computers

MDPI

Article

Algebraic Zero Error Training Method for Neural Networks Achieving Least Upper Bounds on Neurons and Layers

Juraj Kacur

Institute of Multimedia Information and Communication Technologies, Faculty of Electrical Engineering and Information Technology STU, 812 19 Bratislava, Slovakia; juraj.kacur@stuba.sk

Abstract: In the domain of artificial neural networks, it is important to know what their representation, classification and generalization capabilities are. There is also a need for time and resource-efficient training algorithms. Here, a new zero-error training method is derived for digital computers and single hidden layer networks. This method is the least upper bound on the number of hidden neurons as well. The bound states that if there are N input vectors expressed as rational numbers, a network having $N-1$ neurons in the hidden layer and M neurons at the output represents a bounded function F: $R^D \rightarrow R^M$ for all input vectors. Such a network has massively shared weights calculated by $1 + M$ regular systems of linear equations. Compared to similar approaches, this new method achieves a theoretical least upper bound, is fast, robust, adapted to floating-point data, and uses few free parameters. This is documented by theoretical analyses and comparative tests. In theory, this method provides a new constructional proof of the least upper bound on the number of hidden neurons, extends the classes of supported activation functions, and relaxes conditions for mapping functions. Practically, it is a non-iterative zero-error training algorithm providing a minimum number of neurons and layers.

Keywords: algebraic training; mapping; least upper bound; activation functions; single hidden layer; floating points

Citation: Kacur, J. Algebraic Zero Error Training Method for Neural Networks Achieving Least Upper Bounds on Neurons and Layers. *Computers* **2022**, *11*, 74. https://doi.org/10.3390/computers11050074

Academic Editor: Seyedali Mirjalili

Received: 10 March 2022
Accepted: 29 April 2022
Published: 4 May 2022

1. Introduction

In recent years, artificial neural networks (ANN) have become important tools in many areas of research as well as in industry [1], healthcare [2], finance, media [3], etc. They process all types of data, e.g., pictures, videos, speech, audio, time series, texts, etc. They are particularly important in areas where precise mathematical or physical models are unknown or are difficult to calculate and use and this is because there are several structures of very complex ANNs [4] that can be well adapted to wide range of problems. As ANNs become a part of our lives, it is important to expand our theoretical and practical knowledge about their use.

Since the first application of ANNs, great efforts have been made to analyze their basic abilities from a theoretical and practical point of view [5–7], i.e., mainly density and complexity problems. The density aims to determine whether a particular ANN can represent, or at least approximate, a given set of functions. The complexity sets limits on how big or complex such a network must be in order to perform certain representations or approximations. Another major issue is the training process [8], i.e., how to optimally (in terms of accuracy) and efficiently (training times, computational load) set all free ANN parameters with respect to training data. Therefore, a better understanding of theoretical and practical limits can significantly alleviate these problems. There are many useful theorems and measures for evaluating the abilities of various ANNs from different aspects, e.g., approximation, representation, generalization, and related bounds on the number of neurons, layers, free parameters, etc.

This article introduces a new zero-error training/construction process of single hidden layer feed forward networks (SLFN). This method is also the least upper bound on the number of hidden neurons. The presented method and the derived bound differ from the previously published methods mostly by introducing a natural assumption of input samples processed in digital computers, i.e., rational (floating-point) numbers. This allows to derive a new constructional least upper bound achieving zero-error mapping in a fixed number of steps. It defines a network structure with a massive weight sharing, introduces a simple training algorithm that is reduced to solving regular systems of linear equations (upper or lower triangular matrices), relaxes conditions for mapping functions, and extends supported sets of activation functions. Moreover, this process has proved to be fast and robust using only a small number of free parameters compared to similar methods. Finally, this method addresses all three major problems, i.e., density, complexity and training.

2. Related Work

2.1. Approximation Theorems for Single Hidden Layer Networks

In, e.g., [5], it was shown that continuous functions of N variables can be represented by a linear combination of $2N + 1$ different nonlinear functions; however, it is not clear what these functions look like. In [6], it was shown that a superposition of sigmoid-like functions can approximate with an arbitrary accuracy any bounded continuous function (Universal Approximation Theorem). This is directly applicable to a standard SLFN; however, it is not known how many neurons are needed. In, e.g., [7], it was further proven that any continuous non-polynomial activation function can be used. There are also several constructional proofs. In, e.g., [9], so-called Fourier neural series were used for approximation. A different approach was presented in [10], where instead of combining many hidden neurons with fixed activation functions, a single neuron and a more general sigmoidal function were used.

2.2. Representation Theorems, Bounds and Construction Methods for Single Hidden Layer Networks

It is possible to construct networks with a zero-error mapping observed on a finite number of input samples. In [11], a proof was derived of an upper bound on the number of hidden neurons in a SLFN that uses signum activation functions. It states that $N - 1$ neurons are sufficient to perform a mapping of N samples, which is also the least upper bound (LUB). In [12], this result was extended by showing that a SLFN with N hidden neurons can represent N samples implementing any bounded nonlinear activation function that has a limit in $-\infty$ or ∞. Assuming an input sample matrix has at least one row with completely different values, a training approach using N hidden neurons and the rectified linear unit (Relu) was presented in [13]. In [14], weights for N hidden neurons were generated randomly, whereas the biases were calculated with respect to inputs and weights. In, e.g., [15], it was further shown that randomly set weights and biases of N hidden neurons can perform a mapping of N samples using sigmoidal activation functions. In the domain of extreme learning machine (ELM), it was shown in [16] that a network with N hidden neurons can achieve a zero-error approximation with a probability of 1 for all N input samples. It assumes infinitely differentiable activation functions, whereas the weights and biases of hidden neurons can be randomly selected.

2.3. Approximation and Representation Theorems and Bounds for Networks with More Hidden Layers

For networks with more hidden layers, there are several articles proving they can approximate continuous functions. In [17], an approximation power of additional hidden layers was demonstrated, showing that an extra hidden layer can dramatically reduce the number of neurons. An analysis of approximation capabilities for deep ANNs can be found in [18], where the lower bounds are given by the complexity of functions to be approximated and the required accuracy. In [19], an expressive number, i.e., a number of samples that can be expressed by a network, was investigated for a two hidden layer

network. Both the lower and upper bounds on the expressive number of networks using Relu activation functions were derived.

2.4. Generalization Theorems and Bounds

There are also several measures related to generalization capabilities, such as the Vapnik–Chervonenkis (VC) dimension [20]. In [21], tighter upper and lower bounds on VC dimensions for deep ANNs with Relu activation functions were presented; however, for special activation functions, even a single hidden layer can have an infinite VC dimension. Another important measure to limit generalization errors is the Rademacher complexity, and there are several articles estimating this measure, e.g., [22]. As a consequence, various regularization techniques [23], often related to these bounds, are used in the training process to increase its generalization capabilities.

3. Construction Method and the Least Upper Bound on Single Hidden Layer Networks

Since the construction process of a zero-error mapping SLFN is the proof of the least upper bound as well, first the mapping theorem is given, followed by its constructional proof.

Theorem 1. *If $F : R^D \rightarrow R^M$ is a real bounded function defined in a subspace $S \subset R^D$ formed by a D dimensional rectangular cuboid of the size $L_1 \times L_2 \times L_3 \ldots \times L_D$ and $\exists\, k \mid L_k > 0 \wedge L_j < \infty\, \forall j$, and there is a finite number N of rational samples $\boldsymbol{x} \in S \bigcap Q^D$, then a single hidden layer feed forward network having $N - 1$ neurons in a hidden layer and M in an output layer is sufficient to realize a mapping defined by $F(\boldsymbol{x_i}) = \boldsymbol{y_i}$, $i = 1, \ldots, N$. Such a network has $D - 1 + M(N - 1)$ trainable weights and $N - 1 + M$ trainable biases that can be computed by solving $1 + M$ regular systems of linear equations. The output neurons have identity activation functions and the hidden neurons implement arbitrary activation functions f: R→R satisfying:*

$$f(x) = \left\langle \begin{array}{c} 0,\ x \leq 0 \\ R \setminus \{0\},\ x > 0 \end{array} \right. \quad or\ f(x) = \left\langle \begin{array}{c} A\ ,\ x \rightarrow \infty \\ B,\ x \rightarrow -\infty \end{array} \right. ,\ A \neq B \ \wedge \ A \neq 0$$

3.1. Constructional Proof-Preliminaries

The basic proof is primarily given for $Q^D \rightarrow R$ mapping, but it can be analogically and easily extended for more outputs, i.e., $Q^D \rightarrow R^M$; as shown later in the text. For an exact formulation for how rational and floating-point numbers can be represented, e.g., as given in the IEEE 754 standard, see Appendix A, as this is crucial for the proof. For the proof only, it is assumed that N input samples $\boldsymbol{x}_i$ are sorted in ascending order along all dimensions D, as stated in Appendix B. Finally, it is assumed the space bounding rectangular cuboid originates in the beginning of a Cartesian coordinate system (to make the proof more concise), $\Delta > 0$, $0 < step < \Delta$, $L_1 > 0$, and $\boldsymbol{x}_i(k)$ denotes k-th dimension of i-th sample. The proof is divided into several logical steps as follows.

3.2. Separating Affine Hyperplanes

This step is essential for the proof and can be easily and graphically represented in three dimensions (1D and 2D cases are omitted as they can be easily inferred). Then the process will be generalized for a D dimensional case using mathematical induction.

3.2.1. 3D Case

Since the design is geometric, see Figure 1 showing the finite precision grid bounded by a prism of the size $2 \times 2 \times 1$, i.e., $L_1 \times L_2 \times L_3$ originating at $(0, 0, 0)^T$ that contains all rational numbers with $\Delta = 1$ (a difference between grid points). The task is to construct affine planes that can gradually separate all grid points (potentially all possible input samples) one by one from the remaining ones. Such a process starts at $(0, 0, 0)^T$ then moves to $(1, 0, 0)^T$, $(2, 0, 0)^T$, $(0, 1, 0)^T$, etc., and ends at the last but one point of the bounding prism, i.e., $(1, 2, 1)^T$. It is important that once the grid point is separated, it must remain separated by all other planes in the following steps. A grid point separating plane is defined by v_{12}

and v_{123} vectors. v_{12} with a 3rd dimension equal to 0 that satisfies the property of gradual separation (see Figure 2) can be expressed as follows:

$$v_{12} = (L_1,\ 0\ ,\ 0)^{\mathrm{T}} - (0, step, 0)^{\mathrm{T}} = (L_1, -step, 0)^{\mathrm{T}} \tag{1}$$

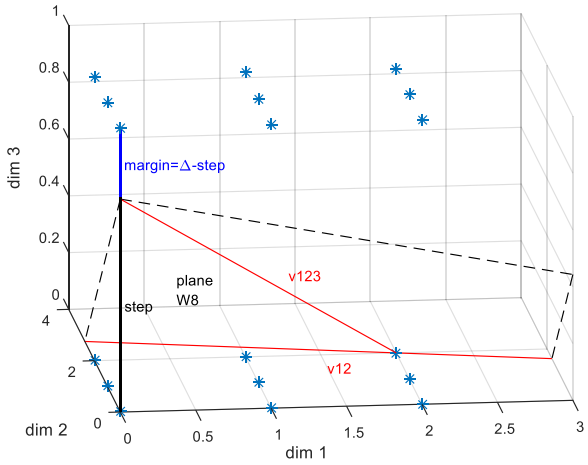


Figure 1. 3D construction of separating planes and the visualization of finite precision grid points (*) located in a $2 \times 2 \times 1$ prism, i.e., $L_1 \times L_2 \times L_3$, separating plane w_8 intersecting point $(2, 2, 0)^{\mathrm{T}}$, plane defining vectors v_{12}, v_{123}, *step*, $\Delta = 1$, and *margin*.

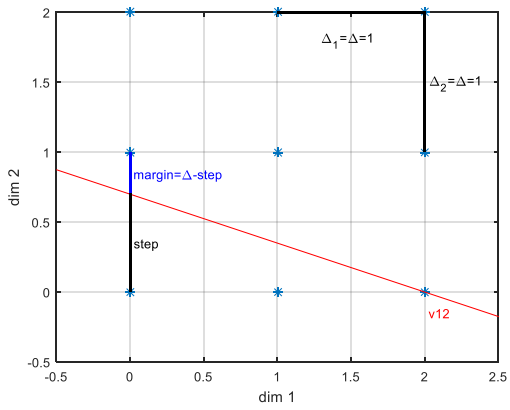


Figure 2. Construction of v_{12} located in a 2D subspace (3rd dimension = 0) forming a separating plane. v_{12} separates $(0, 0, 0)^{\mathrm{T}}$, $(1, 0, 0)^{\mathrm{T}}$, $(2, 0, 0)^{\mathrm{T}}$ from the remaining points in 2D. Visualization of finite precision grid points (*) located in a 2×2 rectangle, i.e., $L_1 \times L_2$, *step*, $\Delta = 1$, and *margin*.

It is clear from Figure 2 that *step* must be in range $(0, \Delta)$ so as not to interfere with other points having 3rd dimension equal to 0. Next, choosing v_{123} as shown in Figure 1 secures the separating plane will not intersect any points in the above parallel plane, i.e., 3rd dimension = Δ. Thus, v_{123} can be expressed as:

$$v_{123} = (L_1,\ L_2,\ 0)^{\mathrm{T}} - (0,\ 0, step)^{\mathrm{T}} = (L_1,\ L_2, -step)^{\mathrm{T}} \tag{2}$$

It is then possible to calculate a normalized normal vector $\boldsymbol{n}$ ((A4), Appendix C) of such a plane allowing a gradual separation process using (A5) as follows:

$$-\begin{pmatrix} v_{12}(2) & v_{12}(3) \\ v_{123}(2) & v_{123}(2) \end{pmatrix}\begin{pmatrix} \boldsymbol{n}(2) \\ \boldsymbol{n}(3) \end{pmatrix} = -\begin{pmatrix} -step & 0 \\ L_2 & -step \end{pmatrix}\begin{pmatrix} \boldsymbol{n}(2) \\ \boldsymbol{n}(3) \end{pmatrix} = \begin{pmatrix} L_1 \\ L_1 \end{pmatrix} \tag{3}$$

This forms a lower triangular matrix with non-zero diagonal elements, thus, the solution always exists and can be easily found. Finally, $N - 1$ separating affine planes w_i

sharing the same normal vector $\boldsymbol{n}$, each intersecting different sample $\boldsymbol{x}_i$, $i = 1, \ldots, N-1$, and multiplied by -1 are defined in (4).

$$\boldsymbol{w}_{\mathrm{i}} : -\boldsymbol{x}^{\mathrm{T}}\boldsymbol{n} - d_i = -\boldsymbol{x}^{\mathrm{T}} \begin{pmatrix} 1 \\ \boldsymbol{n}(2) \\ \boldsymbol{n}(3) \end{pmatrix} + \boldsymbol{x}_i^{\mathrm{T}} \begin{pmatrix} 1 \\ \boldsymbol{n}(2) \\ \boldsymbol{n}(3) \end{pmatrix} = 0,\ i = 1, \ldots, N-1 \tag{4}$$

For the proof only, it is assumed that N input samples $\boldsymbol{x}_i$ are sorted as stated in (A3). The -1 multiplication is applied only to swap the subspaces each plane creates, i.e., positive to negative and vice versa, which is what is needed when using common forms of activation functions, e.g., Relu, tanh, Heaviside, etc.

3.2.2. D-Dimensional Case

Using mathematical induction, assuming there are already correctly found vectors $\boldsymbol{v}_{12}$, $\boldsymbol{v}_{123}$, $\ldots$ $\boldsymbol{v}_{123\ldots m-1}$ partially defining the separating affine hyperplane (in $m-1$ dimensions), a new vector $\boldsymbol{v}_{123\ldots m}$ is constructed in the same way as before. Thus, it connects the most distant point in $m-1$ dimensional subspace, i.e., $(L_1, L_2, \ldots\ L_{m-1}, 0, \ldots, 0)^{\mathrm{T}}$ and the "begging" point located in m dimensional subspace, i.e., $(0, \ldots, 0, step_m, 0, \ldots, 0)^{\mathrm{T}}$, while introducing a proper margin given by $step_m$ (here $step_m$ is used to explicitly indicate that it is in the m-th dimension, otherwise $step$ is regarded to be the same across dimensions), i.e.,

$$\begin{gathered} \boldsymbol{v}_{12\ldots m} = (L_1, L_2, \ldots, L_{m-1},\ 0, \ldots,\ 0)^{\mathrm{T}} - (0,\ 0, \ldots,\ 0,\ step_m, 0, \ldots,\ 0)^{\mathrm{T}} = \\ (L_1, L_2, \ldots, L_{m-1}, -step_m, 0, \ldots,\ 0)^{\mathrm{T}} \end{gathered} \tag{5}$$

Once having $\boldsymbol{v}_{123\ldots m}$, it is possible to reach points $\boldsymbol{p}_{12\ldots m}$ in m dimensional subspace from points $\boldsymbol{p}_{12\ldots m-1}$ in $m-1$ dimensional subspace simply by adding properly scaled (c) vector $\boldsymbol{v}_{123\ldots m}$ to $\boldsymbol{p}_{12\ldots m-1}$ as:

$$\begin{gathered} \boldsymbol{p}_{12\ldots m} = \left(\boldsymbol{p}_{12\ldots m-1}(1), \boldsymbol{p}_{12\ldots m-1}(2), \ldots, \boldsymbol{p}_{12\ldots m-1}(m-1),\ \ 0, \ldots\ 0\right)^{\mathrm{T}} + c\boldsymbol{v}_{12\ldots m} = \\ (k_1\Delta, k_2\Delta, \ldots, k_{m-1}\Delta,\ 0,\ \ldots,\ 0)^{\mathrm{T}} + c\boldsymbol{v}_{12\ldots m} = \\ (k_1\Delta + cL_1,\ k_2\Delta + cL_2, \ldots,\ k_{m-1}\Delta + cL_{m-1},\ c(-step_m), 0, \ldots\ 0)^{\mathrm{T}} \end{gathered} \tag{6}$$

Now, to prove the existence of the separation property, it is sufficient to show that any grid point in the m-th dimension closest to the $m-1$ subspace (minimal distance = Δ), i.e., $(k_1\ \Delta, k_2\ \Delta, \ldots, k_{m-1}\ \Delta, \Delta, 0, \ldots, 0)^{\mathrm{T}}$ cannot be reached from the $m-1$ subspace by $\boldsymbol{v}_{123\ldots m}$ as given in (6) within the allowable grid limits. Using the assumptions, the finite precision grid limits can be applied to any valid point $\boldsymbol{p}_{12\ldots m}$ and dimension $i < m$ as:

$$0 \le \boldsymbol{p}_{12\ldots m}(i) \le L_i \rightarrow\ 0 \le k_i\Delta + cL_i \le L_i\ \wedge\ k_i \in \left[0,\ \frac{L_i}{\Delta}\right] \tag{7}$$

where c is calculated such that when $\boldsymbol{v}_{123\ldots m}$ is added to any grid point in $m-1$ subspace as given in (6), it just intersects Δ (minimal separation distance) in m-th dimension, i.e.,

$$c(-step_m) = \Delta\ \Rightarrow\ \ c = -\frac{\Delta}{step_m} \tag{8}$$

Substituting c into (7) yields:

$$\begin{gathered} k_i\Delta + cL_i = \Delta\left(k_i - \tfrac{L_i}{step_m}\right) \le \Delta\left(\tfrac{L_i}{\Delta} - \tfrac{L_i}{step_m}\right) < 0 \\ \text{as } step_m < \Delta\ \wedge k_i \le \tfrac{L_i}{\Delta}\ \wedge \exists i \,\Big|\, L_i > 0 \end{gathered} \tag{9}$$

The inequality in (9) violates the boundary limits for a valid grid point in i-th dimension, i.e., any grid point in dimension i must be in a close interval $[0, L_i]$, whereas (9) states it must be strictly negative for at least one dimension where $L_i > 0$. Therefore, $\boldsymbol{v}_{123\ldots m}$

enables the gradual separation process also along the newly incorporated dimension m. By gradually generating vectors as in (5) dimension by dimension, $D-1$ base vectors forming the separating affine hyperplane are found. Using (A5), the normalized normal vector $\boldsymbol{n}$ of the separating affine hyperplane is calculated as in (10):

$$\begin{pmatrix} v_{12}(2) & v_{12}(3) & \dots & v_{12}(D) \\ v_{123}(2) & v_{123}(3) & \dots & v_{123}(D) \\ \vdots & \vdots & \ddots & \vdots \\ v_{1234\dots D}(2) & v_{1234\dots D}(3) & \dots & v_{1234\dots D}(D) \end{pmatrix} \begin{pmatrix} \boldsymbol{n}(2) \\ \boldsymbol{n}(3) \\ \vdots \\ \boldsymbol{n}(D) \end{pmatrix} = \\ \begin{pmatrix} -step & 0 & \dots & 0 & 0 \\ L_2 & -step & 0 & \dots & 0 \\ L_2 & L_3 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & -step & 0 \\ L_2 & L_3 & \dots & L_{D-1} & -step \end{pmatrix} \begin{pmatrix} \boldsymbol{n}(2) \\ \boldsymbol{n}(3) \\ \boldsymbol{n}(4) \\ \vdots \\ \boldsymbol{n}(D) \end{pmatrix} = - \begin{pmatrix} L_1 \\ L_1 \\ L_1 \\ \vdots \\ L_1 \end{pmatrix} \tag{10}$$

This is a lower triangular $D-1 \times D-1$ matrix with non-zero diagonal elements, thus the solution always exists. Such an affine hyperplane having its normal vector calculated as in (10) is able to gradually separate all points in the finite precision grid and thus N input samples as well. Now this hyperplane is gradually shifted to intersect the first $N-1$ sorted samples $\boldsymbol{x}_i$. This creates $N-1$ separating affine hyperplanes as in (11), multiplied by -1 to enable the use of common activation functions:

$$\boldsymbol{w}_\mathrm{i} : -\boldsymbol{x}^\mathrm{T}\boldsymbol{n} - d_i = -\boldsymbol{x}^\mathrm{T} \begin{pmatrix} 1 \\ \boldsymbol{n}(2) \\ \vdots \\ \boldsymbol{n}(D) \end{pmatrix} + \boldsymbol{x}_i^\mathrm{T} \begin{pmatrix} 1 \\ \boldsymbol{n}(2) \\ \vdots \\ \boldsymbol{n}(D) \end{pmatrix} = 0 \tag{11}$$

3.3. Dot-Product Matrix

Having $N-1$ affine hyperplanes (11) (weights and biases of hidden neurons), a dot-product matrix $\boldsymbol{DP}$ can be constructed as in (12). It contains dot products of all input samples (extended by constant 1, i.e., input to a bias term) and vectors $(-\boldsymbol{n}^\mathrm{T}, -d_i)$ defining the separating affine hyperplanes $\boldsymbol{w}_i$, i.e., $\boldsymbol{DP}$ represents the application of weights and biases of hidden neurons to all input vectors $\boldsymbol{x}_i$:

$$\boldsymbol{DP} = - \begin{pmatrix} (\boldsymbol{x}_1^\mathrm{T}, 1) \\ (\boldsymbol{x}_2^\mathrm{T}, 1) \\ \vdots \\ (\boldsymbol{x}_N^\mathrm{T}, 1) \end{pmatrix} \left(\begin{pmatrix} \boldsymbol{n} \\ d_1 \end{pmatrix}, \begin{pmatrix} \boldsymbol{n} \\ d_2 \end{pmatrix}, \dots, \begin{pmatrix} \boldsymbol{n} \\ d_{N-1} \end{pmatrix} \right) = \\ - \begin{pmatrix} \boldsymbol{x}_1^\mathrm{T}\boldsymbol{n} + d_1 & \boldsymbol{x}_1^\mathrm{T}\boldsymbol{n} + d_2 & \dots & \boldsymbol{x}_1^\mathrm{T}\boldsymbol{n} + d_{N-1} \\ \boldsymbol{x}_2^\mathrm{T}\boldsymbol{n} + d_1 & \boldsymbol{x}_2^\mathrm{T}\boldsymbol{n} + d_2 & \dots & \boldsymbol{x}_2^\mathrm{T}\boldsymbol{n} + d_{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x}_N^\mathrm{T}\boldsymbol{n} + d_1 & \boldsymbol{x}_2^\mathrm{T}\boldsymbol{n} + d_2 & \dots & \boldsymbol{x}_N^\mathrm{T}\boldsymbol{n} + d_{N-1} \end{pmatrix} = \begin{pmatrix} 0 & \varepsilon^+ & \dots & \varepsilon^+ \\ \varepsilon^- & 0 & \dots & \varepsilon^+ \\ \vdots & \vdots & \ddots & 0 \\ \varepsilon^- & \varepsilon^- & \dots & \varepsilon^- \end{pmatrix} \tag{12}$$

Due to the gradual point by point separation of N sorted samples by $N-1$ affine hyperplanes $\boldsymbol{w}_i$, such a matrix $(N \times N-1)$ has a special structure, i.e., its main diagonal is equal to 0, samples below the diagonal are negative, for simplicity marked as ε^-, and elements above the diagonal are positive (ε^+). Next, a sufficiently small positive constant ξ_i is added to each column of $\boldsymbol{DP}$, so that the diagonal elements become positive, but still securing that the negative elements remain negative. This can be achieved by directly adding such ξ_i to d_i for each affine hyperplane, i.e.,

$$d_i = -\left(\boldsymbol{x}_i^{\mathrm{T}} \begin{pmatrix} 1 \\ \boldsymbol{n}(2) \\ \boldsymbol{n}(3) \end{pmatrix} + \xi_i \right) \tag{13}$$

A proper ξ_i can be found for each column by inspecting negative elements located below the main diagonal as in (14):

$$0 < \xi_i < \min_{j>i} \left| \boldsymbol{DP}_{j,i} \right| i = 1, \ldots, N-1 \tag{14}$$

Here $\boldsymbol{DP}_{j,i}$ denotes j-th raw and i-th column of the matrix. A single constant satisfying such requirements can be calculated by parameters of the finite precision grid and the normal vector $\boldsymbol{n}$, i.e.:

$$0 < \xi < min\{(-\Delta,\, 0,\, \ldots,\, 0)\boldsymbol{n},\ (L_1, -\Delta,\, 0, \ldots, 0)\boldsymbol{n},\ \ldots, (L_1, L_2\, , \ldots,\, L_{D-1}\, , -\Delta)\boldsymbol{n}\} \tag{15}$$

Although (15) is a faster solution, (14) is more suitable for better numerical stability.

3.4. Hidden Output Matrix

In the next step, a proper nonlinear function is applied element-wise to the adjusted $\boldsymbol{DP}$ matrix by ξ_i (14), thus producing a hidden output matrix $\boldsymbol{H}$. Such a matrix ($N \times N-1$) contains outputs of all hidden neurons for all input samples $\boldsymbol{x}_i$. In this case two broad classes of functions can be used. For simplicity, first consider functions satisfying the definition in (16):

$$f(x) = \begin{cases} 0, & x \leq 0 \\ R \setminus \{0\}, & x > 0 \end{cases} \tag{16}$$

Such functions, when applied to the adjusted $\boldsymbol{DP}$ matrices, produce upper triangular matrices with non-zero diagonal elements. In the following steps, this ensures the existence of a unique solution, e.g., in the case of the Heaviside function satisfying (16), this results in the hidden output matrix $\boldsymbol{H}$, as shown in (17):

$$\boldsymbol{H} = Heavisde\left(\begin{pmatrix} \xi_1 & \varepsilon^+ + \xi_2 & \ldots & \varepsilon^+ + \xi_{N-1} \\ \varepsilon^- + \xi_1 & \xi_2 & \ldots & \varepsilon^+ + \xi_{N-1} \\ \vdots & \vdots & \ddots & \xi_{N-1} \\ \varepsilon^- + \xi_1 & \varepsilon^- + \xi_2 & \ldots & \varepsilon^- + \xi_{N-1} \end{pmatrix} \right) = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ 0 & 1 & \ldots & 1 \\ \vdots & \vdots & \ddots & 1 \\ 0 & 0 & \ldots & 0 \end{pmatrix} \tag{17}$$

3.5. Output Neuron Weights and Biases

First, the hidden output matrix in (17) is extended from the right by a column of units, as shown in (18). This represents a constant input 1 to a bias for each training sample. The output neuron has $N-1$ inputs from hidden neurons and the bias term fed by 1 (the last column of the extended matrix $\boldsymbol{H}$). Thus, the extended matrix $\boldsymbol{H}$ is a square $N \times N$ upper triangular matrix with non-zero diagonal elements. Then, $N-1$ output weights in $\boldsymbol{w}_{21}$ and the bias term b_{21} can be calculated simultaneously using (18):

$$\begin{pmatrix} 1 & 1 & \ldots & 1 & 1 \\ 0 & 1 & \ldots & \vdots & 1 \\ \vdots & \vdots & \ddots & 1 & \vdots \\ 0 & 0 & \ldots & 0 & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{w}_{21}(1) \\ \vdots \\ \boldsymbol{w}_{21}(N-1) \\ b_{21} \end{pmatrix} = \begin{pmatrix} \boldsymbol{y}_1(1) \\ \boldsymbol{y}_2(1) \\ \vdots \\ \boldsymbol{y}_N(1) \end{pmatrix} \tag{18}$$

where $\boldsymbol{y}_{\mathrm{i}}(1)$ $i = 1, \ldots, N$, denotes outputs along the first dimension, i.e., output values of the first output neuron for all N inputs samples that realize $F(\boldsymbol{x}_{\mathrm{i}}) = \boldsymbol{y}_{\mathrm{i}}$, $i = 1, \ldots, N$. The matrix in (18) is by theory regular [24], thus there are exactly $N-1$ weights and the bias of the output neuron realizing the required mapping for all N input samples.

3.6. Comments and Extensions

For the sake of brevity, only the basic proof was provided by introducing some assumptions and limitations that, in fact, are not necessary. Thus, in the following, there are several extensions and implementation comments in the form of remarks to make the proof more general.

Remark 1. *A single hidden layer network having P hidden and M output neurons performing $R^D \rightarrow R^M$ mapping is described as:*

$$\boldsymbol{y}(j) = f_{2j}\left(b_{2j} + \sum_{i=1}^{P} \boldsymbol{w}_{2j}(i) f_{1i}\left(\boldsymbol{x}^{\mathrm{T}}\, \boldsymbol{w}_{1i} + b_{1i}\right)\right), \qquad j = 1, \ldots, M$$

Then the weights of hidden neurons, i.e., $\boldsymbol{w}_{1i}$ i = 1, ... , $N-1$ are equal to $-\boldsymbol{n}$ as calculated in (10), their biases b_{1i} are equal to $-d_i$ as derived in (13), output weights w_{2j} and biases b_{2j} are calculated as in (18), f_{1i} satisfy (16) or (19), f_{2j} are identity functions, and P = N − 1. Such a network has $N-1$ neurons in the hidden layer and M neurons in the output one.

Remark 2. *A unique solution always exists if (18) is in the form of an upper triangular matrix with non-zero diagonal elements. Given the form of the hidden output matrix as appeared in (17) and prior to the use of an activation function, it is obvious that this can be easily achieved by all activation functions satisfying (16), i.e., set all negative (under diagonal) elements to 0 and secure the diagonal (positive) elements remain non-zero. Another class of activation functions ensuring the extended matrix **H** is regular is defined in (19):*

$$f(x) = \left\langle \begin{array}{c} A,\ x \rightarrow \infty \\ B,\ x \rightarrow -\infty \end{array} \right. ,\ A \neq B \ \wedge \ A \neq 0 \tag{19}$$

*By multiplying elements of the **DP** matrix by a positive constant α and applying f(x) from (19), the extended matrix **H**, as appeared in (18), can be written as follows:*

$$\begin{pmatrix} f(\alpha\xi_1) & f(\alpha(\varepsilon^+ + \xi_2)) & \ldots & f(\alpha(\varepsilon^+ + \xi_{N-1})) & 1 \\ f(\alpha(\varepsilon^- + \xi_1)) & f(\alpha\xi_2) & \ldots & \vdots & 1 \\ \vdots & \vdots & \ddots & f(\alpha\xi_{N-1}) & \vdots \\ f(\alpha(\varepsilon^- + \xi_1)) & f(\alpha(\varepsilon^- + \xi_2)) & \ldots & f(\alpha(\varepsilon^- + \xi_{N-1})) & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{w}_{21}(1) \\ \vdots \\ \boldsymbol{w}_{21}(N-1) \\ b_{21} \end{pmatrix} = \begin{pmatrix} \boldsymbol{y}_1(1) \\ \boldsymbol{y}_2(1) \\ \vdots \\ \boldsymbol{y}_N(1) \end{pmatrix}$$

*This multiplication constant is applied to the normal vector (10) and to biases/shifts d_i (13). Based on Lemma A2 (*Appendix D*), there exist α for which the rank of **H** becomes N, i.e., such a matrix is regular and can be used to solve (18). This class of functions comprises sigmoid-type functions that play important theoretical and practical roles in ANNs.*

Remark 3. *The multiplication of affine hyperplanes by −1, as applied in (4) and (11), is completed only to allow a direct use of Relu, Heaviside, and sigmoid-type activation functions that are very common. If this is not needed it can be skipped. Then, however, a different class of activation functions must be used. Such a class of function would be redefined to g(x) = f(−x) as given in (16) and (19), and large enough, but still negative constants would have to be added to columns of the dot-product matrix (17). This is to secure negative diagonal and positive below-diagonal elements prior to the application of g(x). This secures **H** to be regular in the case of (16) or for some positive α in the case of (19).*

Remark 4. *The construction of the dot-product matrix required input samples to be sorted across dimensions. It should be noted that this was only to show that such a matrix is upper triangular with non-zero diagonal elements, and therefore must be regular; however, a permutation of rows (samples) as well as columns (hyperplanes) can be realized by elementary matrix operations, and therefore*

even such an "unordered" matrix is also regular [24]. *Thus, this step is not necessary, although it can significantly simplify the computation of (18). In this case, the input sample* x_i *satisfying:*

$$\boldsymbol{x}_j^{\mathrm{T}} \begin{pmatrix} 1 \\ w_2 \\ \vdots \\ w_d \end{pmatrix} - \boldsymbol{x}_i^{\mathrm{T}} \begin{pmatrix} 1 \\ w_2 \\ \vdots \\ w_d \end{pmatrix} \leq 0, \ \forall\, j = 1, \ldots, N \mid i \neq j,$$

is omitted in the construction of $N - 1$ *hidden layer neurons.*

Remark 5. *The basic proof primarily addressed* $Q^D \rightarrow R$ *mapping. Nevertheless, it can be easily extended to* $Q^D \rightarrow R^M$ *mapping. In this case, the construction of separating affine hyperplanes and the hidden output matrix, as appeared in (18), remain unchanged, i.e., the hidden layer as a whole. Now, the network has M output neurons, while the hidden ones remain intact. Thus, only the weights and biases of additional output neurons must be calculated independently of each other, e.g., in the case of Heaviside function by:*

$$\begin{pmatrix} 1 & 1 & \ldots & 1 \\ 0 & 1 & \ldots & 1 \\ \vdots & \vdots & \ddots & 1 \\ 0 & 0 & \ldots & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{w}_{2j}(1) \\ \vdots \\ \boldsymbol{w}_{2j}(N-1) \\ b_{2j} \end{pmatrix} = \begin{pmatrix} \boldsymbol{y}_1(j) \\ \boldsymbol{y}_2(j) \\ \vdots \\ \boldsymbol{y}_N(j) \end{pmatrix}, j = 2, \ldots, M \tag{20}$$

where $y_i(j)$ $i = 1, \ldots, N$, *denotes N output values for j-th output neuron (j-th dimension of y). Therefore, a SLFN performing* $Q^D \rightarrow R^M$ *mapping of N samples has* $N - 1$ *hidden and M output neurons. There are altogether* $D - 1 + M(N - 1)$ *trainable weights and* $N - 1 + M$ *trainable biases. To calculate all free parameters, it is enough to solve M systems of linear equations as in (20) and one system given in (10), i.e., all expressed as triangular matrices.*

4. Results

This construction method was tested with respect to two aspects, i.e., its practical limitations and in comparison to similar methods. In addition, the performance of back propagation-based (BP) training was evaluated on the test data as well.

4.1. Performance of the Proposed Method

Although Theorem 1 gives an upper bound on the number of hidden neurons, it can be directly used to train SLFNs depending on the precision of the training data and applied arithmetic. To show its abilities, three kinds of random data were generated for each test, i.e., 3000 samples in 2, 50, and 60 dimensions, satisfying $\Delta = 10^{-6}$ (a sample precision). Both classification and mapping errors were measured, whereas the output values were random samples from the set {0, 1}. Classification errors were calculated after converting ANN outputs to binary classes, i.e., $y < 0.5 \rightarrow \{0\}$, and $y \geq 0.5 \rightarrow \{1\}$. The mapping errors were evaluated by mean square errors (MSE). All calculations were realized in the Python 3.6.9 environment [25] using double float arithmetic. The results are listed in Table 1 for the tested activation functions and generated data. As can be seen, for the selected precision (Δ), bounding subspace, double precision arithmetic, and the number of samples, a reasonable limit on the number of dimensions is 50, e.g., for 60 dimensions, no network was found providing error-free classification due to numerical overflows. For a sample precision of $\Delta = 10^{-10}$ (not shown in the table) the allowable number of dimensions dropped to 30, e.g., for 40 dimensions no network was found providing a perfect classification.

Table 1. Performance of the proposed algorithm for different dimensions and activation functions in the case of 3000 training samples; precision $\Delta = 10^{-6}$, number of tests = 100.

Dimension	Input Sample Bounding Space Size	Number of Grid Points	Activation Function	Number of Trainable Parameters	Ratio of Found Nets (0 Classification Error) [%]	MSE
2	1×2	$\approx 2 \times 10^{12}$	Heaviside	6000	100	0.0
50	$1 \times 2 \times 1 \times \ldots \times 1$	$\approx 2 \times 10^{300}$	Heaviside	6048	100	0.0
60	$1 \times 2 \times 1 \times \ldots \times 1$	$\approx 2 \times 10^{360}$	Heaviside	6058	0	-
2	1×2	$\approx 2 \times 10^{12}$	Relu	6000	100	2.37×10^{-18}
50	$1 \times 2 \times 1 \times \ldots \times 1$	$\approx 2 \times 10^{300}$	Relu	6048	100	1.59×10^{-17}
60	$1 \times 2 \times 1 \times \ldots \times 1$	$\approx 2 \times 10^{360}$	Relu	6058	0	-
2	1×2	$\approx 2 \times 10^{12}$	tanh, $\alpha = 1$	6000	100	4.27×10^{-28}
50	$1 \times 2 \times 1 \times \ldots \times 1$	$\approx 2 \times 10^{300}$	tanh, $\alpha = 1$	6048	100	0.0
60	$1 \times 2 \times 1 \times \ldots \times 1$	$\approx 2 \times 10^{360}$	tanh, $\alpha = 1$	6058	0	-

4.2. Comparison to Similar Approaches

The proposed method was compared to other similar approaches, i.e., algorithms that theoretically provide zero-error mapping in a fixed number of steps, while minimizing the number of hidden neurons in a SLFN. The comparison was performed in terms of the number of neurons, the number of trainable and free parameters, training times, MSE, and the ratio of successfully found networks, i.e., networks where the maximum mapping error (absolute difference) did not exceed 10^{-9}. All algorithms were implemented as described in their articles except of [13]. It was found that a bias vector given in [13] (Equation (8)) could not be used because it had a wrong size, i.e., $N - 1$ and it should be N, when applied in [13] (Equation (9)) so that $\boldsymbol{H}$ can be invertible for all N input samples. Thus, in the present implementation, the correct bias vector was derived in the following form: $b = (-\boldsymbol{X}_{i1} + k(\boldsymbol{X}_{i1} - \boldsymbol{X}_{i2}), -\boldsymbol{X}_{i2} + k(\boldsymbol{X}_{i2} - \boldsymbol{X}_{i3}), \ldots, -\boldsymbol{X}_{iN-1} + k(\boldsymbol{X}_{iN-1} - \boldsymbol{X}_{iN}), -\boldsymbol{X}_{iN} + 1)^{\mathrm{T}}$, $0 < k < 1$. $\boldsymbol{X}$ is an input data matrix having training vectors in its columns and i is the dimension in which all samples are different (requirement of the method). This secures to produce a lower triangular matrix $\boldsymbol{H}$ in [13] (Equation (9)). In [12], the authors did not specify exact hyperparameters x_1, and x_2 that were set to 0 and 5, respectively, securing regular matrices $\boldsymbol{H}$ in all experiments.

All methods were implemented in the Python 3.6.9 environment using a double float precision. In the case of [16], the least mean square method as implemented in Python 3.6.9 was used to solve potentially irregular matrices $\boldsymbol{H}$. Training vectors were randomly generated from a hypercube with the finite precision of $\Delta = 10^{-10}$ and labels from the closed interval [0, 1]. Results for easy-to-map data, i.e., 100 vectors in 30 dimensions and a challenging case, i.e., 3000 vectors in 2 dimensions, are listed in Table 2. As can be seen, there were only two algorithms reaching the least upper bound ($N - 1$ neurons), i.e., the proposed one and that one in [11]. When considering the number of parameters that must be computed, the lowest number was reported in [16]. Nevertheless, when the number of all free parameters was regarded, the method in [13] and the proposed one were more efficient. This is because most of the weights in [16] were set randomly from the Normal distribution. The most robust methods, i.e., methods producing the desired networks in 100% of tests, were the proposed one and the method in [12]; however, when comparing the average training times, the proposed one was approx. two times faster, and together with [13] these were the fastest methods.

Table 2. Comparison of zero-error construction methods for different number of samples and dimensions; input precision $\Delta = 10^{-10}$, bounding space: hypercube of the size 1, number of tests = 100.

Methods	Dimension	Number of Samples	Number of Hidden Neurons	Number of Trainable Parameters	Ratio of Found Nets (Approx. Error $\leq 10^{-9}$) (%)	Average Training Time (ms)	MSE
proposed	30	100	99	228	100	1	3.2×10^{-31}
[11]	30	100	99	3169	100	6	1.2×10^{-28}
[12]	30	100	100	3070	100	2.5	1.3×10^{-29}
[16]	30	100	100	100 3200 *	100	5	1.25×10^{-21}
[13] revised	30	100	100	200	93	1	1.67×10^{-21}
proposed	2	3000	2999	6000	100	210	1.2×10^{-28}
[11]	2	3000	2999	11,997	0	-	-
[12]	2	3000	3000	11,998	100	400	3.8×10^{-27}
[16]	2	3000	3000	3000 9000 *	0	-	-
[13] revised	2	3000	3000	6000	0	-	-

* The number of all free parameters; most of which were set randomly from the Normal distribution.

4.3. Comparison to Back Propagation-Based Approach

To assess the performance of the proposed method in the context of the ubiquitous BP algorithm, BP-based training was applied to the test data in terms of both classification and mapping tasks. Feed forward networks with one and two hidden layers were implemented and trained in the Keras (version 2.2.4) system run on PC Intel core i7, 3.5 GH, 16 GB RAM, 64 bit Windows 10 professional, i.e., the same platform for all tests. Several optimizers (Adam, SGD, RmsProbs), in combination with loss functions (MSE, cross entropy) and learning rate schemes were tested. Here, the best performing settings (on average) were used, i.e., Adam optimizer, MSE loss, exponential learning decay (0.001–0.0001), maximum number of epochs 1.5×10^4, and Relu activation functions. Training vectors were randomly generated from a hypercube with a finite precision of $\Delta = 10^{-6}$; no networks were successfully trained for finer precisions, e.g., $\Delta = 10^{-10}$, partly due to numerical limitations. In the case of the classification tests, binary outputs were generated randomly, whereas in the mapping tests the outputs were random float numbers from the interval [0, 1]. The results, i.e., the ratios of networks with zero-classification errors, ratios of networks having mapping errors not greater than 10^{-3}, training times for classification and mapping tests, and the number of free parameters, for the different network structures and the tested data are listed in Table 3. It can be seen that for the challenging case (3000 2D samples) no network having a zero-classification error was found, while networks with one and two hidden layers and a wide range of neurons were tested. The same holds for the mapping tests (max. mapping error $\leq 10^{-3}$), both for the easy-to-map case (100 30D samples) and the challenging one. The most successful networks for 100 30D samples and the classification task were SLFNs with a number of neurons ranging from 1000 to 5000. In that case the average training time was 450 ms, requiring approx. 54 training epochs (not listed in the table).

Table 3. Performance of BP-based training on different data and network structures (number of neurons in hidden layers); input finite precision $\Delta = 10^{-6}$, max. number of epochs 1.5×10^4, number of tests 20.

Neurons in Hidden Layers	Dimension	Samples	Free Parameters	Found Nets (Zero Classification Error) (%)	Average Training Time (Zero Classification Error) (ms)	Found Nets (Approx. Error $\leq 10^{-3}$) (%)	Average Training Time (Approx. Error $\leq 10^{-3}$) (ms)
[100]	30	100	3201	75	587	0	- *
[1000]	30	100	32,001	85	408	0	-
[3000]	30	100	96,001	95	446	0	-
[5000]	30	100	160,001	90	497	0	-
[100, 10]	30	100	4121	75	603	0	-
[500, 10]	30	100	20,501	85	496	0	-
[1000, 10]	30	100	41,021	80	467	0	-
[3000]	2	3000	12,001	0	-	0	-
[6000]	2	3000	24,001	0	-	0	-
[10,000]	2	3000	40,001	0	-	0	-
[3000, 100]	2	3000	$\approx 3 \times 10^5$	0	-	0	-
[6000, 100]	2	3000	$\approx 54 \times 10^5$	0	-	0	-

* Training times not provided for networks that did not converge below the pre-set threshold.

5. Discussion

Based on the proofs, analyses and experiments, several findings and comments can be made as follows:

The assumed constrain, i.e., finite number of rational samples located in a finite size space, is very weak and not limiting when using a standard representation of numbers in digital computers (floating points). Thus, this construction and the least upper bound focus directly on data as they appear in digital computers.

The number of neurons increases linearly with the number of training samples. For a fully connected SLFN, the number of free parameters would be $N(D + M + 1) - D - 1$; however, due to the massive weight sharing in the hidden layer (the same normal vector) and the hyperplane normalization, there are only $(N - 1)(M + 1) + D - 1 + M$ free parameters. Therefore, the sample dependent growth of free parameters is significantly lower, i.e., $(N - 1)(M + 1)$ compared to $N(D + M + 1)$ as in a standard SLFN.

The shared hidden layer weights are given only by the input data format (10).

Several limit testing experiments have shown that this constructional bound can be used directly to train networks on various samples, e.g., it was capable to map 3000 50-dimensional samples expressed to 6 decimal places using Heaviside, Relu and tanh (with $\alpha = 1$) functions. Such calculations required currently the standard double float precision. The worst MSE was observed for the Relu function, but was still less than 1.6×10^{-17}; in the case of Heaviside function the displayed MSE was 0.0, i.e., approx. less than 10^{-320} as tested in the Python 3.6.9 environment.

In practice, such an approach has its numerical limitations, therefore it cannot be safely used for high dimensional data, e.g., $D > 50$ with the sample precision of $\Delta \leq 10^{-6}$ or $D > 30$ and $\Delta \leq 10^{-10}$ using double float arithmetic. These limits can be naturally extended by using, e.g., quadruple or octuple arithmetic.

In the construction process, it is naturally assumed that the precision Δ of input samples is known. If this is not the case for any reason, such a parameter can still be calculated (least common multiple) for each dimension separately, as stated in the proof of Lemma A1. Nevertheless, this can be computationally expensive, but it can still increase numerical stability.

It was not explicitly mentioned, but each hidden neuron can implement a different activation function satisfying (16), as it still secures a triangular matrix in (18).

Compared to other similar approaches (Table 2), the proposed method reaches the least upper bound ($N - 1$ neurons) as well as the approach in [11]; however, the proposed one is approx. six times faster and uses significantly fewer free parameters, e.g., in case of 100 30D samples it is 228 vs. 3169. Moreover, the proposed method is more robust, i.e., for 3000 2D samples the approach in [11] was not able to realize any random mapping in 100 tests meeting the required precision (10^{-9}). The proposed method and the method in [12] were the only ones able to find all networks in all experiments, which shows their higher robustness. Nevertheless, the proposed one is faster and uses fewer free parameters, e.g., in case of 100 30D samples it is 228 vs. 3070 and is approx. 2.5 times faster. In terms of the number of trainable parameters, the best method is in [16], which randomly sets most of the free parameters, therefore, it may not always find the exact mapping; however, considering all free parameters, the most efficient method in [13] is followed by the proposed one. Finally, given the training time, the proposed method and that one in [13] are the fastest.

A standard BP-based training was used on the test data to assess the performance of the proposed and other similar approaches. In mapping tests, i.e., where the maximal absolute error is not greater than 10^{-3}, no networks having one or two hidden layers and a wide range of neurons were found (max. number of epochs was 1.5×10^4). This shows how challenging it is for BP-based approaches to map such data while meeting this precision. On the other hand, the proposed method and the method in [12] found all networks even with maximal errors not greater than 10^{-9}. In the classification task, while deploying proper networks and easy-to-classify data (100 30D samples), the BP-based design led in most cases (85–95%) to zero-classification errors; however, such networks required a lot of neurons and a large number of free parameters compared to the proposed and other similar methods. The same findings also apply to training times that were much longer for BP-based methods, i.e., more than 100 times, requiring on average more than 50 training epochs.

Although it is obvious, it should be emphasized that such a training process produces an inconsistent estimator, as it is not aimed at generalizing training data. It is designed to provide the required mapping in a fixed number of steps and to minimize the number of neurons. Instead, BP-based approaches deploying different regularizations, e.g., [23], are vital to achieve a high degree of generalization and robustness when processing noisy or incomplete data. This is undertaken by introducing some redundancy in the form of additional neurons and free parameters; however, such an algebraic training can be used as a quick initialization before the training process that uses additional data and more complex networks (additional neurons).

6. Conclusions

The article presents a new zero-error construction method applicable to feed forward neural networks. In particular, it is focused on the construction of zero-error mapping networks having the minimum possible number of layers and neurons. The design defines the exact structure of the network, the calculation process of free parameters (weights and biases), the classes of supported activation functions, and the conditions for mapping functions that the network can express. The design is unique because it is based on the representation of numbers in digital computers processing floating-point data. Since this construction process has been shown to provide an exact mapping of N samples, this method is also the least upper bound on the number of hidden neurons. Therefore, the presented method addresses three major problems at the same time, i.e., the complexity, density, and training of feed forward networks.

The main theoretical and practical contributions can be summarized as follows:

A new design of a zero-error mapping neural network with the minimal number of neurons and layers. This design and proof are unique because they assume rational numbers (floating-point data).

The resulting structure is very efficient in terms of free parameters because all weights in the hidden layer are shared and do not depend on the input data, i.e., they are given by the data format instead.

The design and its proof constitute the least upper bound on the number of hidden neurons needed for an exact mapping/classification.

Two new and broad sets of supported activation functions, for which the proof holds, were introduced. Moreover, these sets include common functions in the domain of ANN as Relu, sigmoid, tanh, Heaviside, etc.

A very relaxed condition for mapping functions was derived, i.e., it is enough that they are bounded.

A very efficient calculation process was introduced, i.e., for an exact mapping of N samples from the D dimensional space to the M dimensional one it is enough to solve $M + 1$ regular systems of linear equations. Moreover, all these systems exist in forms of upper or lower triangular matrices, thus the process is very fast and simple.

In comparison to similar methods, the proposed one (as shown in Table 2) achieves the least upper bound (together with [11]), is robust, i.e., it found 100% of networks (together with [12]), is fast (best together with [13]) and uses few free parameters (2nd most efficient).

Funding: This research and the APC was funded by the Operational Program Integrated Infrastructure for the project: International Centre of Excellence for Research of Intelligent and Secure Information and Communication Technologies and Systems—II. Stage, ITMS code: 313021W404, co-financed by the European Regional Development Fund.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The author declares no conflict of interest.

Appendix A

The construction process assumes that input samples are rational numbers, i.e., are expressed in a finite precision. More specifically, this requirement states that each number x must be expressed as:

$$x = k_x \Delta; \;\; k \in Z \, , \Delta \; \in Q^+, \tag{A1}$$

where k_x is an integer number (index) specific for x, and Δ (rational) is a constant, i.e., the minimum possible difference between adjacent digits.

Lemma A1. *Every finite set S of N rational numbers x_i can be expressed as in (A1).*

Proof. Let there be a finite set S of N rational numbers, i.e., $\{x_1, x_2, \ldots, x_N\}$, then:

$$\begin{array}{c} \{x_1, \; x_2, \ldots, \; x_N\} = \left\{ \frac{A_1}{B_1}, \; \frac{A_2}{B_2}, \; \ldots, \; \frac{A_N}{B_N} \right\} = \left\{ \frac{A_1 c_1}{M}, \; \frac{A_2 c_2}{M}, \; \ldots, \; \frac{A_N c_N}{M} \right\} = \\ \{k_1 \Delta, \; k_2 \Delta, \; \ldots, \; k_N \Delta\}; \;\; \Delta = \frac{1}{M} \, , \; x_i \in Q, A_i, B_i, c_i, k_i, \; M \in Z \; \wedge \; B_i \neq 0 \end{array} ,$$

where M is the least common multiple of all B_i in S. □

When using computers with floating-point numbers, the condition in (A1) is always met. A floating-point number (without a sign and except special "numbers", e.g., NaN, Inf) [26], as defined in IEEE 754 standard, is expressed as:

$$x = 2^{E_x - bias} \left(1 + \sum_{i=1}^{P} b x_i 2^{-i} \right),$$

where E_x is an exponent specific for x and bx_i are precision (fraction) bits of x; for a single precision float $bias = 127$, $P = 23$ and $0 \leq E_x < 255$. This can be rewritten as follows:

$$2^{E_x - bias}\left(1 + \sum_{i=1}^{P} bx_i 2^{-i}\right) = 2^{-(P+bias)} 2^{E_x}\left(2^P + \sum_{i=1}^{P} bx_i 2^{P-i}\right) = 2^{-M} 2^{E_x} B_x = 2^{-M} k_x = \Delta k_x,\ k_x \in N^+,\ \Delta \in Q^+,$$

where k_x is a positive integer specific for a particular number x.

In the case of D dimensional samples, each dimension may have different but fixed Δ_i. Therefore, each rational sample is located only in intersection points of a fine precision grid constructed over D dimensional subspace S. Such a grid is defined by a set of Δ_i and thus each rational sample $\boldsymbol{x}$ can be addressed by a unique set of indexes k_{xi} as:

$$\boldsymbol{x}^T = (k_{x1}\Delta_1,\ k_{x2}\Delta_2,\ \ldots,\ k_{xD}\Delta_D);\ k_{xi} \in Z,\ \Delta_i\ \in Q \tag{A2}$$

For simplicity, but without any loss of generality, a unique Δ across all dimensions is assumed, i.e., $\Delta_i = \Delta$.

Furthermore, it is assumed that all finite precision samples belong to a D dimensional rectangular cuboid whose vertexes are combinations of points in the form: ($\{0, L_1\}$, $\{0, L_2\}$, $\ldots$, $\{0, L_D\}$). Thus, L_i is the length of such a cuboid along i-th dimension. The construction process in (10) assumes there is at least one dimension for which $L_i > 0$, such a dimension will be redefined as the first one.

Appendix B

In the construction process showing the solution must exists, it is assumed that the input samples are sorted by dimensions in ascending order, starting from the last dimension. Although not necessary, it simplifies the proof.

Definition A1. *D dimensional vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_i, \boldsymbol{x}_j, \ldots, \boldsymbol{x}_N$ are called to be sorted if the following inequalities hold true:*

$$i < j\ \rightarrow\ \exists\, k \in [1,\ D]\ |\ \boldsymbol{x}_i(k) < \boldsymbol{x}_j(k)\ \wedge\ \boldsymbol{x}_i(k+l) \leq \boldsymbol{x}_j(k+l)\ \forall\, l \geq 0 \tag{A3}$$

where $\boldsymbol{x}_i(k)$ denotes k-th dimension of i-th vector. This definition assumes all input vectors are unique.

Appendix C

In the construction process, normalized D dimensional affine hyperplanes $\boldsymbol{w}$, i.e., $\boldsymbol{n}(1) = 1$, are used as follows:

$$\boldsymbol{w}:\ \boldsymbol{x}(1) + \boldsymbol{n}(2)\boldsymbol{x}(2) \ldots + \boldsymbol{n}(D)\boldsymbol{x}(D) + d = 0 \tag{A4}$$

where $\boldsymbol{n}$ is the normal vector of an affine hyperplane and d is a constant (shift). Such normalization reduces the number of free parameters. Since the construction process introduces $step > 0$, there is always an affine hyperplane that can be calculated as in (10). Finally, an affine hyperplane can be defined by $D - 1$ vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{D-1}$ and a point $\boldsymbol{x}$ it intersects [24]. If the vectors are linearly independent, the normalized affine hyperplane can be calculated as:

$$\begin{pmatrix} \boldsymbol{n}(2) \\ \vdots \\ \boldsymbol{n}(D) \end{pmatrix} = -\begin{pmatrix} \boldsymbol{v}_1(2) & \boldsymbol{v}_1(3) & \ldots & \boldsymbol{v}_1(D) \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{v}_{D-1}(2) & \boldsymbol{v}_{D-1}(3) & \ldots & \boldsymbol{v}_{D-1}(D) \end{pmatrix}^{-1} \begin{pmatrix} \boldsymbol{v}_1(1) \\ \vdots \\ \boldsymbol{v}_{D-1}(1) \end{pmatrix}$$
$$d = -\boldsymbol{x}^T \begin{pmatrix} 1 \\ \boldsymbol{n}(2) \\ \vdots \\ \boldsymbol{n}(D) \end{pmatrix} \tag{A5}$$

Appendix D

Lemma A2. *Let there be a $N \times N$ matrix $\boldsymbol{M}$ of the form:*

$$\begin{pmatrix} f(\alpha x_{11}) & f(\alpha x_{12}) & \dots & f(\alpha x_{1N-1}) & 1 \\ f(\alpha x_{21}) & f(\alpha x_{22}) & \dots & \vdots & 1 \\ \vdots & \vdots & \ddots & f(\alpha x_{N-1N-1}) & \vdots \\ f(\alpha x_{N1}) & f(\alpha x_{N2}) & \dots & f(\alpha x_{NN-1}) & 1 \end{pmatrix}, \tag{A6}$$

where $x_{ij} > 0$ for $i \leq j$ and $x_{ij} < 0$ for $i > j$, and a function $f(x)$ defined as in (19). Then there is a positive constant α for which the matrix $\boldsymbol{M}$ is regular.

Proof. A square matrix is regular if its determinant is non-zero and the determinant of a triangular matrix is given as a product of its diagonal elements [24]. Multiplying the first raw of $\boldsymbol{M}$ by $f(\alpha x_{i1})/f(\alpha x_{11})$, subtracting it gradually from rows $i = 2, \dots, N$, and considering properties of $f(\alpha x)$ for $\alpha \to \infty$, $\boldsymbol{M}$ converges to:

$$\begin{pmatrix} A & A & \dots & A & 1 \\ 0 & A-B & \dots & \vdots & 1-\frac{B}{A} \\ \vdots & \vdots & \ddots & A-B & \vdots \\ 0 & 0 & \dots & 0 & 1-\frac{B}{A} \end{pmatrix}$$

Since $A \neq 0$ and $A \neq B$, diagonal elements are non-zero and therefore such a matrix is regular. Moreover, the multiplication of the first raw by B/A and the subsequent subtractions are regular operations. Thus, there exists α_0 for which $\boldsymbol{M}$ is regular for all $\alpha \geq \alpha_0$. □

References

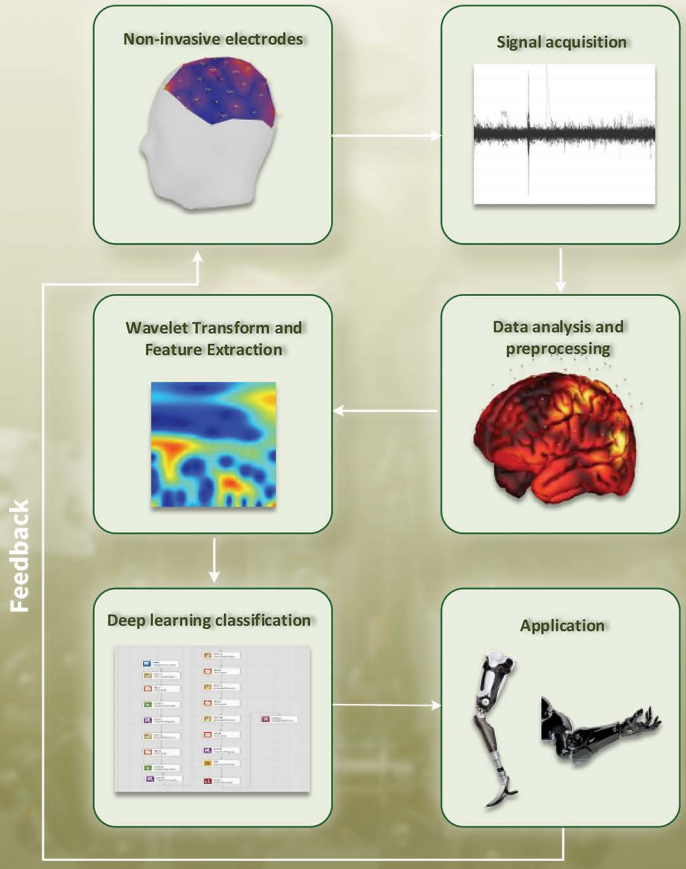
1. Regona, M.; Yigitcanlar, T.; Xia, B.; Li, R.Y.M. Opportunities and Adoption Challenges of AI in the Construction Industry: A PRISMA Review. *J. Open Innov. Technol. Mark. Complex.* **2022**, *8*, 45. [CrossRef]
2. Lötsch, J.; Kringel, D.; Ultsch, A. Explainable Artificial Intelligence (XAI) in Biomedicine: Making AI Decisions Trustworthy for Physicians and Patients. *BioMedInformatics* **2022**, *2*, 1–17. [CrossRef]
3. Di Sotto, S.; Viviani, M. Health Misinformation Detection in the Social Web: An Overview and a Data Science Approach. *Int. J. Environ. Res. Public Health* **2022**, *19*, 2173. [CrossRef] [PubMed]
4. Kastrati, M.; Biba, M. A State-Of-The-Art Survey on Deep Learning Methods and Applications. *Int. J. Comput. Sci. Inf. Secur.* **2021**, *19*, 53–63. [CrossRef]
5. Hecht-Nielsen, R. *Neurocomputing*; Addison—Wesley: Boston, MA, USA, 1990; ISBN 0201093553.
6. Haykin, S. *Neural Network—A Comprehensive Foundation*; Mamillan College Publishing Company: New York, NY, USA, 1994.
7. Leshno, M.; Lin, V.Y.; Pinkus, A.; Schocken, S. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Netw.* **1993**, *6*, 861–867. [CrossRef]
8. Song, H.; Kim, M.; Park, D.; Shin, Y.; Lee, J.G. Learning From Noisy Labels With Deep Neural Networks: A Survey. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, 1–19. [CrossRef] [PubMed]
9. Magnitskii, N.A. Some New Approaches to the Construction and Learning of Artificial Neural Networks. *Comput. Math. Model.* **2001**, *12*, 293–304. [CrossRef]
10. Guliyev, N.J.; Ismailov, V.E. A single Hidden layer Feedforward Network with only One Neuron in the Hidden Layer can Approximate any Univariate function. *Neural Comput.* **2016**, *28*, 1289–1304. [CrossRef] [PubMed]
11. Huang, S.-C.; Huang, Y.-F. Bounds on the number of hidden neurons in multilayer perceptrons. *IEEE Trans. Neural Netw.* **1991**, *2*, 47–55. [CrossRef]
12. Huang, G.B.; Babri, H.A. Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. *IEEE Trans. Neural Netw.* **1998**, *9*, 224–229. [CrossRef] [PubMed]
13. Shen, G.; Yuan, Y. On Theoretical Analysis of Single Hidden Layer Feedforward Neural Networks with Relu Activations. In Proceedings of the 34th Youth Academic Annual Conference of Chinese Association of Automation (YAC), Jinzhou, China, 6–8 June 2019; pp. 706–709. [CrossRef]
14. Ferrari, S.; Stengel, R.F. Smooth function approximation using neural networks. *IEEE Trans. Neural Netw.* **2005**, *16*, 24–38. [CrossRef] [PubMed]

15. Huang, G.-B. Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE Trans. Neural Netw.* **2003**, *14*, 274–281. [CrossRef] [PubMed]
16. Huang, G.B.; Zhu, Q.Y.; Siew, C.K. Extreme learning machine: Theory and applications. *Neurocomputing* **2006**, *70*, 489–501. [CrossRef]
17. Ronen, E.; Shamir, O. The Power of Depth for Feedforward Neural Networks. In Proceedings of the 29th Conference on Learning Theory, COLT 2016, New York, NY, USA, 23–26 June 2016; pp. 907–940.
18. Bölcskei, H.; Grohs, P.; Kutyniok, G.; Petersen, P. Optimal Approximation with Sparsely Connected Deep Neural Networks. *SIAM J. Math. Data Sci.* **2019**, *1*, 8–45. [CrossRef]
19. Kenta, I. Expressive Numbers of Two or More Hidden Layer ReLU Neural Networks. *Int. J. Netw. Comput.* **2020**, *10*, 293–307.
20. Vapnik, V.N. An Overview of Statistical Learning Theory. *IEEE Trans. Neural Netw.* **1999**, *10*, 5. [CrossRef] [PubMed]
21. Harvey, N.; Liaw, C.; Mehrabian, A. Nearly-tight VC-dimension bounds for piecewise linear neural networks. In Proceedings of the 2017 Conference on Learning Theory in Proceedings of Machine Learning Research, Amsterdam, The Netherlands, 7–10 July 2017; pp. 1064–1068.
22. Golowich, N.; Rakhlin, A.; Shamir, O. Size-Independent Sample Complexity of Neural Networks. In Proceedings of the 31st Conference on Learning Theory. In Proceedings of Machine Learning Research, Stockholm, Sweden, 6–9 July 2018; pp. 297–299.
23. Marin, I.; Kuzmanic Skelin, A.; Grujic, T. Empirical Evaluation of the Effect of Optimization and Regularization Techniques on the Generalization Performance of Deep Convolutional Neural Network. *Appl. Sci.* **2020**, *10*, 7817. [CrossRef]
24. Meyer, C.D. *Matrix Analysis and Applied Linear Algebra*; SIAM: Philadelphia, PA, USA, 2000. [CrossRef]
25. Python 3.6 Documentation. Available online: https://docs.python.org/3.6/ (accessed on 9 September 2021).
26. Kahan, W. Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic. Available online: https://people.eecs.berkeley.edu/~{}wkahan/ieee754status/IEEE754.PDF (accessed on 1 September 2021).

computers

COVERED BY EMERGING SOURCES CITATION INDEX CLARIVATE ANALYTICS

CITESCORE 3.3 SCOPUS



Robustly Effective Approaches on Motor Imagery-Based Brain Computer Interfaces

Volume 11 · Issue 5 | May 2022

MDPI

mdpi.com/journal/computers
ISSN 2073-431X