Applied Soft Computing 112 (2021) 107797

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A constructive approach to data-driven randomized learning for feedforward neural networks

Grzegorz Dudek

Czestochowa University of Technology, Faculty of Electrical Engineering, 17 Armii Krajowej Ave., 42-200, Czestochowa, Poland

ARTICLE INFO

Article history:
Received 9 September 2020
Received in revised form 2 July 2021
Accepted 7 August 2021
Available online 18 August 2021

Keywords:
Data-driven randomized learning
Feedforward neural networks
Neural networks with random hidden nodes
Randomized learning algorithms

ABSTRACT

There is an issue with the way in which feedforward neural networks with random hidden nodes generate random parameters in order to obtain a good projection space. Typically, random weights and biases are both drawn from the same interval, which is misguided as they have different functions. Recently, more sophisticated methods of random parameters generation have been developed, such as the data-driven approach, where the sigmoids are placed in randomly selected regions of the input space and then their slopes are adjusted to the local fluctuations of the target function. In this work, we propose a new constructive data-driven method that builds iteratively the network architecture. This method successively generates new candidate hidden nodes and accepts them when the training error falls significantly. The threshold of acceptance is adapted throughout training, accepting at the beginning of the training process only those nodes which lead to the largest reductions in error. In the next stages, the threshold is successively reduced to accept only those nodes which model the target function details more accurately. This leads to a more compact network architecture, as it includes only "significant" nodes. It is worth noting that redundant, random nodes, which are usually generated by existing randomized learning methods, are not accepted by the proposed method. We empirically compared our approach with several alternative methods, including its predecessor, competitive randomized learning solutions, a gradient-based network and a generalized additive model. We found that our proposed approach outperformed its competitors in terms of fitting accuracy.

1. Introduction

A feedforward neural network (FNN) with a single hidden layer containing a finite number of neurons is able to approximate any continuous function under mild assumptions on the activation function (AF). Unfortunately, any learning process which uses a gradient-based algorithm is sensitive to the local minima of the error function as well as its flat regions and saddle points. Moreover, the gradient calculations are time consuming, especially for large network architectures, complex target functions (TFs) and big training datasets. In randomized learning, the optimization problem, which is highly non-convex when gradient descent algorithms are used, becomes convex [1]. This is because the parameters of the hidden nodes are not learned but are selected randomly and stay fixed. The only adaptation occurs in the output weights. As the optimization problem is brought to a linear form, the output weights can be learned using a standard least-squares method. The learning process in such a case is easier to implement and much faster than when using gradient descent algorithms for the full learning of all parameters, i.e. weights and biases of the hidden and output nodes. Many simulation studies reported in the literature show the high accuracy of randomized neural models when compared to fully adaptable ones.

Randomized learning originates from the beginning of 90s [2,3]. One of the most popular randomized learning algorithms is the random vector functional link network (RVFL) [4]. It is a single hidden layer FNN with direct links between input and output layers. Recently, [5] provided a corrected rigorous proof that RVFL is a universal approximator for continuous functions on compact domains, with the approximation error decaying asymptotically. Although RVFL in its standard version overcomes some of the limitations of gradient-based learning such as slow convergence, over-fitting and trapping in local minima, and achieves good generalization performance in real-world problems [6], in order to improve further its effectiveness, modified versions of it have recently been proposed. In [7], an orthogonal polynomial expanded RVFL (OPE-RVFL) was proposed. In this model, to enhance a network's representation capabilities in modeling nonlinear relationships and capture higher order information, a nonlinear expansion of inputs into orthogonal polynomials was used. OPE-RVFL adds direct links between the nonlinear transformation of the input patterns and the output layer. This modification helps to model complex mappings. A simulation study showed the great effectiveness and efficiency of novel OPE-RVFL which

E-mail address: grzegorz.dudek@pcz.pl.

https://doi.org/10.1016/j.asoc.2021.107797
1568-4946/

can achieve better generalization performance than competitive randomization-based methods such as standard RVFL and extreme learning machine (ELM), i.e. RVFL without direct links and output node biases. RVFL enriched with a new learning paradigm called learning using privileged information was proposed recently in [8]. This is a teacher–student interaction mechanism, which can leverage additional sources of information into the RVFL and offers an alternative way to train the RVFL. Moreover, the proposed RVFL+ can perform in conjunction with a kernel trick for highly complex nonlinear learning. The experimental results illustrate that these novel proposed solutions outperform state-of-the-art competitors.

A random projection may generate redundant information, leading to suboptimal solutions. To overcome this problem, in [9] a successive projection algorithm for randomized learning was developed, combining feature selection, hidden node pruning and ensemble methods. The resulting parsimonious ensembles of randomized NNs aim to reduce the complexity of the final model without compromising accuracy. Other positive effects of randomized NN ensembling were noted in [10]. In this study, to ensure the appropriate diversity of the RVFL base learners, additional regularization or randomization in their architecture was introduced. This was realized using deep learning techniques such as Dropout and DropConnect. In the experimental study, it was observed that stronger randomization helps RVFL ensembles generalize better. Interesting results in terms of RVFL ensemble development and applications were presented in [11]. In this work, an ensemble of RVFL was used for time series forecasting. The time series was decomposed, via an empirical mode decomposition method into several intrinsic mode function components. Each component was predicted using RVFL, and then individual component forecasts were aggregated to produce the final results. Based on a simulation study, the authors concluded that their proposed approach is significantly superior to all other benchmarking methods in terms of accuracy, speed and robustness. Promising future directions for randomized NN learning include randomization-based multi-layer or deep NNs [12–14] as well as unsupervised and semi-supervised learning [15–17].

1.1. Generating hidden node parameters - a challenge in randomized NN learning

The main problem in randomized learning is finding a way of generating the random hidden node weights and biases so as to obtain a good projection space. According to the standard approach, the random weights and biases are selected randomly with a fixed interval from any continuous sampling distribution. It was proven that when the interval is symmetric, the FNN has a universal approximation property if the function to be approximated meets Lipschitz condition [18]. The problem of how to select an appropriate interval for parameters has not been solved as yet and is considered to be one of the major research challenges in the area of FNN randomized learning [19,20]. In many practical applications, the interval for random parameters is selected as $[-1, 1]$ without any justification, regardless of the problem type (classification or regression), data, and AF type. Some works have shown that such an interval is misleading because the FNN is unable to model nonlinear maps, no matter how many training samples are provided or how many hidden nodes are used [21]. So, in order to improve the performance in a specified task, the optimization of this interval is recommended in many papers, e.g. [4,18,22]. The authors of the last work proposed a way of assigning random parameters with inequality constraints from the adaptively selecting scope, thus maintaining the universal approximation property of the built randomized learner models.

Noting that weights and biases have different functions (weights express slopes of the sigmoids and biases represent sigmoid positions or shifts), in [23] a method of selecting the random parameters was proposed which generates them separately, not both from the same interval. This method takes into account the data scope and type of the AFs. First, it adjusts the AF slopes to the TF and then shifts the AFs into the input space. Another way of generating random parameters was recently proposed in [24]. In this approach, the slope angles of the sigmoids are randomly generated from the interval adjusted to TF. Then the sigmoids are randomly rotated and shifted into the input hypercube. Both these methods performed very well on a TF with strong fluctuations, outperforming the standard approach with a fixed interval.

In [25] unsupervised pre-training was proposed to generate hidden node parameters. This method is derived from deep learning where it helps optimization by initializing weights in a region near a good local minimum, giving rise to internal distributed representations that are high-level abstractions of the input, and achieving a better generalization [26,27]. In [28] an unsupervised pre-training with autoencoders was applied for a shallow RVFL. In this case, a sparse autoencoder with ℓ_1-norm regularization is applied to generate more sparse and meaningful network parameters. In [13] pre-training was used to generate the hidden node parameters of deep RVFL. To extract better higher-level representations, randomization based stacked autoencoders with a denoising criterion were applied. The network is built hierarchically with high level feature extraction followed by a final classification layer, which is RVFL with direct links.

Another idea for generating random parameters, a data-driven method of randomized FNN learning (D-DM), was recently proposed in [29]. According to this method, the hidden node weights are not selected from a specified interval but, instead, the sigmoids building the fitted function (FF) are adjusted individually to the local complexity of the TF. To do so, first, the proposed method selects the input space regions (by randomly choosing the training points), and then places the sigmoids in these regions and adjusts the sigmoid slopes to the TF slopes in the neighborhoods of the chosen points. A linear combination of the sigmoids reflects the TF features in different regions of the input space. Simulation studies have shown that this approach brings very good results in the approximation of complex TFs when compared to both standard fixed interval methods and new state-of-the-art methods proposed recently in the literature.

1.2. Summary of contribution

To develop randomized learning methods for FNNs, the main contribution of this work is a new constructive approach for data-driven randomized learning of FNNs. This extended version of D-DM constructs the network architecture iteratively by adding new hidden nodes. The candidate nodes are successively generated and are accepted or rejected depending on the approximation error. If the error is reduced significantly by more than the assumed threshold, the node is accepted. The threshold of acceptance is adapted to the current training stage, accepting in the initial iterations only those nodes which approximate the TF roughly. In the next iterations, the threshold is successively reduced by half to accept only those nodes which lead to a more accurate modeling of the TF details. The final architecture of the resulting network is compact as it includes only "significant" nodes, i.e. those which are necessary to construct a well-fitted function to the TF. Compared to other recently proposed constructive algorithms for randomized FNN learning [22,30] our proposed method does not search for the optimal interval for the random parameters. Instead, the weights and biases of nodes are

determined by the TF slopes in randomly selected input regions. So, the resulting FNN is more dependent on data and includes only those nodes which are necessary to model the TF details with the required accuracy.

The contribution of this paper can be summarized as follows.

1. A new constructive data-driven learning for FNNs is proposed. This approach gradually extends the network architecture by adding new nodes to the hidden layer. New nodes introduced in the successive iterations allow the network to model the TF with increasing accuracy. The node positions in the input space and their slopes are related to TF complexity. This new construction process results in a parsimonious architecture with only relevant nodes.
2. This work empirically demonstrates that the proposed constructive approach outperforms its predecessor, D-DM, and is a strong competitor to alternative randomization-based and gradient-based solutions such as OPE-RVFL, standard RVFL, ELM and single hidden layer perceptron as well as statistical methods such as generalized additive model.

The remainder of this paper is organized as follows. Section 2 details our proposed constructive approach to data-driven randomized learning. Section 3 describes the experimental framework used to evaluate the proposed model's performance. Finally, Section 4 concludes this paper.

2. Constructive approach for data-driven randomized learning

2.1. Data-driven randomized learning

A FNN with a single hidden layer, including sigmoids as AFs, is considered. D-DM adjusts the sigmoids individually to the local complexity of the TF [29]. It places the sigmoids into randomly selected input regions and adjusts their slopes to the TF slopes in these regions. Firstly, the algorithm randomly selects the input space regions by drawing points $\mathbf{x}^*$ from the training set. Then, at each point $\mathbf{x}^*$ the sigmoid S is located in such a way that it has one of its inflection points, P, in $\mathbf{x}^*$. The sigmoid value in the inflection point is 0.5, thus:

$$h(\mathbf{x}^*) = \frac{1}{1+\exp\left(-\left(\mathbf{a}^T\mathbf{x}^* + b\right)\right)} = 0.5 \tag{1}$$

where $\mathbf{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]^T \subset \mathbb{R}^n$, $\mathbf{a} = [a_1, a_2, \ldots, a_n]^T \subset \mathbb{R}^n$ is a vector of sigmoid weights and b is a sigmoid bias.

The slope of sigmoid S is adjusted to the TF slope in $\mathbf{x}^*$. To do so, the slope of the TF is approximated by hyperplane T, which is fitted to the neighborhood of $\mathbf{x}^*$. This neighborhood, $\Psi(\mathbf{x}^*)$, includes $\mathbf{x}^*$ and its k nearest neighbors among the training points. We use Euclidean distance to select the nearest neighbors. Hyperplane T is of the form:

$$y = a_1'x_1 + a_2'x_2 + \cdots + a_n'x_n + b' \tag{2}$$

where coefficient a_j' expresses a slope of the hyperplane in jth direction.

If we assume that this hyperplane is tangent to sigmoid S at P, then the partial derivatives of sigmoid S and hyperplane T must be the same in P, and we obtain the sigmoid weights (see [29] for details):

$$a_j = 4a_j', \quad j = 1, 2, \ldots, n \tag{3}$$

Having weights a_j we can calculate the bias of sigmoid S based on the equation we obtain from (1):

$$b = -\mathbf{a}^T\mathbf{x}^* \tag{4}$$

Note the completely different way of generating the hidden node parameters in D-DM when compared to the standard way. Weights a_j are not randomly generated from a fixed interval but correspond to the TF local slope. The biases are also not randomly selected from the fixed interval, but are calculated as linear combinations of points $\mathbf{x}^*$ and sigmoid weights $\mathbf{a}$. They express the shifting of the sigmoids to the randomly selected input regions.

After randomly selecting m points $\mathbf{x}^*$ we generate a set of sigmoids reflecting the local features of the TF in different regions. These sigmoids are the basis functions which are linearly combined to obtain the FF. The least squares estimate of the weights in this combination are calculated as follows:

$$\boldsymbol{\beta} = \mathbf{H}^+\mathbf{Y} \tag{5}$$

where $\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_m]^T$ is a vector of output weights, $\mathbf{Y} = [y_1, y_2, \ldots, y_N]^T$ is a vector of target outputs and $\mathbf{H}^+$ is the Moore–Penrose pseudoinverse of matrix $\mathbf{H}$, which is an N-by-m hidden layer output matrix for all N training samples and m nodes.

As reported in [29], D-DM produces very good results in the approximation of complex TFs when compared to the standard method and new methods proposed recently in the literature. This is because it places the steepest sigmoid fragments inside the input hypercube and adjusts the sigmoid slopes to the TF. So the saturated fragments of sigmoids are avoided in modeling the TF.

2.2. Constructive extension of D-DM

The constructive extension of D-DM (CD-DM) proposed in this work generates sigmoids in randomly selected regions and adjust their slopes according to the algorithm described above. After generating a new node, an approximation error is calculated, and the node is accepted or rejected. It is accepted when the error is reduced by more than threshold θ. Otherwise it is rejected. Threshold θ adapts as the construction process advances. Its initial value should be low enough to accept only the "key" nodes, which model roughly the shape of the TF. These nodes are searched randomly by generating candidate nodes, one by one, in different input space regions. The generation process is paused when, during consecutive Q iterations, no new node is accepted among the candidate nodes. In such a case, threshold of acceptance θ is halved, and the process of generating new candidate nodes continues. At the new value of the acceptance threshold, those nodes which model the TF in more detail are accepted. If after Q iterations, no node is accepted, threshold θ is halved again. This process of generating candidate nodes and reducing the threshold by half is continued until one of the stop conditions is met. Three stop conditions are used:

1. assumed number of nodes m is reached or
2. maximum allowable error $RMSE_{\max}$ is reached or
3. maximum number of consecutive unaccepted candidate nodes $q_{\max}$ is reached.

The third condition prevents an infinite loop if the first two stop conditions cannot be met. The first two conditions can be treated alternatively, i.e. one of them can be ignored ($RMSE_{\max} = 0$ or $m = \infty$).

At each successive level of threshold θ new nodes can be added to the network, which ensures a more and more accurate fitting of the FF to the TF. To avoid overfitting, the process should be stopped at the right moment. This moment is controlled by the total number of hidden nodes m or/and maximum allowable error $RMSE_{\max}$, which should be selected in cross-validation.

In CD-DM the number of samples and their size do not affect the number of nodes. Only the TF complexity affects the network size. TFs without fluctuations can be modeled using a small

number of nodes. In such a case, new candidate nodes considered in the network construction process do not reduce the error by more than threshold θ and are rejected. In the case of the complex TF with fluctuations, new nodes are added in the fluctuation regions until they cause error reduction. If all TF fluctuations are modeled by a set of nodes, new candidate nodes are rejected. The level of fitting accuracy and bias–variance tradeoff are dependent on the size of neighborhood k' and the stop conditions, i.e. the final number of nodes m or/and the maximum allowable error $RMSE_{\max}$. These regularization hyperparameters are adjusted in cross-validation to prevent underfitting or overfitting.

The proposed method has six hyperparameters, defined as follows:

- m – the final number of hidden nodes. The algorithm adds successive nodes to the hidden layer if they reduce the error by more than threshold θ until the final number of nodes m is reached. Intuitively, more complex TF needs more hidden nodes to model accurately the function fluctuations. However, too many nodes can lead to overfitting.
- k' – size of the neighborhood including $k' = k + 1$ training points, i.e. $\mathbf{x}^*$ and its k nearest neighbors. The neighborhood $\Psi(\mathbf{x}^*)$ expresses the local features of the TF around selected point $\mathbf{x}^*$. The sigmoid slope is determined on this set $\Psi(\mathbf{x}^*)$ to model the selected region of the TF. The optimal value of k' depends on the noise level observed in the data, the TF complexity and the data density. A low value of k' at a high level of noise leads to overfitting. On the other hand, too large k' causes underfitting. The size of the neighborhood controls the bias–variance tradeoff of the model, as well as the number of nodes m and the maximum allowable error $RMSE_{\max}$.
- $\theta \leq 0$ – the threshold for the error change. This is the threshold of acceptance for any new candidate hidden node which is added during the construction process. When, after adding a node, the reduction in error is greater than this threshold the node is accepted. Otherwise, it is rejected. If no node has been added for the subsequent Q iterations, threshold θ is halved. At successive levels of θ those new nodes which model the TF in more and more detail are accepted, up until the final number of nodes m or error $RMSE_{\max}$ is reached. The initial value of θ should be low enough. When it is too low, it does not accept any nodes and increases rapidly in the first iterations of the construction process (so due to self-adaptation a too low initial value of θ is not a problem). When it is too high, it accepts most nodes without selection. This means the proposed method behaves just like original D-DM.
- Q – the threshold for the number of successive iterations without a node accepted. The method searches for the nodes to place them in the input space and so reduce the error by more than threshold θ. If such a node is not found by Q iterations, threshold θ is halved and the searching process is repeated, i.e. new candidate nodes are generated. Reaching the total number of m nodes in the hidden layer or reaching the fitting error $RMSE_{\max}$ completes the construction process.
- $RMSE_{\max}$ – the maximum allowable error deciding about the final fitting error. The constructive process adds new hidden nodes to the network until the error decreases to $RMSE_{\max}$. This hyperparameter can be considered as an alternative to m.
- $q_{\max}$ – the maximum number of the consecutive unaccepted candidate nodes which stops the construction process. If the algorithm shows stagnation by $q_{\max}$ iterations, i.e. no new node is added to the network, it stops.

The proposed CD-DM can be formulated in Algorithm 1. This is discussed in detail with an example in the next subsection. A Matlab implementation of CD-DM is presented in Appendix.

Algorithm 1 Constructive Approach to Data-Driven Randomized Learning for FNNs

Input:
Number of hidden nodes m
Number of nearest neighbors $k \geq n$
Initial value of the threshold for the error change $\theta \leq 0$
Q-threshold for adaptation of θ
$q_{\max}$-threshold for the stop condition
Maximum allowable fitting error $RMSE_{\max}$ for the stop condition
Training set $\Phi = \{(\mathbf{x}_l, y_l) | \mathbf{x}_l \in \mathbb{R}^n, y_l \in \mathbb{R}, l = 1, 2, \ldots, N\}$
Output:
Weights $\mathbf{A} = \begin{bmatrix} a_{1,1} & \ldots & a_{m,1} \\ \vdots & \ddots & \vdots \\ a_{1,n} & \ldots & a_{m,n} \end{bmatrix}$
Biases $\mathbf{b} = [b_1, \ldots, b_m]$
Procedure:
$i = 1$, $l = 1$, $q = 1$, $RMSE_0 = RMSE_{\max} + 1$, $\mathbf{H} = [\ ]$
while $(i \leq m)$ and $(RMSE_{i-1} > RMSE_{\max})$ and $(q < q_{\max})$ **do**
  if l mod $N == 1$ **then**
    $\Phi^* = \Phi$
  end if
  (a) Choose randomly $\mathbf{x}^*$ from Φ^*, update $\Phi^* = \Phi^* \setminus \{\mathbf{x}^*\}$
  (b) Create set $\Psi(\mathbf{x}^*)$ containing $\mathbf{x}^*$ and its k nearest neighbors in Φ
  (c) Fit the hyperplane to $\Psi(\mathbf{x}^*)$: $y = a'_1 x_1 + a'_2 x_2 + \ldots + a'_n x_n + b'$
  (d) Compute the weights for the i-th node: $a_{i,j} = 4a'_j, \quad j = 1, 2, \ldots, n$
  (e) Compute the bias for the i-th node: $b_i = -\sum_{j=1}^{n} a_{i,j} x^*_j$
  (f) Add the column corresponding to the i-th node to the hidden layer output matrix:

$$\mathbf{H}' = \begin{bmatrix} & 1/\left(1 + \exp\left(-\left(\mathbf{a}_i^T \mathbf{x}_1 + b_i\right)\right)\right) \\ \mathbf{H} & \vdots \\ & 1/\left(1 + \exp\left(-\left(\mathbf{a}_i^T \mathbf{x}_N + b_i\right)\right)\right) \end{bmatrix}$$

  (g) Compute the output weights for i nodes: $\boldsymbol{\beta} = \mathbf{H}'^{+}\mathbf{Y}$
  (h) Compute the network output for the training set Φ: $\mathbf{Y}' = \mathbf{H}'\boldsymbol{\beta}$
  (i) Compute the network error: $RMSE_i = \sqrt{\frac{1}{N}\sum_{l=1}^{N}(y'_l - y_l)^2}$
  (j) Compute the error change: $\Delta RMSE = RMSE_i - RMSE_{i-1}$
  (k) Accept or reject the i-th node:
    $q = q + 1$
    if $(i == 1)$ or $(\Delta RMSE \leq \theta)$ **then**
      $i = i + 1$, $q = 1$, $\mathbf{H} = \mathbf{H}'$
    end if
  (l) Adapt the threshold for the error change:
    if $q \geq Q$ **then**
      $\theta = \theta/2$
    end if
    $l = l + 1$
end while

2.3. Illustrative example

To illustrate the proposed CD-DM, we consider the case of a single-variable function approximation. TF is in the form:

$$g(x) = 0.2e^{-(10x-4)^2} + 0.5e^{-(80x-40)^2} + 0.3e^{-(80x-20)^2} \tag{6}$$

The training dataset has 1000 points (x_l, y_l), where x_l are uniformly randomly distributed on [0, 1] and y_l are calculated from (6) and then scaled to [0, 1]. A test set has 300 points distributed regularly on [0, 1].

The algorithm starts with one hidden node that is placed in a randomly selected training point $\mathbf{x}^*$, Algorithm 1 (a). Its neighborhood $\Psi(\mathbf{x}^*)$ is selected (b) and hyperplane T is fitted to $\Psi(\mathbf{x}^*)$ (c). Then its weights and bias are determined, (d) and (e), respectively. In the next steps, one-node hidden layer output matrix $\mathbf{H}$ is computed (f), output weight β_1 is computed (g) and

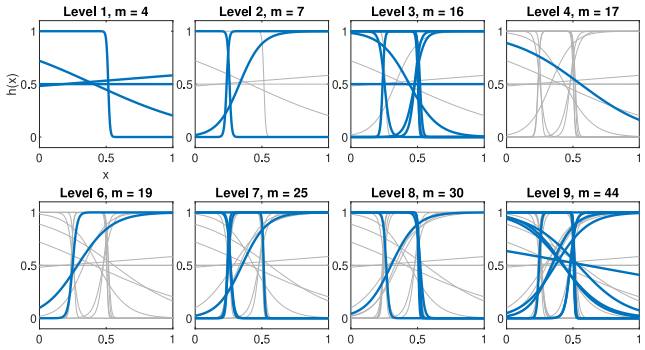

Fig. 1. Sigmoids introduced at successive θ levels (new sigmoids in thick lines).

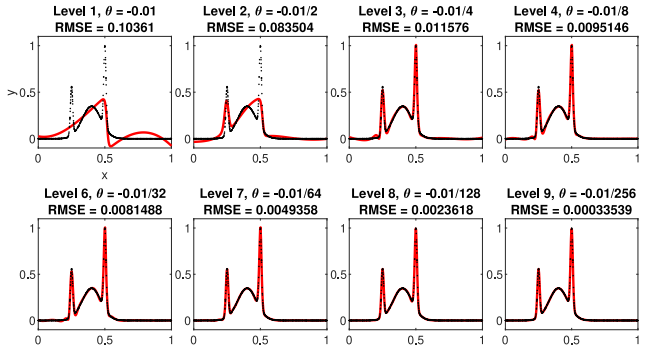

Fig. 2. Fitted curves at successive θ levels.

the predicted output is computed (h). The first hidden node is accepted regardless of the fitting error (k). Then we start the second iteration, going through the same steps. In step (j), the error difference between the two-hidden node and one-hidden node network is calculated. The new node is accepted if this difference is more than the acceptance threshold, which is $\theta = -0.01$ in our example (k). Generating new nodes is repeated until Q consecutively generated nodes do not improve results, i.e. $\Delta RMSE > \theta$ for Q consecutive candidate nodes. In such a case the acceptance threshold is halved (l) and the algorithm continues using the second threshold level. In our case, for $Q = 50$, four hidden nodes are introduced to the network at the first θ level. They are shown in the left upper panel of Fig. 1. The fitted curve composed of these nodes is shown in the left upper panel of Fig. 2. As can be seen in this figure, this curve roughly maps the shape of the TF. At the second θ level, three new hidden nodes are accepted which results in a slightly better fitting. A significant improvement is seen at level three where 9 new nodes are introduced. The fitting error decreased from 0.0835 to 0.0116. At level 4, only one neuron was added, and at level 5, none. The TF is modeled more and more accurately at subsequent levels. At the last level, 14 new hidden nodes are introduced, which model the TF's minor details. The fitting error was further reduced to $RMSE = 0.00034$ at this level.

The fitting error and threshold adaptation during the network construction process are shown in Fig. 3. The red dots in the right panel of this figure represent error reduction greater than the threshold, which is obtained after a new hidden node is accepted.

A comparison of the modeling performance between D-DM and CD-DM is depicted in Fig. 4. In this figure sigmoid $h(x)$ distributions in the input space are shown for both D-DM and CD-DM as well as the resulting fitted curves. Note the many flat nodes in the D-DM case. Introducing such flat nodes rarely decreases the error so CD-DM does not accept them during the construction process. Consequently, only 44 nodes, usually those steep ones which correspond to the steep fragments of the function, are selected by CD-DM to approximate the TF.

Fig. 5 depicts the error on the test set as a function of the number of hidden nodes for both versions of the data-driven method. The median curves over 100 runs are shown with the intervals between 10th and 90th percentiles. Note the much faster convergence of CD-DM with the number of hidden nodes. To get the same error level as D-DM with 250 nodes, CD-DM needs only 44 nodes. Note also the less dispersed results for CD-DM.

2.4. Complexity of CD-DM

In the proposed CD-DM, a hyperplane is fitted to $k+1$ points for each hidden node. The complexity of the least squares regression used for this, when $k \geq n$, is $O(kn^2)$. Selection of k nearest neighbors of $\mathbf{x}^*$ takes $O(Nn + Nk)$ time (the first component expresses distance measure complexity and the second one expresses selection of k nearest neighbors), but for $k \geq n$ it reduces to $O(Nk)$. In the CD-DM construction process, we generate $m' \geq m$ hidden nodes (m of them are accepted but others are rejected; the exact value of m' is not known). So, the total time complexity of generating weights for m' hidden nodes in CD-DM is $O(m'kn^2 + m'Nk)$.

The output weights are computed from (5). This operation requires the least-square estimation which, for m hidden nodes, is $O(Nm)$ and Moore–Penrose pseudoinverse based on a singular value decomposition which is $O(N^2m+Nm^2)$. Thus computing the output weights takes $O(N^2m + Nm^2)$ time. Because the output weights are calculated m' times in the main loop of the construction process, the complexity of this is $O(m'N^2m + m'Nm^2)$. Note that the number of nodes m is not fixed in this loop but changes form 1 to m in the successive iterations. Unfortunately, this change is unpredictable which means the above complexity estimate is overestimated.

The whole CD-DM construction process including determination of the hidden and output weights takes $O(m'kn^2 + m'Nk + m'N^2m + m'Nm^2)$. For comparison, D-DM complexity is $O(mkn^2 + mNk + N^2m + Nm^2)$. In a typical case $N > n^2$, $N > m$ and $m > k$, so the complexity of CD-DM can be expressed as $O(m'N^2m)$, and the complexity of D-DM as $O(N^2m)$.

Taking into account the optimization process, i.e. selection of hyperparameters, the time complexity is as follows. In the basic variant, CD-DM has two optimized hyperparameters, k and m (or alternatively $RMSE_{\max}$). They need to be found using cross-validation. Therefore, the computational load increases linearly with the number of data splits used in cross-validation, v, and also with the number of points of the grid which is $l_k l_m$, where l_k and l_m are the number of searching values for k and m, respectively. Thus, hyperparameter selection using grid search and v-fold cross validation, with the assumptions and simplifications mentioned above, takes $O(v l_k l_m m'N^2m)$. Note that the greatest computational cost is related to the singular value decomposition.

3. Simulation study

In this section, to assess the performance of CD-DM, a two-part experiment was conducted. In the first part, we compare CD-DM with its predecessor, D-DM. In the second part, we compare CD-DM with other randomization-based algorithms such as RVFL, OPE-RVFL and ELM as well as with the gradient-based learning algorithm and classical statistical fitting method — generalized additive model (GAM).

To evaluate the models' performance we use root mean square error (RMSE). For CD-DM, in all cases, the following hyperparameters were assumed to be fixed: $\theta = -0.01$, $Q = 50$, $RMSE_{\max} = 0$,

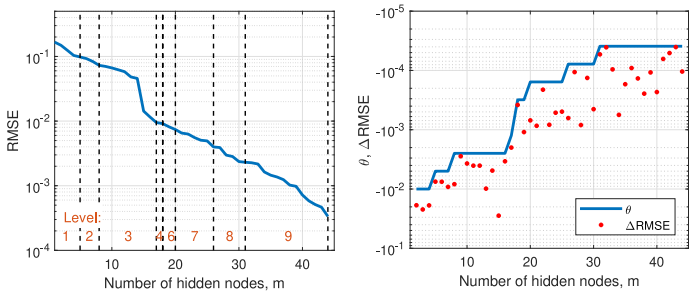

Fig. 3. Fitting error (left) and the threshold adaptation (right) during the network construction process.

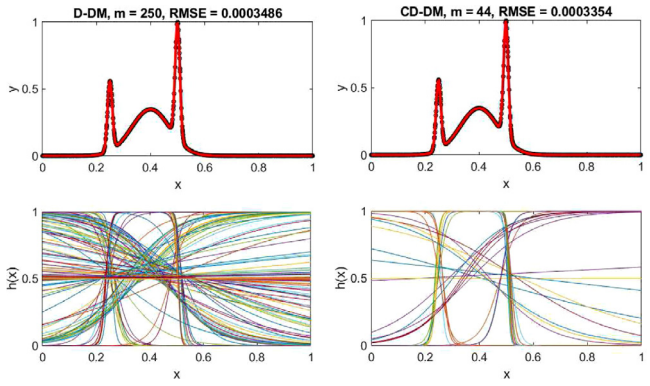

Fig. 4. Fitted curves and the sigmoids constructing them for D-DM (left) and CD-DM (right).

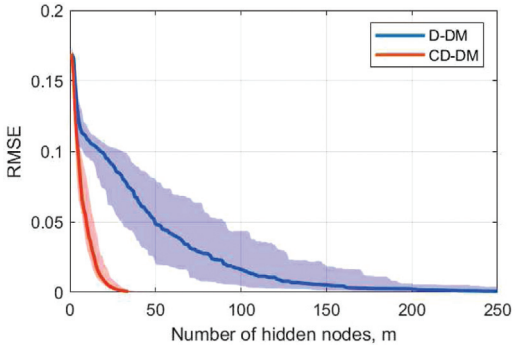

Fig. 5. Fitting error as a function of the number of hidden nodes for D-DM and CD-DM.

and $q_{\max} = 1000$. They were set on the basis of preliminary simulations. The initial value of θ is not that important provided it is low enough. When it is too low, it does not accept any nodes and increases rapidly in the first iterations of the construction process. The simulations were carried out on Windows-10 platform, in MATLAB 2018a environment, running an Intel i7-6950X 3.00 GHz processor, with 48 GB RAM memory.

3.1. CD-DM vs. D-DM

We compare CD-DM with D-DM on the following regression problems, which include a two variable function approximation task and three real-world modeling tasks. All the results reported in this subsection are averages over 100 independent trials.

- Approximation of two variable TF:

$$g(\mathbf{x}) = \sin\left(20 \cdot \exp\left(x_1\right)\right) \cdot x_1^2 + \sin\left(20 \cdot \exp\left(x_2\right)\right) \cdot x_2^2 \qquad (7)$$

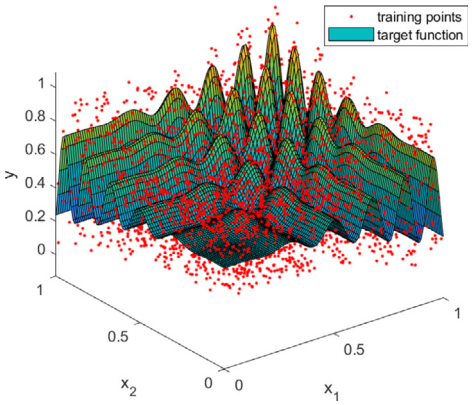

Fig. 6. TF (7).

The training set contains 5000 points $(\mathbf{x}_l, y_l)$, where components of $\mathbf{x}_l$ are independently, uniformly, randomly distributed on $[0, 1]$ and y_l are generated from (7), then normalized to the range $[0, 1]$ and distorted by adding the uniform noise distributed in $[-0.2, 0.2]$. A test set of the same size is created in a similar way. TF (7) is depicted in Fig. 6.

- Stock – daily stock prices from January 1988 through October 1991, for ten aerospace companies. The task is to approximate the price of the 10th company given the prices of the others. There are 950 samples composed of nine input variables and one output variable.
- Concrete – the dataset contains the concrete's compressive strength, age, and seven ingredients. The task is to approximate the highly nonlinear relationship between the concrete's compressive strength and the ingredients and age. There are 1020 samples composed of eight input variables and one output variable.
- Compactiv – the Computer Activity dataset is a collection of computer systems activity measures. The data was collected from a Sun Sparcstation 20/712 with 128 Mbytes of memory running in a multi-user university department. The task is to predict the portion of time that CPUs run in user mode. There are 8192 samples composed of 21 input variables (activity measures) and one output variable.

The datasets Stock, Concrete and Compactiv were divided into training sets containing 75% of samples selected randomly, and test sets containing the remaining 25% of samples. The input and output variables are normalized into $[0, 1]$. These three datasets were downloaded from KEEL (Knowledge Extraction based on Evolutionary Learning) dataset repository (http://www.keel.es). Neighborhood size k' for D-DM was determined in 10-fold cross-validation. The same value of k' was also assumed for CD-DM.

Table 1
Performance comparison of D-DM and CD-DM.

Data	D-DM		CD-DM	
	RMSE	Hyperparameters	RMSE	Hyperparameters
Function (7)	0.1204 ± 0.0003	$m = 300, k' = 35$	0.1191 ± 0.0005	$m = 160, k' = 35, \theta = -0.01, Q = 50$
Stock	0.0285 ± 0.0028	$m = 250, k' = 30$	0.0265 ± 0.0014	$m = 110, k' = 30, \theta = -0.01, Q = 50$
Concrete	0.0770 ± 0.0055	$m = 150, k' = 8$	0.0748 ± 0.0034	$m = 50, k' = 8, \theta = -0.01, Q = 50$
Compactiv	0.0247 ± 0.0006	$m = 500, k' = 25$	0.0240 ± 0.0003	$m = 120, k' = 25, \theta = -0.01, Q = 50$

Table 2
Benchmark regression datasets.

Dataset name	#samples	#features	Description
Auto MPG	386	7	Predict fuel consumption (MPG)
Treasury	1049	15	Predict one month CD rate
Kinematics 32NH	8192	32	Forward kinematics of an 8 link robot arm (nonlinear and highly noisy)
Computer activity	8192	21	Predict usr, the portion of time that CPUs run in user mode from all attributes
Triazines	186	60	Predict the activity from the descriptive structural attributes
Pyramidines	74	27	Inhibition of dihydrofolate reductase by pyrimidines
Machine CPU	209	6	Predict relative CPU performance
Puma32NH	8192	32	Predict the angular acceleration of one of the robot arm's links (nonlinear and highly noisy)
Bank32nh	8192	32	Predict the fraction of bank customers who leave the bank because of full queues (nonlinear and highly noisy)
Elevators	16599	18	Predict the control action on the elevators of the F-16 aircraft (target is absolute value)

The number of nodes was determined for these two methods individually in cross-validation.

Fig. 8 shows RMSE as a function of the number of nodes for D-DM and CD-DM. For each dataset, the proposed CD-DM converged much faster and achieved lower test RMSE than D-DM. Table 1 shows the errors and hyperparameters of the models. For RMSE, the medians and interquartile ranges (IQR) over 100 runs are shown. Note that the number of hidden nodes for CD-DM is from two to five times smaller than for D-DM.

The changes in threshold θ during the construction processes are shown in Fig. 9. Note the rapid increase in the threshold in the first iterations and long segments without change at its higher levels, where many nodes which model the TF in more and more detail were added to the hidden layer.

Fig. 7 shows the function fitted to TF (7) by CD-DM and detailed absolute errors for this case. Note the highest errors near the input square boundaries. In this regions, the neighborhood $\Psi(\mathbf{x}^*)$ may have problems accurately reflecting the shape of the TF.

Taking into account the performance comparison reported in [29], where the original D-DM are compared with the standard methods (with fixed and optimized intervals) as well as with two more sophisticated methods of random parameter generation recently proposed in the literature, [23,24] we can conclude that CD-DM outperforms standard and state-of-the-art methods in terms of accuracy, convergence speed with the number of hidden nodes and more compact network architecture.

3.2. CD-DM vs. other methods

In the second part of the experimental study, we compare CD-DM with a standard RVFL, ELM, state-of-the-art RVFL variant, i.e. OPE-RVFL, gradient-based FNN, and a popular statistical method, i.e. GAM. OPE-RVFL is a orthogonal polynomial expanded RVFL with ridge regression regularization to prevent overfitting. OPE-RVFL was tested over 120 variants including four orthogonal polynomials (Chebyshev, Hermite, Laguerre and Legendre) and three activation functions (tansig, logsig, tribas). The number of hidden nodes in OPE-RVFL and the ridge regression parameter were selected using cross-validation procedure (see [7] for details). Gradient-based FNN is a single hidden layer perceptron, denoted as MLP. MLP was trained using Levenberg–Marquardt backpropagation with early stopping to avoid overtraining (20% of training data was used as validation data). The number of hidden nodes was selected by 5-fold cross validation. For GAM, we used Matlab implementation, function `fitrgam`, which fits a model using a gradient boosting algorithm. GAM hyperparameters were selected by cross-validation using Bayesian optimization. CD-DM hyperparameters, i.e. number of hidden nodes m and number of nearest neighbors k, were selected using grid search in 5-fold cross validation.

Table 3
RMSE comparison of CD-DM with other methods. Best results are underlined.

Dataset name	CD-DM	RVFL [7]	ELM [7]	OPE-RVFL [7]	MLP	GAM
Auto MPG	<u>0.0779</u>	0.0790	0.0817	0.0788	0.0924	0.0809
Treasury	0.0117	0.0129	0.0323	0.0126	<u>0.0116</u>	0.0149
Kinematics 32NH	<u>0.1288</u>	0.1325	0.1502	0.1319	0.1388	0.1314
Computer activity	<u>0.0262</u>	0.0386	0.0580	0.0552	0.0263	0.0265
Triazines	0.2065	0.9574	0.9025	0.8592	0.2848	<u>0.1868</u>
Pyramidines	0.2149	0.2560	0.4828	0.1654	0.2926	<u>0.1086</u>
Machine CPU	<u>0.0553</u>	0.1671	0.1102	0.1394	0.1166	0.0730
Puma32NH	0.1216	0.1565	0.1565	0.1565	<u>0.0386</u>	0.0553
Bank32nh	<u>0.0993</u>	0.1279	0.1328	0.1322	0.1122	0.1035
Elevators	0.0347	0.0920	0.0939	0.0917	<u>0.0312</u>	0.0353

Table 2 shows the 10 regression benchmark datasets used in the experiments. These datasets were taken from [31–33].

Results for RVFL, ELM and OPE-RVFL are taken from [7]. In the interests of a fair comparison, we use exactly the same data and we followed the same experimental test setup for CD-DM, MLP and GAM as in [7]. Namely, each dataset was divided randomly into the training set (70% of data) and test set (30% of data). The training was carried out 50 times for each model with a different random division of the datasets. Input and output variables were normalized into [0, 1].

Table 3 shows RMSE averaged over 50 independent trials. As can be seen from this table, CD-DM outperforms standard RVFL and ELM in all cases, OPE-RVFL in 9 out of 10 cases, and MLP and GAM in 7 out of 10 cases.

Fig. 10 shows distributions of fitting errors $E = Real - Predicted$ for CD-DM. As can be seen from this figure, they are very similar to a normal distribution for Auto MPG, Kinematics 32N,

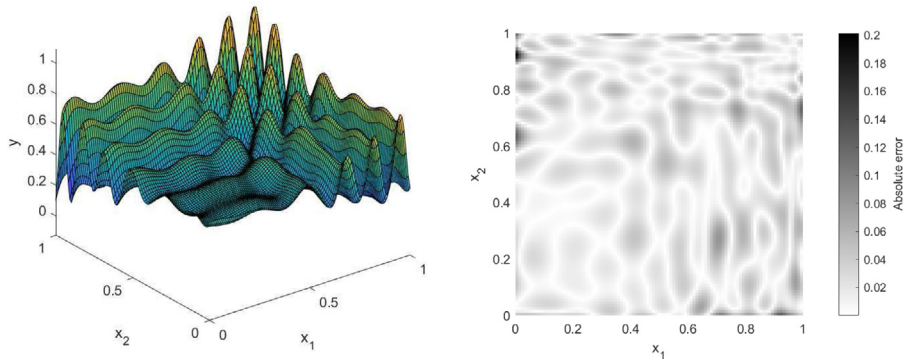

Fig. 7. Function fitted by CD-DM to TF (7) (left) and detailed absolute errors for this case (right).

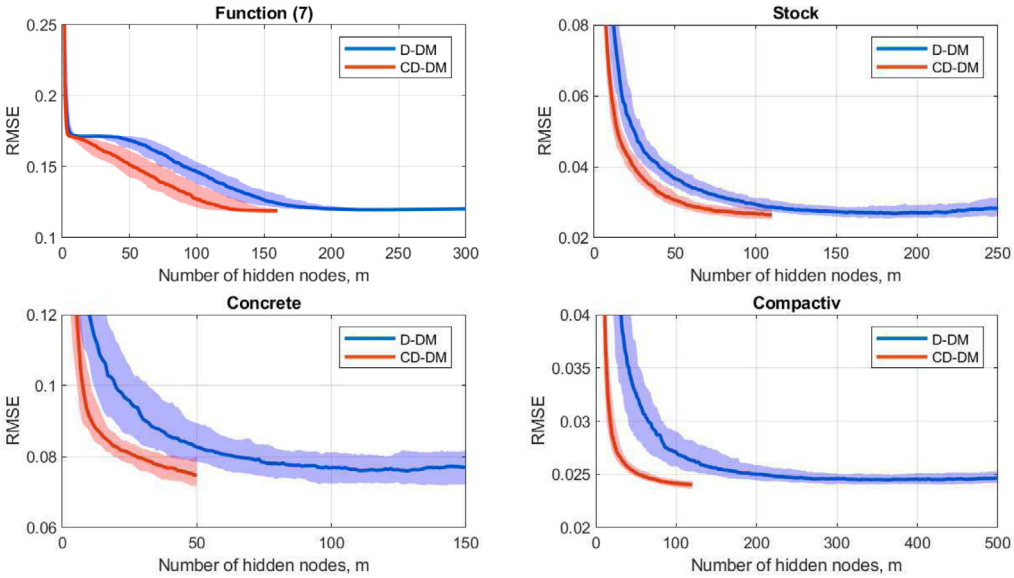

Fig. 8. RMSE as a function of the number of hidden nodes for D-DM and the proposed CD-DM (shaded regions are 10th and 90th percentiles, measured over 100 trials).

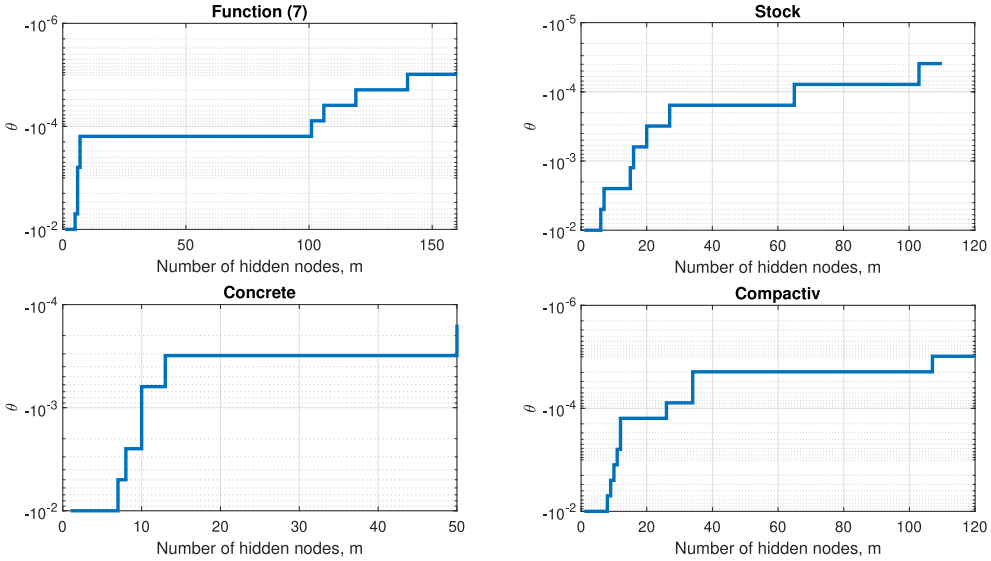

Fig. 9. Threshold θ in the proposed CD-DM.

Table 4
Descriptive statistics of errors E.

Dataset name	mean(E)	median(E)	std(E)	Skewness	Kurtosis
Auto MPG	−2.58E−04	−1.49E−03	0.0335	0.57	5.17
Treasury	−2.95E−04	−7.41E−04	0.0118	0.48	21.55
Kinematics 32NH	−8.09E−05	−3.31E−03	0.1290	0.16	3.22
Computer activity	3.77E−04	2.58E−03	0.0257	−0.49	8.09
Triazines	3.56E−03	1.93E−02	0.2080	−0.58	4.71
Pyramidines	6.62E−03	−3.92E−03	0.2521	1.37	26.08
Machine CPU	3.06E−04	−1.03E−03	0.0618	−0.80	74.20
Puma32NH	−1.62E−03	−1.02E−03	0.1249	−0.04	3.47
Bank32nh	−8.86E−04	−9.63E−03	0.0986	0.89	6.84
Elevators	−5.53E−05	−1.46E−03	0.0347	0.50	17.27

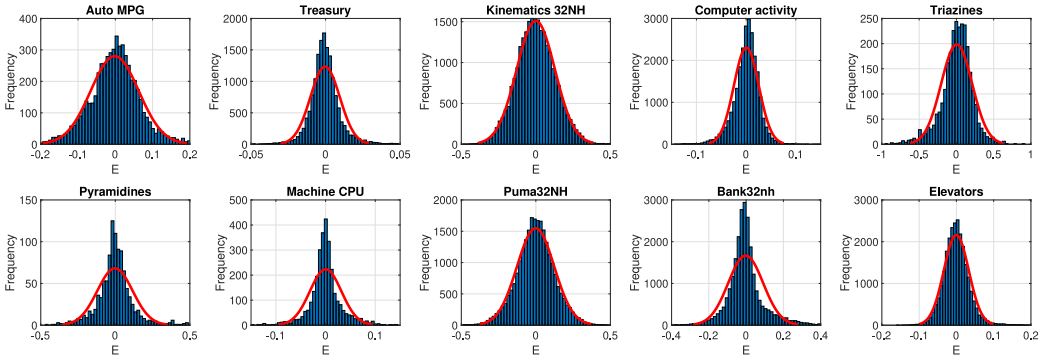

Fig. 10. Fitting error distributions for CD-DM.

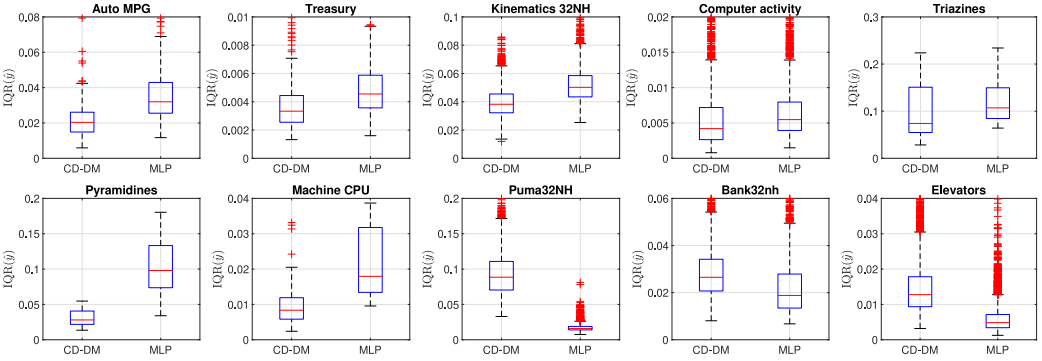

Fig. 11. Repeatability of results for CD-DM.

Puma32NH, and Elevators. For other datasets, they are characterized by a more slender distribution than the normal distribution. Unfortunately, in no case the distributions pass the test for normality (p-value of Anderson–Darling test in all cases was below 0.0005). Table 4 shows basic descriptive statistics of errors E. The underlined cases in this table indicate mean values of E that are not significantly different from 0. The most skewed error distributions are those for Machine CPU and Bank32nh (|Skewness|> 0.8). The least skewed is the distribution for Puma32NH (Skewness $=$ -0.04). In all cases, the kurtosis are greater than 3, which means leptokurtic distributions with fatter tails than for the normal distribution (more outliers). The greatest kurtosis are observed for Machine CPU, Pyramidines, Treasury, and Elevators.

To assess the repeatability of the result in CD-DM, we repeated 50 training runs for each dataset, using the same division of datasets into training and test parts for each run (to avoid the impact of data shuffling on the result repeatability). Fig. 11 shows the variability of predictions ($\hat{y}$) for each dataset. Each box represents the distribution of IQR($\hat{y}$). The results for MLP are also shown for comparison. As can be seen from this figure, the greatest variation in the results was for Triazines, which is a highly dimensional dataset (60 features) containing only 186 samples. The lowest variation in the results was for Treasury and Computer activity datasets. There was less variability for CD-DM than for MLP in seven out of ten datasets. In CD-DM, result variability is produced by randomly selected points $\mathbf{x}^*$ which determine the CD-DM hidden node parameters, while in MLP it is produced by the initial values of weights and training algorithm (including dataset random division into training and validation parts for early stopping) that determine the training trajectory and the final local minimum.

The optimization process of CD-DM is depicted in Fig. 12. This figure shows RMSE depending on the hyperparameters. Note that in most cases, better results are observed for higher values of m and k'. Exceptions are: Elevators, where generally lower errors are observed for lower values of m and k', Pyramidines, where for a higher number of nodes, the lower number of nearest neighbors brought lower errors, and Triazine and Machine CPU,

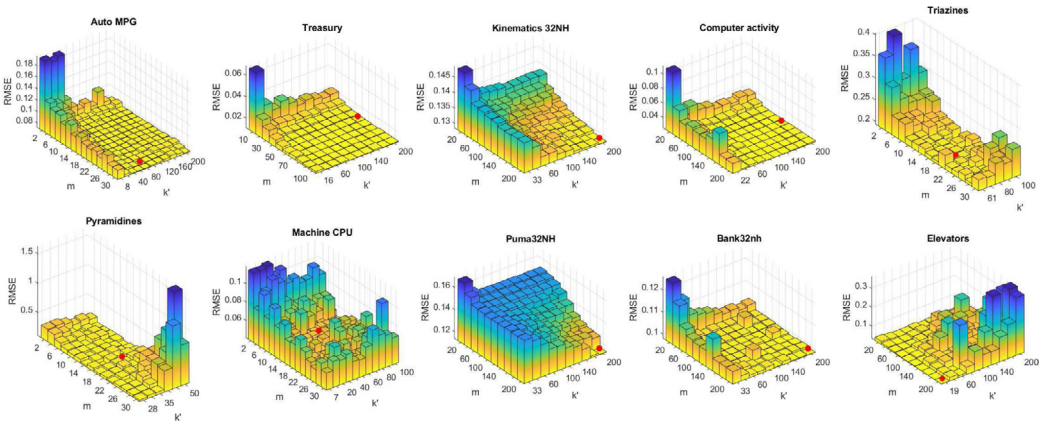

Fig. 12. RMSE depending on the number of hidden nodes (m) and the neighborhood size (k'). Red dots indicate the best solutions.

which demonstrate a very irregular pattern. Therefore, for these irregular datasets, there is no certainty that the selected hyperparameter values are optimal. But CD-DM copes well with these datasets, bringing the best result for Machine CPU compared to other methods, and the second-best result for Triazine.

3.3. Discussion

The results presented in this section show that our proposed CD-DM is very effective at solving regression problems. Its high expressive power allows it to solve nonlinear stochastic complex regression problems. The simulation study showed that CD-DM competes well against other models including randomization-based and gradient-based FNNs as well as statistical models such as GAM. In comparison to the state-of-the-art OPE-RVFL, which uses nonlinear expansions by orthogonal polynomials and ridge regression to prevent overfitting (note that all these extensions complicate the algorithm due to the additional parameters and hyperparameters to tune), our method uses simple sigmoidal AFs and no regularization (although regularization can be easily implemented in place of the Moore–Penrose pseudoinversion). The simple architecture of CD-DM gives it its interpretability. We can interpret sigmoids as local fitting components (basis functions) with slopes adjusted to the TF local slopes. The linear combination of the sigmoids performed by the output node provides the fitted function.

The learning procedure of CD-DM introduces the hidden nodes into the crucial regions of the input space, searching these regions at random. This is completely different from other randomization-based algorithms including RVFL, ELM, and OPE-RVFL, where the nodes are introduced into the random regions, in many cases outside the input hypercube (this issue was discussed in [23]). This approach wastes nodes because many of them have saturated parts which are in the input hypercube, and are completely useless for modeling TF nonlinearities. Others are introduced into the flat regions of the TF, which do not need nodes.

In gradient-based learning, both the sigmoids positions and their slopes are learned. This is a very useful, flexible feature but it should be remembered that the gradient descent method applied to do this is very sensitive to the loss function landscape and gets stuck in local minima. Our method avoids gradient-based searching, using random searching. This is cheaper in terms of the computational burden. Moreover, CD-DM does not use a fixed number of hidden nodes but introduces successively new nodes if the TF complexity needs more nodes to be modeled with the required accuracy. This accuracy is expressed by the CD-DM hyperparameter $RMSE_{\max}$ and can be used as the stop condition for the construction process.

The strong competitors to FNNs for regression problems are statistical nonlinear models such as kernel smoothers, multivariate adaptive regression splines, or GAM. We used the latter in our simulation study. A Matlab implementation of GAM uses a gradient boosting algorithm. It builds sets of boosted trees for linear terms for predictors and then sets of interaction trees. When building a set of trees, the function trains one tree at a time. The iterative learning process successively adds trees as we add the nodes in our CD-DM but using a different boosting mechanism. The process is very complicated and needs to select at least seven hyperparameters such as initial learning rates, maximum number of splits, number of trees and interactions. This process took a lot of time in our simulations. It was the particularly time-consuming for large datasets in comparison to the optimization time of other models. It should be noted that a variant of GAM, XGBoost, is considered as the leading model for working with standard tabular data and dominates many competitions, e.g. Kaggle. Thus, it is one of the state-of-the-art machine learning models for general use in regression and classification whose performance has been confirmed in many practical applications. Our CD-DM outperformed GAM in 7 out of 10 regression problems. Moreover, it is less complex (only two hyperparameters to tune), and can be implemented very easily (see Appendix).

Our proposed method should be further tested on a variety of regression problems, including real-world big data problems with millions of samples and hundreds of features, so that more reliable and general conclusions can be drawn.

The impact of the hyperparameters on the final results should be investigated further using different datasets. In our implementation, the number of nearest neighbors is a global hyperparameter. To achieve better performance in the case of TFs with different properties, e.g. density, variance or noise level, in different regions of the input space, hyperparameter k can be locally adjusted. One possible way of introducing locally adjusted k into the proposed algorithm is to divide the input space into subregions with different properties, and to represent these subregions by a set of prototype points or a k-d tree (or its variants such as a ball tree). Each prototype or leaf node of the k-d tree is labeled with k dependent on the data properties (density, TF variance and noise) in the subregion it represents. In the simplest case, the prototype points can be determined using a clustering algorithm. The key to this approach is input space partitioning and labeling, i.e. assigning k to prototypes or leaf nodes. For locally adjusted hyperparameters k, the CD-DM algorithm shown in Algorithm 1 changes only slightly. It includes a new step between (a) and (b): "Find the nearest prototype to $\mathbf{x}^*$ and read k from its label" or, alternatively, "Input $\mathbf{x}^*$ to the k-d tree and read k from the label of the leaf node reached by $\mathbf{x}^*$".

Another issue that should be considered in further research is the impact of the first node on the final solution. The first node is randomly introduced into the input hypercube. Each subsequent node is dependent on the previously accepted nodes. Thus, the first node determines the subsequent ones. It seems that this is not a strong determination, but it should be examined in the context of TF properties such as complexity or noise level.

In our implementation, we use the training points as $\mathbf{x}^*$. This is not the only solution. As points $\mathbf{x}^*$ we can select randomly any points in the input hypercube or we can take the prototypes of the training data clusters. We can also select them from the regions where the TF is the most variable or steep [23]. The problem of $\mathbf{x}^*$ selection also needs to be considered in further research.

The proposed CD-DM reduces to D-DM when we set $\theta = 0$. In such a case, each candidate node is accepted, regardless of whether it improves the result or not. The algorithm is more random in the D-DM version. It introduces nodes in random regions without selecting their best positions from the perspective of the current advancement of the construction process.

The proposed solution is restricted to a single hidden layer FNN. This gives it readability, interpretability, and speed of learning. Although deep architecture is more flexible, it also has its drawbacks. Some works show that shallow randomized NNs can be advantageous over deep ones in some cases, especially in modeling large-volume and time-varying systems [34]. The universal approximation property of shallow NNs justifies their use in modeling any nonlinear function [5]. Note the simplicity of the proposed solution which does not need backpropagation using tedious and time-consuming gradient calculations. The architecture of CD-DM can be coded in around 40 lines of code in standard Matlab syntax, as shown in the code listing presented in Appendix.

4. Conclusion

The key issue in FNNs with random hidden nodes is to generate the random weights and biases in such a way as to ensure good approximation properties of the network. The standard way of generating both parameters from the same fixed interval leads to weak performance, especially for complex target functions. The data-driven mechanism of randomized FNN learning proposed recently in [29] adjusts the random parameters to the local features of the target function and so improves the accuracy of fitting. First, the method randomly selects the input space regions by drawing the points from the training set. Then, the hyperplanes are fitted to the neighborhoods of the selected points and their coefficients are transformed into the sigmoid weights and biases. This results in the placement of the sigmoids in the selected regions of the input space and the adjustment of their slopes to local fluctuations in the target function.

In this work, we propose a new constructive approach for data-driven FNN randomized learning. This alternative paradigm to train FNN constructs iteratively the network architecture by successively adding new hidden nodes. These are accepted or rejected depending on the error. A node is accepted only if the error reduction is greater than the threshold. A low initial value of the threshold accepts only those nodes which ensure a rough function approximation. In the next steps, the threshold is increased successively which leads to the acceptance of those nodes which more accurately model the details of the target function. To prevent overfitting the number of hidden nodes is limited to a value estimated in the cross-validation. The proposed constructive approach can also be employed for RVFL based networks.

As simulation studies have shown, the proposed constructive method leads to faster convergence with the number of nodes and more compact network architecture than its predecessor, D-DM, as it includes only "significant" nodes. It outperformed the standard randomization-based NNs such as RVFL and ELM as well as the state-of-the-art OPE-RVFL. In most cases, it achieved lower fitting errors than gradient-based learning and GAM.

Future work will focus on further analysis and improvement of the proposed method as well as other methods from this family, and their adaptation to classification problems. Ensembles of randomized FNNs and their deep variants are promising directions for further research [35].

CRediT authorship contribution statement

Grzegorz Dudek: Conceptualization, Methodology, Formal analysis, Validation, Visualization, Writing – original draft, Writing - review & editing, Supervision, Software, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by Grant 2017/27/B/ST6/01804 from the National Science Centre, Poland. I am very grateful to Professor Milica Petrovic with University of Belgrade for supplying data listed in Table 2. I am also grateful to the anonymous Reviewers whose comments helped improve this work.

Appendix. CD-DM implementation

The source code is available here: https://github.com/GMDudek/CD-DM.

```
function [a, b, beta] = CDDM(X, Y, m, k, ...
    theta, Q, q_max, RMSE_max)
%CDDM - constructive data-driven method ...
    for randomized learning of FNNs
%
% X - inputs, n x N, n - #features, N - ...
    #samples
% Y - target outputs, N x 1
% m - number of hidden nodes
% k - number of nearest neighbours, k≥n
% theta - initial value of the threshold ...
    for the error change, theta≤0
% Q - threshold for adaptation of theta
% q_max - maximum number of the ...
    consecutive unaccepted candidate nodes ...
    for stop condition
% RMSE_max - maximum allowable error for ...
    stop condition
% a - hidden node weights, n x m
% b - hidden node biases, 1 x m
% beta - output weights, m x 1

[n,N] = size(X);

a = nan(n,m);
b = nan(1,m);
h = nan(N,m);
rmse = nan(1,m);
drmse = nan(1,m);

d = dist(X); %distance between input points
[~,is] = sort(d); %indices of the nearest ...
    neighbours

i = 1; l = 1; q = 1; rmse1 = RMSE_max + 1;
```

```
while (i ≤ m) && (rmse1 > RMSE_max) && (q ...
    < q_max) %main loop

    if rem(l,N) == 1
        ig = randperm(N,m); %choose ...
            randomly x*-points
        xg = X(:,ig); %x*
        ik = is(1:k,ig); %indices of the k ...
            nearest neighbours of x*
    end

    xp = [ones(k,1) X(:,ik(:,i))'];
    yp = Y(ik(:,i));
    xp = xp + rand(size(xp)) * 1e-10; %to ...
        avoid numerical errors
    ap = xp \ yp; %hyperplane fitting to ...
        the neighborhood

    a(:,i) = 4 * ap(2:end); %hidden node ...
        weights
    b(i) = -a(:,i)' * xg(:,i); %hidden ...
        node bias
    h(:,i) = 1 ./ (1 + exp(-(a(:,i)' * X + ...
        b(i)))); %hidden layer output

    beta = pinv(h(:,1:i)) * Y; %output weights

    fr = h(:,1:i) .* repmat(beta',N,1);
    Y1 = sum(fr,2); %predicted output
    rmse(i) = (mean((Y1 - Y).^2))^0.5; %RMSE

    if i > 1
        drmse(i) = rmse(i) - rmse(i-1); ...
            %error change
    end
    q = q + 1;

    %add node if i == 1 or error reduction ...
        over theta
    if (i == 1) || (drmse(i) ≤ theta)
        rmse1 = rmse(i);
        i = i + 1;
        q=1;
    end

    if rem(q,Q)==0
        theta = theta / 2; %theta is halved
    end

    l = l + 1; %main loop counter
end
end



function [Yp] = CDDMpredict(a, b, beta, X)
%CDDMpredict - prediction function for CCDM
%
% a - hidden node weights, n x m
% b - hidden node biases, 1 x m
% beta - output weights, m x 1
% X - inputs, n x N, n - #features, N - ...
    #samples

m = length(b);
[n,N] = size(X);
h = nan(N,m);

for i=1:m
    h(:,i) = 1 ./ (1 + exp(-(a(:,i)' * X + ...
        b(i)))); %hidden layer output
end

fr = h .* repmat(beta',N,1);
Yp = sum(fr,2); %predicted output

end
```

References

[1] J. Principe, B. Chen, Universal approximation with convex optimization: Gimmick or reality? IEEE Comput. Intell. Mag. 10 (2015) 68–77.

[2] W.F. Schmidt, M.A. Kraaijveld, R.P.W. Duin, Feedforward neural networks with random weights, in: Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol. II. Conference B: Pattern Recognition Methodology and Systems, 1992, pp. 1–4, https://doi.org/10.1109/ICPR.1992.201708.

[3] P.N. Suganthan, R. Katuwal, On the origins of randomization-based feedforward neural networks, Appl. Soft Comput. 105 (2021) 107239.

[4] Y. Pao, G. Park, D. Sobajic, Learning and generalization characteristics of the random vector functional-link net, Neurocomputing 6 (2) (1994) 163–180.

[5] D. Needell, A.A. Nelson, R. Saab, P. Salanevich, Random vector functional link networks for function approximation on manifolds, 2020, arXiv:2007.15776.

[6] L. Zhang, P. Suganthan, A comprehensive evaluation of random vector functional link networks, Inform. Sci. 367–368 (2016) 1094–1105.

[7] N. Vuković, M. Petrović, Z. Miljković, A comprehensive experimental evaluation of orthogonal polynomial expanded random vector functional link neural networks for regression, Appl. Soft Comput. 70 (2018) 1083–1096.

[8] P.-B. Zhang, Z.-X. Yang, A new learning paradigm for random vector functional-link network: RVFL+, Neural Netw. 122 (2020) 94–105.

[9] D.P. Mesquita, J.P.P. Gomes, L.R. Rodrigues, S.A. Oliveira, R.K. Galvão, Building selective ensembles of randomization based neural networks with the successive projections algorithm, Appl. Soft Comput. 70 (2018) 1135–1145.

[10] R. Katuwal, P.N. Suganthan, Dropout and DropConnect based ensemble of random vector functional link neural network, in: 2018 IEEE Symposium Series on Computational Intelligence, SSCI, 2018, pp. 1772–1778, https://doi.org/10.1109/SSCI.2018.8628640.

[11] L. Tang, Y. Wu, L. Yu, A non-iterative decomposition-ensemble learning paradigm using RVFL network for crude oil price forecasting, Appl. Soft Comput. 70 (2018) 1097–1108.

[12] L. Zhang, P.N. Suganthan, Visual tracking with convolutional random vector functional link network, IEEE Trans. Cybern. 47 (10) (2017) 3243–3253.

[13] R. Katuwal, P. Suganthan, Stacked autoencoder based deep random vector functional link neural network for classification, Appl. Soft Comput. 85 (2019) 105854.

[14] P.A. Henríquez, G.A. Ruz, Twitter sentiment classification based on deep random vector functional link, in: 2018 International Joint Conference on Neural Networks, IJCNN, 2018, pp. 1–6.

[15] G. Huang, S. Song, J. Gupta, C. Wu, Semi-supervised and unsupervised extreme learning machines, IEEE Trans. Cybern. 44 (12) (2014) 2405–2417.

[16] S. Scardapane, D. Comminiello, M. Scarpiniti, A. Uncini, A semi-supervised random vector functional-link network based on the transductive framework, Inform. Sci. 364–365 (2016) 156–166.

[17] Y. Peng, Q. Li, W. Kong, F. Qin, J. Zhang, A. Cichocki, A joint optimization framework to semi-supervised RVFL and ELM networks for efficient data classification, Appl. Soft Comput. 97 (2020) 106756.

[18] D. Husmeier, Random vector functional link (RVFL) networks, in: Neural Networks for Conditional Probability Estimation: Forecasting beyond Point Predictions, Springer-Verlag London, 1999, pp. 87–97.

[19] L. Zhang, P. Suganthan, A survey of randomized algorithms for training neural networks, Inform. Sci. 364 (2016) 146–155.

[20] W. Cao, X. Wang, Z. Ming, J. Gao, A review on neural networks with random weights, Neurocomputing 275 (2018) 278–287.

[21] M. Li, D. Wang, Insights into randomized algorithms for neural networks: Practical issues and common pitfalls, Inform. Sci. 382–383 (2017) 170–178.

[22] D. Wang, M. Li, Stochastic configuration networks: Fundamentals and algorithms, IEEE Trans. Cybern. 47 (10) (2017) 3466–3479.

[23] G. Dudek, Generating random weights and biases in feedforward neural networks with random hidden nodes, Inform. Sci. 481 (2019) 33–56.

[24] G. Dudek, Improving randomized learning of feedforward neural networks by appropriate generation of random parameters, in: I. Rojas, G. Joya, A. Catala (Eds.), Advances in Computational Intelligence, IWANN 2019, Springer International Publishing, Cham, 2019, pp. 517–530.

[25] L. Kasun, H. Zhou, G.-B. Huang, C. Vong, Representational learning with ELMs for big data, IEEE Intell. Syst. 28(6) (2013) 31–34.

[26] G. Hinton, S. Osindero, Y. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (2006) 1527–1554.

[27] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy layer-wise training of deep networks, in: Proc. NIPS'06, 2006, pp. 153–160.

[28] J. Zhang, J. Wu, Z. Cai, B. Du, P. Yu, An unsupervised parameter learning model for RVFL neural network, Neural Netw. 112 (2019) 85–87.

[29] G. Dudek, Data-driven randomized learning of feedforward neural networks, in: 2020 International Joint Conference on Neural Networks, IJCNN, 2020, pp. 1–8, https://doi.org/10.1109/IJCNN48605.2020.9207353.

[30] W. Dai, D. Li, P. Zhou, T. Chai, Stochastic configuration networks with block increments for data modeling in process industries, Inform. Sci. 484 (2019) 367–386.

[31] K. Bache, M. Lichman, UCI machine learning repository, 2017 (Accessed 22 July 2016).
[32] L. Torgo, Regression DataSets, 2017, http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets. (Accessed 2 July 2016).
[33] H. Guvenir, I. Uysal, Function approximation repository, 2017, http://funapp.cs.bilkent.edu.tr/DataSets/. (Accessed 2 July 2016).
[34] C.L.P. Chen, Z. Liu, Broad learning system: An effective and efficient incremental learning system without the need for deep architecture, IEEE Trans. Neural Netw. Learn. Syst. 29 (1) (2018) 10–24.
[35] Q. Shi, R. Katuwal, P. Suganthan, M. Tanveer, Random vector functional link neural network based ensemble deep learning, Pattern Recognit. 117 (2021) 107978.