Density Encoding Enables Resource-Efficient Randomly Connected Neural Networks

Denis Kleyko, Mansour Kheffache, E. Paxon Frady, Urban Wiklund, and Evgeny Osipov

***Abstract*—The deployment of machine learning algorithms on resource-constrained edge devices is an important challenge from both theoretical and applied points of view. In this article, we focus on resource-efficient randomly connected neural networks known as Random Vector Functional Link (RVFL) networks since their simple design and extremely fast training time make them very attractive for solving many applied classification tasks. We propose to represent input features via the density-based encoding known in the area of stochastic computing and use the operations of binding and bundling from the area of hyperdimensional computing for obtaining the activations of the hidden neurons. Using a collection of 121 real-world datasets from the UCI Machine Learning Repository, we empirically show that the proposed approach demonstrates higher average accuracy than the conventional RVFL. We also demonstrate that it is possible to represent the readout matrix using only integers in a limited range with minimal loss in the accuracy. In this case, the proposed approach operates only on small n-bits integers, which results in a computationally efficient architecture. Finally, through hardware Field-Programmable Gate Array (FPGA) implementations, we show that such an approach consumes approximately eleven times less energy than that of the conventional RVFL.**

***Index Terms*—random vector functional link networks, hyperdimensional computing, density-based encoding**

I. Introduction

An ability to provide insights and predictive analytics in real-time is the greatest demand from businesses and industries to data-driven technologies. The vector of the current development targets enabling machine learning applications on connected devices (edge computing) such as smartphones, robots, vehicles, etc. The benefits of computing at the edge are tremendous: higher reliability of solutions due to the decoupling from the network connectivity and bandwidth availability; very low latency; higher security and privacy as sensitive data are processed locally on a device.

Randomly connected neural networks such as the recently proposed class of advanced randomized learning techniques called Stochastic Configuration Networks [1] and the well-known Random Vector Functional Link (RVFL) [2] have become an increasingly popular topic of modern theoretical and applied research. On the theoretical side, the main result is that RFVLs provide universal approximation for continuous maps and functional approximations that converge in Kullback-Leibler divergence, when the target function is a probability density function [3]. When this is combined with the simplicity of RVFL's design and training process, it makes them a very attractive alternative for solving practical machine learning problems in edge computing.

The aim of this article is to present an approach for an order of magnitude increase of the resource-efficiency (memory footprint, computational complexity, and energy consumption) of RVFLs operations. The proposed approach combines techniques from two fields of computer science: stochastic computing [4] and hyperdimensional computing [5]. The fundamental idea is in the realization of activations of the hidden layer with the computationally simple operations of hyperdimensional computing, and the usage of the density-based encoding of the input features as in stochastic computing. Moreover, we enhance this approach with the integer-only readout matrix. This combination allows us to use integer arithmetics end-to-end. The novel contributions of the article are as follows:

- A resource-efficient approach to RVFLs is proposed, which uses only integer operations;
- The empirical evaluation on 121 real-world classification datasets demonstrates that the accuracy of the proposed approach is higher than that of the conventional RVFL;
- Field-Programmable Gate Array (FPGA) implementation of the proposed approach is an order of magnitude more energy-efficient and 2.5 times faster than the conventional RVFL. [1]

The article is structured as follows. The background of methods used for the proposed approach is presented in Section II. The approach itself is described in Section III. The performance evaluation follows in Section IV. Section V covers related work. Section VI presents the concluding remarks.

Manuscript received Feburary 10, 2020; revised and July 2, 2020; accepted August 8, 2020. This work was supported in part by the Swedish Research Council (grant No. 2015-04677). The work of Denis Kleyko was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Individual Fellowship grant agreement No. 839179, and DARPA's VIP program under Super-HD project.

*D. Kleyko is with the Redwood Center for Theoretical Neuroscience at the University of California, Berkeley, CA 94720, USA and also with Intelligent Systems Lab at Research Institutes of Sweden, 164 40 Kista, Sweden. E-mail: denis.kleyko@ri.se

M. Kheffache is with Netlight Consulting AB, 111 53 Stockholm, Sweden. E-mail: mansour.kheffache@netlight.com

E. P. Frady is with the Redwood Center for Theoretical Neuroscience at the University of California, Berkeley, CA 94720, USA. E-mail: epaxon@berkeley.edu

U. Wiklund is with the Department of Radiation Sciences, Biomedical Engineering, Umeå University, 901 87 Umeå, Sweden. E-mail: urban.wiklund@umu.se

E. Osipov is with the Department of Computer Science Electrical and Space Engineering, Luleå University of Technology, 971 87 Luleå, Sweden. E-mail: evgeny.osipov@ltu.se

[1] For a network with: 16 features, 4 classes, and 512 hidden neurons, which are the median values for the considered 121 datasets.

II. Background and Methods

A. Random Vector Functional Link

This subsection briefly describes the conventional RVFL. For a detailed survey of RVFLs diligent readers are referred to [6]. Fig. 1 depicts the architecture of the conventional RVFL, which includes three layers of neurons. The input layer with K neurons represents the current values of input features denoted as $\mathbf{x} \in [K \times 1]$. The output layer ($L$ neurons) produces the prediction of the network (denoted as $\mathbf{y}$) during the operational phase. The layer in the middle is the hidden layer of the network, which performs a nonlinear transformation of input features. The hidden layer contains N neurons and its state is denoted as $\mathbf{h}$.

In general, the connectivity of an RVFL is described by two matrices and a vector. A matrix $\mathbf{W}^{\text{in}} \in [N \times K]$ describes connections between the input layer neurons and the hidden layer neurons. This matrix projects the given input features to the hidden layer. Each neuron in the hidden layer has a parameter called a bias. Biases of the hidden layer are stored in a vector and denoted as $\mathbf{b} \in [N \times 1]$. The other matrix of readout connections $\mathbf{W}^{\text{out}} \in [L \times N]$ between the hidden and the output layers transforms the current activations in the hidden layer stored in $\mathbf{h}$ into the network's output $\mathbf{y}$.[2]

The main feature of the RVFL is that matrix $\mathbf{W}^{\text{in}}$ and vector $\mathbf{b}$ are randomly generated at the network initialization and stay fixed during the network's lifetime. There are no strict limitations for the generation of $\mathbf{W}^{\text{in}}$ and $\mathbf{b}$. They are usually randomly drawn from either normal or uniform distributions. Here, both $\mathbf{W}^{\text{in}}$ and $\mathbf{b}$ are generated from a uniform distribution. Following [8], the range for $\mathbf{W}^{\text{in}}$ is $[-1, 1]$ while the range for $\mathbf{b}$ is $[-0.1, 0.1]$. Since $\mathbf{W}^{\text{in}}$ and $\mathbf{b}$ are fixed the process of training RVFL is focused on learning the values of the readout matrix $\mathbf{W}^{\text{out}}$. The main advantage of training only $\mathbf{W}^{\text{out}}$ is that the corresponding optimization problem is strictly convex, thus, the solution could be found in a single analytical step.

The activations of the network's hidden layer $\mathbf{h}$ are described by the following equation:

$$\mathbf{h} = g(\mathbf{W}^{\text{in}}\mathbf{x} + \mathbf{b}), \tag{1}$$

where $g(x)$ is a nonlinear activation function applied to each neuron. Here, the sigmoid function $g(x) = \frac{1}{1+e^{-x}}$ is used. Thus, the activation function restricts the range of possible activation values in the hidden layer to the range $[0, 1]$.

The predictions issued by the output layer are calculated as:

$$\mathbf{y} = \mathbf{W}^{\text{out}}\mathbf{h}. \tag{2}$$

With respect to the training of RVFLs, the article focuses on classification tasks[3] considering only supervised-learning scenarios when the network is provided with the ground truth label for each training example. The total size of the training dataset is denoted as M. In this setting, the standard way of acquiring weights of the trainable connections between the hidden and the output layers in $\mathbf{W}^{\text{out}}$ matrix is via solving the ridge regression (which is a special case of Tikhonov regularization) problem, which minimizes the mean square error between predictions (2) and the ground truth. In particular, the activations of the hidden layer $\mathbf{h}^T$ for each training example are collected together in matrix $\mathbf{H} \in [M \times N]$. Matrix $\mathbf{Y} \in [M \times L]$ stores the corresponding ground truth classifications using one-hot encodings. Given $\mathbf{H}$ and $\mathbf{Y}$, $\mathbf{W}^{\text{out}}$ is calculated as follows:

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

where $\mathbf{I}$ denotes an identity matrix of the suitable dimensionality ($\mathbf{I} \in [N \times N]$); λ is a hyperparameter (scalar) determining the weight of the regularization part.[4]

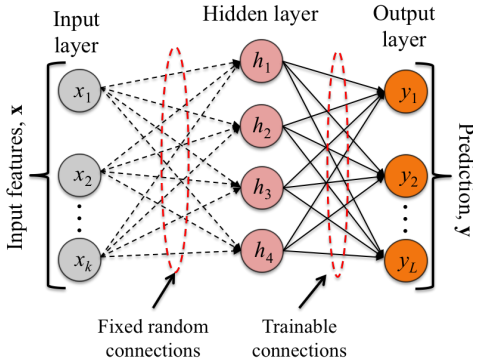

Fig. 1. The architecture of the conventional Random Vector Functional Link. In the presented example, the number of hidden neurons is set to $N = 4$.

B. Density-based encoding of scalars

The idea of representing scalars as vectors is not new. It has been independently proposed in several areas. The area of stochastic computing [4] is probably the most notable example since the whole idea of the stochastic computing is that it is possible to implement arithmetics on scalars using boolean operations on vectors (in general, streams) of bits. The rate coding model of neuronal firing used, e.g., in spiking neural networks is another notable example. Stochastic computing operates with scalars between 0 and 1, which are represented as random bit vectors where the scalar being encoded determines the probability of generating ones. Thus, the density of ones in the obtained bit vector encodes the scalar, hence such a representation method is called the density-based encoding. Generating random streams is important because the independence of two vectors is a prerequisite for using boolean operations to implement the arithmetics on them (e.g., AND for multiplication). Note that for the proposed approach no arithmetic operations will be performed with the density-based encodings of scalars. Therefore, the randomness of

[2]Strictly speaking, in the most general case, the readout matrix could also include connections between the input layer and the output layer. However, in the scope of this study, we only consider the case when the output layer predictions are obtained from the activations of the hidden layer. The interested readers are referred to work [7], which performed a comprehensive evaluation of different design choices for the RVFL.

[3]Though, the proposed approach is also applicable to regression problems. One may, however, expect that the quality of predictions might be more sensitive to the use of the density-based encoding.

[4]Note that (3) is computationally simpler compared to, e.g., the backprop algorithm and it is implementable efficiently on CPUs as well as on GPUs.

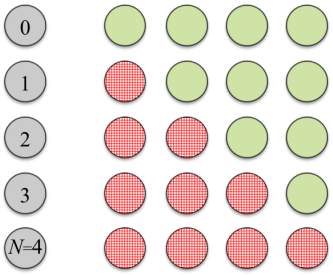

Fig. 2. An example of the density-based encoding when the dimensionality of representation is set to $N = 4$.

representations for encoding scalars is not compulsory in this study. In fact, from the simplicity point of view, it is more advantageous to use a structured version of the density-based encoding, which does not require a source of randomness. We will use the structured version of the density-based encoding also known under the name thermometric encoding [9] for the rest of this paper.

The most intuitive way of presenting the concept of the density-based encoding is via visualization. Fig. 2 illustrates all possible values, which could be encoded when dimensionality of the representation[5] is set to $N = 4$. Fig. 2 indicates that using the density-based encoding it is possible to represent $N + 1$ different values. The most convenient way of denoting these values is by using integers in the range $[0, N]$ (nodes on the left in the figure). In this case, in order to obtain the encoding of a given value v, it is necessary to set v leftmost positions of the vector to "one" (hashed red nodes in the figure) while the rest of the vector is set to "zero" (filled green nodes). In the case of bipolar representations used below, "one" corresponds to -1 while "zero" corresponds to 1.

Recall, however, that input features are not integers in the range $[0, N]$. Instead, it is assumed that a feature x_i is represented by a real number in the range $[0, 1]$. The task is to represent the current value of the feature as a vector $\mathbf{f} \in [N \times 1]$ using the above density-based encoding. Since, the encoding requires a finite set of values between 0 and N, real numbers are first discretized using a fixed quantization step, which is determined by N. Given the current value of the feature, it is quantized to the closest integer as:

$$v = \lfloor x_i N \rceil, \tag{4}$$

where $\lfloor * \rceil$ denotes rounding to the the closest integer. The obtained v will determine the density-based encoding $\mathbf{f}$. The presented procedure allows generating density-based encodings for the whole feature vector $\mathbf{x}$. Matrix $\mathbf{F} \in [N \times K]$, where K denotes the number of features, contains the density-based encodings $\mathbf{f}$ of the current values of $\mathbf{x}$.

C. Hyperdimensional computing

Hyperdimensional computing [10], [11] also known as Vector Symbolic Architectures is a family of bio-inspired methods of representing and manipulating concepts for cognitive architectures and their meanings in a high-dimensional space. Vectors of high (but fixed) dimensionality (denoted as N) are the basis for representing information in hyperdimensional computing.[6] The information is distributed across HD vector's positions, therefore, HD vectors use distributed representations. Distributed representations [12] are contrary to the localist representations since any subset of the positions can be interpreted. This is very relevant to the density-based encoding introduced in the previous subsection since the encoding in $\mathbf{f}$ is also distributed.

In the scope of this article, columns of $\mathbf{W}^{\text{in}}$ matrix are interpreted as HD vectors, which are generated randomly. These HD vectors are bipolar ($\mathbf{W}^{\text{in}} \in \{-1, +1\}^{[N \times K]}$) and random with equal probabilities for $+1$ and -1. It is worth noting that an important property of high-dimensional spaces is that with an extremely high probability all random HD vectors are dissimilar to each other (quasi-orthogonal). In order to manipulate HD vectors, hyperdimensional computing defines operations on them. In this article, we implicitly use only two key operations: binding and bundling.

The binding operation is used to associate two HD vectors together. The result of binding is another HD vector. Here, the result of binding (denoted as $\mathbf{z}$) two vectors $\mathbf{x}$ and $\mathbf{y}$ is calculated as follows: $\mathbf{z} = \mathbf{x} \odot \mathbf{y}$, where the notation $\odot$ for the Hadamard product is used to denote the binding operation since this article uses position-wise multiplication for binding. An important property of the binding operation is that the resultant HD vector $\mathbf{z}$ is quasi-orthogonal to the HD vectors being bound.

The second operation is called bundling. The bundling operation combines several HD vectors into a single HD vector. Its simplest realization is a position-wise addition. However, when using the position-wise addition, the vector space becomes unlimited, therefore, it is practical to limit the values of the result. This could be achieved with, e.g., a clipping function (denoted as $f_\kappa(*)$):

$$f_\kappa(x) = \begin{cases} -\kappa & x \leq -\kappa \\ x & -\kappa < x < \kappa \\ \kappa & x \geq \kappa \end{cases} \tag{5}$$

In the clipping function, κ is a configurable threshold parameter. Thus, in this article, the bundling operation is implemented via position-wise addition limited via the clipping function. For example, the result (denoted as $\mathbf{a}$) of bundling HD vectors $\mathbf{x}$ and $\mathbf{y}$ is simply: $\mathbf{a} = f_\kappa(\mathbf{x} + \mathbf{y})$. In contrast to the binding operation, the resultant HD vector $\mathbf{a}$ is similar to all bundled HD vectors, which allows, e.g., storing information in HD vectors [13]. For example, we have demonstrated the usefulness of the clipping function for resource-efficient implementations of Self-Organizing Maps [14] and Echo State Networks [15], [16].

III. RVFL WITH DENSITY-BASED ENCODINGS

This section presents an architecture of the RVFL utilizing the density-based encoding. The approach is illustrated in

[5]It will become evident in Section III why the same notation N as for the number of hidden neurons is used.

[6]These vectors are referred to as high-dimensional vectors or HD vectors.

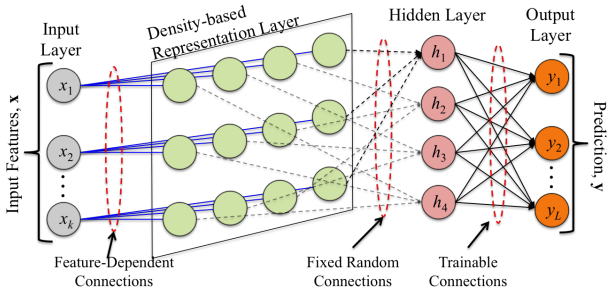

Fig. 3. The architecture of the Random Vector Functional Link, which relies on the density-based encoding. In the presented example, the number of hidden neurons as well as the dimensionality of encoding are set to $N = 4$.

Fig. 3. The architecture is intentionally depicted to be as structurally identical to the conventional RVFL (Fig. 1) as possible. The major difference is that the proposed approach is illustrated with four layers of neurons: input layer ($\mathbf{x}$, K neurons); density-based representation layer ($\mathbf{F}$, $N \times K$ neurons); hidden layer ($\mathbf{h}$, N neurons); and output layer ($\mathbf{y}$, L neurons). Thus, in contrast to the conventional RVFL, the hidden layer is not connected directly to the input layer. Instead, each input feature is first transformed to a row of neurons storing its density-based encodings. These vectors constitute the density-based representation layer, which in turn is connected to the hidden layer. Note also, that the input and density-based representation layers are not fully-connected. Each neuron in the input layer is only connected to N neurons in the corresponding row of the next layer. Moreover, these connections (blue lines in Fig. 3) are called "feature-dependent" because the activation of the i-th input neuron $\mathbf{x}_i$ will be quantized to the closest integer v according to (4); in turn v determines the number of the rightmost connections, which transmit -1, the remaining connections from that neuron transmit $+1$. Since each neuron in the density-based representation layer has only one incoming connection, the input activations are projected in the form of the bipolar matrix $\mathbf{F}$.

It is also important to mention that the density-based representation and hidden layers are not fully-connected. In fact, each neuron in the density-based representation layer has only one outgoing connection. Therefore, the matrix $\mathbf{W}^{\text{in}}$ describing the fixed random connections to the hidden layer is still $\mathbf{W}^{\text{in}} \in [N \times K]$. Moreover, these connections have a clear structure. In Fig. 3 the connections are structured in such a way that each column in $\mathbf{F}$ is connected to one of the hidden layer neurons. It explains why the number of hidden neurons N also determines the dimensionality of the density-based encoding of features: each hidden neuron has its corresponding column in $\mathbf{F}$ (see Fig. 4). Note that in Fig. 4 N is set to 10 only for visualization purposes. In practice, the values of N are larger.

Similar to the conventional RVFL, the values of $\mathbf{W}^{\text{in}}$ are also generated randomly. However, the values are drawn equiprobably from $\{-1, +1\}$. Thus, similar to $\mathbf{F}$, $\mathbf{W}^{\text{in}}$ is also a bipolar matrix. When reflecting to the ideas of hyperdimensional computing, $\mathbf{W}^{\text{in}}$ should be interpreted as K N-dimensional bipolar HD vectors. In other words, each feature is assigned with the corresponding HD vector. Thus, a conceptual intermediate step before getting input values of the hidden neurons is the binding operation between features' HD vectors and their current density-based encoding.

Finally, the proposed approach uses different nonlinear activation function in the hidden layer, the clipping function (5) is used instead of the sigmoid function. The clipping function is characterized by the threshold value κ regulating nonlinear behavior of the neurons and limiting the range of activation values. Summarizing aforementioned differences, activations of the hidden layer $\mathbf{h}$ are obtained as follows:

$$\mathbf{h} = f_{\kappa}(\sum \mathbf{F} \odot \mathbf{W}^{\text{in}}), \tag{6}$$

where $\sum$ is a column-wise summation. Note that in contrast to (1) there is no bias term since it has been found empirically that its presence does not improve classification performance. Once the activations of the hidden layer $\mathbf{h}$ are obtained, the rest of the network works in the same way as the conventional RVFL. The predictions in $\mathbf{y}$ are calculated according to (2).

In order to make operations of the proposed approach more intuitive, Fig. 4 presents a numerical example of acquiring the activations of the hidden layer. First, the input layer with $K = 5$ neurons sets the values of the current feature vector. These values are quantized to integers in the range [0,10] (since $N = 10$). The quantized values determine the neurons of the density-based encoding, which are set to -1 (the rest is $+1$). For example, since the third feature is quantized to $v = 10$ all values of its density-based encoding are set to -1. The bottom left panel shows a randomly generated $\mathbf{W}^{\text{in}}$. Once $\mathbf{F}$ is obtained, we calculate the Hadamard product $\mathbf{F} \odot \mathbf{W}^{\text{in}}$, which is denoted as "bound representations" in Fig. 4. The row-wise summation of the resultant matrix represents the input values of the hidden layer. Finally, the clipping function ($\kappa = 2$ in Fig. 4) is used in the hidden layer to get $\mathbf{h}$.

Note that due to the way of forming $\mathbf{F}$ and $\mathbf{W}^{\text{in}}$, the input to the hidden layer neurons is always integers in the range $[-K, K]$. Moreover, even after the clipping the activations of neurons are integers in the range $[-\kappa$ and $\kappa]$ (practically, $\kappa < K$). Thus, each hidden neuron can be represented using only $\lceil \log_2(2\kappa{+}1) \rceil$ bits of memory. For example, when $\kappa = 3$, there are seven unique activations of a neuron, which can be stored with just three bits. Last but not least, it is worth mentioning that for an efficient implementation the explicit calculation of $\mathbf{F}$ is redundant. As it could be seen from Fig. 4, the same result as $\mathbf{F} \odot \mathbf{W}^{\text{in}}$ could be obtained if for each feature we use v as an indicator of which signs should be changed in $\mathbf{W}^{\text{in}}$. As it will be shown in the next section, these properties give a major advantage over the conventional RFVL for resource-efficient implementation on digital hardware.

Since in the proposed approach the part of the network between the hidden and output layers is not modified, the simplest case is to train the readout matrix $\mathbf{W}^{\text{out}}$ in the same manner as for the conventional RVFL (Section II-A). Note that while training the readout matrix $\mathbf{W}^{\text{out}}$ according to (3) there is no need to normalize the activation values in $\mathbf{h}$. Moreover, since the goal is to obtain a very simplistic implementation, it is worth considering alternatives where $\mathbf{W}^{\text{out}}$ would contain only integer values in a small limited range. In particular, we have considered three options: quantizing the result of

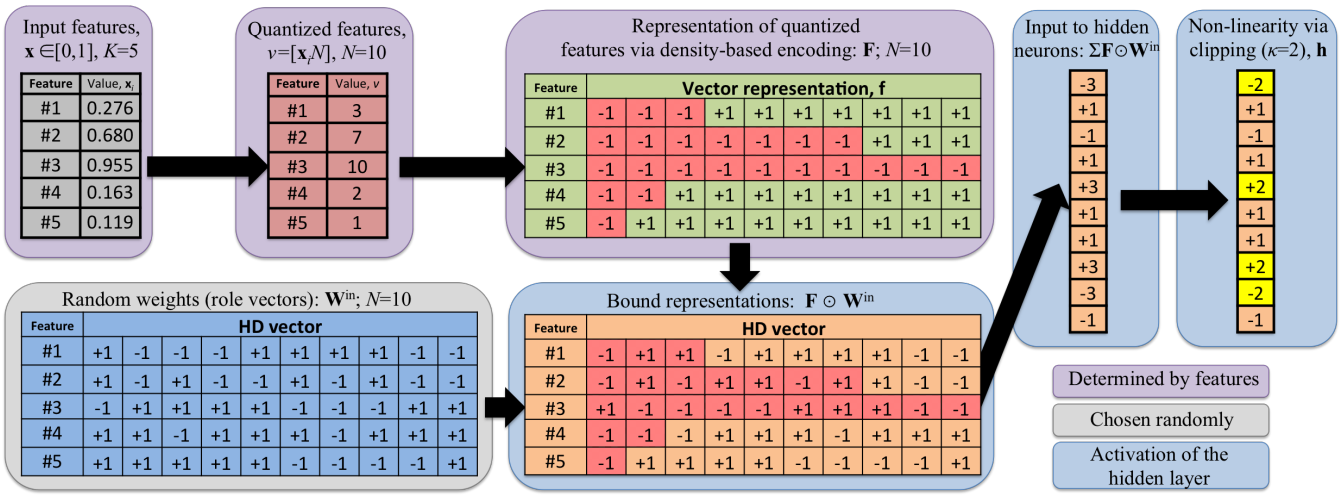

Fig. 4. An example of activating the hidden layer with density-based encodings: $K = 5$; $N = 10$. Note that N is set to 10 for visualization purposes only.

regression (3); using a genetic algorithm (GA) initialized randomly; using GA initialized with the quantized result of regression. During the search, GA used the cost function for the generalized Learning Vector Quantization [17].

IV. Performance Evaluation

In this section, the proposed approach is verified in three scenarios.[7] The first scenario compares it against the conventional RVFL in the case when the weights of the readout matrix are real numbers for both approaches. The second scenario compares the results for the real-valued readout matrix against the considered strategies of obtaining integer-valued readout matrix. The final scenario compares FPGA implementations of the proposed approach and the finite precision RVFL [8] in the case of a limited energy budget. All reported results[8] are based on 121 real-world classification datasets obtained from the UCI Machine Learning Repository[9] [18]. The considered collection of datasets has been initially analyzed in a large-scale comparison study of different classifiers and the interested readers are kindly referred to the original work [19] for more details. The only preprocessing step was to normalize features in the range $[0, 1]$. Finally, the reported accuracies were averaged across five independent initializations.

A. Comparison with the conventional RVFL

First, we compare the conventional RVFL with the proposed approach when computational resources for both approaches are not limited. The search of the hyperparameters has been done according to [19] using the grid search over λ and N in the case of the conventional RVFL and additionally considering κ for the proposed approach; N 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 obtained optimal hyperparameters were used to estimate the cross-validation accuracy on all datasets. In order to visualize the results, we rely on the same approach as reported in [20]. Fig. 5 presents the accuracy of the conventional RVFL against the proposed approach. First, it is important to note that as expected the correlation coefficient between the obtained results is high (0.86). Moreover, the average accuracy for the conventional RVFL is 0.76 while that for the proposed approach is 0.80.[10] The difference in accuracy was statistically significant using 5% significance level according to two-sample hypothesis testing. It is not absolutely intuitive why the proposed approach demonstrates higher accuracy. Nevertheless, one hypothesis is that the quantization for the density-based encoding might provide extra regularization.[11]

B. The effect of quantized readout weights

Fig. 6 presents the average accuracy of the proposed approach for three considered strategies of obtaining the readout matrix with integer values against the average accuracy from the previous experiment. The considered ranges are symmetric and the figure indicates only positive boundaries. It is clear that if the result of regression is quantized (dashed line) to very few levels the accuracy is affected significantly. However, with the increased number of levels, the accuracy approaches the baseline and it is concluded that 5-bits per weight results in a very close approximation. Refining the quantized result of regression with GA (dash-dot line) certainly improved the accuracy for a small number of quantization levels, which is in line with the results in [8]. However, using GA for the number of levels larger than six was not beneficial. Random GA initialization (dotted line) decreased the accuracy.

[7]The diligent readers are kindly referred to the Supplementary materials, which provide additional experimental evaluation to further justify the proposed approach.

[8]The implementation of the experiments reported in the article is available online via https://github.com/denkle/Density-Encoding-Enables-Resource-Efficient-Randomly-Connected-Neural-Networks.

[9]Available online: http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/jmlr/data.tar.gz.

[10]In [19] the highest mean accuracy 0.82 was obtained for Random Forest.

[11]For the conventional RVFL with quantized inputs the mean accuracy was 0.753 (0.755 for non-quantized). The correlation coefficient was 0.986. Thus, the improvement cannot be caused barely by the input quantization. Also, for the case when the conventional RVFL additionally used the connections between the input layer and the output layer, there was no significant improvement in the accuracy as the mean accuracy was 0.762.

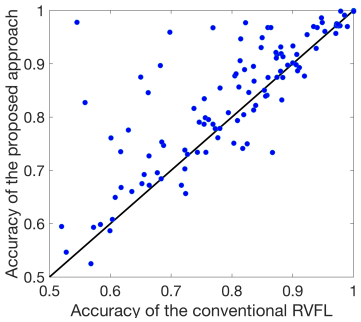

Fig. 5. Cross-validation accuracy of the conventional RVFL against the proposed approach. A point corresponds to a dataset.

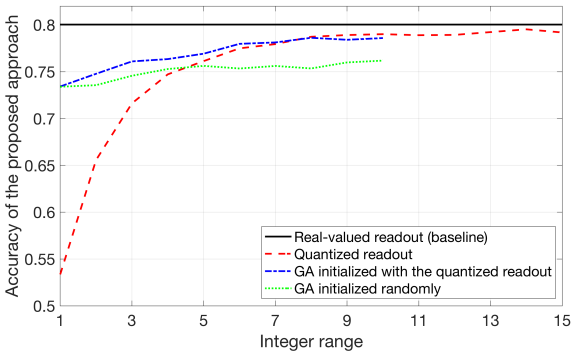

Fig. 6. Average cross-validation accuracy of the proposed approach for different integer readout strategies.

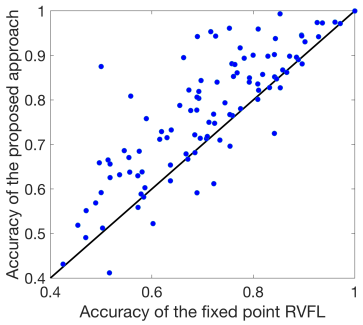

Fig. 7. Cross-validation accuracy of the finite precision RVFL against the proposed approach in case of the fixed energy budget.

C. Performance in the case of limited resources

The third experiment compares FPGA hardware implementations of the proposed approach and the finite precision RVFL in the case of a fixed energy budget per one classification pass. The idea of restricting the energy budget could be seen as an intuitive set-up for comparing bounded-optimality [21] of two approaches. Finite precision RVFL [8] with 8-bits per neuron/weight was used since it is more efficient than the conventional RVFL.[12] Following the conclusions from the previous experiment, the resolution of the readout weights of the proposed approach was set to 5-bits. Both approaches were deployed on ZedBoard FPGA and the energy consumption was estimated with the Xilinx Power Estimator tool. The energy budget was set to $3.2\,\mu$J to reflect a network with typical parameters.[13] Fig. 7 presents the accuracy of the proposed approach (average 0.73) against the finite precision RVFL (average 0.65). The difference in accuracy was statistically significant using 5% significance level according to two-sample hypothesis testing. Due to the limited resources, values are lower than in the first experiment, nevertheless, the results are impressive when the performance of our approach is compared to the fixed point RVFL.

V. Related work

This section briefly describes the related work. First of all, the readers generally interested in neural networks, which rely on randomly created connections, are kindly referred to the survey in [6].

A. Paradigms used for the proposed approach

In order to design a resource-efficient RVFL algorithm, the proposed approach combines the ideas from two areas. These are the density-based encoding from stochastic computing and the binding and bundling operations from hyperdimensional computing. Since both are research fields on their own, here we only indicate the introductory papers facilitating entrance to the areas. The recent magazine article [4] is probably the most approachable reading for stochastic computing. With respect to hyperdimensional computing, the best starting point is the tutorial-like article [5] by Kanerva.

B. Resource-efficient RVFL

Recall that even the conventional RVFL networks are considered to be one of the simplest approaches for machine learning. This fact explains why the efforts on pushing the resource-efficiency of RVFL networks to the extreme are rather limited. The most relevant works in this direction are [8], [22]. Similar to the present study, both works use FPGA for hardware experiments. Moreover, both works rely on finite precision implementation for improving the resource-efficiency. The work [22], however, heavily focuses on the process of obtaining the weights of the readout matrix, which is not the case here. The work [8], which focuses on the operational phase, is used here as the baseline for comparison with the proposed approach. However, none of the previous works in the area of RVFL, to the best of our knowledge, have been focusing on using the combination of the density-based encoding with the binding operation. As an important topic for future research, we see the theoretical characterization of the classification performance improvement obtained with the proposed approach. As indicated earlier, one hypothesis is that quantization and the density-based encoding provide extra regularization. In order to move in this direction, the relevant works are related to a phenomenon of network generalization improvement, e.g., via adding a noise [23] or discretizing quantitative features [24].

C. Simplification of neural networks

Finally, it is worth mentioning that in recent years the simplification of computing architectures for neural networks is an important research topic. Notable examples are works [25], [26], [27], which have been evaluated on convolutional neural networks; work [28] that has introduced networks with ternary activations and work [29] that has introduced networks where all parameters are binary. It is worth mentioning that in contrast to the bit-wise networks [29], the proposed use of

[12]When comparing FPGA implementations of the conventional RVFL and the proposed approach (integer readout) for a network where $K = 16$, $N = 512$, and $L = 4$ (median values for the 121 UCI datasets) the proposed approach consumed about 11 times less energy and was 2.5 times faster.

[13]Since in each dataset input and output sizes are fixed, the budget was enforced by determining number of hidden layer neurons within the budget.

the density-based encoding does not require the binarization of the input features, which often worsens the accuracy.

VI. Conclusions

This article proposed a resource-efficient fully-integer approach to randomly connected neural networks. The key enabler for efficiently obtaining activations of the hidden neurons is the combination of the representation of input features via the density-based encoding used in the stochastic computing, and the use of binding and bundling operations from hyperdimensional computing area. Integer values of the readout matrix could be obtained with minimal loss in the accuracy, e.g., by simple rounding of the ridge regression solution, which in turn could be fine-tuned by the genetic algorithm. The empirical evaluation was performed on 121 real-world datasets. The proposed approach demonstrated a higher average accuracy than the conventional RVFL networks while being 2.5 times faster and consuming eleven times less energy (typical network on FPGA). Finally, the accuracy of the proposed approach significantly prevailed that of the finite precision RVFL networks when both networks implemented on hardware were constrained to a fixed energy budget.

Despite that this work has focused only on classification tasks it is worth mentioning that the proposed approach of forming the activations of the hidden layer should be seen as a generic structured representation scheme based on high-dimensional random projections that allow for direct learning of complex nonlinear functions. Therefore, a promising direction for future work is to develop an analytical theory similar to the capacity theory of such representations [13] that would relate the quality of approximations based on the complexity of a nonlinear function, number of its inputs and outputs, and a number hidden neurons and their resolution.

Last but not least, we conjecture that the density-based encoding will be useful for developing resource-efficient versions of other neural networks. For example, as it has been recently shown in [30] that binarizing initial layers of convolutional neural networks could easily harm their accuracy. We expect that the density-based encoding of input features will solve this issue.

References

[1] D. Wang and M. Li, "Stochastic Configuration Networks: Fundamentals and Algorithms," *IEEE Transactions on Cybernetics*, vol. 47, no. 10, pp. 3466–3479, 2017.

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

[3] H. D. Nguyen, D. Wang, and G. J. McLachlan, "Randomized Mixture Models for Probability Density Approximation and Estimation," *Information Sciences*, vol. 467, pp. 135–148, 2018.

[7] L. Zhang and P. Suganthan, "A Comprehensive Evaluation of Random Vector Functional Link Networks," *Inf. Sci.*, vol. 367-368, pp. 1094–1105, 2016.

[4] A. Alaghi and J.P. Hayes, "Computing with Randomness," *IEEE Spectrum*, vol. 55, no. 3, pp. 46–51, 2018.

[5] 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.

[6] S. Scardapane and D. Wang, "Randomness in Neural Networks: an Overview," *Data Min. and Know. Disc.*, vol. 7, pp. 1–18, 2017.

[8] A. Rosato, R. Altilio, and M. Panella, "Finite Precision Implementation of Random Vector Functional-Link Networks," in *DSP*, 2017, pp. 1–5.

[9] D. A. Rachkovskij, S. V. Slipchenko, E. M. Kussul, and T. N. Baidyk, "Sparse Binary Distributed Encoding of Scalars," *Jou. of Aut. and Inf. Sci.*, vol. 37, no. 6, pp. 12–23, 2005.

[10] T. A. Plate, *Holographic Reduced Representations: Distributed Representation for Cognitive Structures*. Stanford: CSLI, 2003.

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

[12] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart, "Distributed Representations," in *PDP*, 1986, pp. 77–109.

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

[14] D. Kleyko, E. Osipov, D. De Silva, U. Wiklund, and D. Alahakoon, "Integer Self-Organizing Maps for Digital Hardware," in *IJCNN*, 2019, pp. 1–8.

[15] D. Kleyko, E. P. Frady, and E. Osipov, "Integer Echo State Networks: Hyperdimensional Reservoir Computing," *arXiv:1706.00280*, pp. 1–10, 2019.

[16] O. Nepomnyashchiy, A. Khantimirov, D. Galayko, and N. Sirotinina, "Method of Recurrent Neural Network Hardware Implementation," in *Computer Science On-line Conference: Artificial Intelligence and Bioinspired Computational Methods (CSOC)*, ser. Advances in Intelligent Systems and Computing, vol. 1225, 2020, pp. 429–437.

[17] A. Sato and K. Yamada, "Generalized Learning Vector Quantization," in *NeurIPS*, 1996, pp. 423–429.

[18] D. Dua and C. Graff, "UCI machine learning repository," 2019. [Online]. Available: http://archive.ics.uci.edu/ml

[19] M. Fernandez-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?" *JMLR*, vol. 15, pp. 3133–3181, 2014.

[20] M. Olson, A. J. Wyner, and R. Berk, "Modern Neural Networks Generalize on Small Data Sets," in *NeurIPS*, 2018, pp. 1–10.

[21] S. J. Russell and D. Subramanian, "Provably Bounded-Optimal Agents," *Journal of Artificial Intelligence Research*, vol. 2, pp. 575–609, 1995.

[22] J. M. Martinez-Villena and A. Rosado-Munoz and E. Soria-Olivas, "Hardware Implementation Methods in Random Vector Functional-Link Networks," *Applied Intelligence*, vol. 41, no. 1, pp. 184–195, 2014.

[23] C. M. Bishop, "Training with Noise is Equivalent to Tikhonov Regularization," *Neural Computation*, vol. 7, no. 1, pp. 108–116, 1995.

[24] N. A. Zaidi, Y. Du, and G. I. Webb, "On the Effectiveness of Discretizing Quantitative Attributes in Linear Classifiers," *arXiv:1701.07114*, pp. 1–24, 2017.

[25] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," *arXiv:1712.05877*, pp. 1–14, 2017.

[26] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks," in *NeurIPS*, 2016, pp. 1–9.

[27] ——, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *JMLR*, vol. 18, pp. 1–30, 2018.

[28] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Petrot, "Ternary Neural Networks for Resource-Efficient AI Applications," in *IJCNN*, 2017, pp. 2547–2554.

[29] M. Kim and P. Smaragdis, "Bitwise Neural Networks," in *ICML*, 2015, pp. 1–5.

[30] A. G. Anderson and C. P. Berg, "The High-Dimensional Geometry of Binary Neural Networks," in *ICLR*, 2018, pp. 1–13.