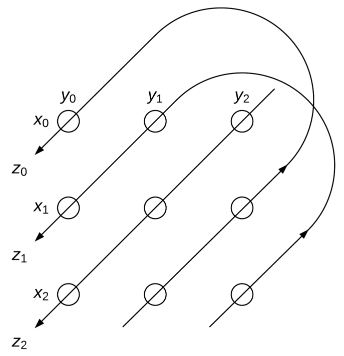

Fig. 2: An example of the circular convolution operation.

E. Compression of the classifier

As discussed in the previous section, agents need to exchange information about locally computed versions of $\mathbf{W}^{\text{out}(p)}$. Since communicating information can be a costly operation we are interested in minimizing it. HDC/VSA can be seamlessly used to compress $\mathbf{W}^{\text{out}}$ into a single D dimensional vectors. To do so we use HRR framework [7]. The first step is to form L hypervectors (do not confuse with $\mathbf{W}^{\text{in}}$) corresponding to key-value pairs where in each pair a value is a classifier for class i, $\mathbf{W}_i^{\text{out}}$ while a key is a random hypervector corresponding to class i (denoted as $\mathbf{K}_i$). The key-value pair hypervector is formed via the binding operation, which in the HRR framework is implemented via the circular convolution. The circular convolution (denoted as $\circledast$) can be seen as a compressed version of the outer product of vectors being bound. Fig. 2 shows an example for three dimensions when performing the binding:

$$\mathbf{z} = \mathbf{x} \circledast \mathbf{y}.$$

The individual components of the result in $\mathbf{z}$ are calculated as:

$$\mathbf{z}_0 = \mathbf{x}_0\mathbf{y}_0 + \mathbf{x}_2\mathbf{y}_1 + \mathbf{x}_1\mathbf{y}_2;$$
$$\mathbf{z}_1 = \mathbf{x}_1\mathbf{y}_0 + \mathbf{x}_0\mathbf{y}_1 + \mathbf{x}_2\mathbf{y}_2;$$
$$\mathbf{z}_2 = \mathbf{x}_2\mathbf{y}_0 + \mathbf{x}_1\mathbf{y}_1 + \mathbf{x}_0\mathbf{y}_2.$$

In general case, the value of the jth component is calculated as:

$$\mathbf{z}_j = \sum_{k=0}^{D-1} \mathbf{y}_k \mathbf{x}_{j-k \mod D}$$

Thus, in our scenario, we form L (which has been already defined as the number of classes) bound hypervectors: $\mathbf{K}_i \circledast \mathbf{W}_i^{\text{out}}$. These hypervectors are used to form the compressed version of $\mathbf{W}^{\text{out}}$ (denoted as $\mathbf{w}$), which contains their superposition:

$$\mathbf{w} = \sum_{i=1}^{L} \mathbf{K}_i \circledast \mathbf{W}_i^{\text{out}}.$$

Hypervector $\mathbf{w}$ is the compression version of the classifier, which is going to be sent to other agents when exchanging the information in order to enhance the performance of local classifiers. It is worth noting that it is assumed that each agent will generate its own random matrix $\mathbf{K}$ storing key hypervectors but all agents will be able to regenerate $\mathbf{K}$ on their own. This is not an unrealistic assumption since $\mathbf{K}$ can be generated using, e.g., agent's ID as a seed to initialize a pseudorandom number generator.

When an agent receives $\mathbf{w}$ from some other agent q it has perform the decompression procedure to reconstruct the approximate version of $\mathbf{W}^{\text{out}}$. The decompression is done for each $\mathbf{W}_i^{\text{out}}$ using the inverse of the corresponding key hypervector[2] of the agent q as follows:

$$\hat{\mathbf{W}}_i^{\text{out}} \approx \mathbf{w} \circledast \mathbf{K}_i^{-1}.$$

The reconstructed classifier $\hat{\mathbf{W}}^{\text{out}}$ is not going to be the exact replica of the original $\mathbf{W}^{\text{out}}$ since the compression with the superposition operation is lossy, as other key-value pairs act as a crosstalk noise during the reconstruction process. Nevertheless, our hypothesis is that when combining $\hat{\mathbf{W}}^{\text{out}}$ from different agents, their crosstalk noise will average out without having a large effect on the classification results. Thus, this approach should act as a trade-off between the advantages of exchanging the information amongst agents and the communication overheads associated with this exchange.

III. Experimental setup

A. Materials

For assessing the performance of the proposed approach, we carried out experiments on a collection of 121 benchmark classification datasets from the UCI Machine Learning Repository [41]. The collection was prepared as a part of the seminal work [42]. Here, we followed the same experimental protocol as in [42]. The only addition we introduced to the preprocessing of the datasets was the normalization of input features to the range $[0, 1]$ prior to providing input samples to a network. There was no additional preprocessing other than that. In the experiments involving only the centralized version, we used the full collection. In experiments involving distribution of the datasets between agents, we had the need to discriminate the datasets based on their size. This was done by setting up a threshold to ensure there were at least some data points in each and every local subset. Therefore, we used only datasets with more than $1,000$ samples in the training part. There are 42 such datasets in the collection.

[2]Please refer to [7] for the details of constructing the inverse of $\mathbf{K}_i$.

B. Setup

In order to obtain a good grasp on what to expect from the proposed approach for distributed classification, we experimented with three different classification models introduced below.

a) Centralized version: this model simulates the situation where all training samples are gathered in a central location. The dataset is, thus, analyzed as a whole, and the performance is evaluated on a single RVFL network. It should be noted that this version is an extreme case benchmark where all the information is available to a single agent. Therefore, this model can be considered as a kind of "upper-bound" benchmark, in the sense that the other two versions below are expected to perform worse or on a par with it, given their limited access to the data. As explained in Section I, there are practical scenarios where the centralized version might be infeasible.

b) Local version: the most elementary implementation of the proposed approach that we analyzed stems from the scattering of the computational power in the network of interconnected agents. In this case, each agent in the network is considered to have access only to local subset of the dataset, without the possibility of sharing any information with its neighbors. For evaluation purposes, in this version each subset of dataset is taken randomly without replacement from the full dataset, mimicking the real-world decentralized case.

c) Distributed version: this case represents the realization of the proposed approach to distributed classification, where the information flow among agents is restricted to the means already detailed in Sec. II-D. We allow agents sharing their locally computed classifiers $\mathbf{W}^{\text{out}}$ either as is or after a compression procedure (detailed in Section II-E). Also, in the case studied here, all agents can communicate with each other (i.e., the network is fully connected). This way enables isolating the effect of learning on decentralized dataset without taking into account additional consequences stemming from different protocols for sharing the information.

C. Hyperparameters

The search of the (D, λ, and κ) was done according to [42] with the grid search for each dataset using the RLS classifier and the centralized version. D varied in the range $[50, 1500]$ with step 50; λ varied in the range $2^{[-10,5]}$ with step 1; and κ varied between $\{1, 3, 7, 15\}$. The chosen values were used for both classifiers and all the versions reported in the next section.

It is worth noting that for all considered models, it was assumed that the agents share the same values of $\mathbf{W}^{\text{in}}$, which were chosen equiprobably from $\{-1, +1\}$. Practically, it is easy to ensure the same $\mathbf{W}^{\text{in}}$ by letting all agents to share the same seed for their pseudorandom number generators. In order to avoid the influence of a particular random selection of $\mathbf{W}^{\text{in}}$ on model's performance, all results reported below were averaged for 10 random initializations of $\mathbf{W}^{\text{in}}$.

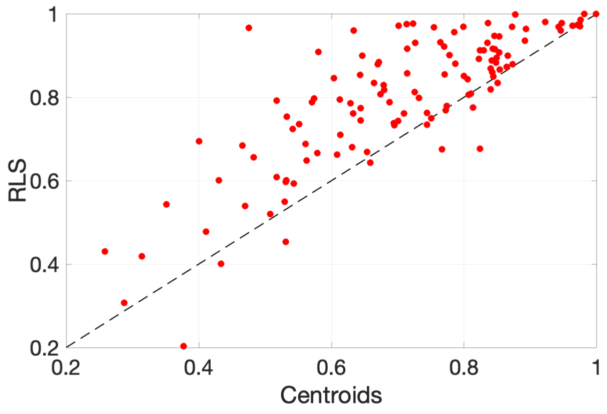

Fig. 3: The cross-validation accuracy of RLS classifier (mean 0.80) against the centroids classifier (mean 0.70) on all 121 datasets. Each point corresponds to a dataset. The results were averaged over 10 simulation runs.

IV. Experiments

A. *RLS vs. centroids in the centralized setup*

In this subsection, our goal is to compare the two classifiers considered in this study: RLS and centroids. To do so, we use the most standard setup – the centralized version. Since data is not distributed between agents, in this experiments we were able to use the full collection.

Fig. 3 presents the results in the form of the cross-validated accuracies obtained by each of the classifiers. The Pearson correlation coefficient between the results obtained from the classifiers was 0.80, which indicates high dependence between the results. We could clearly see, however, that the RLS would usually outperform the predictions obtained from the centroids. In this regard, the average accuracy for the RLS was 0.80 while that for the centroids was only 0.70. At the same time, it is worth pointing out that there were numerous datasets where the centroids performed on a par with the RLS. For example, there were 32 datasets where the accuracy of the centroids was worse to less than 0.02 of the RLS accuracy. In other words, for approximately a quarter of the datasets, the centroids classifier was a viable option. This indicates that while being very simplistic, the centroids classifier could still be a useful approach.

B. *Local and distributed vs. centralized versions: no compression*

In this subsection, our goal is to assess the difference between the local and distributed versions. We used the centralized version of the corresponding classifier as a baseline. No compression was used in the experiments reported herein. To get a grasp on the performance of the distributed paradigm, we considered three different experiments, varying only the number of agents in the network N, in the set $\{10, 50, 100\}$. For every experiment, we used local independent subsets for both training and test, and then computed the average accuracy over the whole network of agents. In Fig. 4, the results are reported for the three values of N (columns in Fig. 4) and two different classifiers (rows in Fig. 4). The results are based on 42 datasets selected from the collection as described in Section III-A.

TABLE I: Average accuracies for different versions and classifiers.

		$N = 1$	$N = 10$	$N = 50$	$N = 100$
Cent.	Local	0.70	0.67	0.63	0.60
	Distr	0.70	0.70	0.70	0.70
RLS	Local	0.83	0.74	0.66	0.62
	Distr	0.83	0.82	0.80	0.78

The results allow us making several interesting observations. The main one is that the distributed version clearly performed better than the local one, bringing the performance very close to the centralized version. This is important since it justifies that the exchange of information between the agents had its positive impact for the classification performance. Getting into more specific observations, we see that, for both classifiers, local versions are getting noticeably worse with increased N. Table I reports the average accuracies. Note that $N = 1$ corresponds to the centralized version and local and distributed versions are equivalent to each other. This tendency is not unexpected since the data was split between the agents without replacement, which means that the amount of training samples provided to a single agent was decreasing with increased N. When it comes to the distributed classification, the results for the centroids classifier always matched the centralized version. This result is also expected since in our setup after exchanging their local centroids, each agent should obtain the exact replica of the centralized version classifier. The situation is more subtle with the RLS classifier. It is clear that for any number of agents the distributed version is better than the local one (see Table I). Interestingly, the relative improvement of the distributed version over the corresponding local version was increasing with N: 10.1%, 21.1%, and 25.8%, respectively. At the same time, we see that the results were decreasing with increased N. That is because aggregating together local RLS classifiers is not equivalent to computing a single classifier from the full dataset (the average accuracy of the centralized version was 0.83) and, therefore, we do not see exactly the same performance as in the case of the centroids classifier.

C. *Distributed version with compression*

The fact that the distributed version without compression compares favorably to the local version makes it interesting to evaluate the performance of the distributed version with compression and assess it in relation to the results of the previous experiment.

Fig. 5 reports the results using the same datasets and format of presentation as in Fig. 4. Circle and square markers correspond to the results from the previous experiment for local and distributed (without compression) versions, respectively. Asterisk markers correspond to the results for the distributed version with compression.

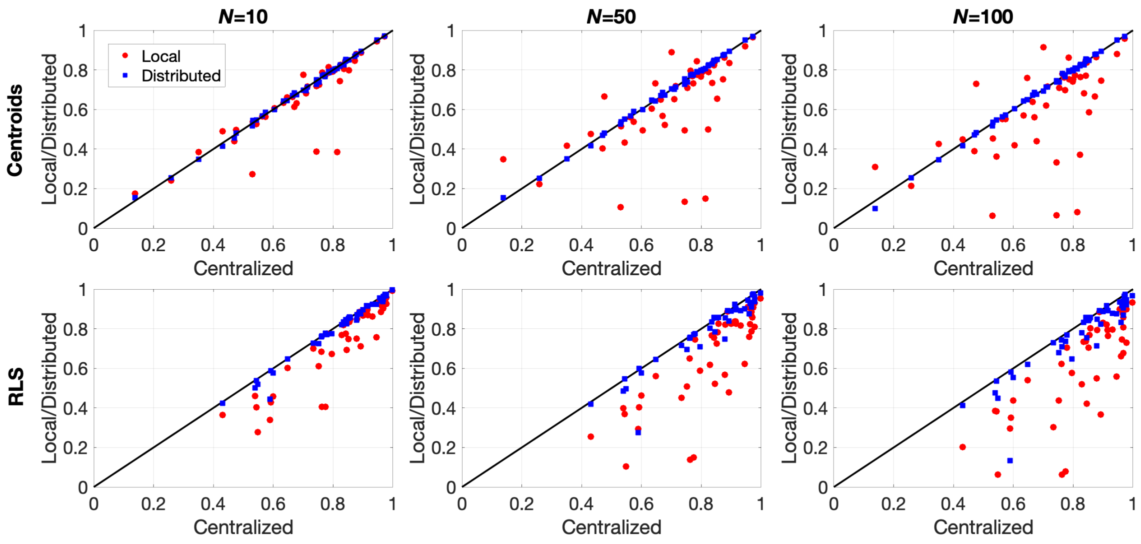

Fig. 4: The cross-validation accuracy of the local and distributed version vs. the centralized version for three different number of agents ($N \in \{10, 50, 100\}$). There was no compression for the distributed version. Upper panels: centroids classifier; Lower panels: RLS classifier. Each point corresponds to a dataset. The results were averaged over 10 simulation runs.

The centroids classifier provided some unexpected results since for some datasets the accuracy was higher than that of the centralized version. We conjecture that the cause of this effect should be the fact that for some of the datasets, the noise introduced by the compression procedure acted as some kind of regularization. Nevertheless, there were also datasets for which the distributed version with compression performed worse than the local one. However, the average accuracy for $N = 10$ was the same as for the local version (0.67). For larger values of N the average accuracy was higher than for $N = 10$ (0.70 for both $N = 50$ and $N = 100$), which demonstrated the tendency opposite to the local version where the average accuracy decreased for larger N. Moreover, the average accuracy of the distributed version with compression for $N = 50$ and $N = 100$ was the same (i.e., 0.70; Pearson correlation coefficient: 0.93) as for the distributed without compression and centralized versions.

Similar to the centroids classifier, the average results for the RLS classifier for $N = 10$ in the case of compression were the same as for the local version (0.74; Pearson correlation coefficient: 0.93). The results have, however, improved for larger values of N to 0.76 and 0.75, respectively. This results have the same trend as in the case of the distributed version without compression: the relative improvement over the local version was increasing with N: 0.0%, 15.2%, and 20.1%, respectively. Moreover, the difference in performance between the distributed version with and without compression was shrinking with N as the version without compression was better by 10.1%, 5.3%, and 4.0%, respectively.

V. Discussion

A. Related work

In this work, we have considered two classifiers. The RLS classifier is a standard choice for the RVFL networks. The centroids classifier is less common because, as we have seen in the experimental results, its average accuracy was lower than that of the RLS classifier. At the same time, there were quite a few datasets, where the centroids classifier demonstrated comparable accuracy. When taking into account its simplicity, this explains why this classifier is often a common choice in HDC/VSA literature [43]–[47]. Nevertheless, there is understanding that the centroids are not always the best choice of the classifier, which motivates to refine them using, e.g., the perceptron learning rule [48]–[50]. A more general approach to classification, which is based on centroids, is Learning Vector Quantization [40]. It has recently been introduced in the context of HDC/VSA [39] and it will be important to explore how the proposed approach will deal with the classifiers obtained via Learning Vector Quantization.

The compression procedure is similar in its spirit to the recent idea of using the binding and superposition operations of HDC/VSA to represent parameters of many deep neural networks in a single hypervector [51]. The difference in the presented procedure is that the classifier was reconstructed back from a single hypervector, which was not the case in [51]. The attempts to apply HDC/VSA in the communication domain [52]–[55] are also related but there the goal would usually be to extract the data back from the hypervector without any losses, which is not the case in our scenario.

Regarding the distributed classification domain, there are some works worth mentioning, which are related to the proposed approach. First of all, let us discuss the works pertaining to distributed classification. In this field, most of the authors presented solution that can be ascribed to the broad field of wireless sensor networks. A good review of such applicative methods can be found in [56].

In more specific terms, regarding purely distributed classification (i.e., considering only algorithms that do not imply

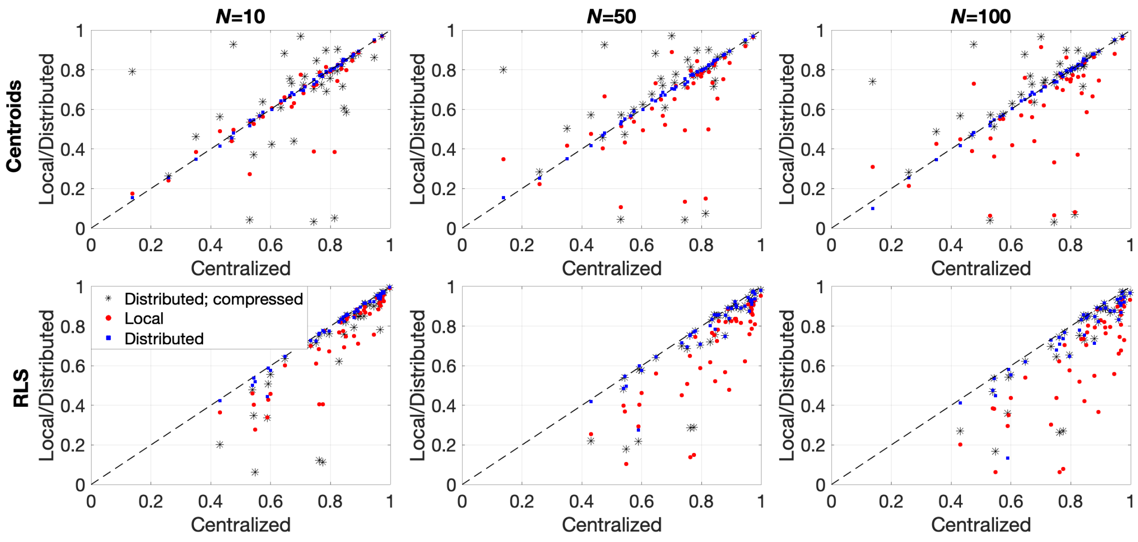

Fig. 5: The cross-validation accuracy of the local and distributed version vs. the centralized version for three different number of agents ($N \in \{10, 50, 100\}$). The results for the distributed version were obtained using the compression procedure from Section II-E. Upper panels: centroids classifier; Lower panels: RLS classifier. Each point corresponds to a dataset. The results were averaged over 10 simulation runs.

a master/slave interaction nor sharing of data of any kind), there are several works proposing a detailed analysis of the communication and convergence of the global problem. For instance, in [57] a distributed classification is studied, based on the Broad Learning System [58], which is quite complex in its computation, since it involves optimizing the Broad Learning System via Alternative Direction Method of Multipliers (ADMM). In [59] a decentralized classification solution is proposed, similar in is premises to ours, but the focus of the work is on parallel classification performance. Another work that focuses on a similar distributed approach, levering a divide-and-conquer logic, is [60]. In effect, such works studying the parallelization of a task on multiple agents in a network are more focused on analysing the scalability and convergence of the problem, veering from the actual distribution. Still regarding the diffusion of information through the network, in [61] the probability distribution is the focus of this analysis, getting to the point of studying the complexity of the combination of unbalanced agents. While interesting in the general distributed framework, we believe that the simplicity of the network computation must be at the core of the distributed learning study. Similarly, works like [62], in which the transfer of information among agents is carried out via sharing the gradients through backpropagation, are related to the present study, but do not offer a worth comparison in terms of intelligibility and effortlessness.

Moving to the specific distributed randomized network framework, there are some works in which this very scheme is studied. For instance, [63] was one of the first works in which authors introduced a distributed RVFL scheme, but, differently from us, they derived it for sequential data. In [64], a very similar technique has been studied, focusing on a single problem distributed in several agents, optimizing it via ADMM. As it is evident from this discussion, the ADMM is a well-known solution for distributed problems, and it is used also for more complex distributed network optimization [65]. By contrast, we want to examine not the optimization of the single problem, but how well a single agent can approximate the global result by only sharing some insights on the classification. Finally, it is worth mentioning a good framework studying a generalized, sparse, time varying implementation of parallel and distributed learning with neural networks [66], to give to the reader a glimpse of a broader perspective on the problem.

In the context of HDC/VSA, there are a couple of studies [67], [68], which dealt with the case where a training dataset is distributed across the network but, in contrast to this work, both studies assumed some centralization in their approaches.

B. *Extension of the current work*

We have made several assumptions when performing the experiments in this study. While the current setup is very useful for proof of concept, it is important to extend it in the future work to be able to make stronger claims about the propose approach. Below, we indicate the directions for future work.

a) Strategies for splitting data between agents: In this study, we only considered the case when data was split between agents randomly without replacement. While it allowed us abstracting from the questions of exploring strategies for splitting data, this decision had an effect on the results since the average accuracy was decreasing with increased number of agents due to the shrinking number of training sample per agent. In the future work, we need to consider other scenarios such as sub-sampling the full dataset with replacement or allowing partial sharing of samples in the agents.

b) Comparison with standard compression methods: Here, we proposed to use of HDC/VSA operations as a way to compress agent's classifier before sharing it with its neighbors. We have compared the results with the version without compression to make a proof of concept of the idea but in the future work we will make a comparison with the standard compression methods in order to see how HDC/VSA-based approach position itself.

c) Topology of the connectivity graph: In this work, we have assumed fully connected graph as a topology between the agents. It is a simple and intuitive topology to work and experiment with but it is not very practical as it is often the case that all-to-all connectivity is not available. While reporting results for other topologies falls outside of the scope of this paper, investigating alternative choices of connectivity graph that better emulate real-world scenarios by restricting number of neighbors will be an important direction for future work. It will also be important to study how the aggregated classifiers converge with iterations of information exchange.

d) Advanced classifiers: In this paper, we used only two rather simple classifiers. Future work would benefit from considering more advanced approaches to form classifiers in RVFL networks (see, e.g., [69], [70]).

VI. Conclusion

In this paper, we have proposed to improve the efficiency of RVFL networks in distributed classification problems by employing ideas from the HDC/VSA frameworks. Notably, we considered a decentralized approach where the obtained classifiers are able to reach satisfactory results when dealing with local subsets of training data, even without sharing the actual data samples. In fact, we choose to employ the RVFL networks to further benefit from its simple design, while retaining the benefits of the fast training and simple RLS or centroids solution. In the work, we made use of the compression procedure, which uses the binding operation (implemented via the circular convolution) to efficiently implement a distributed classification in which each local agent is able to reach a compelling classification accuracy without overcrowding the network. The results of the experiments carried out on fully connected networks with varying number of agents, have assessed the performance of the proposed approach with respect to the centralized and local versions. Furthermore, we highlighted the results expected in the case of relying on only the local version of a classifier, without sharing any information between the agents.

Certainly, to have a more comprehensive picture of the application of distributed learning theory to the randomized concepts, there are other areas which must be considered. Namely, the assumption of simply splitting the local subset should be challenged, while the variegated topologies the network can assume and other different compression techniques should be investigated. Nonetheless, we believe that this work is able to shine some light on how new paradigms of learning theory can be employed in diverse neural learning schemes, providing a solution to balancing the efficiency/accuracy equilibrium in such methods, which is still an open problem.

References

[1] W. Cao, X. Wang *et al.*, "A Review on Neural Networks with Random Weights," *Neurocomputing*, vol. 275, pp. 278–287, 2018.

[2] C. Gallicchio, J. D. Martín-Guerrero *et al.*, "Randomized Machine Learning Approaches: Recent Developments and Challenges," in *European Symposium on Artificial Neural Networks (ESANN)*, 2017, pp. 1–10.

[3] G. Montavon, W. Samek *et al.*, "Methods for Interpreting and Understanding Deep Neural Networks," *Digital Signal Processing*, vol. 73, pp. 1–15, 2018.

[4] C. Gallicchio and S. Scardapane, "Deep Randomized Neural Networks," in *INNS Big Data and Deep Learning Conference (INNSBDDL)*, 2019, pp. 43–68.

[5] J. Verbraeken, M. Wolting *et al.*, "A Survey on Distributed Machine Learning," *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–33, 2020.

[6] A. Rosato, R. Altilio *et al.*, "Recent Advances on Distributed Unsupervised Learning," in *International Symposium on Neural Networks, (ISNN)*, 2016, pp. 77–86.

[7] T. A. Plate, *Holographic Reduced Representations: Distributed Representation for Cognitive Structures*. Stanford: Center for the Study of Language and Information (CSLI), 2003.

[8] D. A. Rachkovskij, "Representation and Processing of Structures with Binary Sparse Distributed Codes," *IEEE Transactions on Knowledge and Data Engineering*, vol. 3, no. 2, pp. 261–276, 2001.

[9] R. W. Gayler, "Multiplicative Binding, Representation Operators & Analogy," in *Advances in Analogy Research: Integration of Theory and Data from the Cognitive, Computational, and Neural Sciences*, 1998, pp. 1–4.

[10] 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.

[11] S. I. Gallant and T. W. Okaywe, "Representing Objects, Relations, and Sequences," *Neural Computation*, vol. 25, no. 8, pp. 2038–2078, 2013.

[12] E. P. Frady, D. Kleyko *et al.*, "Variable Binding for Sparse Distributed Representations: Theory and Applications," *arXiv:2009.06734*, pp. 1–16, 2020.

[13] G. E. Hinton, J. L. McClelland *et al.*, "Distributed Representations," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, 1986, pp. 77–109.

[14] T. van Gelder, "Distributed vs. Local Representation," in *The MIT Encyclopedia of the Cognitive Sciences*, 1999, pp. 235–237.

[15] E. P. Frady, D. Kleyko *et al.*, "A Theory of Sequence Indexing and Working Memory in Recurrent Neural Networks," *Neural Computation*, vol. 30, pp. 1449–1513, 2018.

[16] D. Kleyko, A. Rosato *et al.*, "Perceptron Theory for Predicting the Accuracy of Neural Networks," *arXiv:2012.07881*, pp. 1–12, 2020.

[17] D. Kleyko, E. Osipov *et al.*, "Integer Self-Organizing Maps for Digital Hardware," in *International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–8.

[18] O. Nepomnyashchiy, A. Khantimirov *et al.*, "Method of Recurrent Neural Network Hardware Implementation," in *Computer Science Online Conference: Artificial Intelligence and Bioinspired Computational Methods (CSOC)*, 2020, pp. 429–437.

[19] D. Kleyko, E. P. Frady *et al.*, "Integer Echo State Networks: Efficient Reservoir Computing for Digital Hardware," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–14, 2020.

[20] D. Kleyko, A. Rahimi *et al.*, "Autoscaling Bloom Filter: Controlling Trade-off Between True and False Positives," *Neural Computing and Applications*, vol. 32, pp. 3675–3684, 2020.

[21] T. Yerxa, A. Anderson *et al.*, "The Hyperdimensional Stack Machine," in *Cognitive Computing*, 2018, pp. 1–2.

[22] E. Osipov, D. Kleyko *et al.*, "Associative Synthesis of Finite State Automata Model of a Controlled Object with Hyperdimensional Computing," in *Annual Conference of the IEEE Industrial Electronics Society (IECON)*, 2017, pp. 3276–3281.

[23] D. Kleyko, E. Osipov *et al.*, "Fly-The-Bee: A Game Imitating Concept Learning in Bees," *Procedia Computer Science*, vol. 71, pp. 25–30, 2015.

[24] D. Kleyko, M. Davies *et al.*, "Vector Symbolic Architectures as a Computing Framework for Nanoscale Hardware," *arXiv*, pp. 1–28, 2021.

[25] A. Rahimi, P. Kanerva *et al.*, "Efficient Biosignal Processing Using Hyperdimensional Computing: Network Templates for Combined Learning and Classification of ExG Signals," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 123–143, 2019.

[26] D. Kleyko, A. Rahimi *et al.*, "Classification and Recall with Binary Hyperdimensional Computing: Tradeoffs in Choice of Density and Mapping Characteristics," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 5880–5898, 2018.

[27] L. Ge and K. K. Parhi, "Classification using Hyperdimensional Computing: A Review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

[28] D. A. Rachkovskij, "Linear Classifiers based on Binary Distributed Representations," *Journal of Information Theories and Applications*, vol. 14, no. 3, pp. 270–274, 2007.

[29] D. Kleyko, E. Osipov *et al.*, "Distributed Representation of n-gram Statistics for Boosting Self-Organizing Maps with Hyperdimensional Computing," in *International Andrei Ershov Memorial Conference on Perspectives of System Informatics (PSI)*, 2019, pp. 64–79.

[30] T. Bandaragoda, D. D. Silva *et al.*, "Trajectory Clustering of Road Traffic in Urban Environments using Incremental Machine Learning in Combination with Hyperdimensional Computing," in *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 1664–1670.

[31] K. Shridhar, H. Jain *et al.*, "End to End Binarized Neural Networks for Text Classification," in *Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, 2020, pp. 29–34.

[32] P. Alonso, K. Shridhar *et al.*, "HyperEmbed: Tradeoffs Between Resources and Performance in NLP Tasks with Hyperdimensional Computing enabled Embedding of n-gram Statisticss," in *International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–9.

[33] B. Igelnik and Y. Pao, "Stochastic Choice of Basis Functions in Adaptive Function Approximation and the Functional-Link Net," *IEEE Transactions on Neural Networks*, vol. 6, pp. 1320–1329, 1995.

[34] G. Huang, Q. Zhu *et al.*, "Extreme Learning Machine: Theory and Applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.

[35] S. Scardapane and D. Wang, "Randomness in Neural Networks: an Overview," *Data Mining and Knowledge Discovery*, vol. 7, no. 2, pp. 1–18, 2017.

[36] D. Kleyko, M. Kheffache *et al.*, "Density Encoding Enables Resource-Efficient Randomly Connected Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–7, 2020.

[37] D. A. Rachkovskij, S. V. Slipchenko *et al.*, "Sparse Binary Distributed Encoding of Scalars," *Journal of Automation and Information Sciences*, vol. 37, no. 6, pp. 12–23, 2005.

[38] J. Daugman, "The Importance of being Random: Statistical Principles of Iris Recognition," *Pattern Recognition*, vol. 36, pp. 279–291, 2003.

[39] C. Diao, D. Kleyko *et al.*, "Generalized Learning Vector Quantization for Classification in Randomized Neural Networks and Hyperdimensional Computing," in *International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–9.

[40] D. Nova and P. Estevez, "A Review of Learning Vector Quantization Classifiers," *Neural Computing and Applications*, vol. 25, pp. 511–524, 2013.

[41] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[42] M. Fernández-Delgado, E. Cernadas *et al.*, "Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?" *Journal of Machine Learning Research*, vol. 15, no. 90, pp. 3133–3181, 2014.

[43] A. Rahimi, P. Kanerva *et al.*, "A Robust and Energy-Efficient Classifier Using Brain-Inspired Hyperdimensional Computing," in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2016, p. 64–69.

[44] F. R. Najafabadi, A. Rahimi *et al.*, "Hyperdimensional Computing for Text Classification," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 1–1.

[45] A. Rahimi, S. Benatti *et al.*, "Hyperdimensional Biosignal Processing: A Case Study for EMG-Based Hand Gesture Recognition," in *IEEE International Conference on Rebooting Computing (ICRC)*, 2016, pp. 1–8.

[46] O. Räsänen and S. Kakouros, "Modeling Dependencies in Multiple Parallel Data Streams with Hyperdimensional Computing," *IEEE Signal Processing Letters*, vol. 21, no. 7, pp. 899–903, 2014.

[47] O. Yilmaz, "Symbolic Computation Using Cellular Automata-Based Hyperdimensional Computing," *Neural Computation*, vol. 27, no. 12, pp. 2661–2692, 2015.

[48] M. Imani, D. Kong *et al.*, "VoiceHD: Hyperdimensional Computing for Efficient Speech Recognition," in *IEEE International Conference on Rebooting Computing (ICRC)*, 2017, pp. 1–.8.

[49] M. Imani, J. Morris *et al.*, "AdaptHD: Adaptive Efficient Training for Brain-Inspired Hyperdimensional Computing," in *IEEE Biomedical Circuits and Systems Conference (BIOCAS)*, 2019, pp. 1–4.

[50] Y. Kim, M. Imani *et al.*, "Efficient Human Activity Recognition Using Hyperdimensional Computing," in *International Conference on the Internet of Things (IOT)*, 2018, pp. 1–6.

[51] B. Cheung, A. Terekhov *et al.*, "Superposition of Many Models into One," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 10 868–10 877.

[52] P. Jakimovski, H. R. Schmidtke *et al.*, "Collective Communication for Dense Sensing Environments," *Journal of Ambient Intelligence and Smart Environments*, vol. 4, no. 2, pp. 123–134, 2012.

[53] D. Kleyko, N. Lyamin *et al.*, "Dependable MAC Layer Architecture based on Holographic Data Representation using Hyper-Dimensional Binary Spatter Codes," in *Multiple Access Communications (MACOM)*, ser. Lecture Notes in Computer Science, vol. 7642, 2012, pp. 134–145.

[54] H. Kim, "HDM: Hyper-Dimensional Modulation for Robust Low-Power Communications," in *IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[55] C. Simpkin, I. Taylor *et al.*, "Constructing Distributed Time-critical Applications using Cognitive Enabled Services," *Future Generation Computer Systems*, vol. 100, pp. 70–85, 2019.

[56] J. Park, S. Samarakoon *et al.*, "Communication-Efficient and Distributed Learning over Wireless Networks: Principles and Applications," *Proceedings of the IEEE*, pp. 1–24, 2021.

[57] Y. Zhai and Y. Liu, "Distributed Broad Learning System," in *International Conference on Machine Learning and Computing (ICMLC)*, 2020, pp. 567–573.

[58] C. L. P. Chen and Z. Liu, "Broad Learning System: An Effective and Efficient Incremental Learning System without the Need for Deep Architecture," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 1, pp. 10–24, 2017.

[59] C. Zhang, P. Zhao *et al.*, "Distributed Multi-task Classification: A Decentralized Online Learning Approach," *Machine Learning*, vol. 107, no. 4, pp. 727–747, 2018.

[60] S.-B. Lin, X. Guo *et al.*, "Distributed Learning with Regularized Least Squares," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 3202–3232, 2017.

[61] P. Montero-Manso, L. Morán-Fernández *et al.*, "Distributed Classification based on Distances between Probability Distributions in Feature Space," *Information Sciences*, vol. 496, pp. 431–450, 2019.

[62] N. Lewis, S. Plis *et al.*, "Cooperative Learning: Decentralized Data Neural Network," in *International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 324–331.

[63] S. Scardapane, R. Fierimonte *et al.*, "Distributed Music Classification using Random Vector Functional-Link Nets," in *International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1–8.

[64] S. Scardapane, M. Panella *et al.*, "Learning from Distributed Data Sources using Random Vector Functional-Link Networks," *Procedia Computer Science*, vol. 53, pp. 468–477, 2015.

[65] R. Fierimonte, M. Barbato *et al.*, "Distributed Learning of Random Weights Fuzzy Neural Networks," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2016, pp. 2287–2294.

[66] S. Scardapane and P. Di Lorenzo, "A Framework for Parallel and Distributed Training of Neural Networks," *Neural Networks*, vol. 91, pp. 42–54, 2017.

[67] D. Kleyko, E. Osipov *et al.*, "Hyperdimensional Computing in Industrial Systems: The Use-Case of Distributed Fault Isolation in a Power Plant," *IEEE Access*, vol. 6, pp. 30 766–30 777, 2018.

[68] M. Imani, Y. Kim *et al.*, "A Framework for Collaborative Learning in Secure High-Dimensional Space," in *IEEE International Conference on Cloud Computing (CLOUD)*, 2019, pp. 435–446.

[69] M. A. Ganaie, M. Tanveer *et al.*, "Minimum Variance Embedded Random Vector Functional Link Network," in *International Conference on Neural Information Processing (ICONIP)*, 2020, pp. 412–419.

[70] M. Tanveer, M. A. Ganaie *et al.*, "Ensemble of Classification Models with Weighted Functional Link Network," *Applied Soft Computing*, pp. 1–39, 2021.